



书 名：ASP.NET 程序设计基础篇

出版日期：2001/06/15

书 号：957-2085-72-7

I S B N：957-2085-72-7

原 作 者：林煌章

[书本简介](#)

序

从事计算机信息工作也一段不算短的时间了，每次遇到新技术的出现还是会有不少的冲击，尤其面对这次重新打造的开发平台，心情真是又痛苦又兴奋。痛苦的是要重新打破旧有的观念，并且面对庞大的原始文件及各种庞大的对象与架构；兴奋的是在面对新对象及架构后，深深为新技术所提供的强悍能力及弹性喝采。虽然导入新技术势必投入时间以及其它成本来学习，但是这个新技术在上线后所带来的优点，势必远远超过先期所投入的成本。新技术并不是要带来麻烦，而是要带来许多利益与优点，更正旧技术的缺点与问题，让开发人员更轻松愉快的快速开发出功能强大、执行效率佳，以及执行稳定的解决方案。

本书是针对.NET 初学者而写作的入门书，不是以讨论新技术到底可以做到什么程度为前提，而是希望可以帮读者打好.NET 的基础，并且愉快的学习新技术。身为计算机知识工作者深深体会到导入新技术的痛苦，所以本书力求信息单纯化、技术原理化、理论实务化，并避免让读者直接面对杂乱无章、未经消化过的信息，而以大量的插图以及示意图，让各位读者可以愉快的享受新技术所带来的好处。

本书在一月就已经开始进行策划撰写，感谢微软出版社的黄鸿模先生以及谢慕萍副理，给予相当大的自主权以及写作时程；并忍受作者连封面都要自己画的吹毛求疵，导致本书无法在四月顺利付梓。感谢华彩软件教育教育中心同仁的体谅，没有在微软.NET 架构尚未确定、教材还尚未完善的状况下，要求作者教授.NET 相关课程，让作者可以完善的整理.NET 技术。感谢家人及朋友的支持及谅解，让我可以心无旁骛专心写作。最后要感谢各位读者，您的购买就是对作者最大的支持及鼓励。

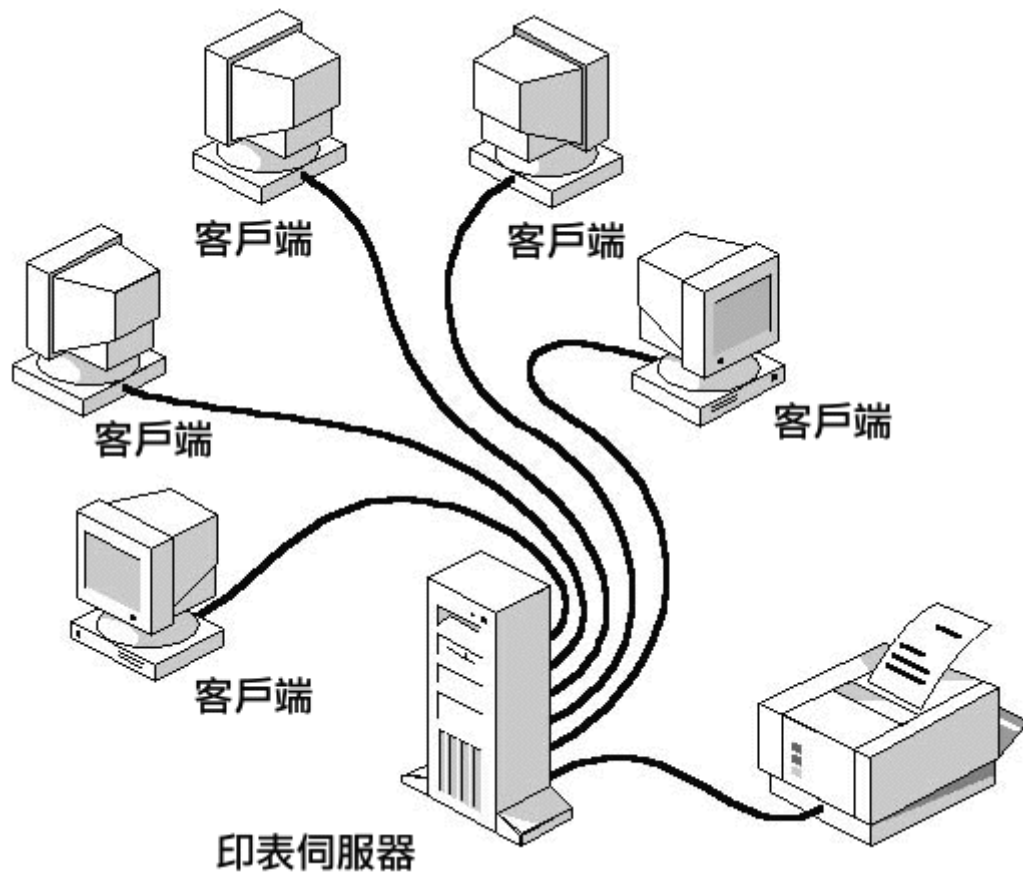
林煌章

1. ASP.NET 准备工作

基本概念

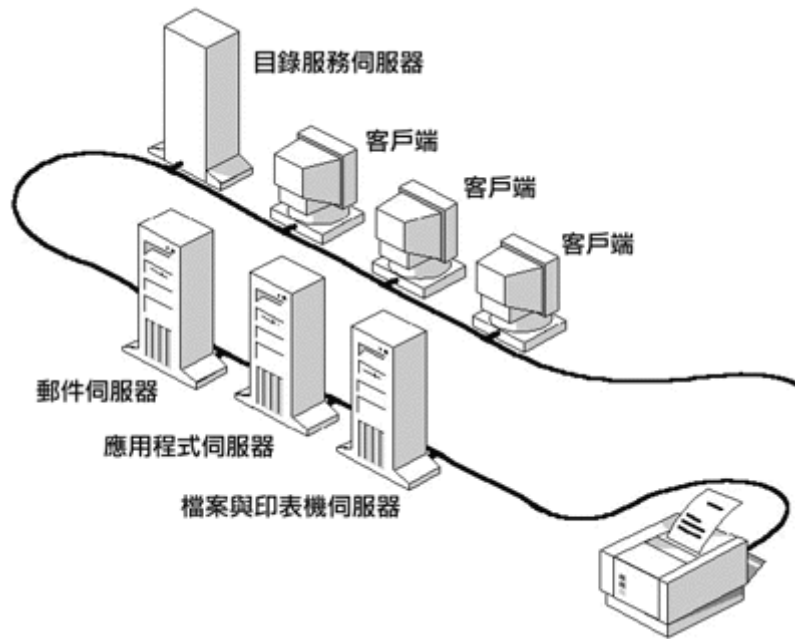
什么是客户端 / 伺服器端（Client/Server）

要了解 ASP 之前，我们先来了解 Client 及 Server 间的关系。在计算机的世界里，凡是提供服务的一方我们称为伺服器端（Server），而接受服务的另一方我们称作客户端（Client）。我们最常接触到例子是局域网络里的打印服务器所提供的打印服务：提供打印服务的计算机，我们可以说它是打印服务器；而使用打印服务器提供打印服务的另一方，我们则称作客户端。但是谁是客户端谁是伺服器端也不是绝对的，例如倘若原提供服务之伺服器端要使用其它机器所提供之服务，则所扮演之角色即转变为客户端。



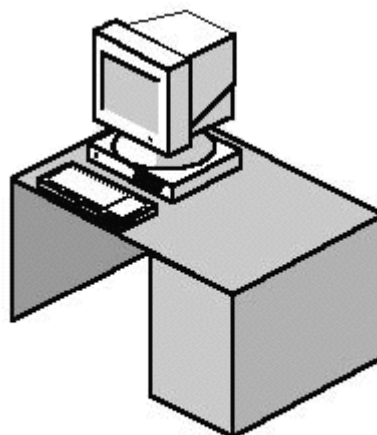
使用者向和打印服务器之间的关系。

而这种关系在因特网上，就变成使用者和网站的关系了。使用者透过调制解调器等设备上网，在浏览器中输入网址，透过 HTTP 通讯协议向网站提出浏览网页的要求（Request）。网站收到使用者的要求后，将使用者要浏览的网页数据传输给使用者，这个动作称为响应（Response）。网站提供网页数据的服务，使用者接受网站所提供的数据服务；所以使用者在这里就是客户端，响应使用者要求的网站即称为伺服器。



因特网上的 Client/Server。

不过客户端及伺服器端的关系不见得一定建立在两台分开的机器上，同一台机器中也有这种主从关系的存在。提供服务的伺服器端及接受服务的客户端也有可能都在同一台机器上，例如我们在提供网页的服务器上执行浏览器浏览本机所提供的网页，这样在同一台机器上就同时扮演伺服器端及客户端。



使用本機電腦的服務

Client/Server 都在同一台机器上。

因特网应用程序的开发

当因特网刚开始时，网站只不过是让你简单的读取一些只读的文章或档案，只有很少数的网站另外扩充一些外部的程序来协助网站的运行。这是因为当时要开发网站的功能不是用 C 语言，就是要用 CGI (Common Gateway Interface) 才能做到。用 C 及 CGI 来开发动态网站，不仅有经验的程序设计师少，而且开发的成本非常的高。另外，以前的网站都是以两层式的架构所建设，这样会让网站的延展性以及应用程序的整合性发生了一些问题。因为以前的网站程序设计师在开发网页时，完全没有顾虑到网站之外的应用，所以使用者接口和程序代码是结合在一起的。如果别的应用程序要使用这个相同的组件，或要连结一些站台一起工作，或进行些交易等的动作，程序的撰写会变得很困难。但是在 1996 年后，微软的 COM (Component Object Model) 以及 ASP (Active Server Pages) 技术让这些工作变的轻松简单。微软的 ASP 藉由简单的描述 (Script) 语言来呼叫企业法则 (企业处理数据的规则) 以及服务器上的服务程序；而 COM 的技术则让程序设计师可以轻易的利用 VB、C++，或是其它支持 COM 这种规格的程序语言，将企业法则包装到组件里，并把这些组件提供出来分享大家使用。所以现在的网站开发累积了丰富的经验，并且有大致的步骤及方法来克服一些问题，例如利用网页框架 (Frame) 将两个网站的内容显示在同一个浏览器中，这样可以使用者执行作业较为直觉及便利。但是这样做也有些缺点：例如某个网站改变了网址或是关门大吉，这些断裂连结的处理问题也蛮浪费时间。另外现在网站的开发已经不再像以前的两层式架构那么单纯，现今的网站运用已经发展为运用大量的企业法则，或是中间阶层组件的 N 层式架构和其它应用程序一起工作。所以让开发一个可靠、稳定的网站变得极富挑战，这就是为什么一个强而有利的开发工具是如此的重要。

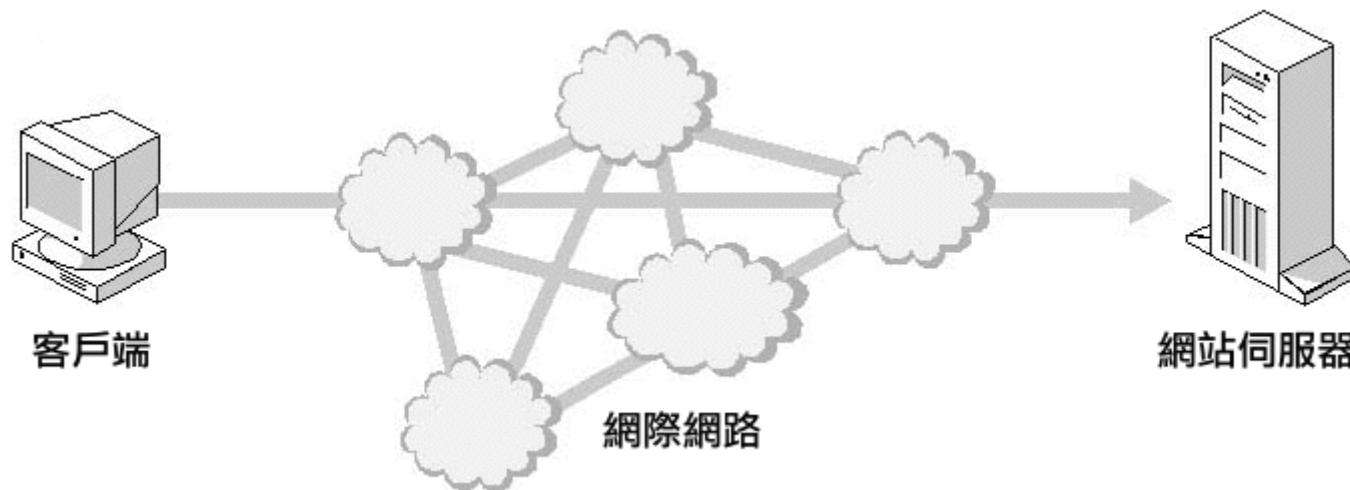
网页的种类

要让计算机成为网站服务器，需要安装 IIS (Internet Information Services) 的服务软件，后面会提到如何安装。网站服务器安装完毕后，接下来就是要设计供使用者下载的网页了。相信各位都有浏览网页的经验，不过可能还不清楚网页还有动态网页及静态网页的差别，就让我们来厘清这两种网页的差别。

静态网页

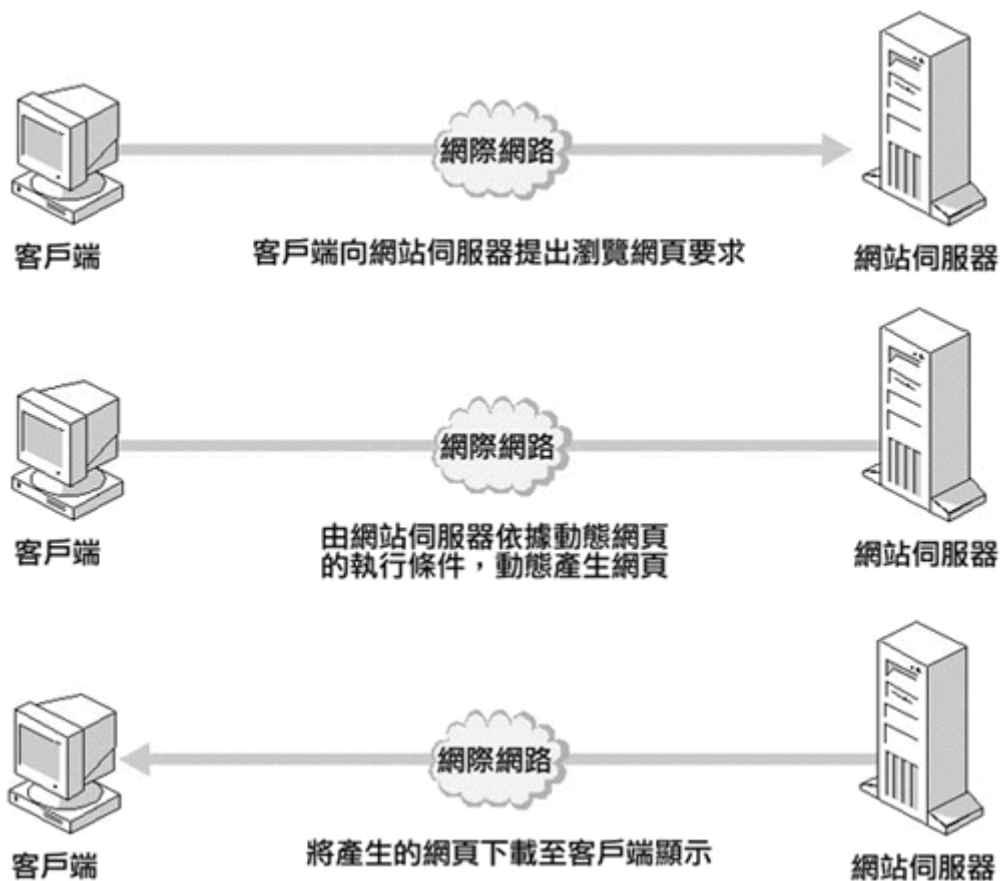
所谓静态网页，就是网页里面没有程序代码，不会被伺服器端执行。这种网页通常在伺服器端以扩展名 .htm 或是 .html 储存，表示里面的内容是以 HTML 语言所撰写。HTML 语言是由许多叫做标注 (Tag) 的元素所组成。这种语言指示了文字、图形等元素在浏览器上面的配置、样式以及这些元素实际上是存放于因特网上的哪个地方 (地址)，或是点选了某段文字或图形后，应该要连

结到哪个网址。我们在浏览这种扩展名为 **.htm** 的网页的时候，网站服务器不会执行任何程序就直接会把档案传给客户端的浏览器直接进行解读的工作。所以除非网站设计师有更新过网页档案的内容，否则网页的内容是不会因为执行程序而出现不同的内容。我们在第二章：**HTML 语言** 会提到更多如何使用 **HTML 语言**。



动态网页

所谓动态网页，就是网页内含有程序代码，并会被伺服器端执行。这种网页通常在伺服器端以扩展名 **asp** 或是 **aspx** 储存，表示里面的内容是 **Active Server Pages (ASP)** 动态网页，有需要执行的程序。使用者要浏览这种网页时必需由伺服器端先执行程序后，再将执行完的结果下载给客户端的浏览器。这种动态网页会在伺服器端执行一些程序，由于执行程序时的条件不同，所以执行的结果也可能会有所不同，所以称为动态网页。



.NET Framework

Microsoft Visual Studio.NET 簡介

Web 应用程序（以因特网为基础的应用程序）的优点在于可以让企业间的商业数据及交易等行为，透过因特网的通讯来彼此交换讯息。这样不但可以节省数据交换的时间，而且可以简化流程。但是在新一代的 .NET 开发平台还没有出现之前，要让因特网应用程序达到上述的功能是一项浩大的工程。牵涉到的技术及程序开发语言，可能包括了 HTML、ASP、VBScript、JavaScript、C++、ADO、SQL、COM、MTS 等。这样的环境对于开发人员来说，想要快速的开发一个功能强大且稳定可靠的 Web 应用程序，不是一项轻松的工作。

之前的 Microsoft Visual StudioTM（微软的解决方案开发平台）及 Windows 上的一些架构及服务，已经帮助程序设计师由单机平台的程序开发转为主从式（Client/Server）的架构来开发应用程序。但是对于现在的环境而言，因为企业的客户或供货商等所使用的系统或软件可能不尽相同，

开发人员所要面临的挑战是要如何整合所有的异质性资源，所以必须把焦点转到如何发展分布式的因特网应用程序架构，好整合这些存在于不同平台或不同软件的异质资源。

目前全世界有超过六百万的专业程序设计师，而百分之 70 以上的人使用微软的 Visual Studio 开发平台；现今最受欢迎的架构则为主/从式的架构，而发展最快速的架构则是以 Web 为基础的架构。在这种架构下程序逻辑及数据的处理都是在伺服端，使用者是透过网络以浏览器来存取伺服端的数据。这种架构的开发工具，微软早在前两版的 Visual Studio 中透过 Active Server Pages (ASP) 以及 IIS 3.0 就已经提供了。为了因应 Web 架构的快速发展及广大程序开发人员的需求，微软亦举办了不下百场的研讨会和开发人员沟通、交换意见；就是为了提供给开发人员更好的解决方案开发平台。而这个新一代开发平台目前已经推出了，即为 Visual Studio.NET。Visual Studio.NET 开发平台里面包含 Visual Basic.NET、C#、Visual C++、ASP.NET 以及 Visual FoxPro。为了让这套开发平台更容易开发以因特网为基础的应用程序，这个开发平台做了许多和以往不同的改革，这个改革就是 .NET Framework。**.NET 架构(就是 .NET Framework, .NET 念作 dot Net) 就是为了让开发分布式因特网应用程序架构变得更简单容易而发展出来的。**



微软新一代开发平台 Visual Studio.NET 的标志。

.NET Framework

.NET Framework 是微软的几个开发团队一起努力发展的成果，最主要用来产生一个可以用来快速开发、部署网站服务及应用程序的开发平台。这个架构是两个项目的结果：第一个项目的目的是用来改善 Windows 作业平台上的程序开发，特别是改善 COM (Component Object Model, 组件对象模块。一种微软所制定的软件技术；让对象的功能可以被其它软件所调用，可以让组件重复使用、容易更新及维护)；第二个项目则是制作一个以发展服务 (Service) 软件为目标的开发平台。这两个项目团队三年多前就已经在一起工作，他们希望可以发展出一种可以快速开发出以因特网为基础，而且易学易用的开发平台。为了要达到这些目标，所以 .Net Framework 在设计时加入了下列特色：

透过因特网的标准做整合

以 XML（eXtensible Markup Language，延伸标注语言）及 SOAP（Simple Object Access Protocol，简单对象存取协议）等标准通讯协议，将各种由不同环境所组成的应用程序及组件整合在一起工作。

松散的综合组件

大多数具延展能力（可扩充功能）的系统，现阶段是以「异步讯息」为架构而建立的。要建立这种多层的架构非常复杂，而且工具很少。.NET Framework 不需要很严谨的定义每个组件的结构即可很轻松的整合，这样可提高程序的延展性。

支持多种程序语言

许多程序设计师会使用多种语言来开发他们的解决方案，这是因为每种语言都有它的长处。例如某些语言对于数值计算效率较好，某些语言对于数据库的操作较为方便，而某些语言又有大量的链接库可供使用；所以没有办法强迫别人只学一种程序语言。.NET Framework 把这些语言整合起来，可以让开发人员使用不同的程序语言来开发解决方案，让程序设计师可以选择他们专长的程序语言，企业则可省去重新训练员工的成本。

提高程序设计师的生产力

现今程序设计师人才非常缺乏，程序设计师在人力不足的情形之下就必需提高生产力，因为每个项目的时程很可能很急促；况且公司也希望赶快结案好再进行下一个项目。正因如此，.NET Framework 的开发团队希望尽可能减少写程序会发生的问题，让程序设计师专心于撰写企业法则（企业处理数据的规则）。所以 .NET Framework 有些节省时间的特色，例如容易使用的自动交易机制、自动内存管理，以及丰富的控件。

完善的数据保全

目前因特网最受大家注目的，就是它的安全性。要设计一个安全性完善的因特网应用程序，在设计时必须考虑所有组件的保全设计，而不能仅做一部分而已。.NET Framework 在设计安全模型时时即考虑到这点，将所有的数据与程序代码做完善的安全防护。

可用操作系统的服务

Windows 提供了比其它作业平台更丰富的服务及资源，例如众多的数据存取服务、使用系统所提供的整合安全模式来做身分验证及保全的工作、交互式的使用者接口、成熟的对象模块、交易程序监视以及讯息队列服务。**.NET Framework** 当然也将这些操作系统所提供出来的功能包装起来，以更简单的方式提供程序设计师使用。



微软 .NET Framework 的标志。

.NET Framework 概要

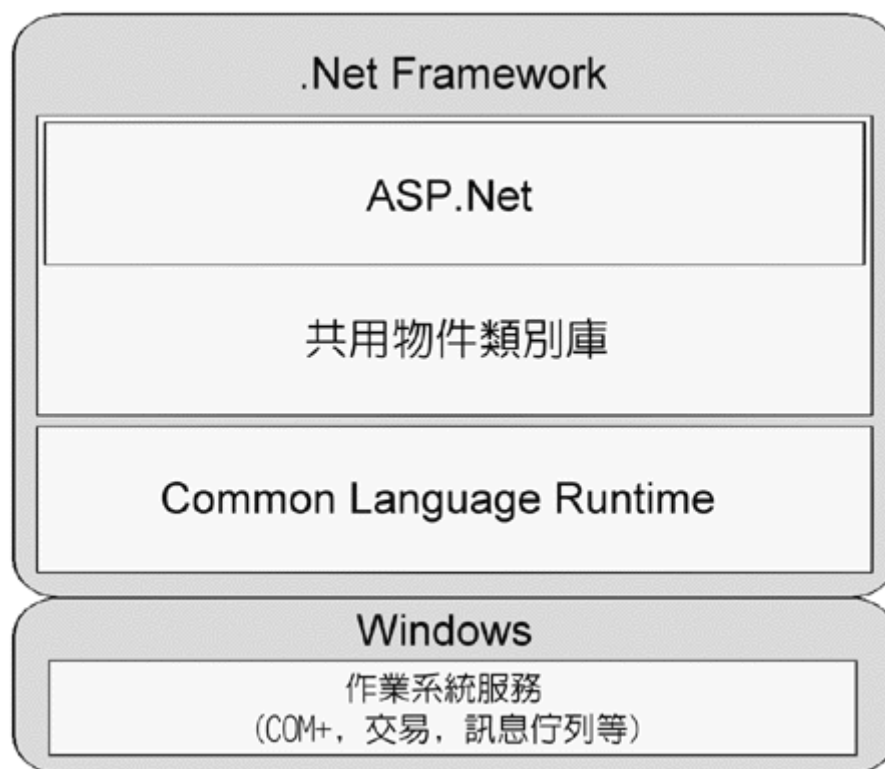
.NET Framework: 就是微软 Web Services 引擎

有许多程序设计师和使用者，非常渴望有一个完善而且透明清楚的基础架构，来建立 **Web Services**（因特网服务）。**.NET Framework** 就是为了这个需求，而提供的基础架构。**.NET Framework** 提供了应用程序模型及关键技术，让开发人员容易以原有的技术来产生、布署，并可以继续发展具有高安全、高稳定，并具高延展的 **Web Services**。对于 **.NET Framework** 而言，所有的组件都可以成为 **Web Services**，**Web Services** 只不过是另一种型态的组件罢了。微软将 **COM** 的优点整合进来，它可以不用像 **COM** 那么严谨的来栓锁两个对象，**.NET Framework** 以松散的方式来栓锁 **Web Services** 这种型态的组件。这样的结果让开发人员非常容易的发展出强而有力的 **Web** 服务组件，提高了整体的安全及可靠性，并且大大的增加系统的延展性。

.NET Framework: 由三个部分组成

.NET Framework 的目的就是要让建立 **Web Services** 以及因特网应用程序的工作变的简单，**.NET Framework** 包括了三大部分：第一个部分是 **Common Language Runtime**（**CLR**，所

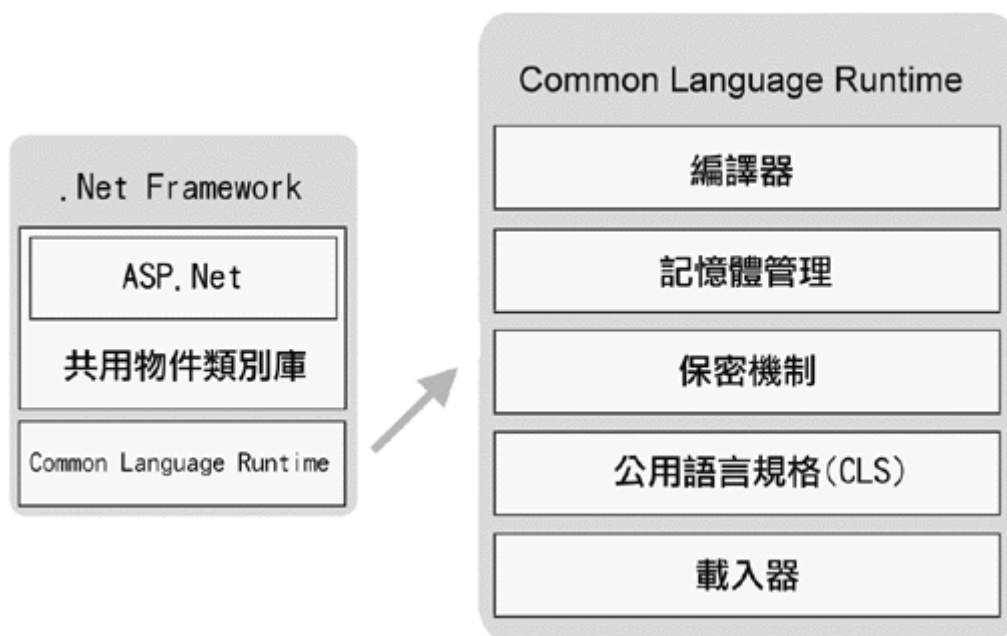
有 .NET 程序语言公用的执行时期组件)，第二部分是共享对象类别库（提供所有 .NET 程序语言所需要的基本对象），第三个部分是重新以组件的方式写成的 ASP.NET（旧版本则是以 asp.dll 提供 ASP 网页所需要的对象）。



.NET Framework 由三部份组成：Common Language Runtime、共享类别库，以及 ASP.NET，并且架构在系统服务之上。

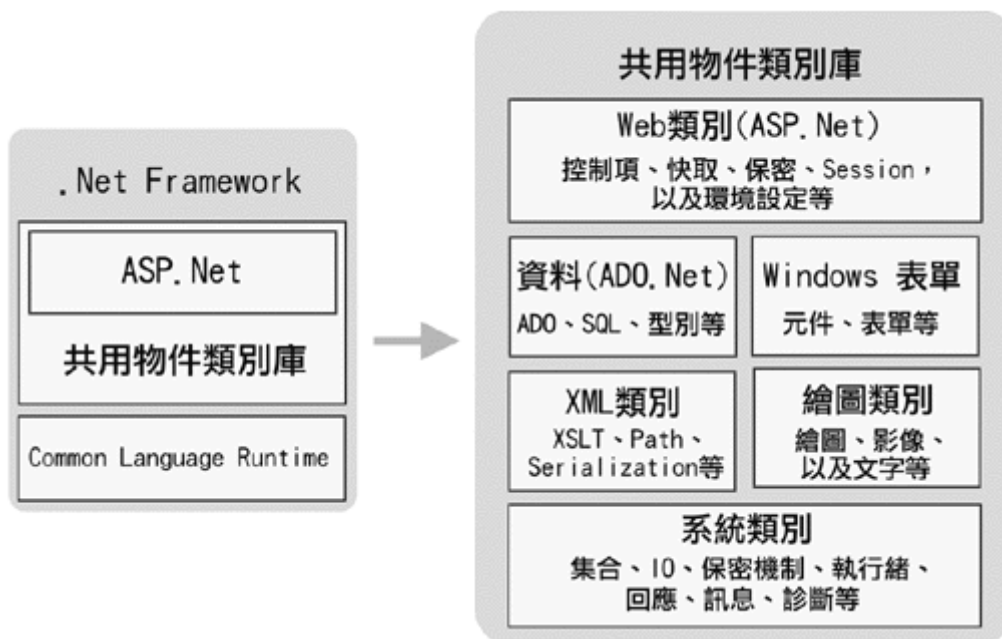
Common Language Runtime（CLR）架构在操作系统的服务上，它负责应用程序实际的执行，满足所有的应用程序的需求；例如内存管理、处理安全问题、整合不同的程序语言等等。Runtime 供了许多帮助程序写作的简化，以及应用程序的部署并同时加强程序稳定可靠的一些服务。不过程序设计师实际上不会被 Runtime 所影响，因为他们所面对的是架构在 CLR 上面的共享对象类别库，这个共享对象类别库可以被任何程序语言所使用。在这个类别中包含了以建构 Web 为基础的应用程序模型，提供以架构 Web 服务与 Web 应用程序为目标的组件及服务，这个就是我们要讨论的 ASP.NET。

Common Language Runtime (CLR)



Common Language Runtime 所负责的工作。

想要执行由某种特定程序语言所开发出来的程序，计算机内部必需装置这种特定程序语言的执行时期（Runtime）组件才可以。例如想要执行以 VB 所撰写的程序，计算机内就必须有安装 VB 的 Runtime 组件（msvbvm.dll）才可执行；而其它诸如由 Java 或是 VC++ 等的程序语言所写成的软件，也是需要 Runtime 组件才能执行。为什么 Runtime 组件如此重要？这是因为 Runtime 组件内部有该种程序语言所需要的一些核心功能，例如提供该种语言所需要的基本函式及对象等等；所以当程序在执行时会动态的连结到 Runtime 组件，取得所需要的功能。但是不同的程序语言所需要的 Runtime 不一样，所以会造成程序设计师在开发时的困难。另外在浏览网页时，如果该网页有包含有类似以 VB 这种程序语言所开发的 ActiveX 控件时，除该控件会被下载外，如果使用者没有安装 VB 的 Runtime 组件，那么执行起来会发生些问题。所以为了解决上述的问题，.NET Framework 在发展时，设计了让所有 .NET 的程序语言共同使用的 Runtime 组件，这个组件的名称就叫做 Common Language Runtime（CLR）。CLR 是一个高效率的执行引擎，程序代码的执行是由 Runtime 所管理，Runtime 负责的工作有产生对象、方法（methods，对象所能执行的动作称之）的呼叫等等，Runtime 也可以提供程序代码一些额外的服务。我们把要透过 CLR 的控制，才能执行的程序代码称为 Managed Code。



CLR 执行引擎。

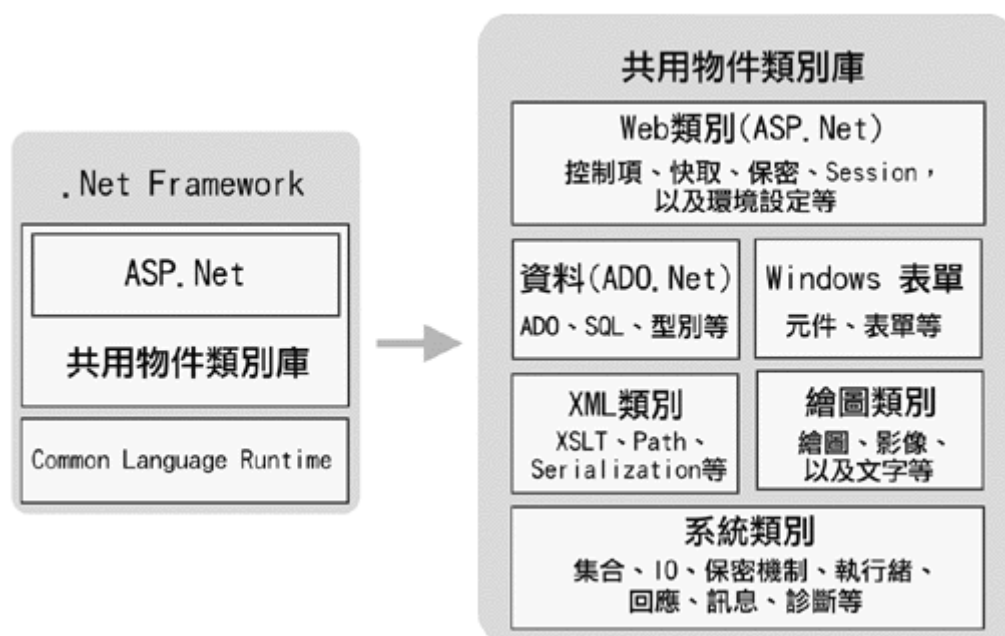
Intermediate Language 及 Just In Time 编译器

在了解 CLR 所扮演的角色后，我们要介绍 IL (Intermediate Language) 以及 JIT (Just In Time) 编译器。旧版的 ASP 是以直译的方式一行一行的执行程序，所以如果网页程序太复杂或是浏览人数变多，或是服务器负载变重，其执行的效率可想而知。ASP.NET 为了改善执行的效率，以及让程序将来可跨平台执行，所以便设计了 IL 以及 JIT 编译器。IL 这种架构非常接近机器码，可以非常有效率的透过 JIT 编译器转换为机器码；而透过 JIT 编译器所编译出来的机器码还是被 CLR 所管理。IL 含有许多广泛的指令，不但包括对象的加载、排序、初始、以及方法呼叫的指令（关于对象、方法等，第三章有详细的讨论），而且还有算数暨逻辑运算、流程控制、直接内存存取，以及例外处理的指令。不过因为每个 CPU 的架构都不一样，所以 IL 不能够直接执行；必需透过 JIT 编译器先转换成被 CPU 所认识的指令后才可执行。只要有支持该种 CPU 架构的 JIT 编译器，就可以把 IL 编译成可以在该 CPU 架构上执行的机器码，这意味着 IL 透过各种 JIT 编译器将可以跨平台。而 IL 这种格式又非常接近机器码，直接由这种格式透过 JIT 编译器编译成机器码的速度又非常的快，所以第一次执行 aspx 网页时需要编译成 IL 效率较差外，尔后只要 aspx 网页没有异动过，就只要从 IL 透过 JIT 编译器编译成机器码就可以执行，效率当然比以前的 ASP 网页以直译器来执行的效率提升许多。编译成 IL 也有另外一个好处，那就是只要该种程序语言可以被编译成 IL，就可以由 JIT 编译器编译执行。所以不管该组件用 VB、C#、Java 或其它语言所写成的，都可以被结合在一起使用；这个结合组件的动作称为组裝 (Assembly)。

组件 Assemblies

组件就是组成 .NET 应用程序的任何元素，可能来自于 .NET Framework 对象类别库中的基础对象，或是我们自行开发的对象。我们利用这些组件来开发 .NET 应用程序，最后将这些组件进行组合的动作，使用这些组件将我们的应用程序制作成执行文件（EXE）或是动态连结函式库（DLL）。

.NET 共享对象类别库



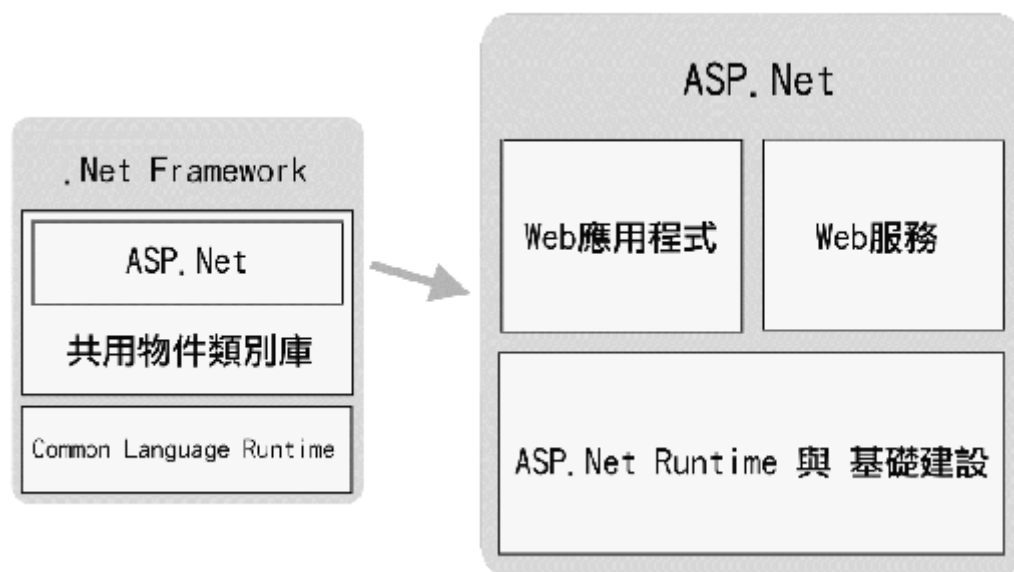
共享对象类别库中所包含的对象类别。

在 .NET Framework 出现之前,使用 Visual C++ 的程序设计师使用 MFC(Microsoft Foundation Classes) 对象类别库来写程序,而 Visual Basic 的程序设计师则使用 VBF (Visual Basic Framework)。现在 .NET Framework 将这些对象库整合并统一,设计了一个让所有程序语言共享的对象类别库。这样一来程序设计师不需要再学习多种对象模型或是对象类别库,就可以做到跨语言的对象继承、错误处理以及除错。因为不管是 VB.NET 或是 C# 等程序语言所使用的对象类别库都一样,所以程序设计师就可以自由的选择他们所偏好使用的程序语言。

.NET 提供了一个让 .NET 所有程序语言使用的共享对象类别库,这个对象类别库提供了几组统一、对象导向、结构化以及可扩充的对象类别库,协助程序设计师快速的开发软件。共享对象类

别库中提供了许多对象，包括集合、IO、数据类型等等，也提供一些对象类别可以存取操作系统服务，例如绘图、讯息、网络、执行绪与数据存取等等。程序设计师可以直接建立 .NET 共享对象类别库所提供的对象，也可以呼叫共享对象类别库的功能，或者藉由继承某个对象的功能来扩充自己建立的对象。

ASP.NET



ASP.NET 包含的部分。

共享对象类别库

ASP.NET 是共享对象类别库中的一员，ASP.NET 提供了一个 Web 应用程序模型。这模型提供了一些窗体、控件及基础架构，让程序设计师简单的建立 Web 应用程序。ASP.NET 提供了一些对应 HTML 元素（例如按钮、清单盒等）的 HTML 控件（HTML Controls，第四章会提到）以及功能更强的 Web 控件（Web Controls，第六章会提到）；这些控件在伺服器端执行，然后在客户端的浏览器以 HTML 元素的方式显示。这种 HTML 控件在伺服器端以对象导向的方式被程控，让程序设计师享受到对象导向程序写作的优点，简化程序的复杂性。

使用者接口感应

ASP.NET 另外一个重要的功能，是这些控件被设计成可以适应客户端，同样的一页可以被广大的客户端平台浏览。换句话说，Web 网页会侦测客户端所需的格式提供适合的网页：提供给行动电话 WML，能力较差的浏览器提供 HTML3.2，而 IE5.5 则提供 DHTML。

Web Services

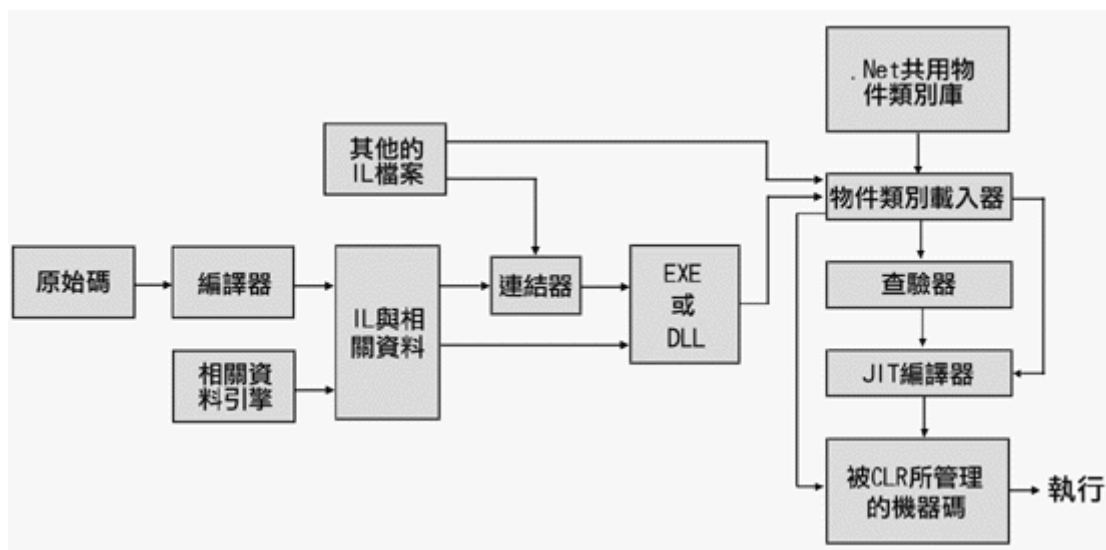
ASP.NET 也可让程序设计师把软件做成服务（Service Software，在服务器中以提供服务的方式所存在的应用程序）的方式执行。Web Services 是一种可以使用在因特网的程序逻辑，和传统的应用程序可以使用操作系统提供的功能一样，以因特网为基础的应用程序也可以利用 Web Services 来增强本身的功能。要解决应用程序间的整合，以及把程序做成服务程序的方法，就是使用 Web Services（网站服务，在网站中提供服务）。Web Services 提供简单、弹性，并以标准模块的方法来建立透过因特网工作的应用软件。因特网应用程序可以将 Web Services 在不管是否不同平台、不同程序语言所开发，以及不管新旧的情形下轻松的做整合。利用 ASP.NET 的新功能可以简单的将企业法则写成 Web 服务组件，这时 ASP.NET 所提供的一些基础架构就负责透过 SOAP 或是 XML 等标准通讯协议来使用这些服务组件。



Web Services。

了解 CLR 实际的运作

对 .NET Framework 有个概念之后，我们再了解 CLR 实际的动作：



CLR 的执行。

1. 首先程序代码先由编译器编译成 IL，同时相关联数据会由相关数据引擎（Metadata Engine）产生。
2. 这时候如果有不同的语言所编译成的 IL 或机器码，连接器（Linker）就可以将它连结进来，并产生包含 IL 的 EXE 或 DLL，编译器的功能在这里就算完成了。
3. 这时当程序在执行的时候，就是 CLR 执行工作的时候。程序中如果有任何使用到 .NET Framework 共享对象类别库的程序代码时，会被对象类别加载器（Class Loader）载入并合并。这时候被合并的程序代码在 JIT 编译器执行之前可以透过查验器（Verifier）来检查型别安全。
4. 最后由 JIT 编译器把程序代码编译成可以被 CLR 所管理的机器码便可执行。

建立 ASP.NET 开发平台

了解整个 .NET 架构后，接下来我们就要建立 ASP.NET 的开发平台了。要建立 ASP.NET 平台需要的软件如下：

- Windows 2000
- IIS 5.0
- NET Framework SDK
- Internet Explorer 5.5

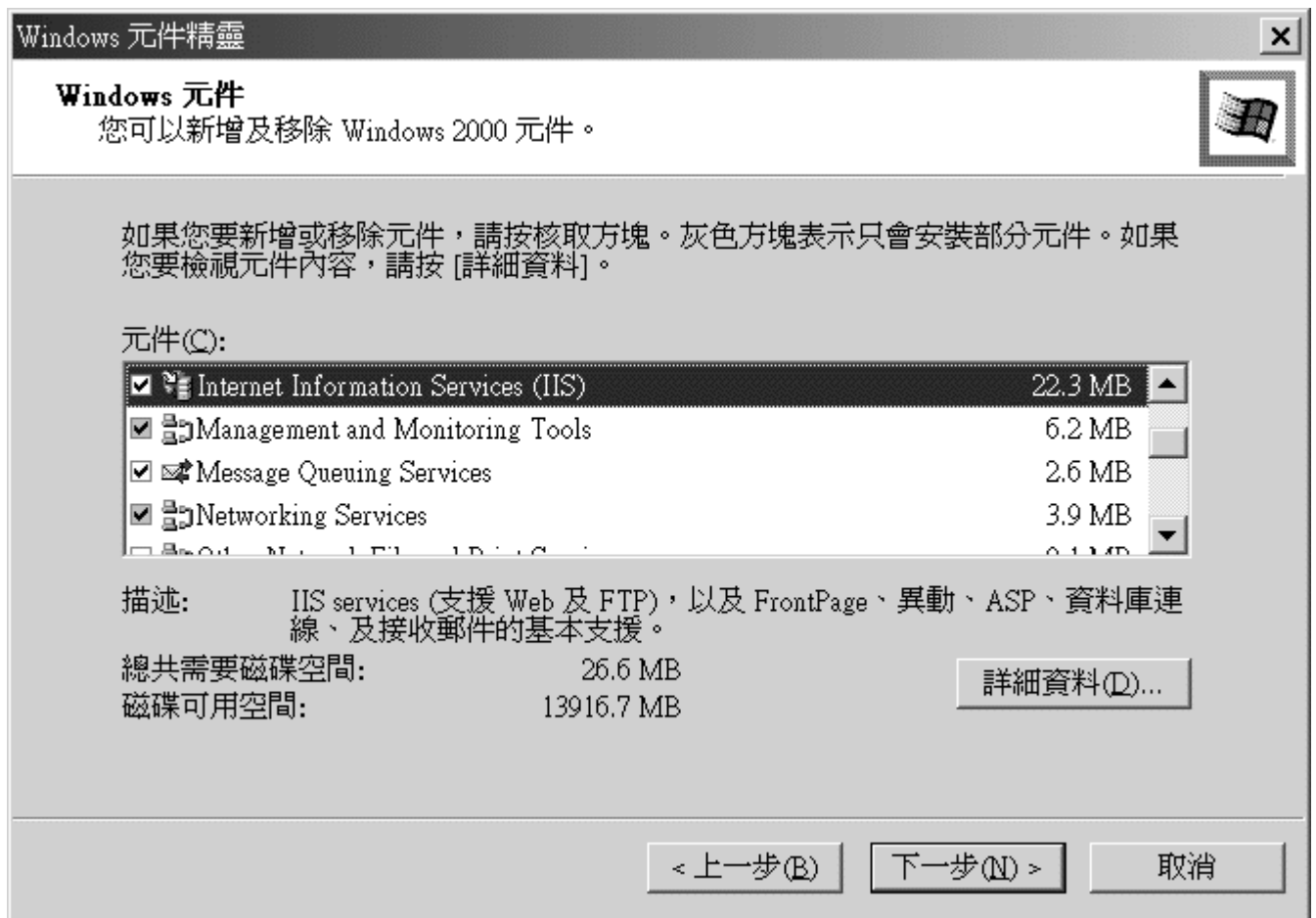
Internet Information Server 的安装

要成为网站服务器，只要有 IIS（Internet Information Services）的服务程序即可。IIS 最主要的功能大略为：

- 响应使用者的要求，将所要浏览的网页内容传输给他们。
- 管理及维护 Web 站台。
- 管理及维护 FTP 站台。
- SMTP（Simple Mail Transfer Protocol）虚拟服务器。

执行 ASP 的程序（要执行 ASP.NET 程序，需要安装 .NET Framework SDK）。目前 IIS 的最新版本 5.0 版，是 Windows 2000 的内建组件，除了 Professional 需使用「控制台」的「新增 / 移除程序」另外安装到系统内外，Server 等其它版在安装 Windows 2000 后就已经在系统内提供服务了。倘若你是使用 NT Server 4.0 版，则必需安装 NT Service Pack 6a 版才可以建置 ASP.NET 的开发平台；所以要建立 ASP.NET 的开发平台使用 Windows 2000 比较方便。若您的 Windows 2000 中没有 IIS 5.0，请按下列步骤安装：

1. 选择「开始」→「设定」→「控制台」
2. 点选「新增/移除程序」，并选取「新增/移除 Windows 组件」。
3. 出现下列窗口后，勾选 Internet Information Server(IIS)，如下图所示：

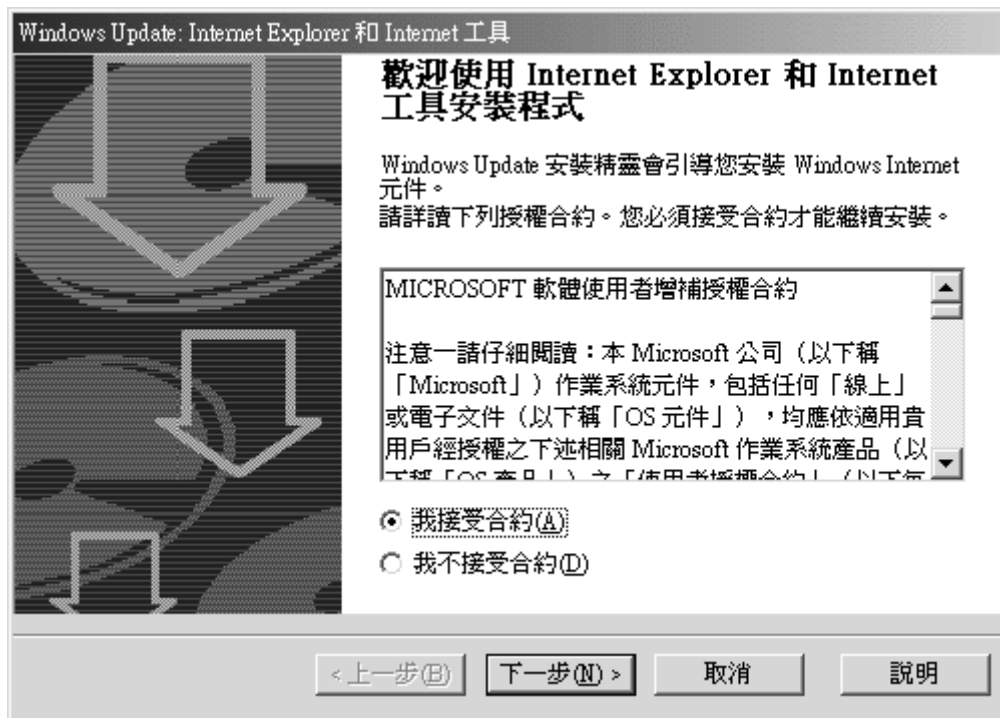


4. 按「下一步」即可完成 IIS 5.0 的安裝。

Internet Explorer 5.5 的安裝

本书附的 .NET Resource CD 中有 IE 5.5，請一下列步驟安裝：

1. 光盤中的 IE55 目錄中執行 ie5setup.exe，即出現下列畫面：



2. 如同意授权合约后，选择我同意后按下一步，即出现下列画面：



3. 按下一步后即出现下列画面开始安装：



4. 重新开机即完成安裝。



.NET Framework SDK 的安裝

要使用 ASP.NET，一定要装核心组件：.NET Framework SDK（Software Develop Kit，程序开发套件，里面有开发程序所需要的各种组件、对象类别模块，以及一些工具软件。）。

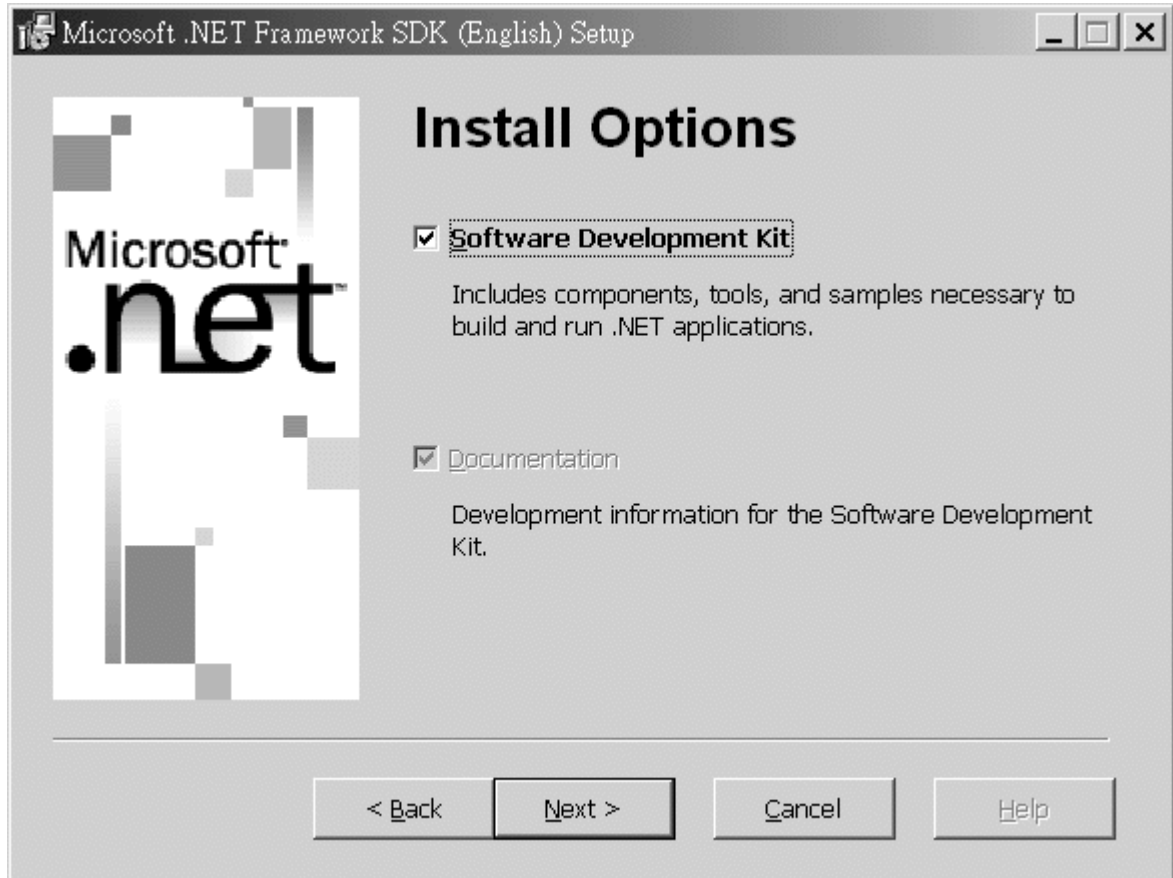
1.在光盘中的「DotNETSDK」数据夹中执行「Setup.exe」，则出现下面画面：



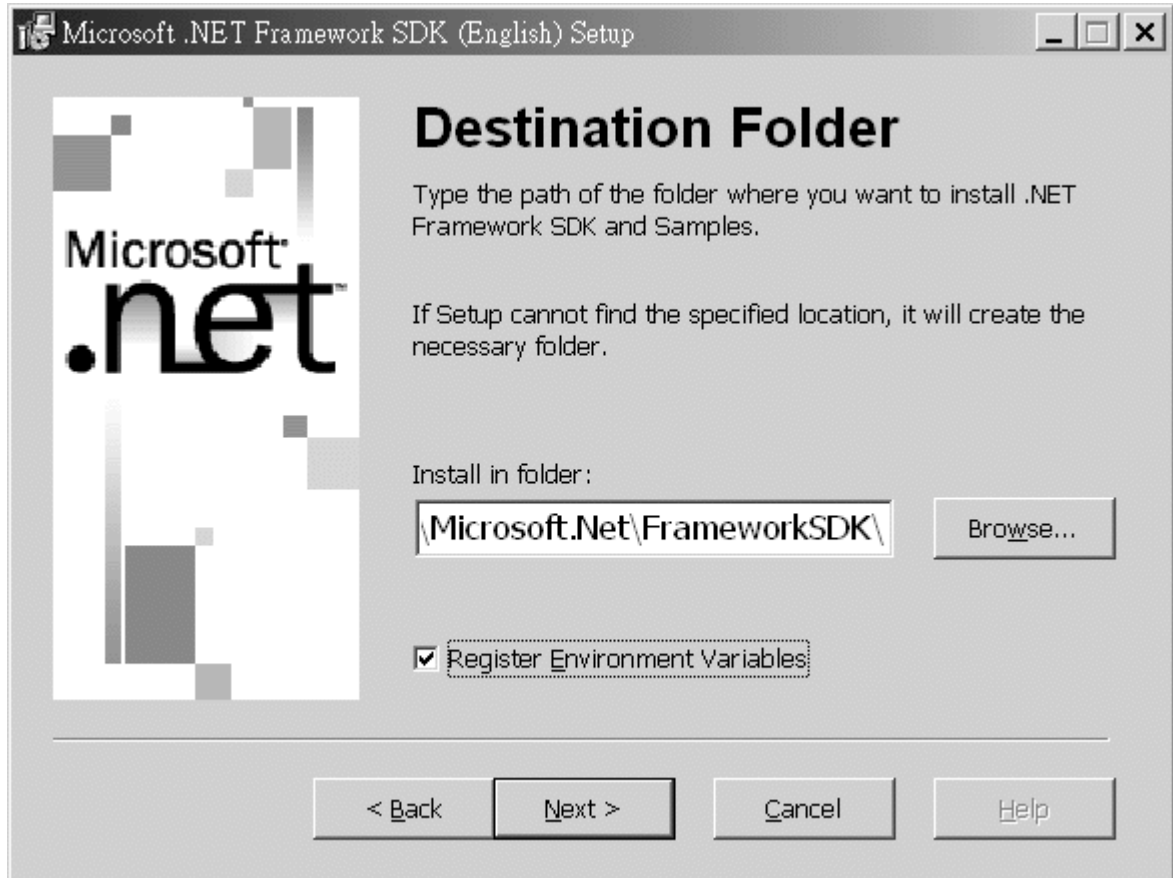
2.点选「Next」后，出现版权宣告画面。选择「I accept the agreement」后按「Next」：



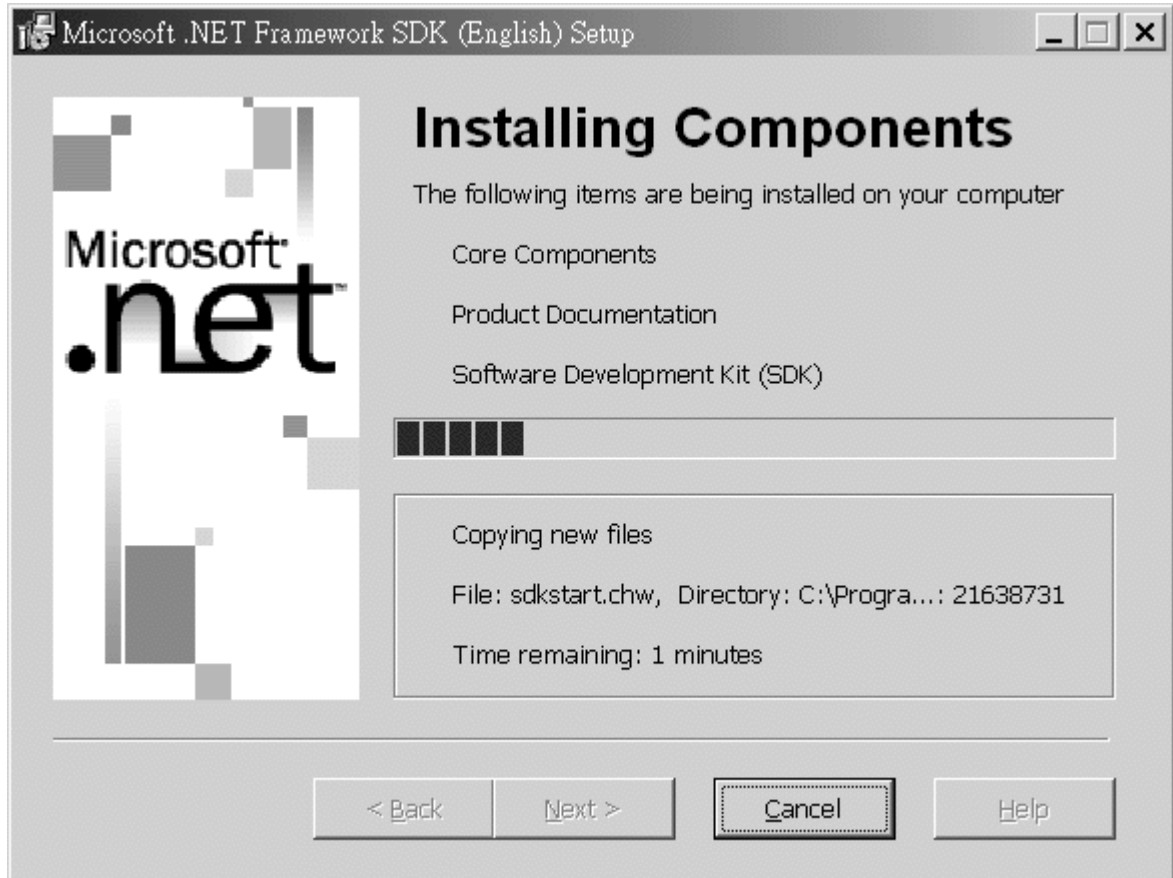
3. 勾选「Software Development Kit」安装.NET Framework SDK:



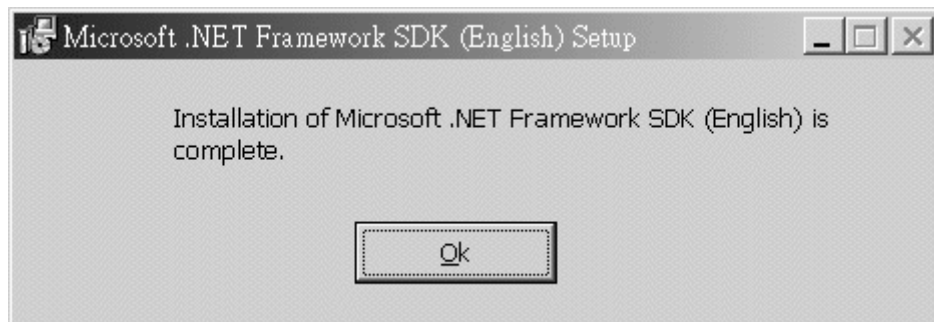
4. 选择所要安装的目的数据夹以及是否注册环境变量，直接按「Next」：



5.接下来安装程序会做些安装准备，然后开始安装：



6.完成后「OK」，.NET Framework SDK 就已经安装完毕：



本书范例程序

本书范例程序存放于光盘中的「CR」这个数据夹中，请将本数据夹以及其子数据夹中的所有档案复制到「C:\inetpub\wwwroot」这个位置，并取消所有档案的只读属性即可。

2. HTML 语言

HTML 简介

HTML 显示原理

了解 ASP.NET 的大致的工作原理后，我们就要开始来进行实作了。我们知道 ASP.NET 的程序代码最终执行完毕后，会将执行结果制成浏览器了解的 HTML（Hypertext Markup Language，超文件标示语言）后再传输给浏览器进行解读，最后再将解读的结果显示给使用者。HTML 语言是全球信息网（World Wide Web）文件（即网页）所使用共同的语言。要让浏览器显示出我们网页想要表现出的样式，必需要用 HTML 语言加以设定版面的编排。虽然现在网页制作的新技术很多，但是基本上还是架构在 HTML 之上；所以在学习 ASP.NET 之前，对 HTML 语言有基本的了解是必要的。



HTML 档的解读及显示

HTML 制作工具

编辑 HTML 的可视化工具很多，使用 FrontPage 比较方便，没有的话用 WordPad 也可以。

1. 我们先用 WordPad 输入如下 HTML 内容：

2. <HTML>

3. Hello

 </HTML>

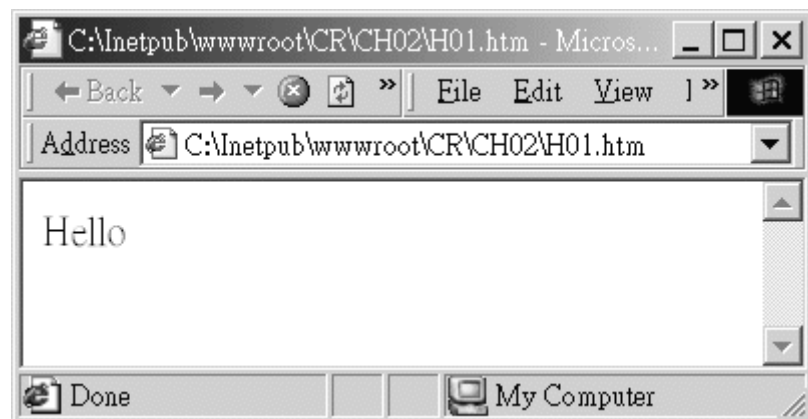


4. 并选择存盘类型为文字文件，取名为 H01.htm:



5. 然后使用档案总管，用鼠标在该档案上双击两下，该 HTML 档案就会被浏览器

打开：



HTML 标注

HTML 是由标注、属性及数据组合而成的，例如我们刚刚所输入的 HTML 档案中，HTML 标注即为 `<HTML>` 以及 `</HTML>`，而「Hello」则是我们要显示的资料。这里我们还没有用到属性，稍后会带领大家了解。HTML 标注有许多种，每一种都有特定的意义和用途，并规定必须被「`<>`」符号包含。刚刚的 `<HTML>` 标注即表示以下为 HTML，浏览器收到后即进行解读，将「Hello」输出至显示窗口，直到读到 `</HTML>`，即表示 HTML 结束。接下来我们将刚刚的 HTML 加入如下内容：

```
<HTML>
```

```
<font size="7" color="red">
```

```
Hello
```

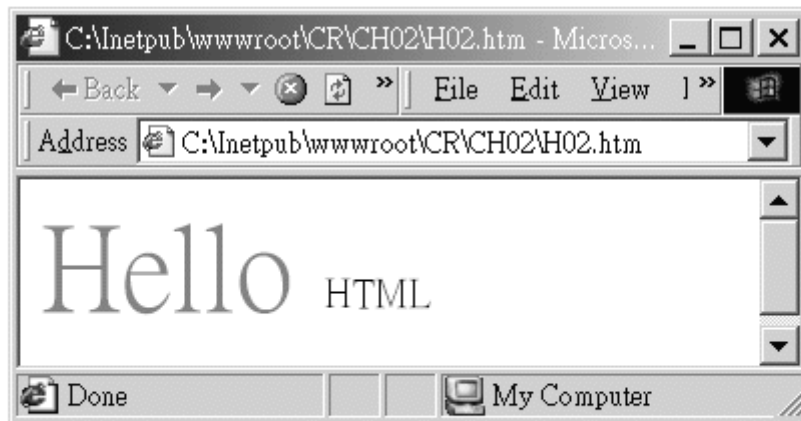
```
</font>
```

```
HTML
```

```
</HTML>
```

我们观察刚才所加入的标注 ``，`font` 标注表示标注后面的文字要设定字型变化，而里面的 `size` 以及 `color` 都是属性，分别设定字型的大小以及颜色，属性和属性之间要空一个空格；由双引号引起来的 `7` 及 `red` 分别是属性的设定值，而 `` 表示影响到此为止。所以被 `` 及 `` 所围起来的文字「Hello」大小为 7 级，颜色为红色；而在 `` 后面的文字「HTML」就没有受到影响。由此我们知道：**HTML 标注的影响范围在区块之内，而且不分大小写**。所以显示的结果就变成：

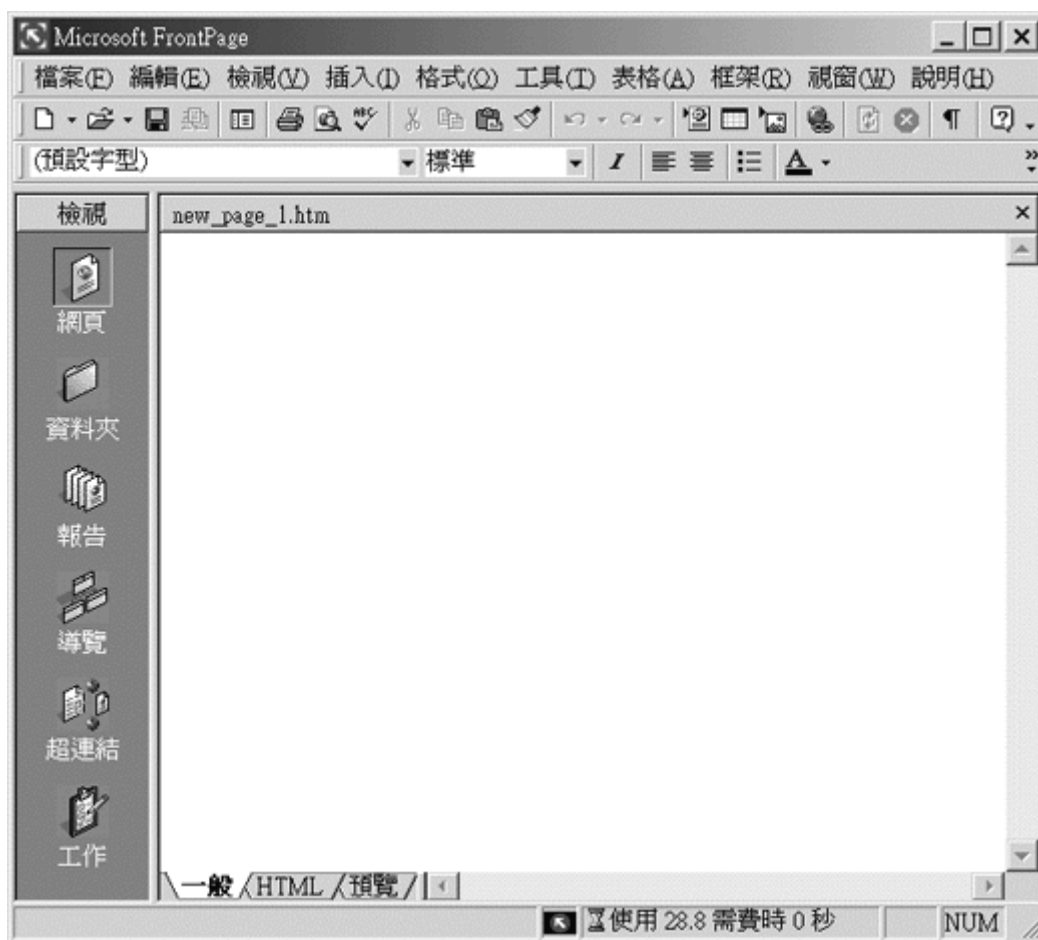
```
<HTML>  
<font size="7" color="red">  
Hello  
</font>  
HTML  
</HTML>
```



HTML 学习工具—FrontPage 2000

使用 FrontPage 来编辑 HTML 是最方便不过了，因为它是即见即所得的工具，可以立即看到结

果。我们先执行 FrontPage 2000，则出现以下画面：

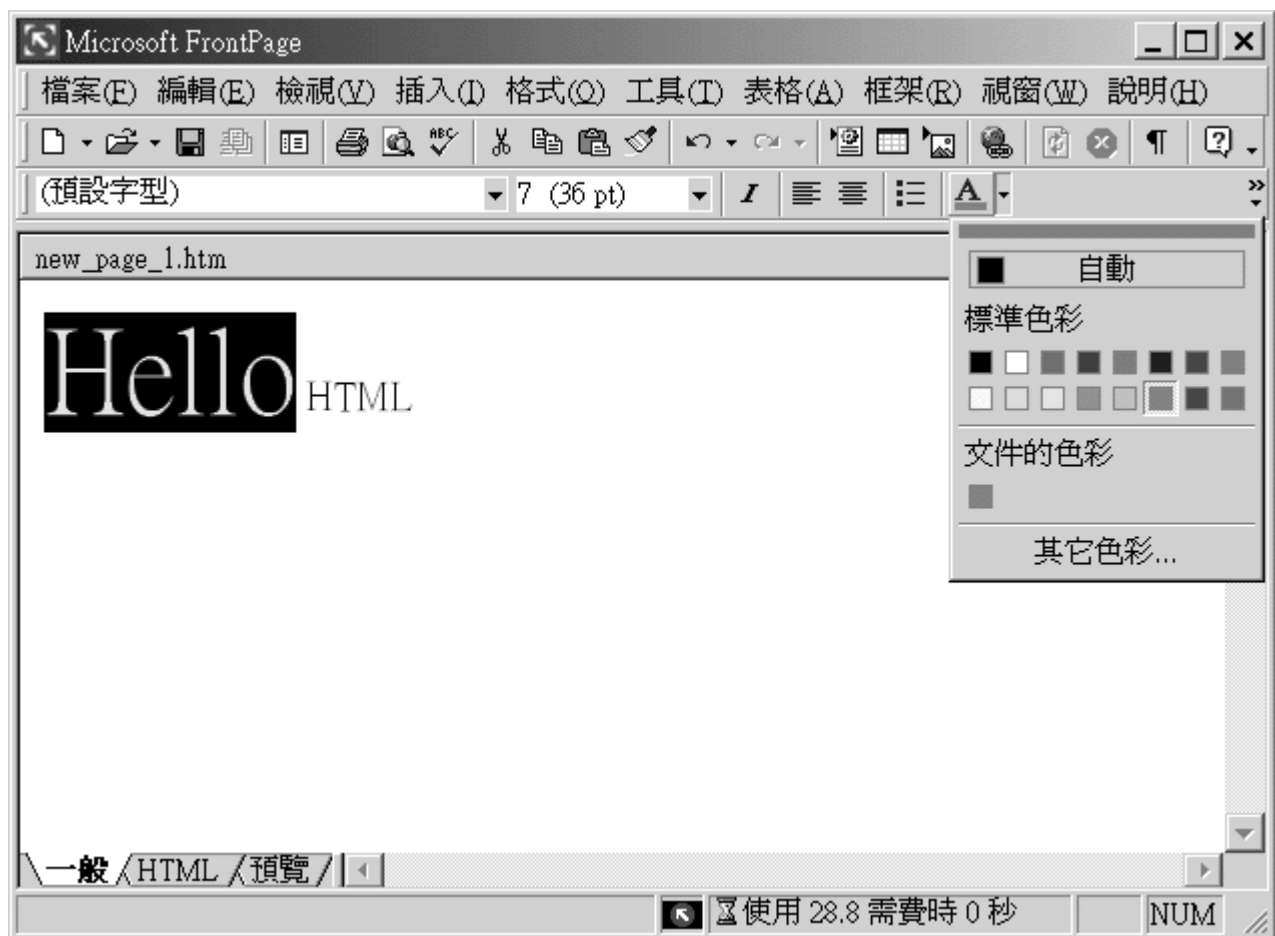


在 FrontPage 下方有三个标签，分别是「一般」、「HTML」以及「预览」。我们最主要就是利用这三种模式进行 HTML 的编辑。



一般窗口

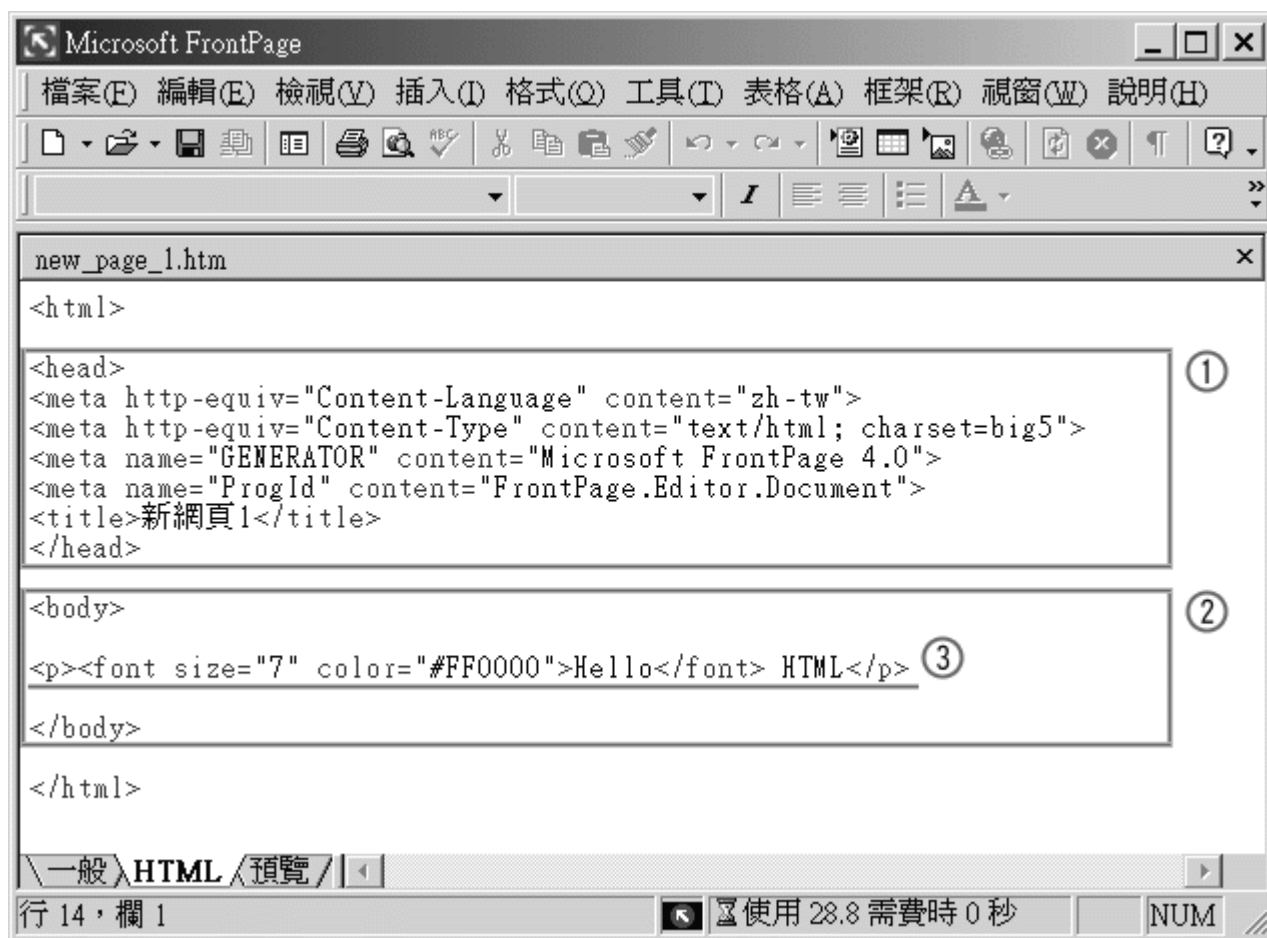
在一般卷标内可以用视觉的方式编辑网页，直接在窗口内输入文字或配置图形，可以不用使用 **HTML** 语言。如果我们只是网页设计师的话，直接使用这个窗口编辑网页即可。可是要动态的产生网页，就必须利用程序产生 **HTML** 才可以。左侧的检视窗口我们目前用不到，可以选择选单的「检视」选项，并将「检视列」的选项取消，让编辑窗口大一点。这个窗口的使用方法和 **Word** 很像，直接输入文字后可以设定文字的字型、颜色、大小等属性。由于是即见即所得，当然会马上出现显示结果：



HTML 窗口

HTML 窗口是让我们以直接输入 HTML 的方式编辑网页，我们切换到 HTML 窗口，并且观察一

下刚刚以直接输入的方式所产生的 HTML 檔：

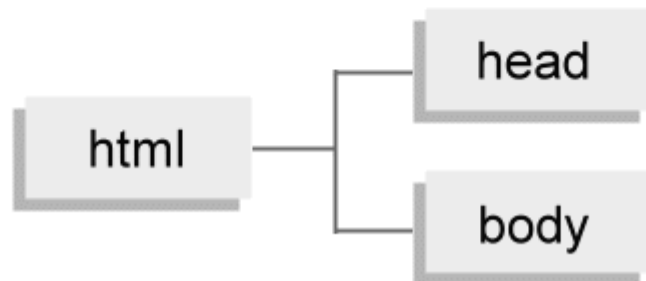


我们看到除了（3）是我们之前所输入的 HTML，并被 <p> 标注起来之外，还多了（1）head

和（2）body 这两个标注。

我们将刚刚 HTML 标注做分析后，发现如下的阶层：

```
<html>
  <head>
    <meta http-equiv="Content-Language" content="zh-tw">
    <meta http-equiv="Content-Type" content="text/html; charset=big5">
    <meta name="GENERATOR" content="Microsoft FrontPage 4.0">
    <meta name="ProgId" content="FrontPage.Editor.Document">
    <title>新網頁1</title>
  </head>
  <body>
    <p><font size="7" color="#FF0000">Hello</font> HTML</p>
  </body>
</html>
```



我们来看看这些标注所代表的意义：

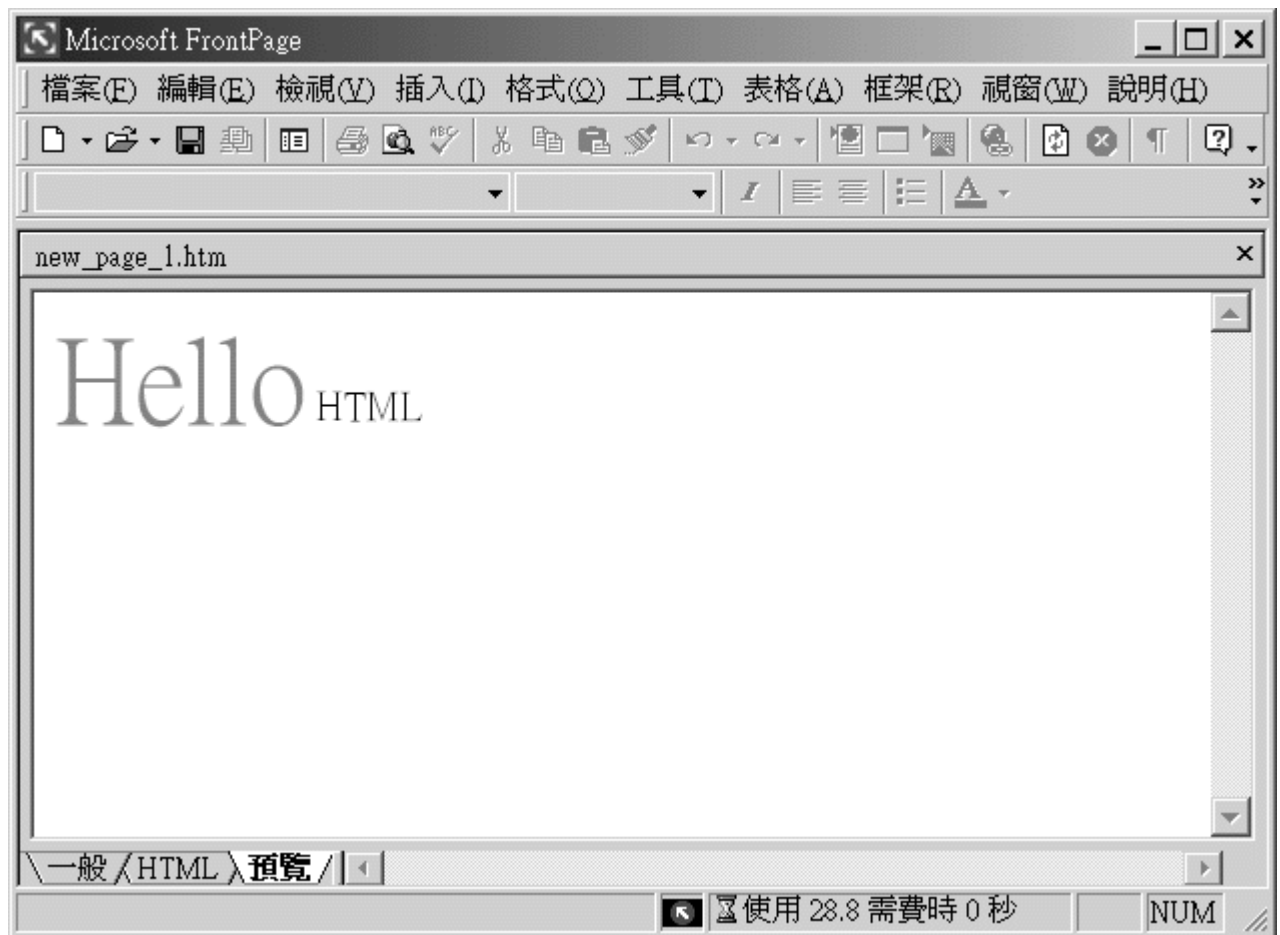
html：表示被<html> 及 </html> 所包围起来的内容是一份 HTML 文件，不过本标注也可以省略。

head：此标注用来注明此份文件作者等信息，除了 <title> 会显示在浏览器的标题列之外，其它并不会显示出来。故 <meta> 可以省略。

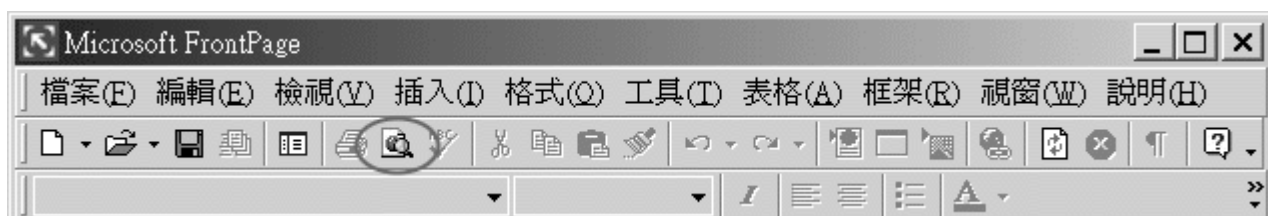
body：被此标注所围起来的数据，即表示是 HTML 文件的内容，会被浏览器显示在显示窗口，不过本标注也可以省略；而里面的 <p> 则是代表段落标注，代表所围起来的文字为一个段落。

预览窗口

我们用一般窗口或是 HTML 窗口所编辑的网页，想要预览在浏览器上输出的状况，可以使用预览窗口：



或是先存盘后选择「用浏览器预览」，使用浏览器来观察：



经常使用的 HTML 标注

断行

**<p>段落及
断行标注:**

要让文字断行，必需使用 `<p>` 或 `
` 标注。

```
<HTML>
```

```
<font size="7" color="red">
```

```
Hello
```

```
</font>
```

HTML

第一行

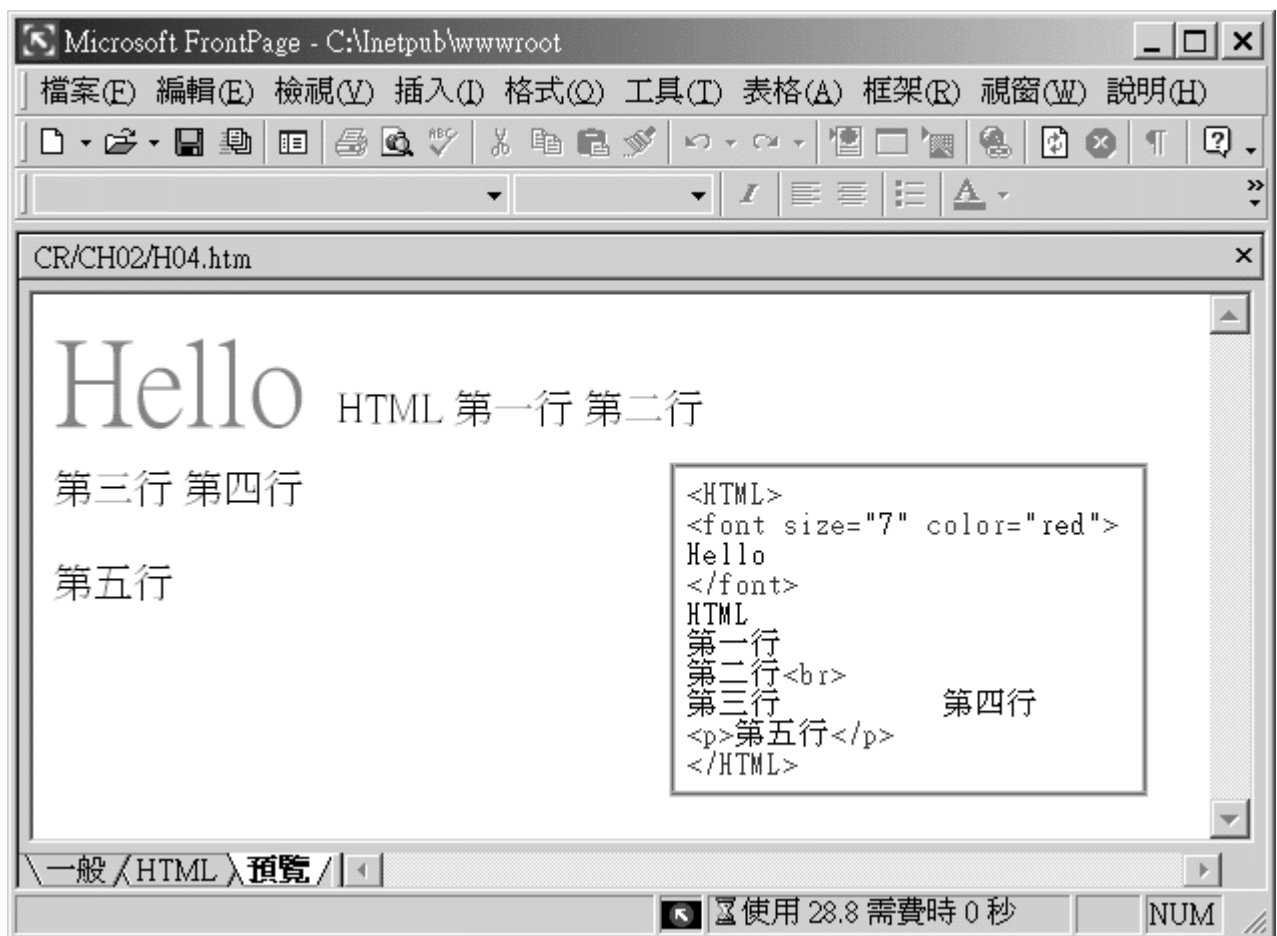
第二行

第三行

第四行

<p>第五行</p>

</HTML>



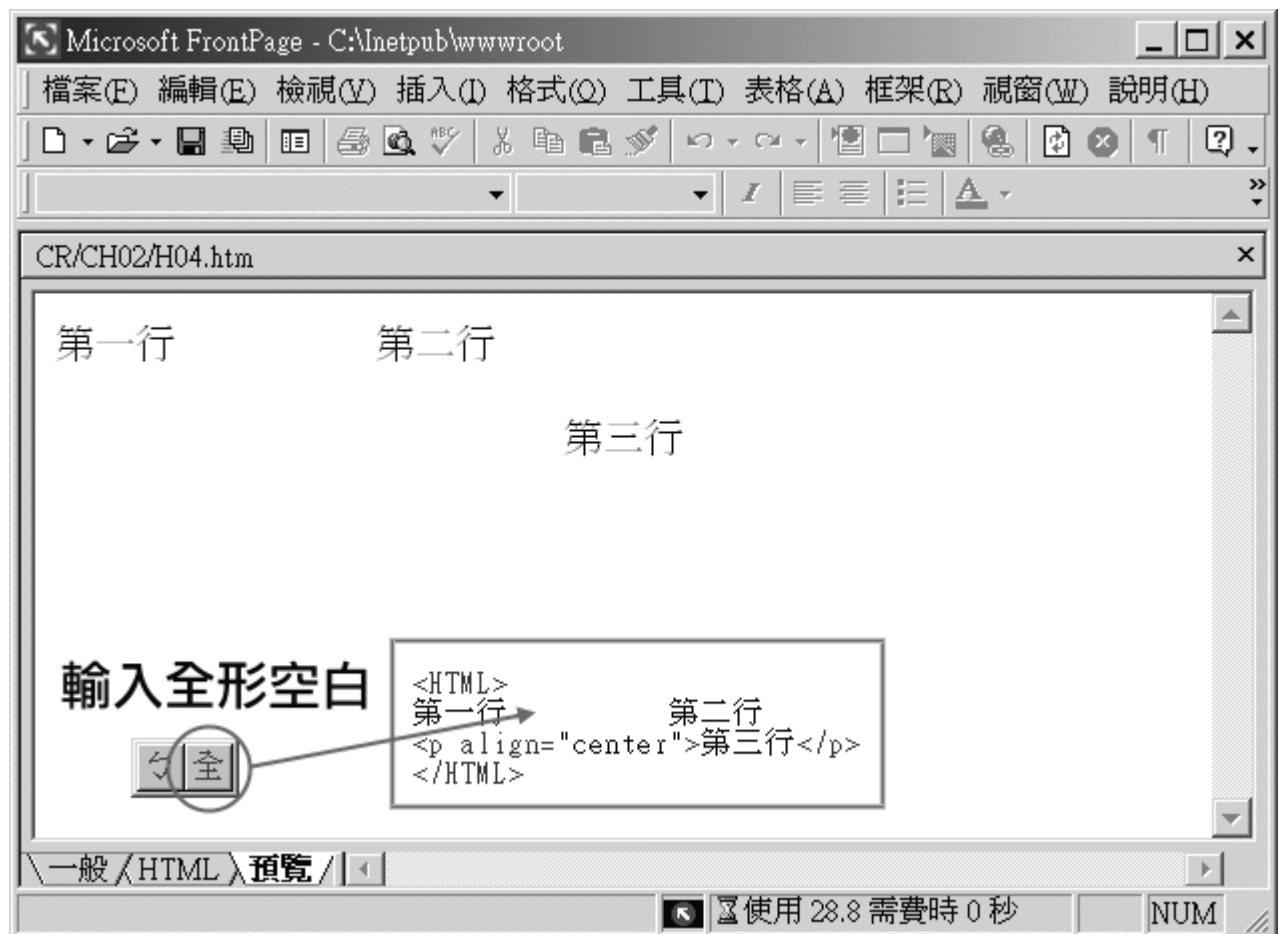
在 HTML 中按「Enter」让文字分行是没有用的；另外在 HTML 文件中空多个空格符，只会空一个。因为在 [HTML 里面断行是没有作用的](#)，故「Hello」、「HTML」、「第一行」、「第二行」都在同一行。第二行后面有一个 `
` 标注，这个标注不需要再写一个对映的 `</br>`，表示后面换行。所以「第三行」、「第四行」换到下一行显示，而「第五行」被 `<p>`、`</p>` 所包围，表示新的段落，所以和 `
` 比起来行距比较大。接下来我们看看下面的范例：

```
<HTML>

第一行          第二行
```

<p align="center">第三行</p>

</HTML>



这个范例中，在「第一行」以及「第二行」中间加入全角空格符，就可以将字距拉宽。这是因为

全角的空白是中文字型，所以可以加大间距；而「第三行」是被 <p> 段落区所包围，<p

`align="center">` 表示此段落的对齐属性为靠中，只要在这一个段落内的文字都生效。除了

`"center"` 外，还有 `"left"` 靠左及 `"right"` 靠右可以设定。

文字的显示

``标注:

本标注用来设定文字的大小、颜色、字体。首先我们先来看 **size** 属性的设定:

```
<HTML>

<font size="1">size="1"</font><br>

<font size="2">size="2"</font><br>

<font size="3">size="3"</font><br>

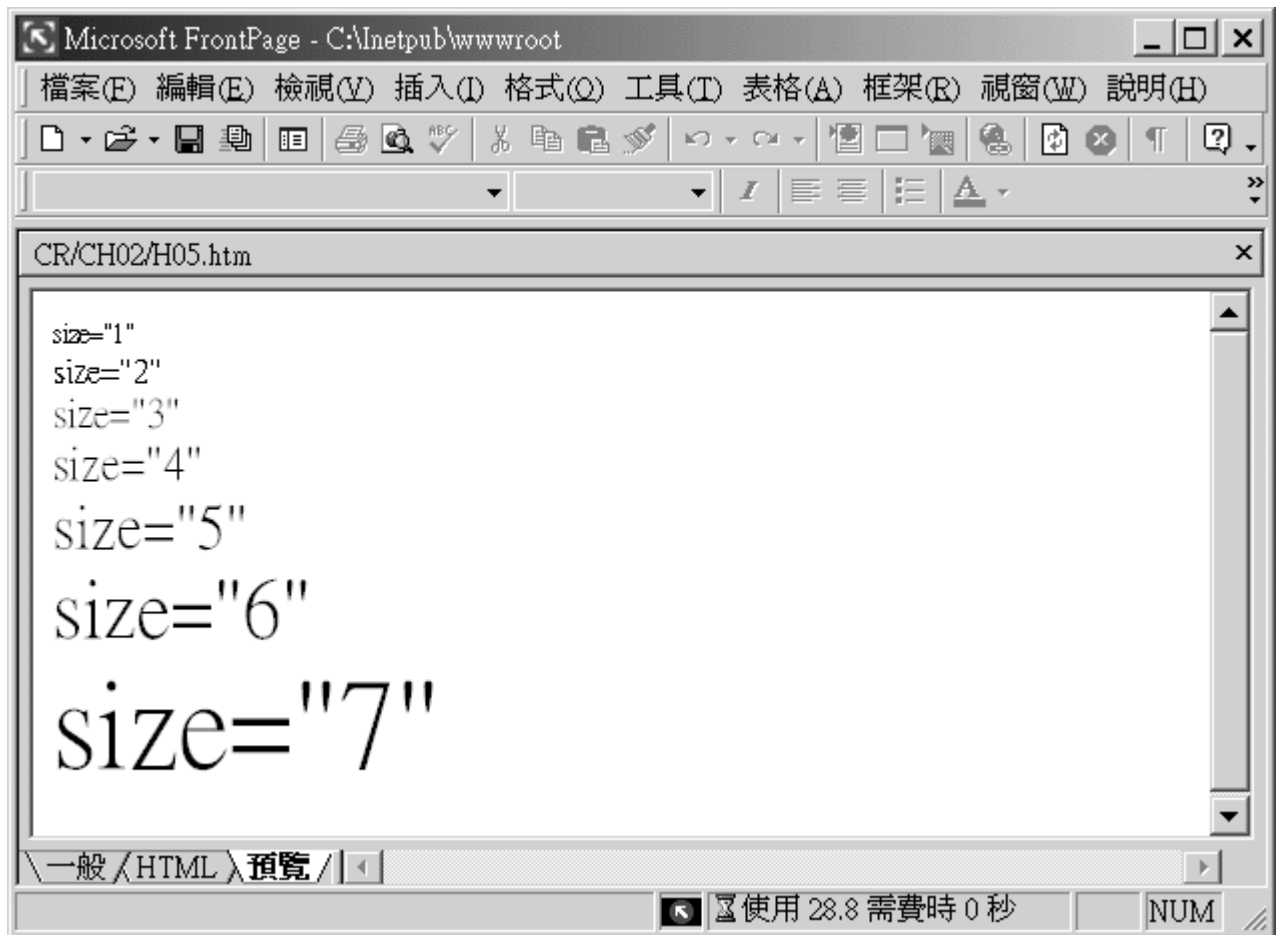
<font size="4">size="4"</font><br>

<font size="5">size="5"</font><br>

<font size="6">size="6"</font><br>

<font size="7">size="7"</font><br>

</HTML>
```



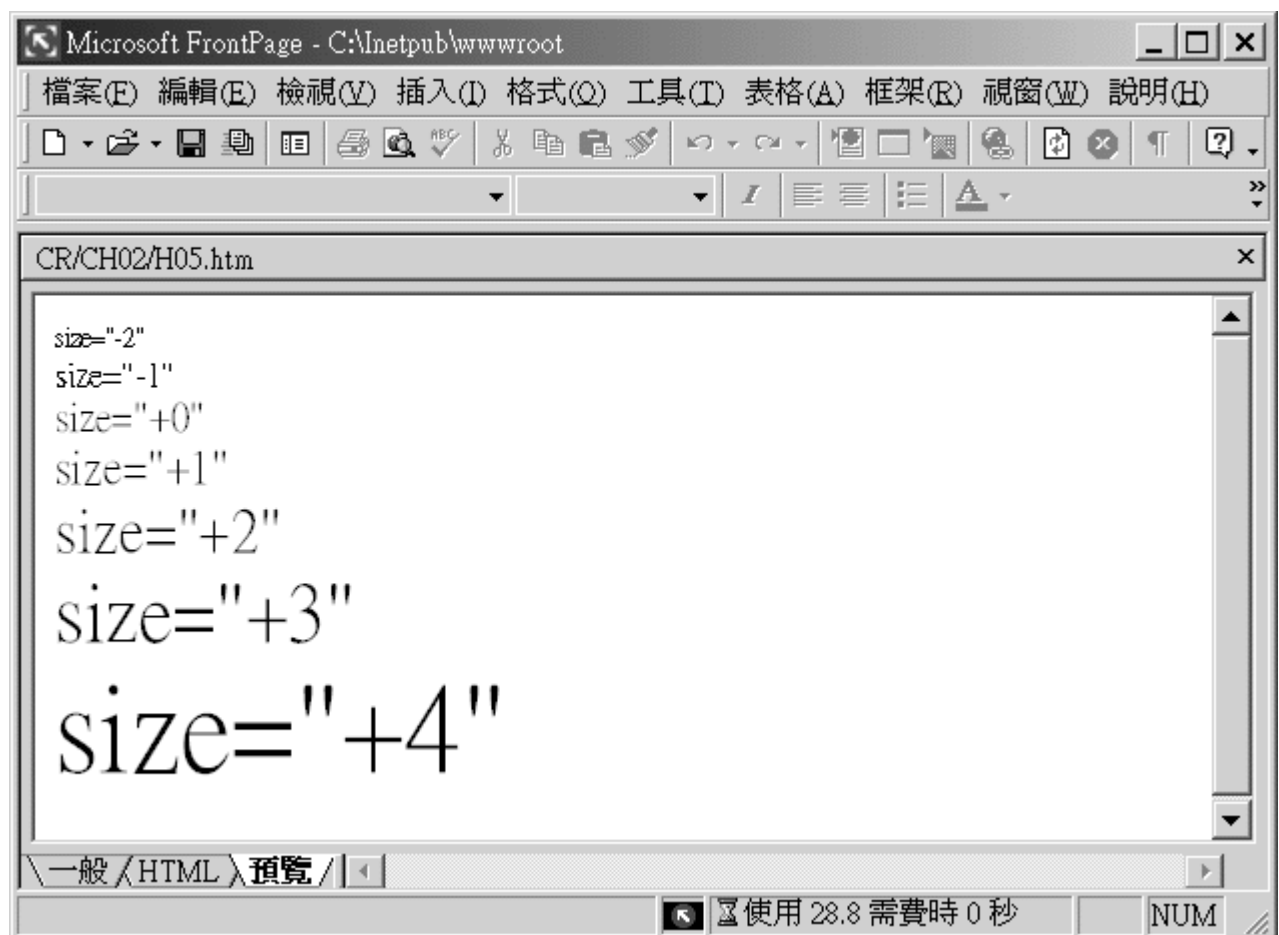
size 属性的设定值最小为 1，最大为 7。也有另一种以第三级算为基准，然后加减几级的用法：

```
<HTML>  
  
<font size="-2">size="-2"</font><br>  
  
<font size="-1">size="-1"</font><br>  
  
<font size="+0">size="+0"</font><br>  
  
<font size="+1">size="+1"</font><br>  
  
<font size="+2">size="+2"</font><br>
```

```
<font size="+3">size="+3"</font><br>
```

```
<font size="+4">size="+4"</font><br>
```

```
</HTML>
```



face 属性是设定文字的字型，接着我们来看看 face 属性的范例，分别将 size 都设定为 5，并且

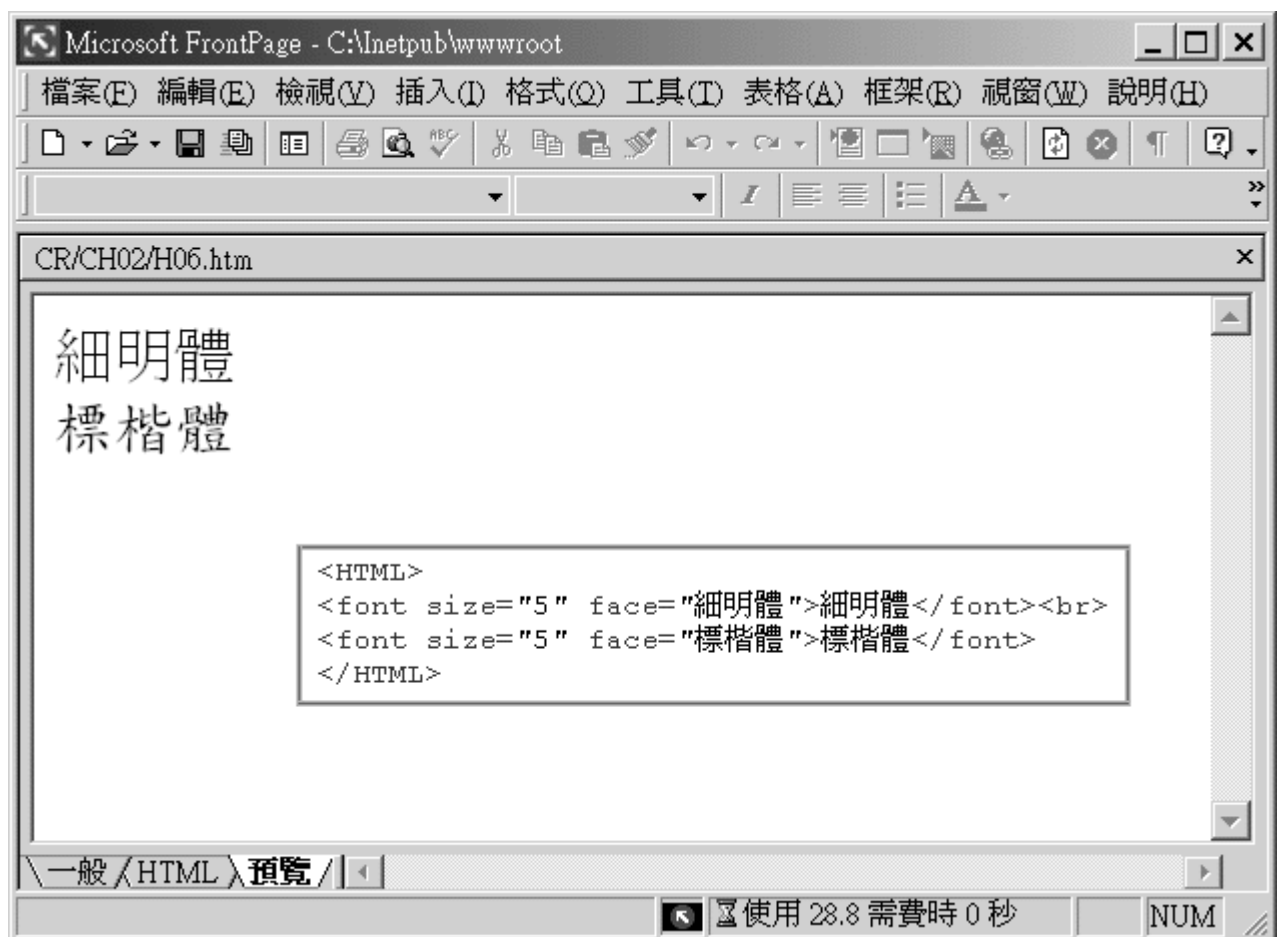
使用细明体级标楷体：

<HTML>

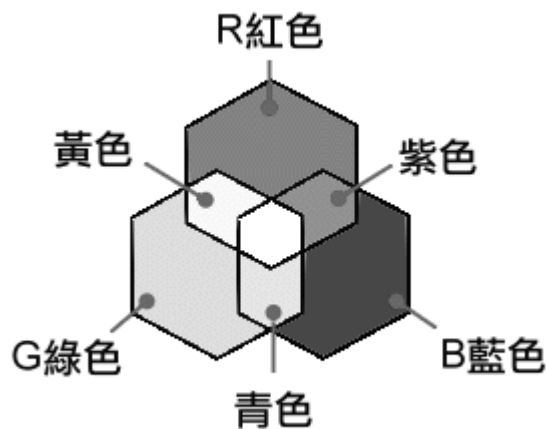
細明體

標楷體

</HTML>



color 属性为设定字体的颜色，其设定方式有两种：一种是直接输入颜色的英文名称，例如红色为 **red**，绿色为 **green**，蓝色为 **blue**；另一种比较常用，为输入 RGB 三原色的强度值，用法为 ****，其中 **rr**、**gg**、**bb** 分别代表红色、绿色、蓝色的强度，分别由最弱的 **00** 到最强的 **FF**（十六进制）。



```
<HTML>

<font color="#FF0000">red</font><br>

<font color="#00FF00">green</font><br>

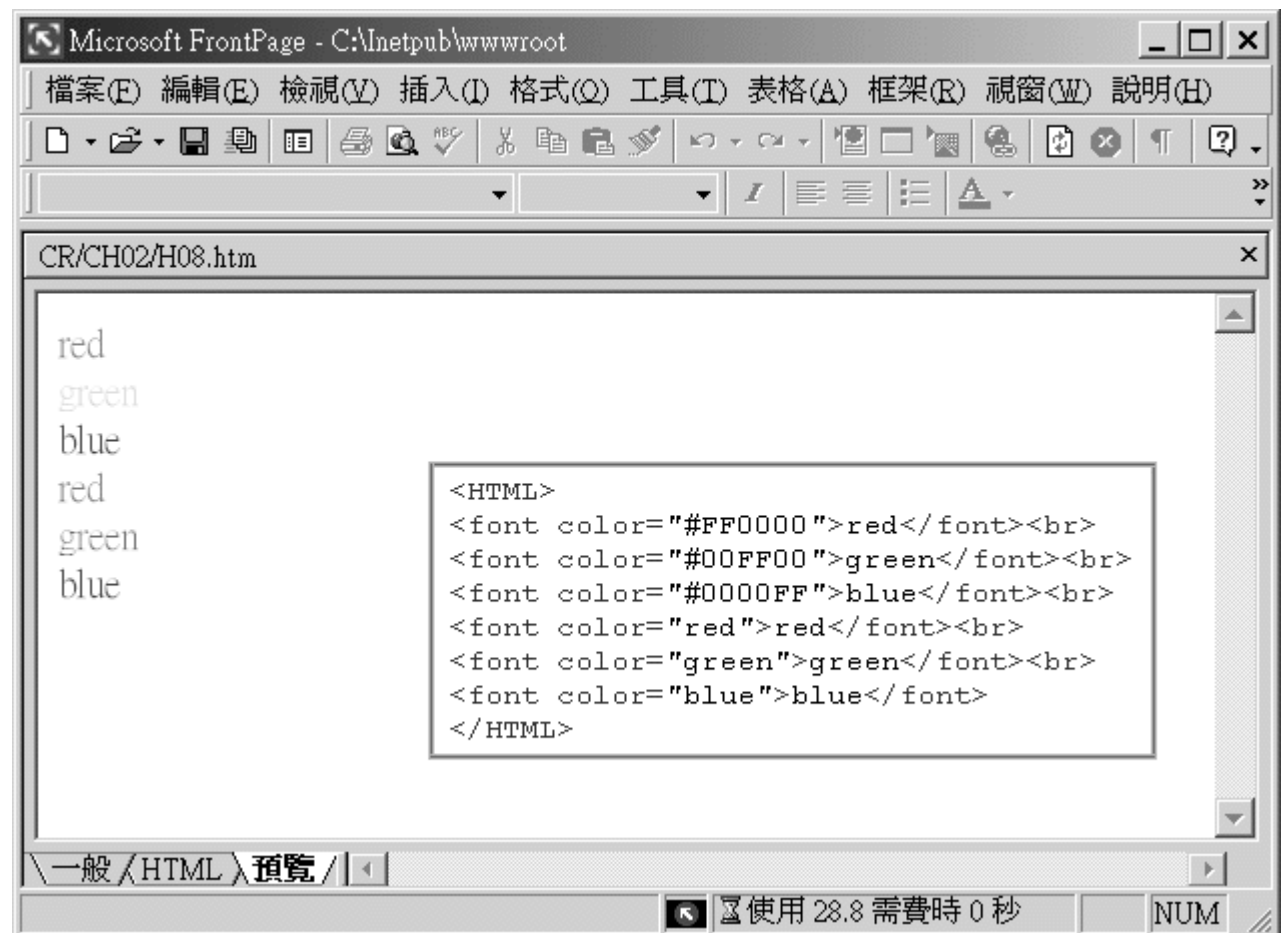
<font color="#0000FF">blue</font><br>

<font color="red">red</font><br>

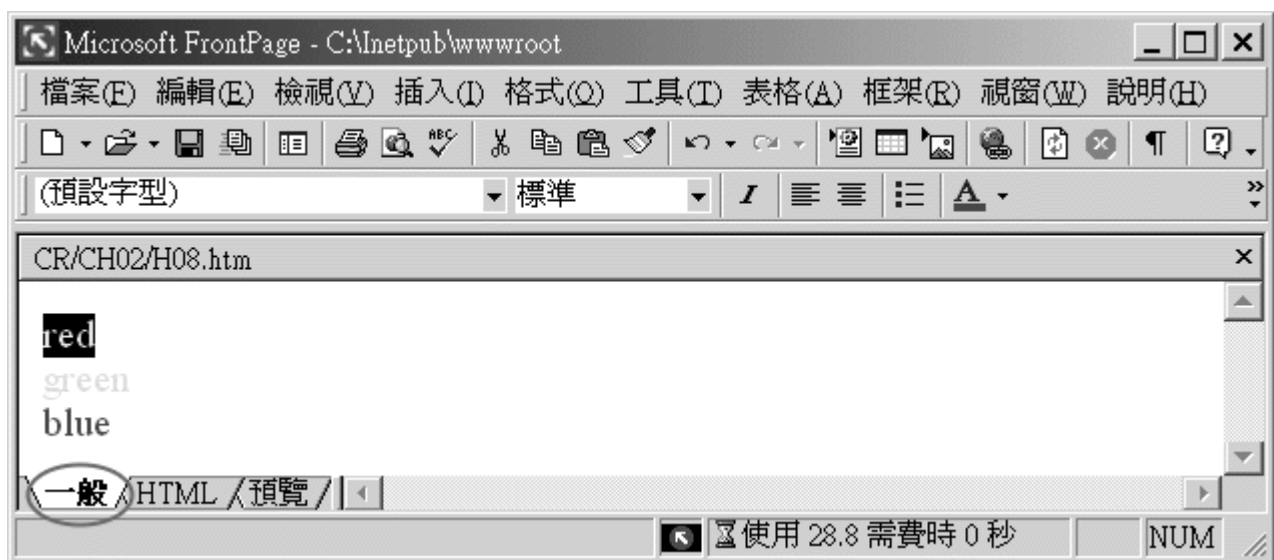
<font color="green">green</font><br>

<font color="blue">blue</font>
```

</HTML>

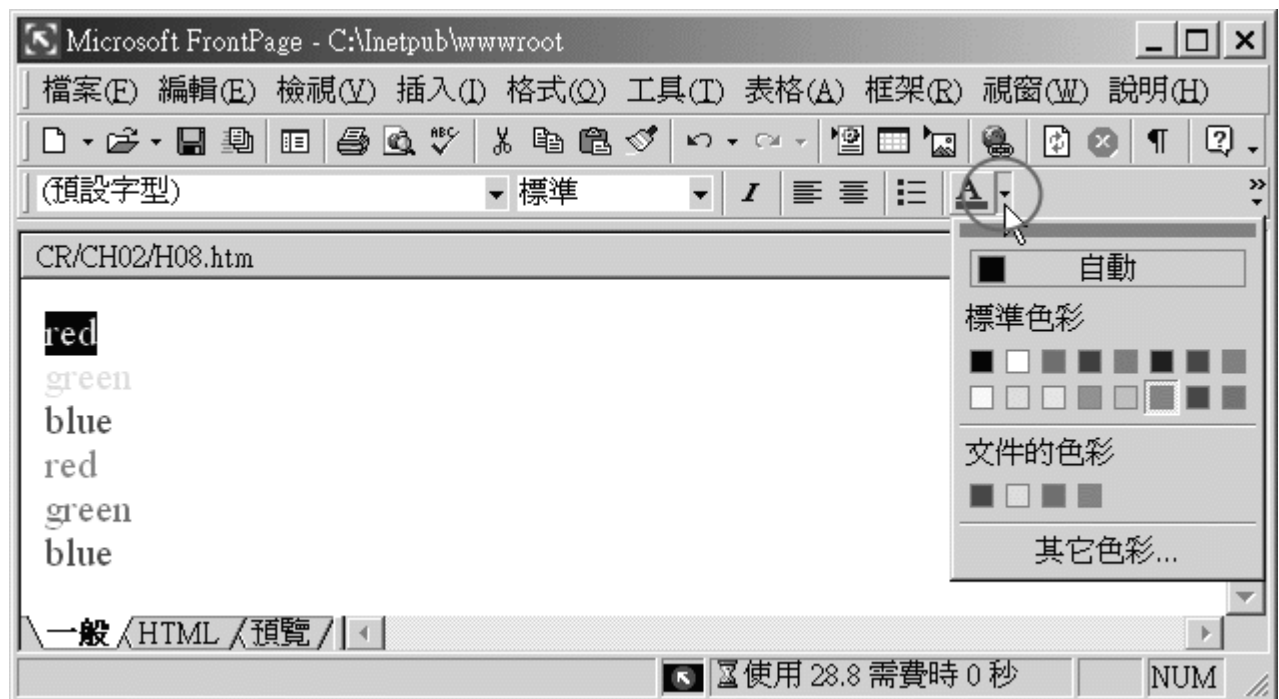


只要调整好 RGB 三原色的强度，即可调配出各种颜色。但是并不建议这做，因为 FrontPage 提供了视觉的方式直接选择你所需要的颜色，不但快速而且方便。首先，先切换到「一般」窗口，并且选择所要调整颜色的文字区域：

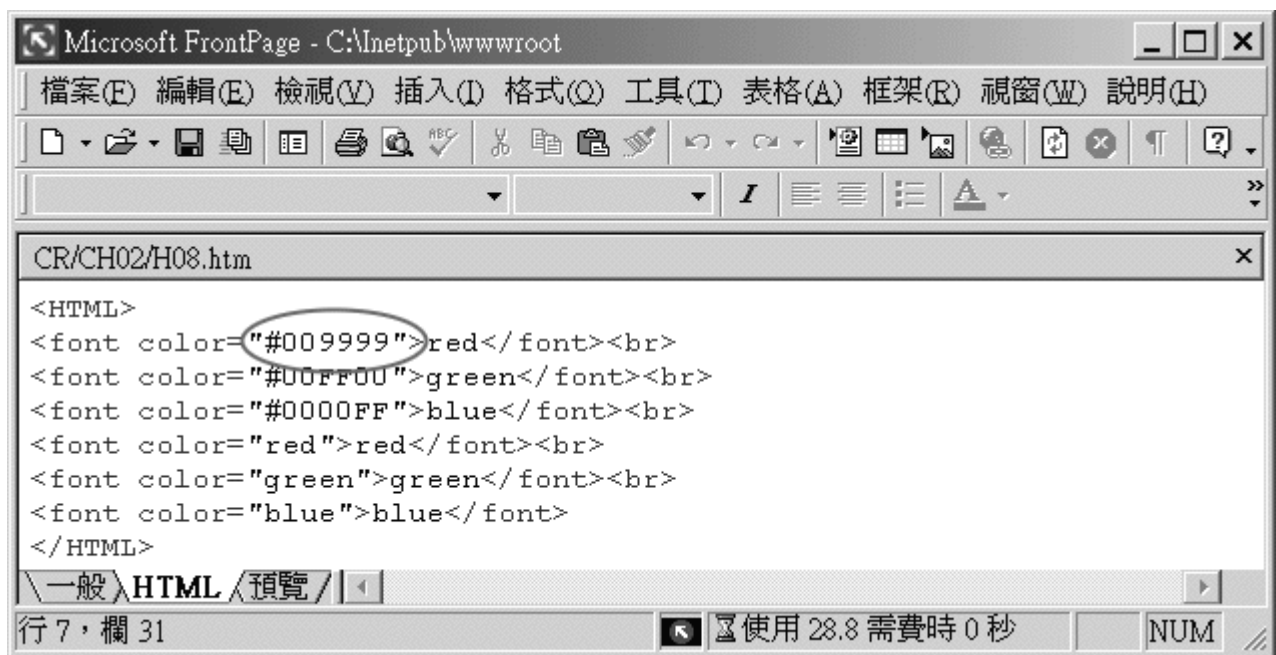


然后把「字型色彩」旁的小箭头拉下来，你可以直接选择上面的颜色，或是点选「其它色彩」选

择丰富的颜色：



选择完毕后切到「HTML 窗口」即可得到所要的颜色值：



**** 粗体、**<i>** 斜体及 **<u>** 底线标注:

接下来的 ****、**<i>** 以及 **<u>** 分别为设定字体是否为粗体、斜体或是加底线:

```
<html>

正常字示范<br>

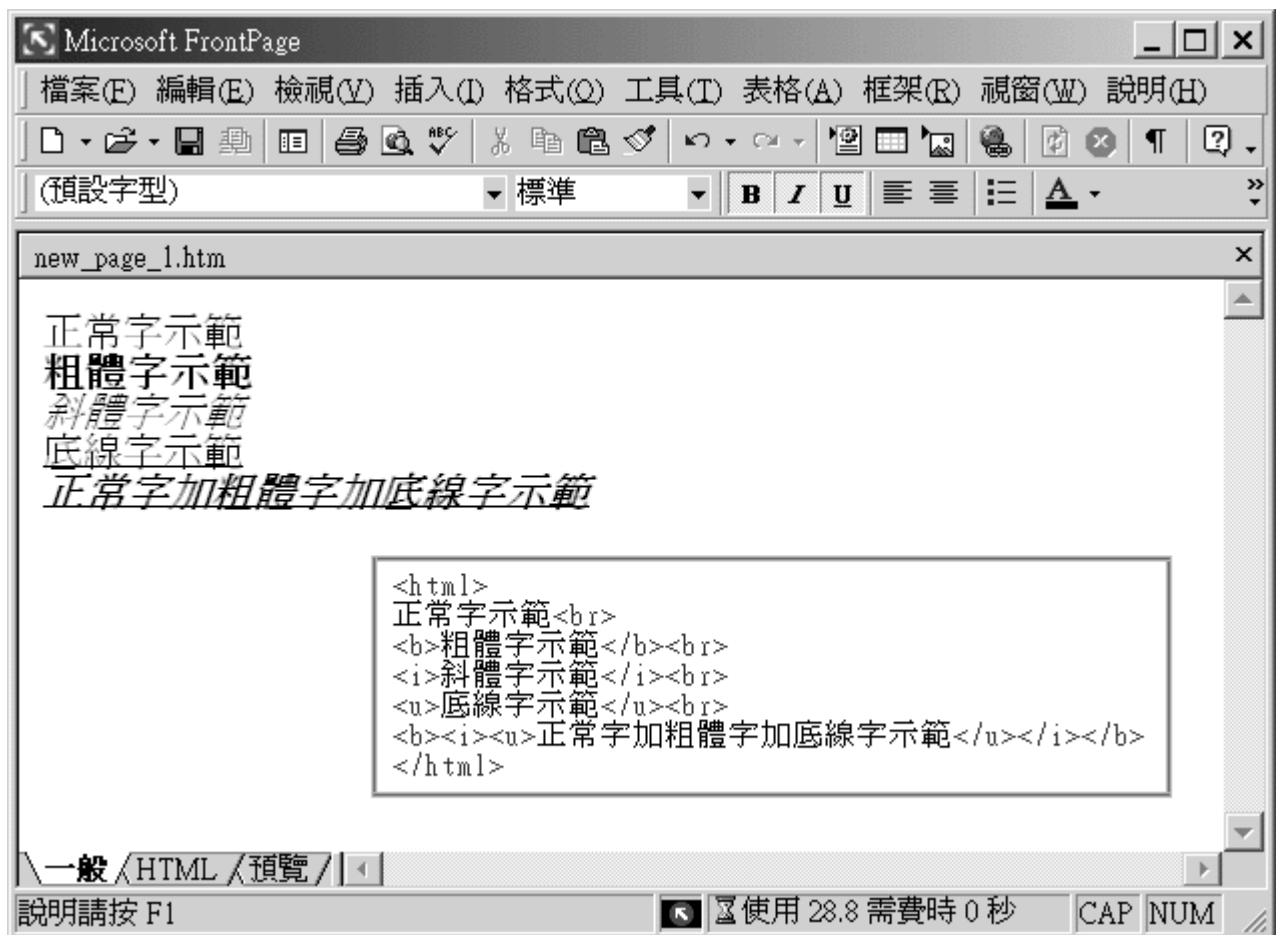
<b>粗体字示范</b><br>

<i>斜体字示范</i><br>

<u>底线字示范</u><br>

<b><i><u>正常字加粗体字加底线字示范</u></i></b>

</html>
```



段落及項目

<h1>～<h6> 標題标注：

这个标注是用来设定用来当作标题的文字大小，从 <h1> 到 <h6> 总共有六级：

```
<html>
```

```
<h1>标题一</h1>
```

```
<h2>标题二</h2>
```

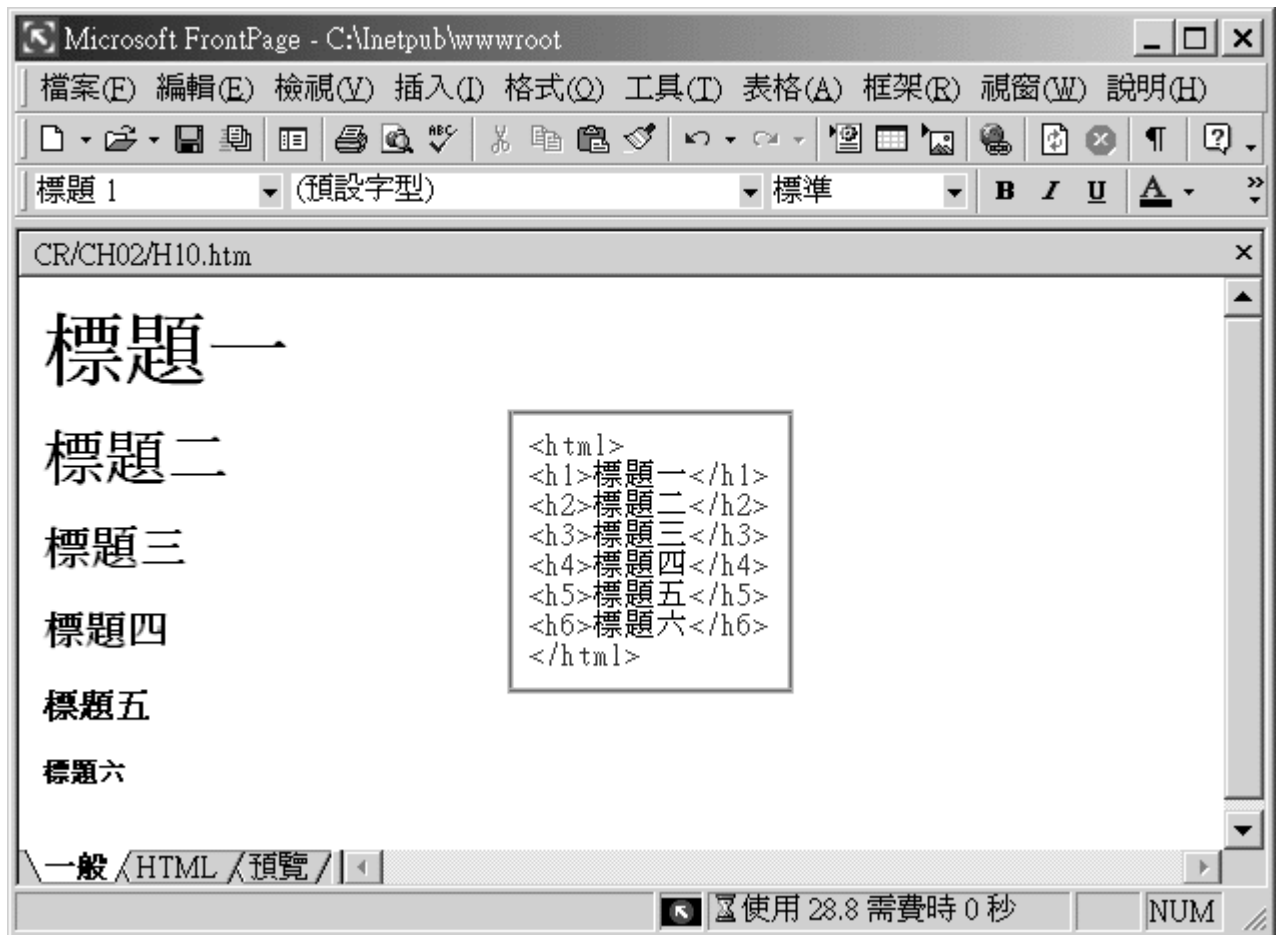
```
<h3>标题三</h3>
```

```
<h4>标题四</h4>
```

```
<h5>标题五</h5>
```

```
<h6>标题六</h6>
```

```
</html>
```



<blockquote>缩排标示:

这个标示可以用来设定某段落是否要缩排:

```
<html>
```

.Net Framework 总共分为三大部分

```
<blockquote>
```

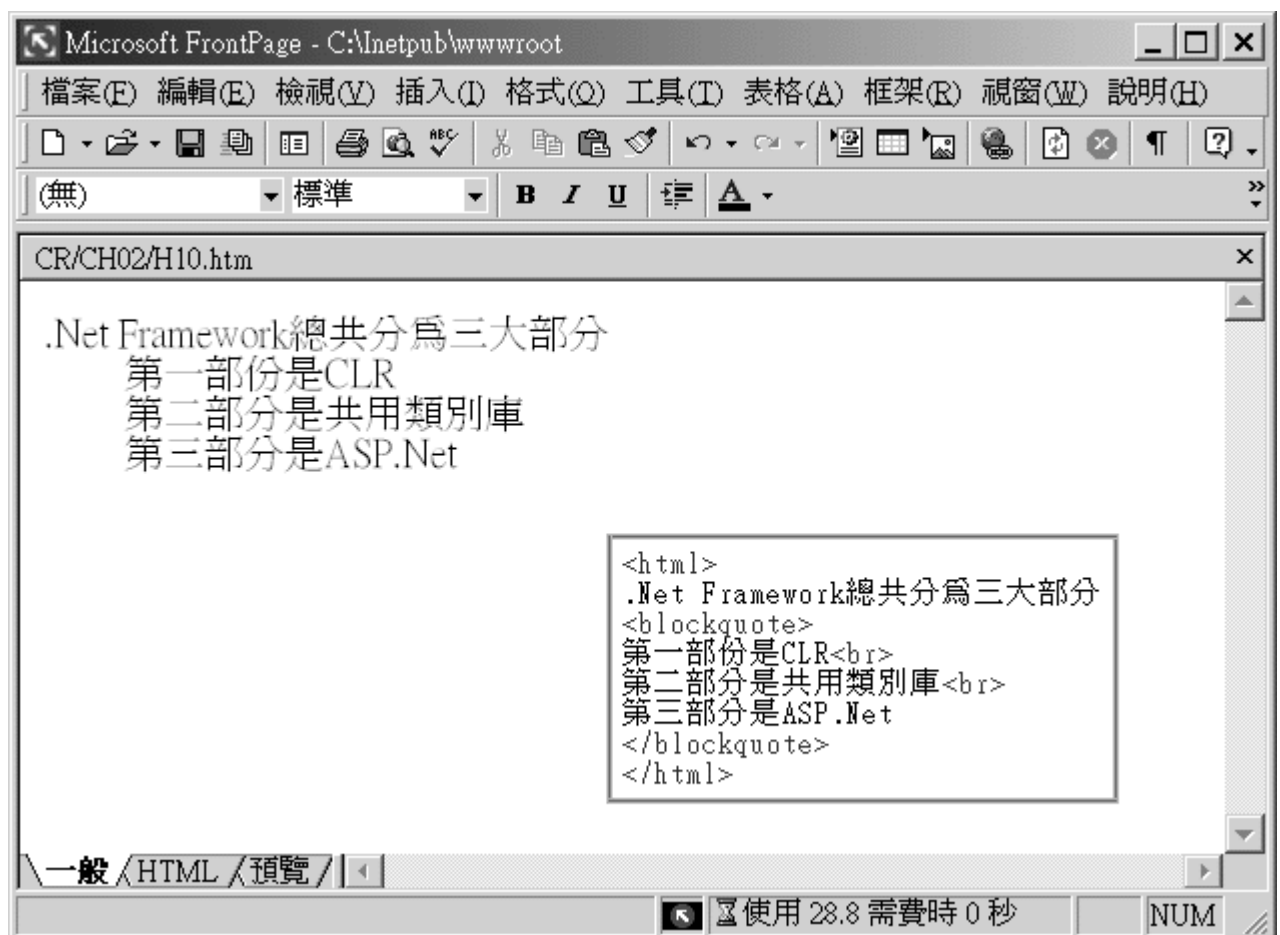
第一部份是 CLR

第二部分是共享类别库

第三部分是 ASP.Net

</blockquote>

</html>



 条列标注:

本标注是用来显示条列的数据，可以自动将条列加入编号，其中要显示编号的条列前面加上 ``

标注即可：

```
<html>

.Net Framework 总共分为三大部分

<ol>

<li>第一部份是 CLR<br>

所有的程序语言都共同使用单一的执行时期

<li>第二部分是共享类别库<br>

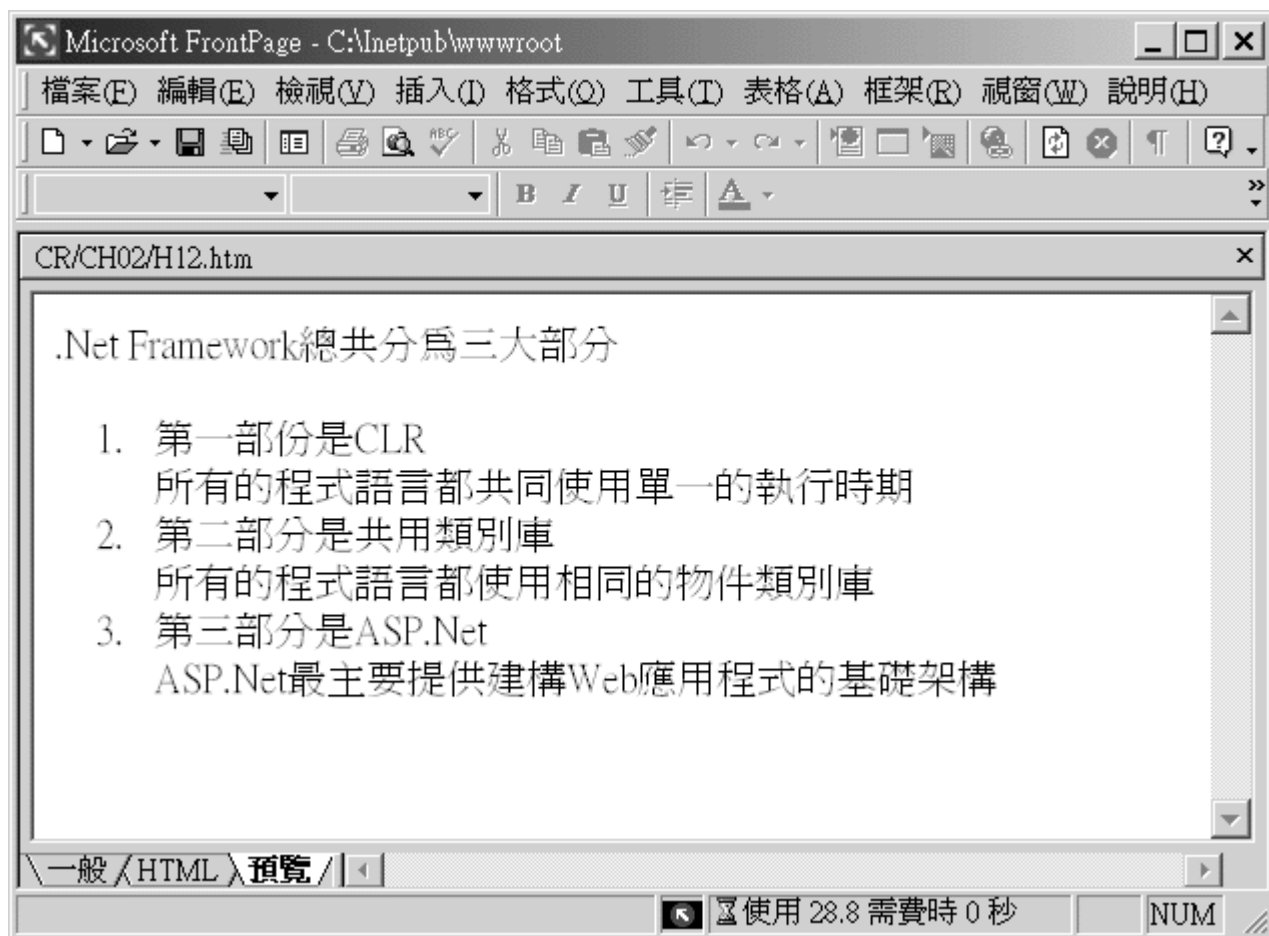
所有的程序语言都使用相同的对象类别库

<li>第三部分是 ASP.Net<br>

ASP.NET 最主要提供建构 Web 应用程序的基础架构

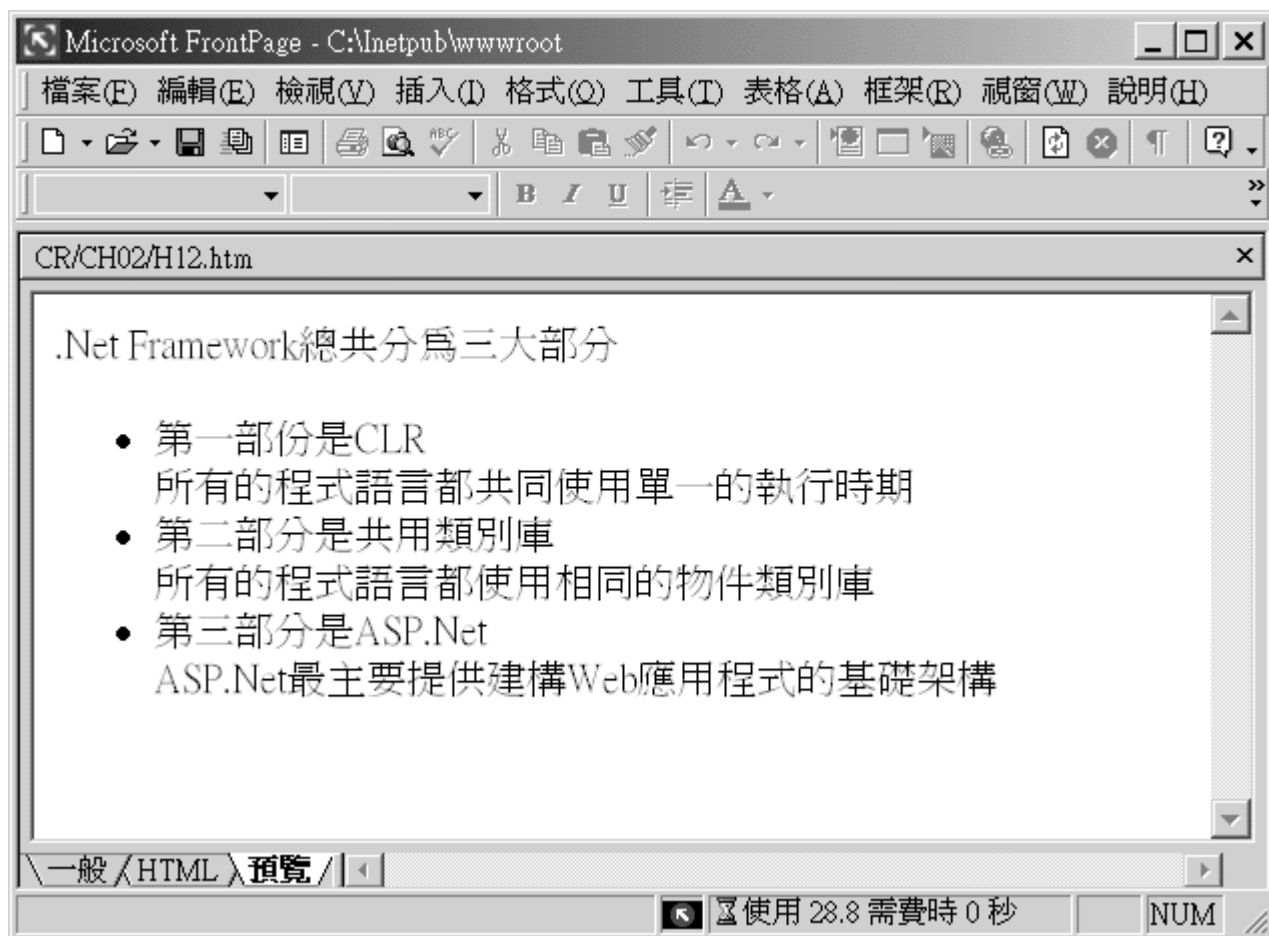
</ol>

</html>
```

条列标注:

本标注是用和 一样用来显示条列的数据，但是以项目符号（•。■）来显示，只要把刚刚的 标注改成 就可以了：



<div> 段落对齐标注:

想要设定一个段落的对齐属性，只要在 `<p>` 标注内设定 `align` 属性就可以了。倘若要设定所有段落的对齐属性，那么每个 `<p>` 标注内都要设定，对我们来说太麻烦了。这时候只要用 `<div>` 标注设定 `align` 属性，并将要影响的段落含括到 `<div>` 标注内就可以了：

```
<html>  
  
<div align="right">
```

.Net Framework 总共分为三大部分

<p>第一部份是 CLR

所有的程序语言都共同使用单一的执行时期</P>

<p>第二部分是共享类别库

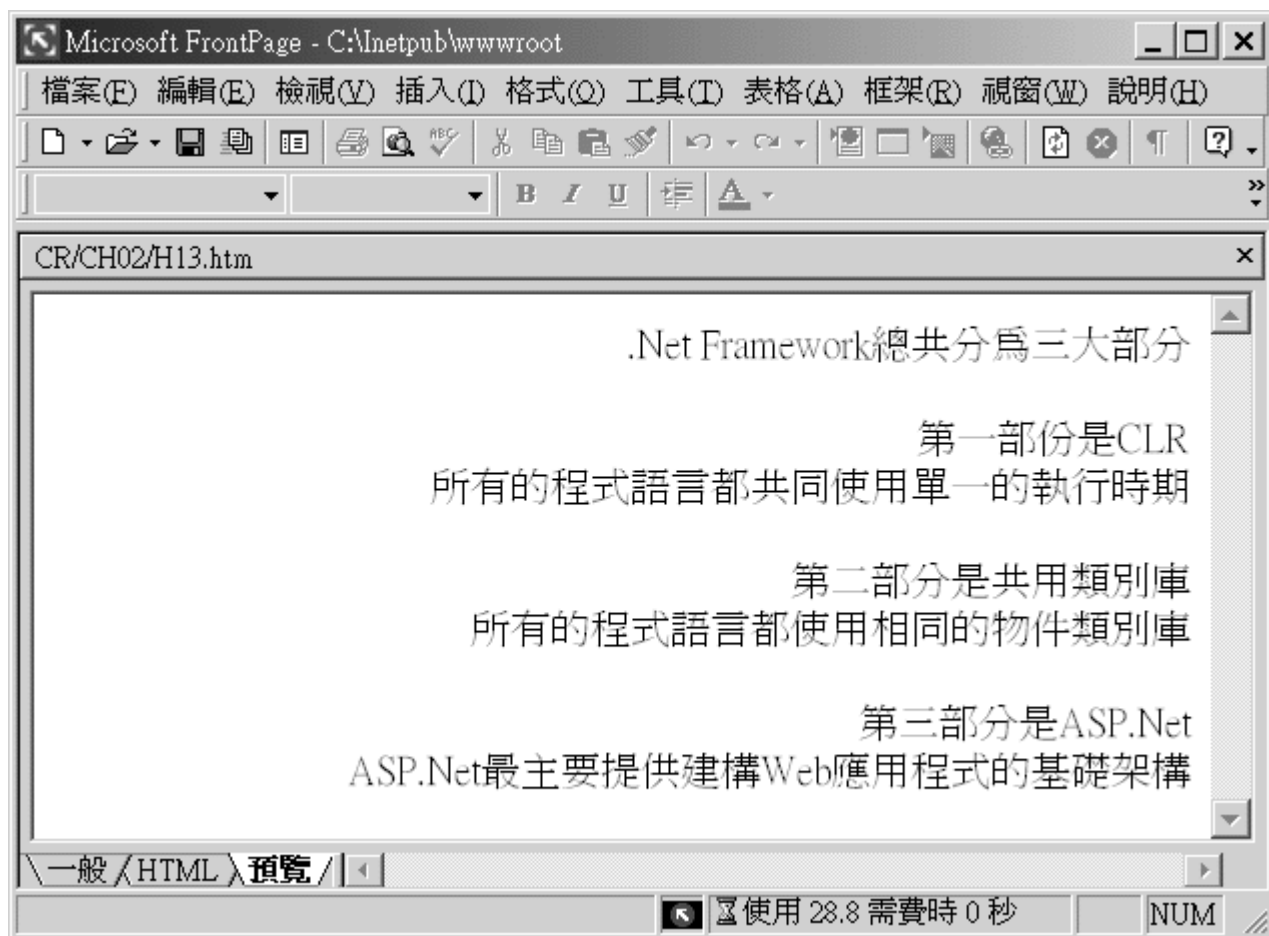
所有的程序语言都使用相同的对象类别库</p>

<p>第三部分是 ASP. Net

ASP. Net 最主要提供建构 Web 应用程序的基础架构</p>

</div>

</html>



其它和文字呈现有关的标注

除了上面所提的标注之外，还有一些和设定文字有关的标注：

```
<html>
```

```
<!-- 这是批注，给开发人员看的，不会被解译 -->
```

```
<big>这是大型字</big><br>
```

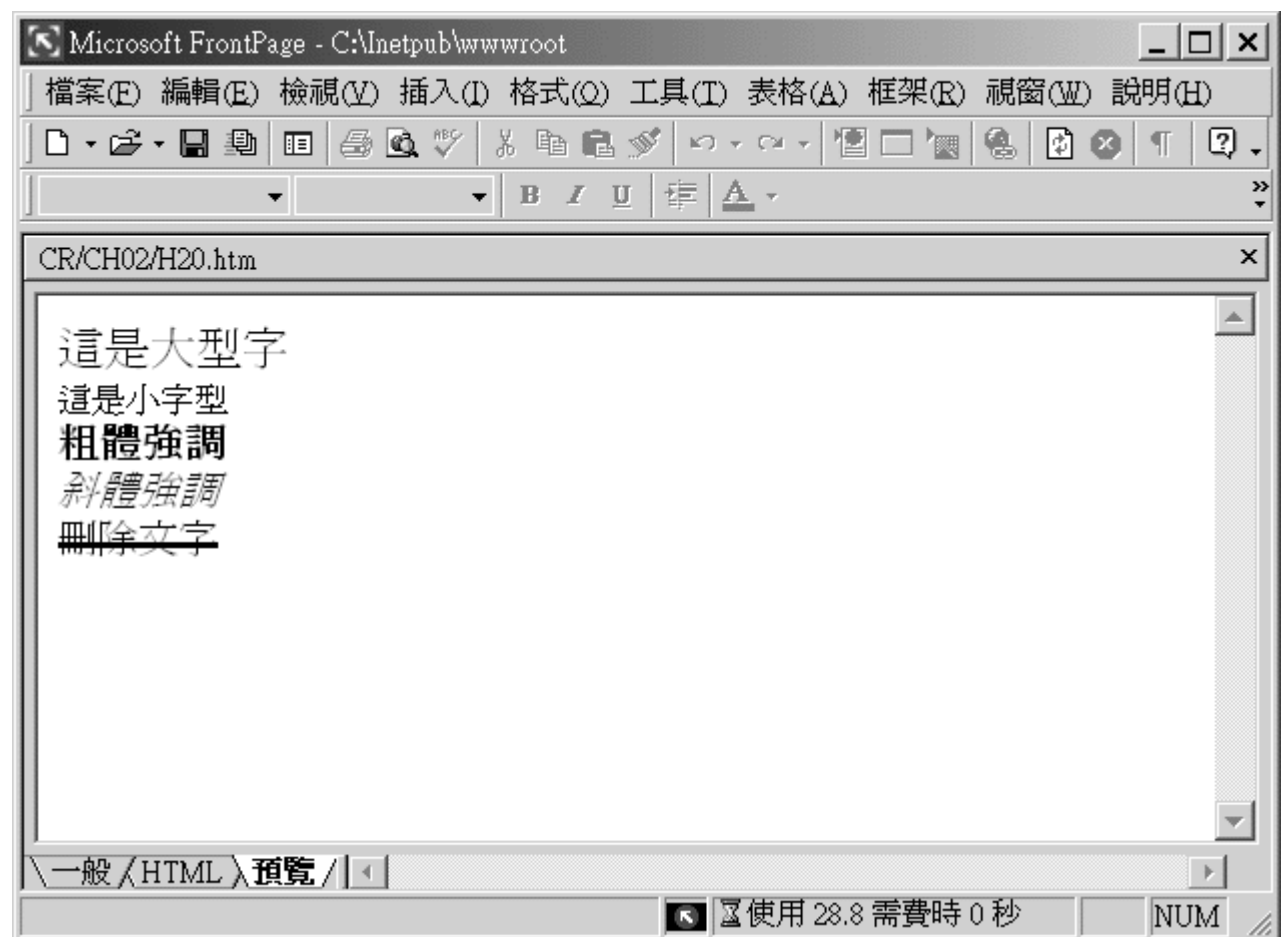
```
<small>这是小字型</small><br>
```

粗体强调

斜体强调

<s>删除文字</s>

</html>



表格

善用表格来群组数据，可以让网页的版面更容易让使用者接收，也可以整齐的配置文字与影像的位置。要产生表格，必需搭配 `<table>`、`<tr>` 以及 `<td>` 这三个标注。`<table>` 标注用来表示表格的开始及结束，`<tr>` 则表示其中列的开始及结束，`<td>` 则表示一列中的字段，而字段我们称为储存格。我们先来看看下面的表格：

```
<html>

<table border="1">

  <tr>

    <td>第一列第一栏</td>

    <td>第一列第二栏</td>

  </tr>

  <tr>

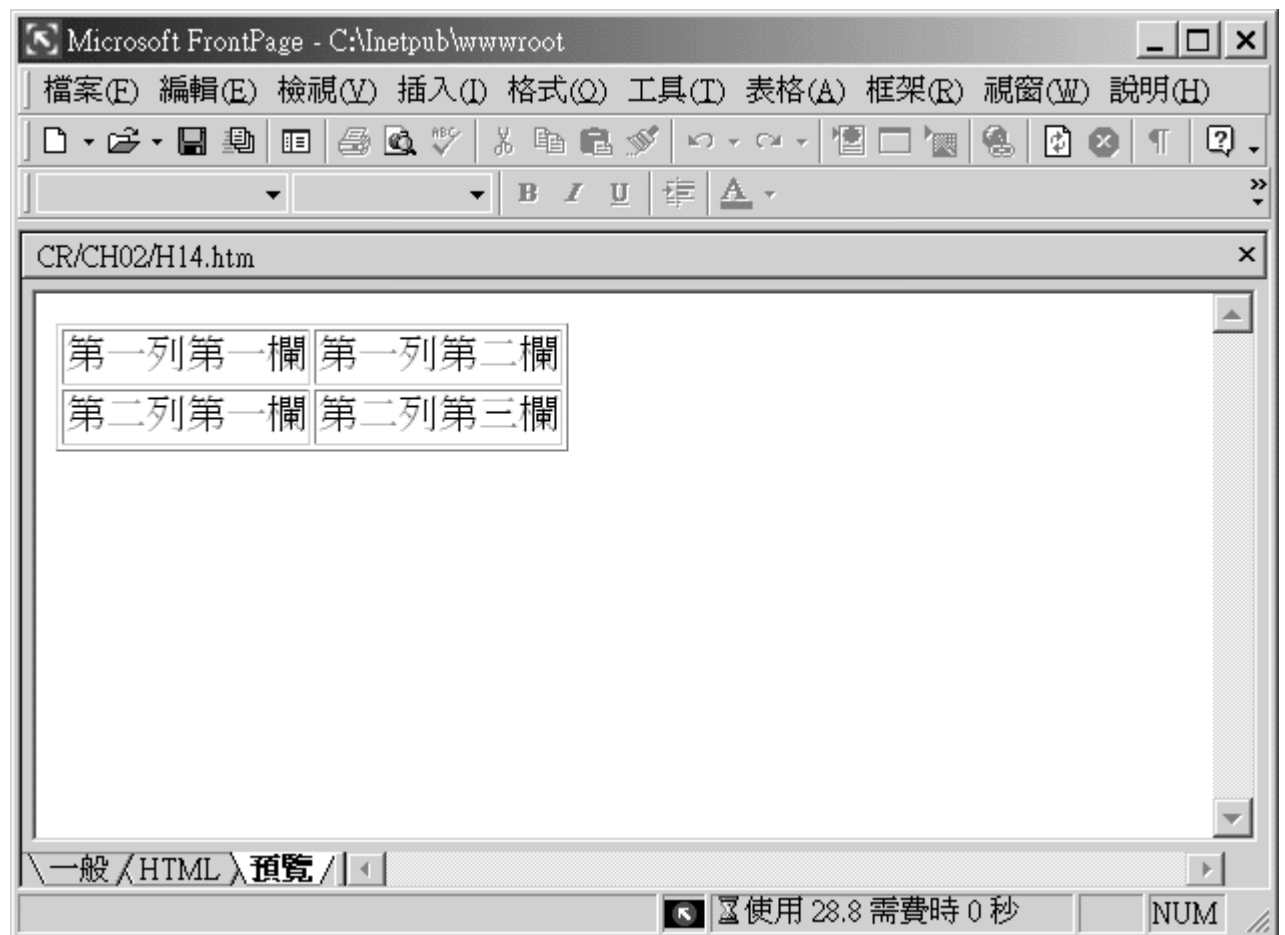
    <td>第二列第一栏</td>

    <td>第二列第三栏</td>

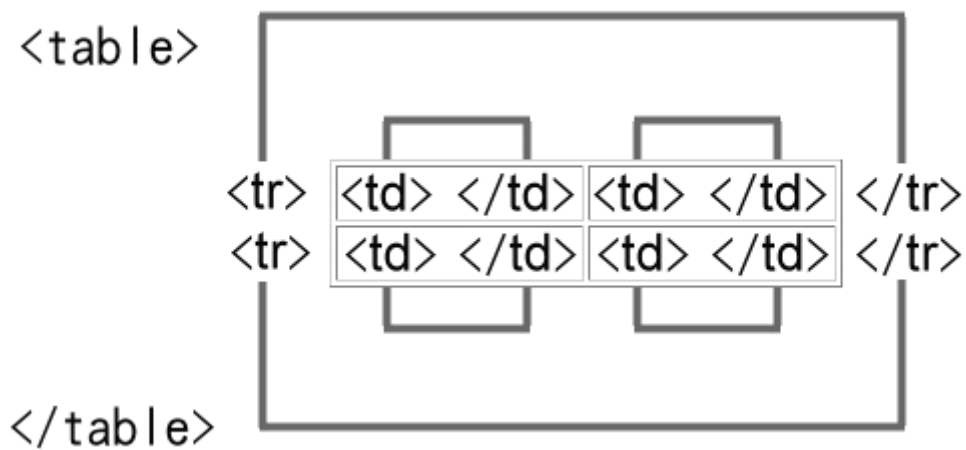
  </tr>

</table>

</html>
```



我们来看看这个表格和这三个标注的关系：



表格要被 `<table>` 标注所包围，而一列内所包含的字段要被 `<tr>` 所包围，接下来才是定义一列内有多少栏 `<td>`。另外 `<table>` 标注的 `border` 属性则表示表格的边框宽度，如为 `0` 则此表格无边框。除 `border` 属性外还有其它有用的属性，我们一一来介绍。

align 属性:

对齐属性，如果是在 `<table>` 标注内设定此属性，则表示是设定整个 `table` 的对齐；如果在 `<tr>` 中设定，则表示整列文字的对齐受影响；若是在 `<td>` 标注内，则表示此字段内文字的对齐会受影响：

```
<html>

<table align="center" border="1">

  <tr>

    <td align="right">靠右</td>
```



```
<td align="left">靠左</td>

</tr>

<tr align="center">

    <td>靠中</td>

    <td>靠中</td>

</tr>

<tr>

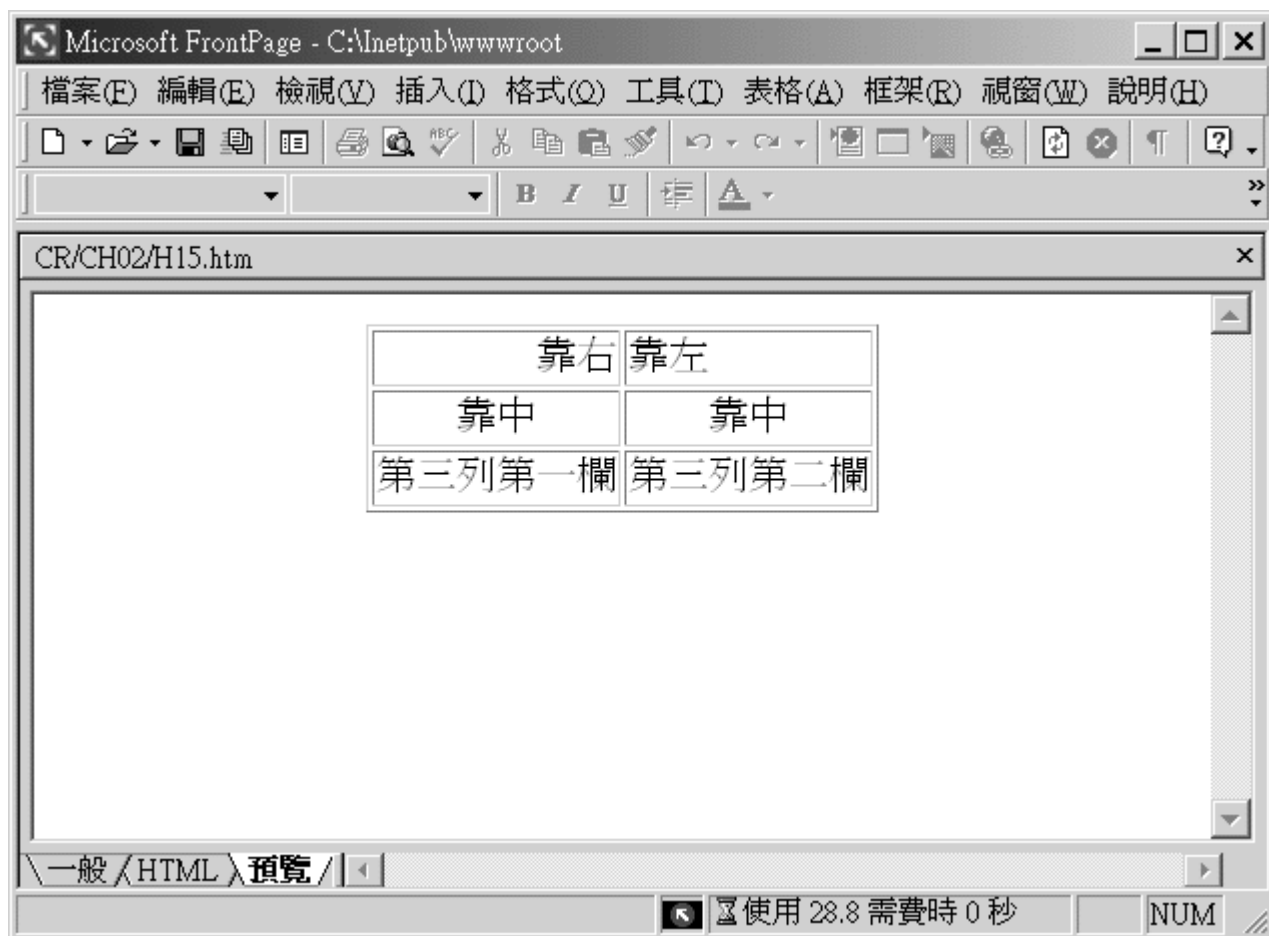
    <td>第三列第一栏</td>

    <td>第三列第二栏</td>

</tr>

</table>

</html>
```



bgcolor 属性:

本属性来设定表格的背景色，一样可以分别设定 `<table>`、`<tr>`、`<td>` 这三个标注:

```
<html>

<table align="center" border="1" bgcolor="cyan">

  <tr bgcolor="yellow">

    <td align="right">靠右</td>
```

```
<td align="left">靠左</td>

</tr>

<tr align="center">

  <td bgcolor="red">靠中</td>

  <td>靠中</td>

</tr>

<tr>

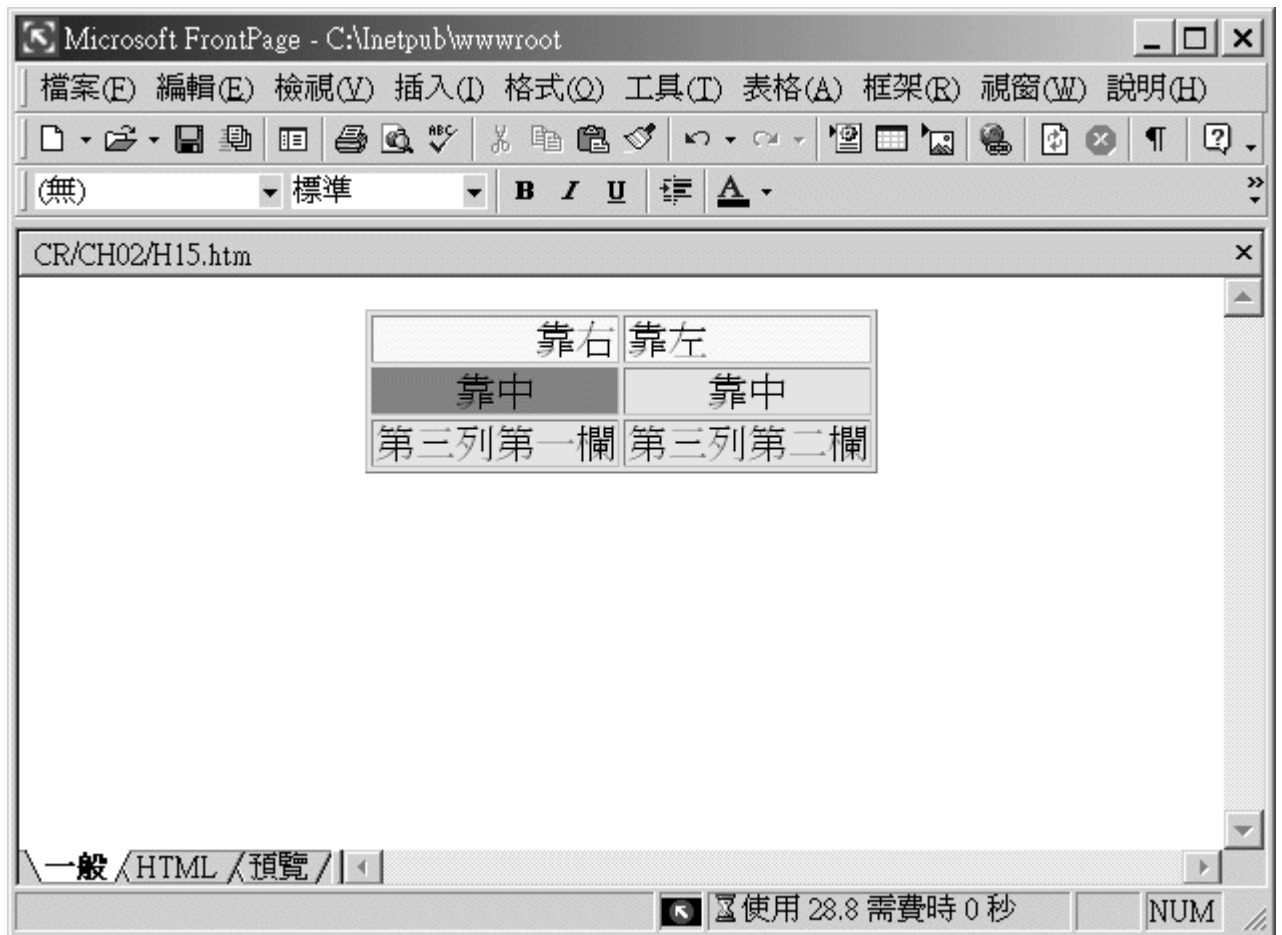
  <td>第三列第一栏</td>

  <td>第三列第二栏</td>

</tr>

</table>

</html>
```



cellpadding 及 cellspacing 属性

这两个属性用来设定储存格与表格边框的距离以及储存格和储存格边框的距离，我们看看下面范

例：

```
<html>

<table border="10" cellpadding="10" cellspacing="10">

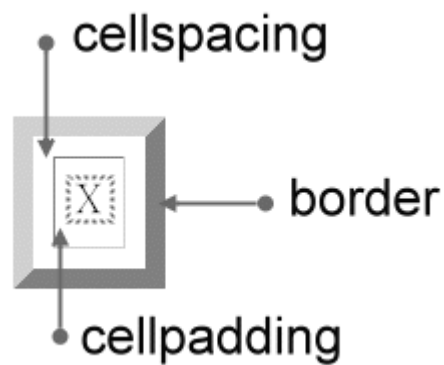
  <tr>
```

```
<td>X</td>

</tr>

</table>

</html>
```



width 属性:

本属性可以用来设定表格及储存格的宽度，单位有像素及百分比两种。如果要以像素来设定，只要只直接给像素值即可；如果要以百分比设定，则在像素值后面加上百分比符号。如果像之前范例没有设定本属性，则表格的宽度以储存格内的数据长度为准。

```
<html>

<table align="center" border="1" width=400>
```

```
<tr>

    <td>第一列第一栏</td>

    <td>第一列第二栏</td>

</tr>

</table><br>

<table align="center" border="1" width=90%>

    <tr>

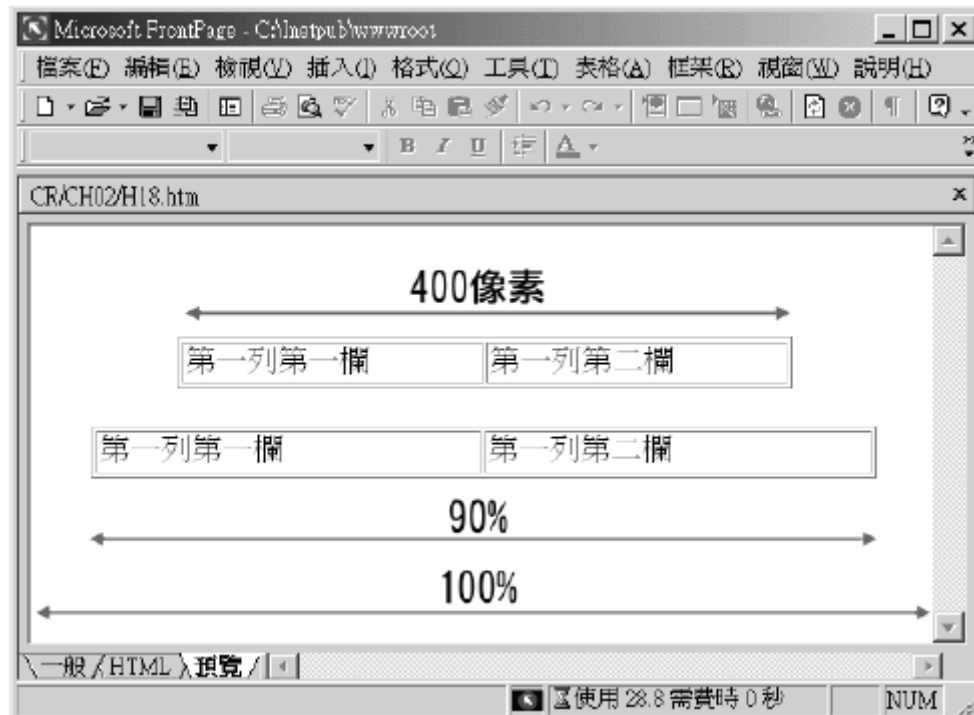
        <td>第一列第一栏</td>

        <td>第一列第二栏</td>

    </tr>

</table>

</html>
```



超级链接

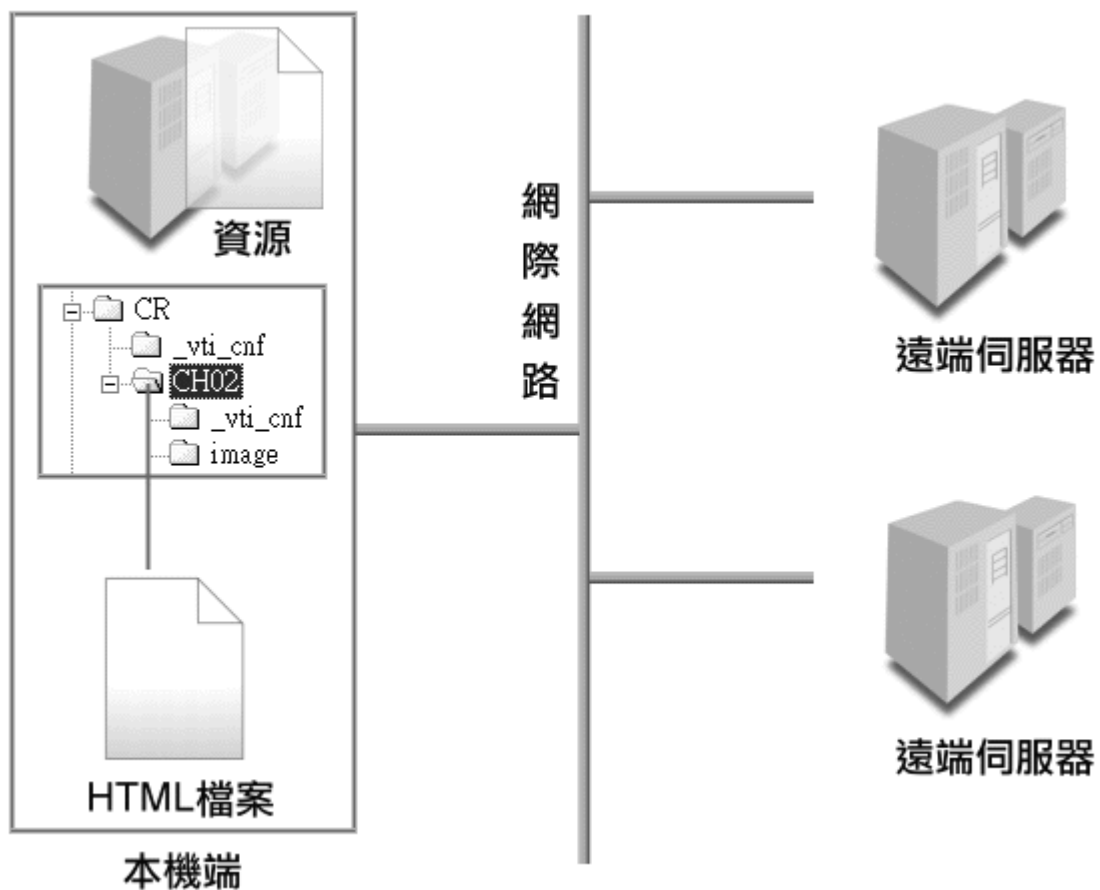
前面我们学了这么多的 HTML 标注后，还有一个重要的观念还没介绍给大家，这个观念就是「超级链接」。什么是超级链接呢？超级链接的意思是指向某个固定的地址，因为该地址有网页需要的图片、声音、档案、影像，甚至是网页等等；而这个位置可能在一台计算机上，也可能在因特网上的某个地址。因特网上的每一个网站都有一个独一无二网址，这个网站的地址我们叫做 URL（Uniform Resource Locator）。如果我们需要的资源放在因特网上面的某个地方，我们就需要为指定到该 URL 上。

地址的指定方式

要使用其它地址的资源，首先我们要知道如何表示资源的地址。地址的表示法有相对位置及绝对位置两种，如果所要使用的资源放在和网页同样的地方，那么这两种方法都可以使用；如果所使用的资源放在因特网其它地址，那就只能用绝对地址表示法来寻址。

相对地址

相对地址表示法表示是以 HTML 网页所在的数据夹作为参考基础。例如 HTML 档案所在的数据夹地址为 `c:\inetpub\wwwroot\CR\CH02`，我们要在该网页中利用 `` 标注显示影像，并设定 `src` 属性用来指明影像资源文件所在的位置。资源档假设名为 `train.jpg`，并放在同一台机器的同一个数据夹内。这时候我们就可以直接指定资源的档名，例如 `img src="train. jpg ">`。假如资源文件放在 HTML 文件所在路径的下一层名为 `image` 的数据夹中，那么要写为 ``。而若是我们的资源放在上一层数据夹中，那么要写成 ``。



绝对地址

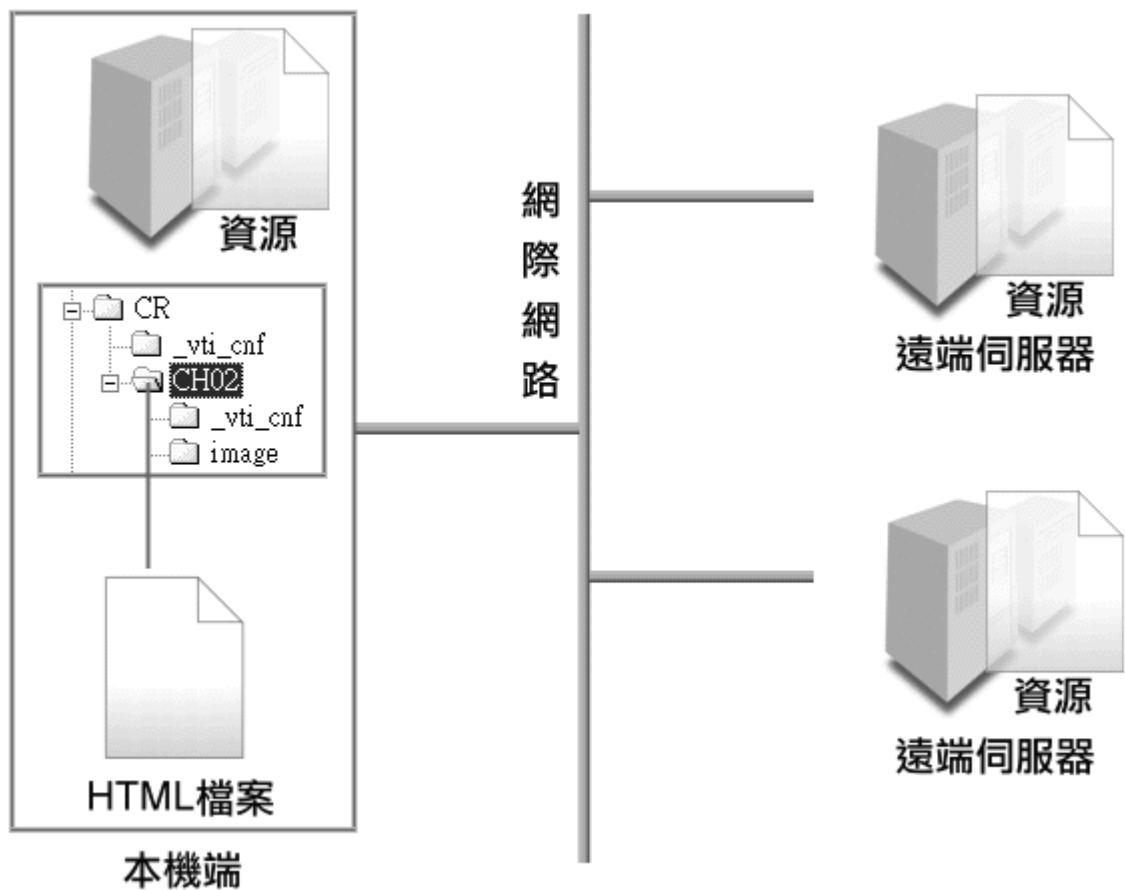
另外一种标示方法为绝对位置表示法，表示资源的地址必需详细指定。以上述例子来说，假设我们所在的服务器的网址为 `www.msn.com`，而且 HTML 档案放在 `c:\inetpub\wwwroot\CR\ch02` 里；如果资源档 `train.jpg` 放在和 HTML 档案同一个数据夹内，指定的方式则为 `<img`

`src="http://www.msn.com/CR/ch02/train.jpg">`。假如资源文件放在 HTML 文件所在路径的下一层名为 `image` 的数据夹中，那么位置的描述则为 `<img`

src="http://www.msn.com/CR/ch02/image/train. jpg ">。若是我们的资源放在上一层数据夹中，

则要写成 。所以如果资源档是放在因特网远程

的服务器中，必须要详细的指定放在所在的 URL 地址。



 图形标注:

要插入图形必需要使用 标注，并指定图片所存放的地址：

```
<html>
```

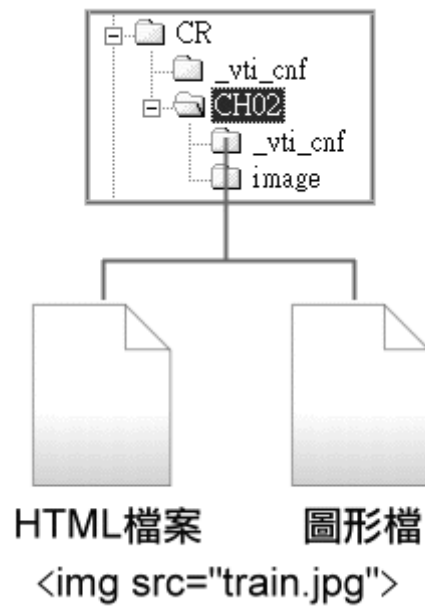
```

```

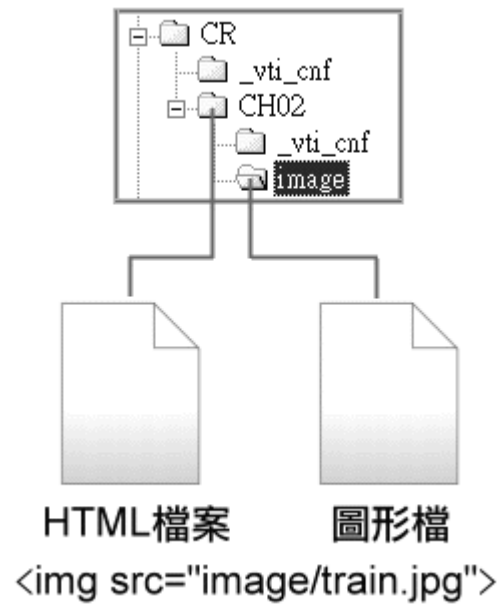
```
</html>
```



其中 `` 标注中的 `src` 属性是指定所要显示的影像文件放在哪的地址上，上面的范例中因为影像文件和网页是放在同一个资料夹中，所以可以直接指定档名。



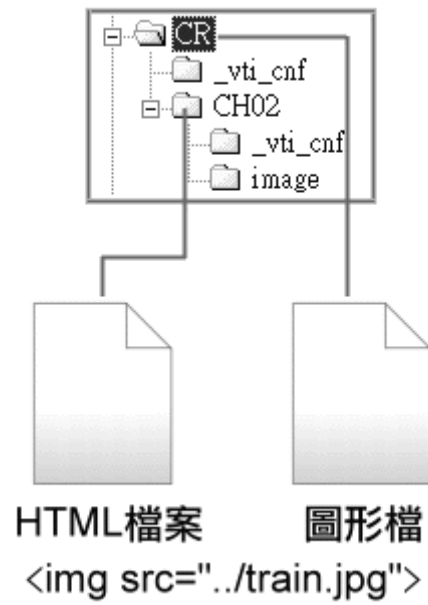
网页上所用到的资源也有可能放在其它的位置，我们来看看资源放在下一层资料夹的情形：



影像资源放在 image 数据夹里

这时候 src 属性的设定就要变成 `` 即可。接下来是资源放在上

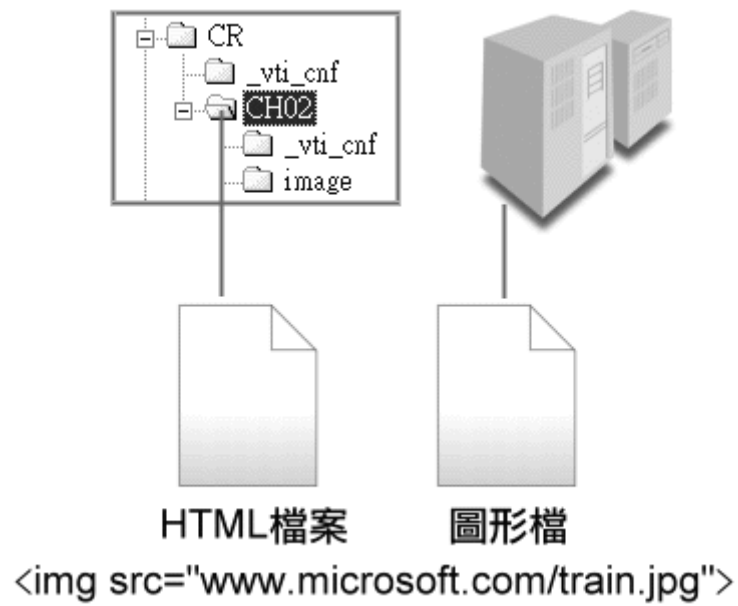
一层数据夹的情形：



影像数据源放在 CR 数据夹里

这时候 **src** 属性设定就要变成 `` 即可，「..」表示是上一层资料夹。

最后一种情形是资源文件放在网络上其它地址，例如放在微软的网站内的情形：



影像资源放在 www.microsoft.com 内

这时候 `src` 属性设定就要变成 ``即可。

<a> 连结标注:

使用者按下被 `<a>` 标注所包围起来的文字或影像, 就会将地址导向 `href` 属性所指定的地址, 如

下范例所示:

```
<html>

<a href="http://www.microsoft.com">



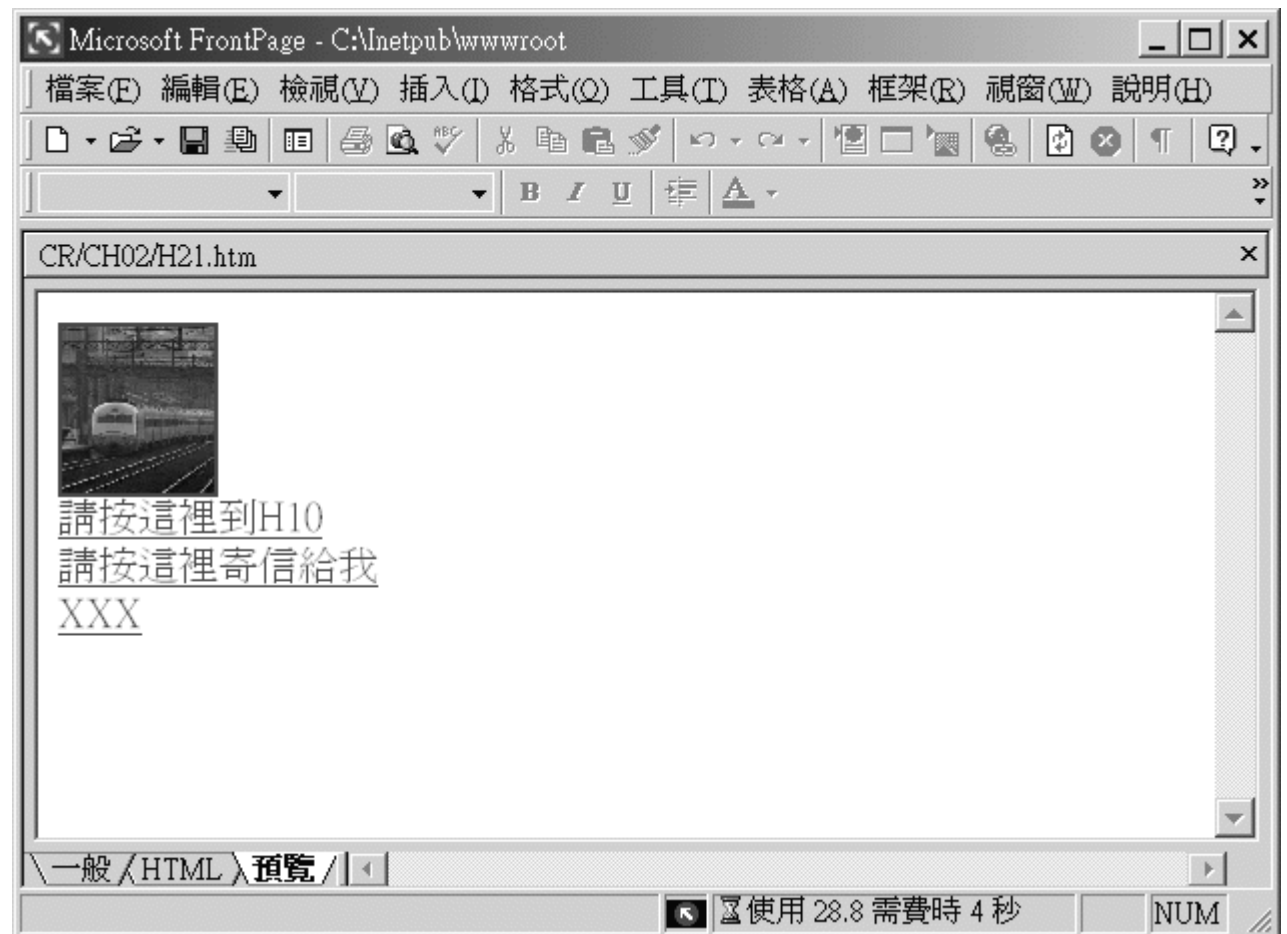
</a><br>
```

请按这里到 H10

请按这里寄信给我

XXX

</html>



如果 href 后面所指定直接是网址，那么表示被 <a> 标注所围起来的文字或图形被点选后，会将超级链接导向该网址。如果 href 后面所指定的直接是资源文件名，例如「 请按这里到 H10
」这行，那么表示按下「请按这里到 H10」时，会将网页导向同一个数据夹的 H10.htm；若 href 属性后是 mailto 的字样，如「 请按这里寄信给我
」这行叙述，表示使用者按下后会开启邮件编辑程序，而收信人就是 mailto 后面的电子邮件信箱。

3. Visual Basic.NET 语言

在第二章〈HTML 语言〉中，我们已经学会如何用 HTML 叙述将我们所要呈现的数据编辑好，并且用 IE 来观看 HTML 文件。但这样的网页是固定不会变动的静态网页，要让网页依照你所设定的条件而显示不同的数据，必需撰写程序才办的到，所以我们要学习如何使用程序语言。

ASP.NET 支持许多种程序语言，例如 VB.NET 及 C# 等，在这里我们使用的是 VB.NET 语言。

很多使用者用过 VB.NET 后下了个结论—「VB 已死」，这一点其实完全不正确；因为 VB 不但没有死，而且经过改造后比以前更强悍，简直可以说是浴火重生。如果新一版的 VB.NET 还和以前的 VB 一样没有什么革新和进步的话，那么才真的会被宣告死刑。微软可以说是将 VB 语言做一个大幅度的翻修，不但把 VB 语言程序的一些特质修正的和其它程序语言一致，还让 VB 更彻底的支持如继承、覆载等对象导向的能力，而且不会自动帮你做一些额外的处理；也因为这些新革新以及功能，让执行 VB.NET 起来更有效率。把 VB.NET 程序语言的特质修改的和其它语言一致，这样一来我们利用其它程序语言撰写程序时，就不会被这些小地方的差异所困扰。

VB.NET 经过了大幅修正的好处，除了语法和其它程序语言不一样之外，语言的特质一样、所面对的对象也都一样（都是从同一个 .NET 基础对象库中产生的）；这样一来开发人员就可以选择他们喜爱的程序语言来开发某个项目或是组件。因为这些语言都是使用 .NET 基础对象库并编译成一致的 IL 格式，所以最后可以将这些组件（Assemblies）组装在一起，成为我们所需要的解决方案。

基础观念

在 ASP.NET 中所使用的语言是 VB.NET，和先前 ASP 网页所使用的 VB Script 有许多地方不一样。如果你学过前几版的 VB 语言或是使用过 VB Script 语言，建议重新将本章看过一遍，因为 ASP.NET 中所用的语言是 VB.NET 而不是 VB Script。

数据的输出

我们在学习许多程序语言的时候，第一个程序就是印出「Hello world」。我们也未能免俗，第一个程序是在 IE 上显示「Hello world」。请打开 FrontPage 后，在 HTML 的窗口中输入以下程序：

```
<html>

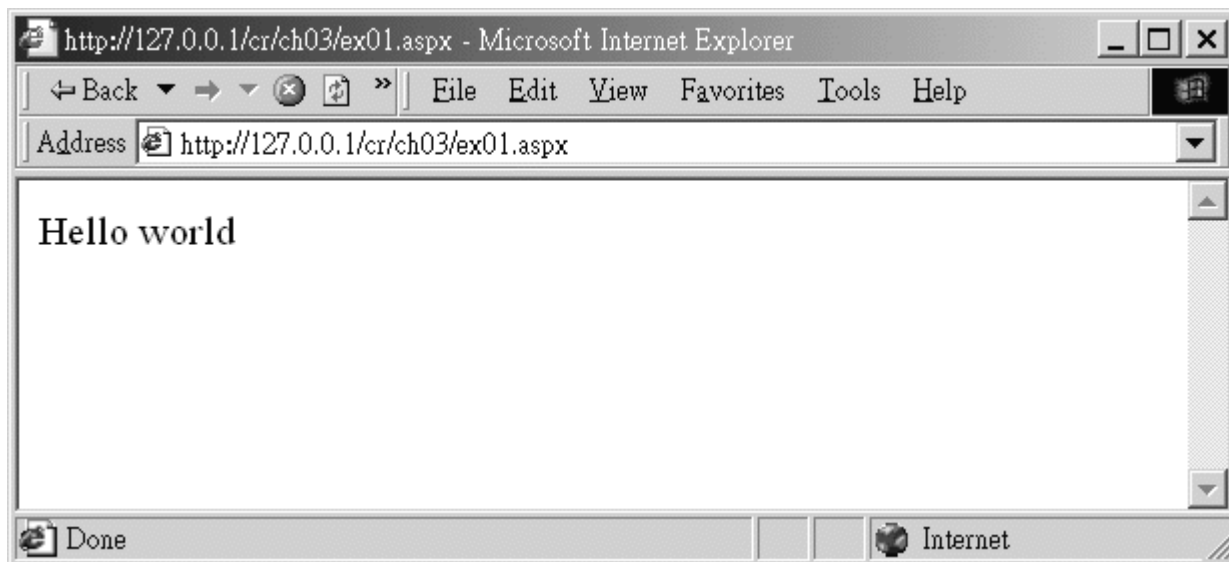
<%

Response.Write("Hello world")

%>

</html>
```

我们将程序储存于 C:\inetpub\wwwroot\CR\CH02 内，选择档类型时选择「所有档案」后，将档案取名为 EX01.aspx。如果 IIS 安装正确，打开浏览器并输入下列地址
<http://127.0.0.1/cr/ch02/ex01.aspx> 后，则可以看到下列结果：

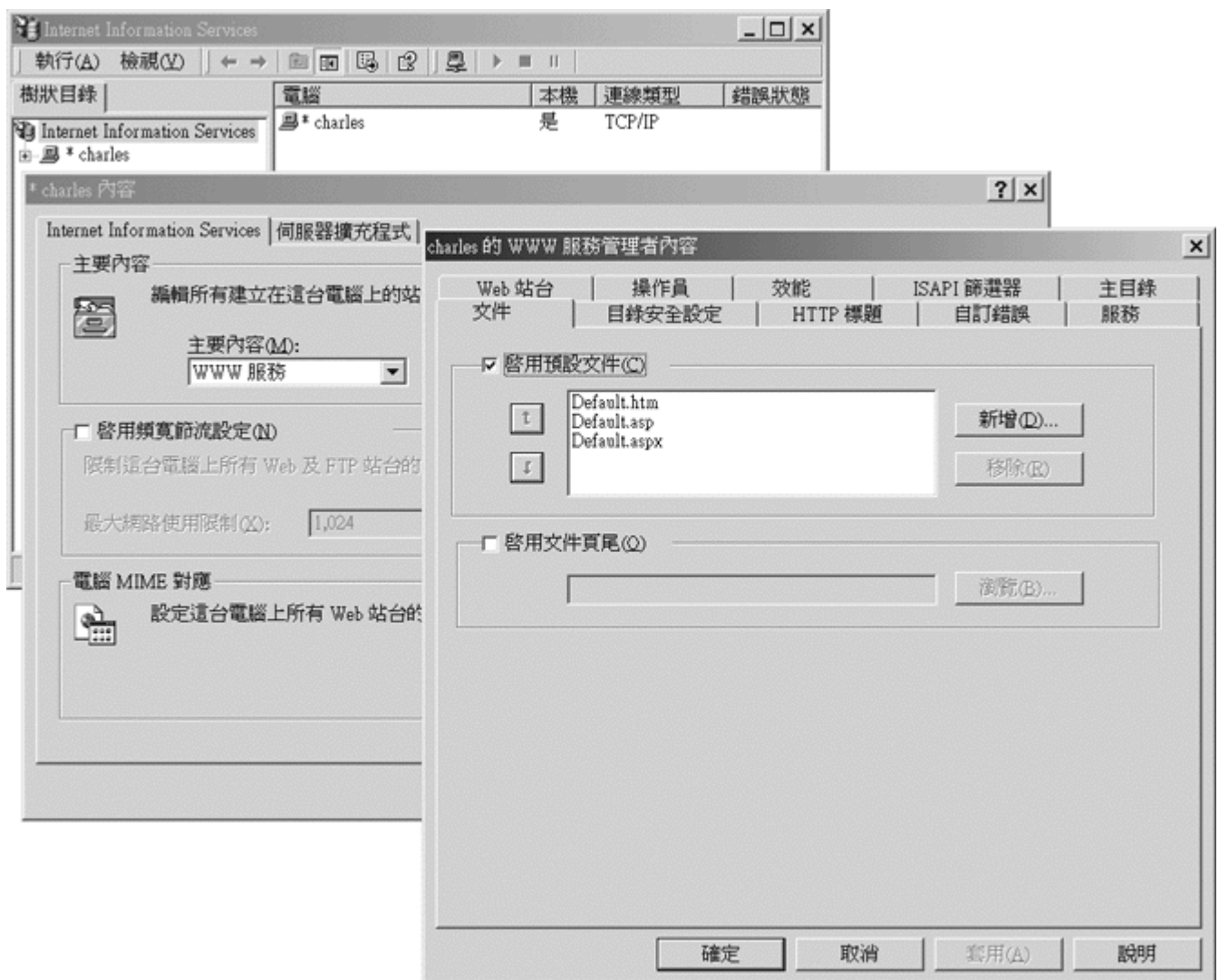


结果这个动态网页执行的结果「Hello world」出现在浏览器中，表示执行成功。

IIS 动作原理

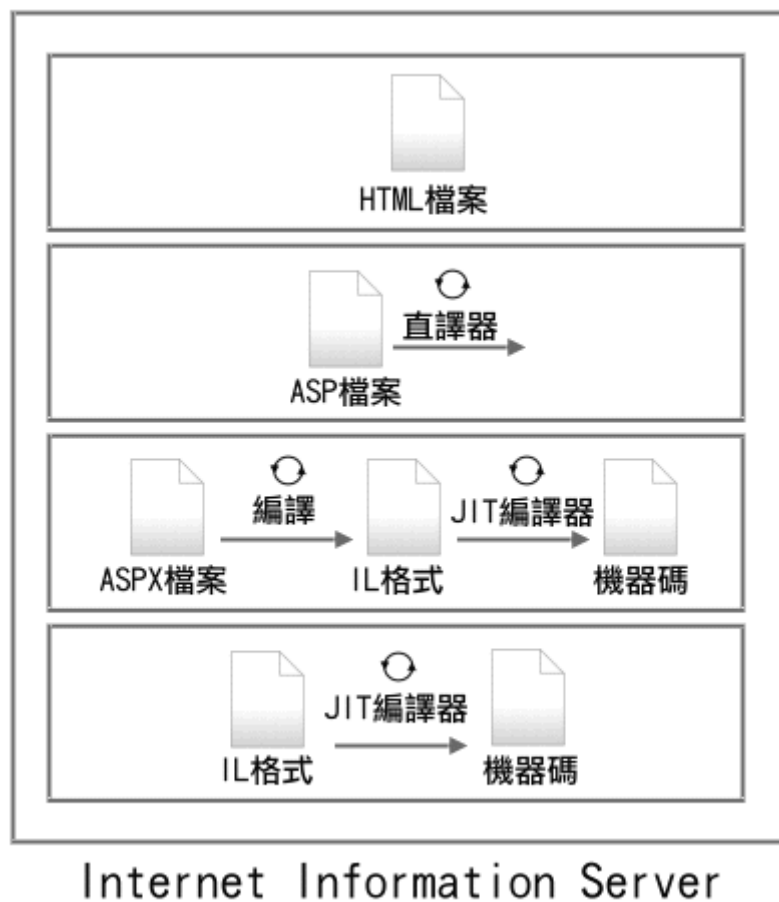
安装好 IIS 后,我们的服务器就有能力提供全球信息网的服务。IIS 在 C 碟中建立一个名为 Inetpub 的数据夹,所有和因特网服务有关的内容,例如 FTP、Mail,以及 WWW 服务都存放在本数据夹内。这里我们的重点在 WWW 服务上,WWW 的服务内容是在 Inetpub 目录内的次数据夹 wwwroot 中,这个数据夹是 WWW 服务的根目录;当使用者连接到我们的服务器后,就是以这个目录作为最外层的数据夹。而在浏览器中输入 127.0.0.1 表示连接到本机端,如果没有指定所要浏览的网页,IIS 会自动去找 Default.htm,如果找不到再找 Default.asp,最后才是 Default.aspx。

这个预设的寻找顺序及档案可以更改，各位可以执行「开始」→「程序集」→「系统管理工具」→「Internet 服务管理员」查看，并可以改变顺序以及新增或是移除预设的网页。



我们要浏览存放在 c:\inetpub\wwwroot\cr\ch02 这个路径中的 EX01.aspx 这个动态网页，不可以直接使用档案总管开启，因为这样做 IIS 不会去执行这个 ASP.NET 的动态网页。要显示动态网

页，必须要透过浏览器向 Web 服务器提出浏览某个网页的要求，IIS 收到我们的要求后再依照网页的扩展名来决定如何执行。

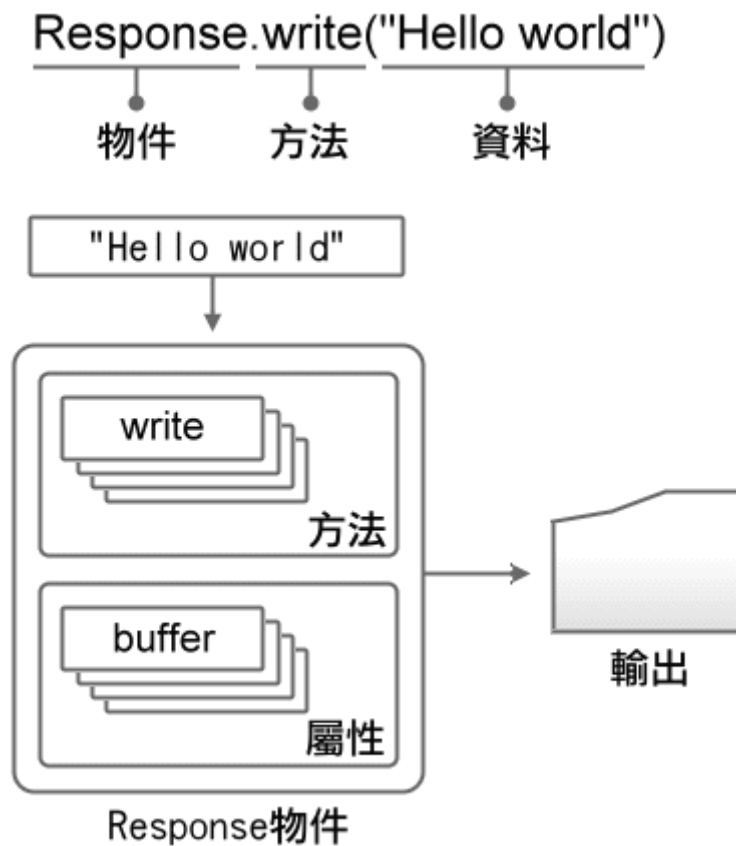


如果扩展名是 **htm** 则表示是 **HTML** 网页，IIS 直接将其内容传给使用者不做处理。若扩展名是 **asp** 则表示是 **ASP** 动态网页，IIS 会启动直译器将其内容实时直译执行，并将执行后的执行结果传给使用者。若扩展名是 **aspx** 则表示是 **ASP.NET** 动态网页，IIS 会视情形启动 **aspx** 网页。如果 **aspx** 网页是第一次被浏览，则启动编译器编译成 **IL** 后，再呼叫 **JIT** 编译器编译成机器码执行；

若 **aspx** 网页不是第一次被浏览且其内容也没有被改变, **IIS** 就直接呼叫 **JIT** 编译器将该网页的 **IL** 档编译成机器码并且执行, 两种情形都是将执行后的执行结果传给使用者浏览。

程序的执行

了解 **IIS** 的执行原理后, 我们来看看程序代码的内容。 **ASP.NET** 的程序代码还是架构在 **HTML** 之上, 而且规定必需要写在 **<%%>** 标注之内。 **ASP.NET** 网页的执行依旧是一行一行执行, **IIS** 在编译 **aspx** 的时候, 针对 **HTML** 标注的部分会一五一十的输出, 对于 **<%%>** 内的内容视为程序代码并执行其中指定的叙述。但是程序的执行是在伺服器端, 使用者是看不到的。除非我们透过 **Response** 对象, 该对象提供 **Write** 方法可将指定的数据输出至使用者的浏览器; 但是所要输出的内容必需是可以被浏览器解读的 **HTML** 等格式之数据, 否则使用者是无法看到执行的结果。刚刚范例中所输出的数据为「**Hello world**」这段文字, **VB.NET** 语言规定对于文字型态的数据, 必需被双引号「**"**」围起来。如果没有使用双引号围起来, **VB.NET** 则视为一个对象或是变量的名称。

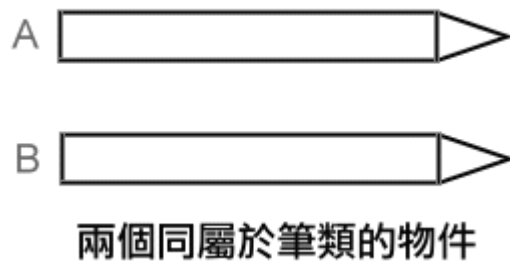


物件（Object）

刚刚我们使用了一个对象—`Response`，我们来看看什么是对象。在现实生活中，我们可以看到的几乎都可以说是对象，例如笔、电话、键盘、屏幕、打印机等，这些对象都有其特殊的功能及特质。我们可以很清楚的分别上述对象是不同的对象，那是因为他们是属于不同的类别（`Class`）。类别定义了对应的特质，对象的特质就是对象的属性、方法及事件。类别好比是蓝图一样，我们在使用的对象的时候，是依照类别的定义来产生对象。我们可以依据类别来产生许多

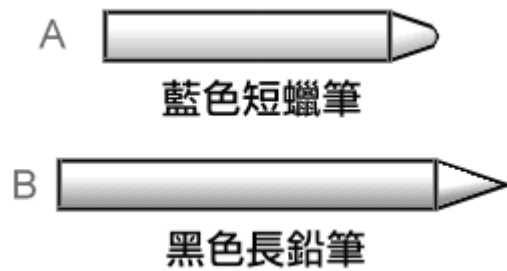
不同的对象，但他们都还是属于同一个类别；所以 **没有类别就没有对象**。我们用笔来举个例子，

假设我们有对象 **A** 及对象 **B** 都是属于笔类，那么你的脑海中大致上会浮出下面的影像：



属性（Property）

不过这两个笔类的对象有点抽象，因为除了知道是笔类外跟本没有其它信息；所以我们为这两个对象加入更具体的特质。例如对象 **A** 笔芯的材质为蜡，颜色为蓝色，而对象 **B** 笔芯材质为铅，颜色为黑色。这样一来我们就知道 **A** 对象是蓝色蜡笔，而 **B** 对象为黑色铅笔。如果再为这两个对象加入长短的特质，假设 **A** 对象为短而 **B** 对象为长，那么这两个笔类的对象就更具体，更可以分辨出是不同的个体。以上如材质、颜色、长短这些特质，我们称为属性（Property）。简单的说， **属性代表一个对象的状态、数据或是设定值**。



了解什么是属性后,我们再来了解程序中如何设定及取回对象的属性。要在程序控制一个对象,首先要为这个对象取一个独一无二的名称,这个用来标示对象的名称我们称为 **ID** 属性。例如刚刚的蓝色短蜡笔我们叫 **A**, 而黑色长铅笔我们叫 **B**。这样一来就算两种笔都是蓝色短蜡笔,我们也可以利用 **ID** 属性来控制我们想要使用的对象。对象的命名规则为: 要以英文字母为开头,中间可包含数字及底线,但是不可以有空格或标点符号。我们在为对象命名的时候,应该要以直觉的方式对物见指定一个有意义的名称; 这样一来我们就不用猜测对象 **A** 和对象 **B** 到底是什么东西。例如我们将上述的 **A** 对象的名称改为 **WaxCrayon**, **B** 对象改成 **Pencil**。要设定对象的属性使用下列的语法:

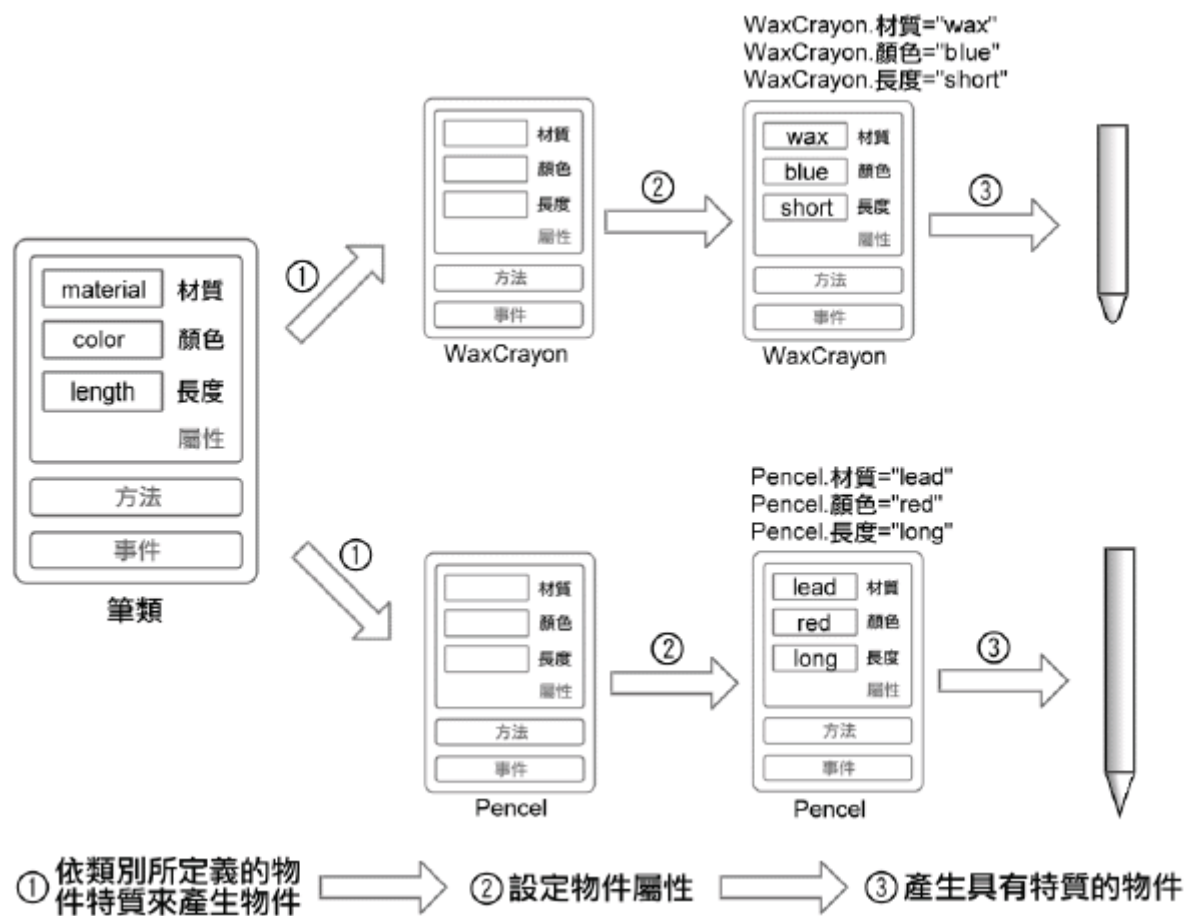
```
物件 ID. 属性=叙述
```

要控制一个对象,必需指定该对象的 **ID** 属性,然后加上连结符号(就是英文句号)并指定属性名称,再利用指定运算符(就是等号)将右边叙述的处理结果存入对象的属性中。「叙述」的内容可以是数学表达式或是直接要设定的值等,在指定运算符右边的叙述 **VB** 会先做处理,处理完后再将处理完的结果存入指定运算符左边的对象属性或变量中,例如:

```
WaxCrayon. 材质 = "wax"
```

WaxCrayon. 颜色 = "blue"

WaxCrayon. 长度 = "short"



方法 (Method)

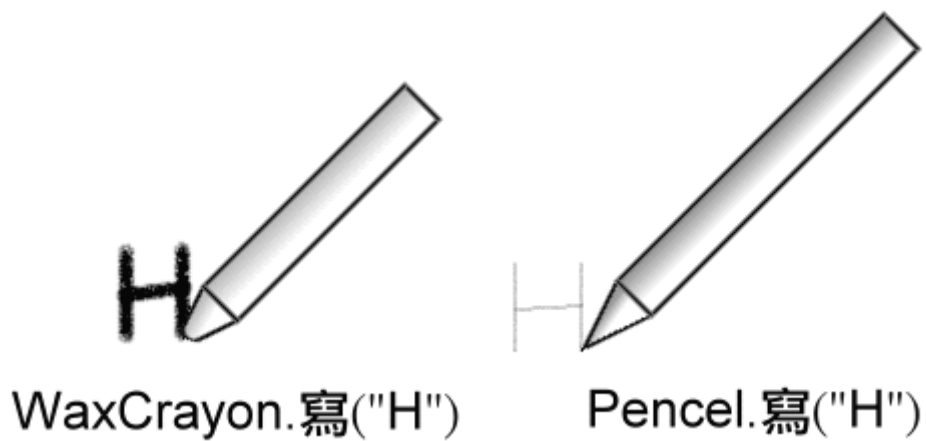
有了属性后我们就可以设定一个对象的状态以及数据，但是这个对象还不会执行任何动作，因为我们还没有为它定义方法；方法就是 [对象可以执行的动作](#)。例如刚刚的笔类别对象，我们为它设计一个叫做「写」的方法，并且规定要使用「写」这个方法的时候，必需要指定所要写的文字为何。当我们使用这个方法的时候，只要呼叫这个方法并将这个方法执行时所需要的数据一并输入即可；这个执行时所需要的数据我们称为「参数」。我们使用如下语法来使用对象的方法：

```
物件 ID. 方法(参数)
```

和指定属性一样，直接指定要使用哪个对象的方法，并且规定要在方法的后面加一个小括号「()」。如果这个方法需要数据才能执行的话，就必须要在括号里面附加这些必要的参数。如果这个数据是文字型态，那必需用双引号「"」引起来，例如：

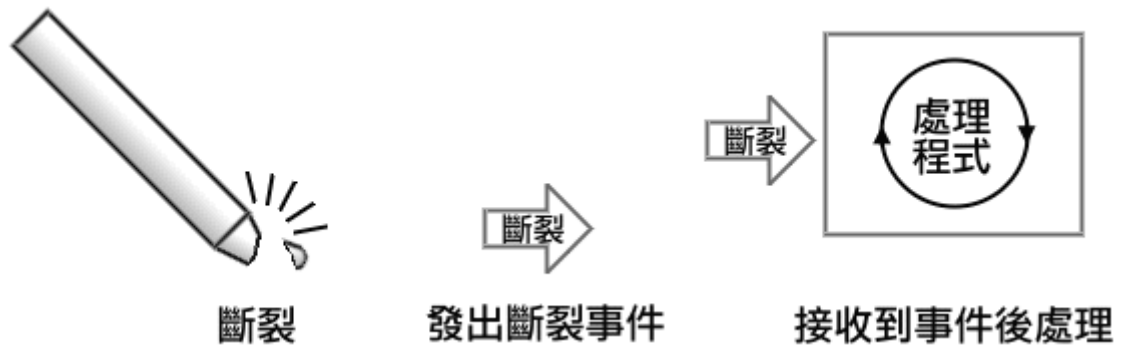
```
Pencil. 写("H")
```

```
WaxCrayon. 写("H")
```



事件 (Event)

有了属性和方法之后，对象的特质可以说是越来越具体了。属性可以让你设定及读取对象的数据及状态，方法可以让对象执行一些动作；但是对象只有属性和方法对我们来说还不够。如果我们希望对象的某些状况有所改变时，可以发出讯息通知我们执行一些因应的动作，这时候就可以靠事件来帮我们完成。举个例子来说，我们为笔类对象加入一个「断裂」事件。如果笔芯断裂的话，我们的对象就自动会发出这个「断裂」事件来通知我们；这时候我们就可以写响应这个事件的程序，例如收到「断裂」事件后，我们就可以撰写削笔的程序来修复笔芯。所以 **事件就是对象所认识的动作**。



了解上述的类别、对象以及属性、方法、事件之后，我们对于对象的概念已经有一个程度的了解。

ASP.NET 提供了许多类别的对象供你使用，我们可以利用这些对象帮你工作。要使用这些对象，首先要了解有哪些类别的对象，这些对象有哪些属性、方法、事件，以及如何使用这些对象所提供的属性、方法、事件；我们会在后面的章节一一介绍。

变数

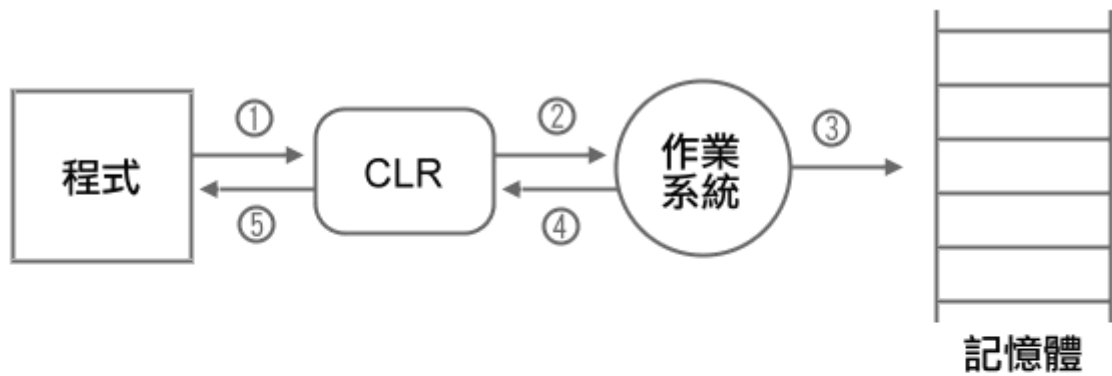
变数的基本概念

在撰写程序的时候，常需要暂时把一些数据存放在一个地方，然后等适当的时机再作处理；这时就需要一个可供我们快速存取数据的地方，这个地方就是计算机的内存。变量可以被用来：

- 作为暂时存对象属性值的地方

- 作为一个计算某个程序或是程序代码块执行次数的计数器
- 作为保留从函式（Function）所传回的值
- 作为存放数据夹名称或是文件名称的地方

当我们使用变量的时候，内存会保留一个空间供程序使用。我们不可能直接以指定内存地址的方式来存取内存内的数据，因为直接去存取内存有相当程度的风险，存取不当甚至会导致数据的损坏或系统当机，所以必需请求操作系统分配记忆空间给我们使用，不过这种低阶的工作交给 CLR（Common Language Runtime）处理即可；因为程序在执行的时候，内存管理是由 CLR 来负责的。这时候我们就可以撰写程序请 CLR 向操作系统要求一个内存空间，至于内存会保留多大的空间，则视我们的需要而定。这个向 CLR 提出内存需求的动作，我们称为变量的宣告。变量的宣告包括了两个部分：变量名称以及数据型态。变量名称是我们利用有意义的代码来取代十六进制的内存地址，让程序开发人员容易使用及管理变量。数据型态是我们要存取的数据型态，因为系统的内存容量是有限制的，所以我们要宣告适当大小的变量以避免浪费内存空间；变量被分配到的内存容量大小，就是由数据型态来决定。



① 程式以宣告的方式宣告變數 ② CLR向作業系統提出記憶體使用要求

③ 作業系統將可用記憶體空間配置出來 ④ 將可用實際位址回應給CLR ⑤ 管理配置到的位址並讓程式使用

变量的数据类型

VB.NET 的变量可以储存文字、数值以及对象等型态的数据。为了要让执行更有效率，VB.NET

提供了几种可以储存特定数据的变量型态，如下表所示：

数据类型 态	储存大小	说明	可储存的数据范围
Char	2Bytes	字符	0 到 65535
String	10Bytes	字符串	0 到大约 2 百万个双位字符

	加上	符 串	(2 乘以字符串长度) (Unicode)
Short	2Bytes	精 简 整 数	-32, 768 到 32, 767
Integer	4Bytes	整 数	-2, 147, 483, 648 到 2, 147, 483, 647
Long	8Bytes	长 整 数	-9, 223, 372, 036, 854, 775, 808 到 9, 223, 372, 036, 854, 775, 807
Single	4Bytes	单 精 浮 点 数	负数部分为 -3. 402823E38 到-1. 401298E-45 正数部分为 1. 401298E-45 到 3. 402823E38
Double	8Bytes	双 精 浮	负数部分为-1. 79769313486231E308 到-4. 94065645841247E-324, 正数部分为 4. 94065645841247E-323 到 1. 79769313486232E308

		点 数	
Boolean	4Bytes	布 尔	True 或 False
Object	4Bytes	物 件	任何型态都可以被对象型态的变量储存
Decimal	12Bytes	数 值	+/-79, 228, 162, 514, 264, 337, 593, 543, 950, 335 以上未带小数，若带小数可存 28 位，为 +/-7. 9228162514264337593543950335 最小为 +/-0. 00000000000000000000 0000001
Date	8Bytes	日 期	公元 1 年 1 月 1 号至 9999 年 12 月 31 日
Byte	1Byte	位	0 到 255

选择变量的数据型态

要使用变量最好先明确的宣告变量名称及数据型态。宣告正确的变量型态不但可以让你的程序更有效率，并且可以减少内存使用的空间。举例来说，VB.NET 处理整数（Integer 或 Short）型别的数据会比处理浮点数（Single 或是 Double）型别的数据来的快；而处理精简整数（Short）又比处理整数（Integer）来的快，所以最好使用精简整数来储存 100 这个数值。除了内存以及执

行效率的考量外，另外我们所要考虑的是数值的精确性。选择 **Decimal** 型态的变量比使用浮点数来的精确，而 **Double** 比 **Single** 来的精确。另外还要考量变量溢位（**Overflow**）的问题，变量可储存的数据被其储存范围所限制。例如精简整数（**Short**）型态的变量可以接受的数据范围为 -32,768 到 32,767，如果超出这个范围则会发生溢位错误。

为变数命名

当我们决定要使用变量的时候，为变量取一个名称是非常重要的。变量的命名不但要考虑容易理解，而且还要考虑变量名称的一惯性，尤其是当你或别人日后可能会再阅读或维护程序的时候。

命名法则

变量的命名和其它对象一样被下列的命名法则所限制，为变量取一个清楚且有意义的名称在大型的项目中特别重要。

命名法则：

- 必需以英文字母为开头。
- 其中不可包含空格，但可包含数字。

- 必需再同一范围内是独一无二的。
- 不可以包含标点或型态宣告字符，但可以包含底线。
- 不可以和 VB.NET 的保留字，或系统对象名称一样。

命名习惯

对变量的命名我们有一个习惯，那就是将该变量数据型态的缩写放在变量名称前面，这样可以
程序代码易于阅读及维护。

数据类型	前缀	数据类型	前缀
Char	chr	Double	dbl
String	str	Boolean	bln
Short	sht	Object	obj
Integer	int	Decimal	dcm
Long	lng	Date	dt
Single	sng	Byte	byt

宣告变量

宣告变量的语法如下：

```
Dim 变量名称 [As 数据类型]
```

我们在 VB.NET 中以 Dim 关键词来宣告变量，并且在 AS 关键词后面指定所要使用的变量型态。

「AS 数据类型」用中括号刮起来，表示这个选项是非必要性的，也就是不给也可以。我们在宣告变量的时候可以不指定变量型态，倘若不指定变量的型态，VB.NET 则预设变量是 Object

型态的变量，例如：

```
Dim strUserName As String
```

以上宣告一个名为 strUserName 的字符串型态变量。

```
Dim shtAge As Short
```

以上宣告一个名为 shtAge 的精简数值型态变量。

```
Dim objPen As Object
```

以上宣告一个名为 objPen 的对象型态变量。

```
Dim objPen
```

以上宣告了一个名为 **objPen** 的对象变量。由于使用者没有指定变量的型态，所以 **VB.NET** 就以预设的对象型态作为变量的型态。

当我们以 **Dim** 关键词宣告变量时，**VB.NET** 就会帮我们吧变量准备好，并将该变量依该变量的型态填入初始值。如果变量型态为数值，**VB.NET** 就会填入 **0** 作为初始值；如果变量型态为字符串，**VB.NET** 就会为我们填入空字符串 `""`；如果变量型态为对象，**VB.NET** 就会为我们填入空值（**Null**）。空值不是零也不是空字符串，而是里面要储存的数据还不知道。

变数的初值化

我们在宣告变量的时候也可以指定一个值作为初始值：

```
Dim strUserName As String="Charles"
```

以上宣告一个名为 **strUserName** 的字符串型态变量，初始值为 **Charles**。

```
Dim shtAge As Short=30
```

以上宣告一个名为 **shtAge** 的精简数值型态变量，初始值为 **30**。

为了方便，**VB.NET** 除了支持初始值的设定外，还可用以让我们在同行中宣告多个变量：

```
Dim shtAge, shtHeight, shtWeight As Short
```

以上宣告了三个变量 **shtAge**、**shtHeight** 与 **shtWeight**，其数据型态都是精简整数。

```
Dim shtAge As Short, strAddress As String
```

以上分别宣告了精简整数型态的变量 **shtAge** 及字符串型态的变量 **strAddress**。

型态宣告字符

VB.NET 为了让我们使用方便，可以利用型态宣告字符来宣告变量，例如：

```
Dim intIncome%  
  
Dim intPayment As Integer
```

以上两个宣告都是宣告为整数型态的变量，这样一来程序的写作就轻松多了。并不是每种数据型态都有型态宣告字符，以下为支持型态宣告字符的数据型态：

数据型态	型态宣告字符
Single	!
Double	#
Integer	%
Long	&
String	\$

使用型态宣告字符有一个要注意的地方，那就是 [同一个宣告的叙述中一般宣告法不可以和型态](#)

[宣告字符混合使用](#)。如下面的范例就是 [错误](#) 的：

```
Dim intIncome%, shtAge As Short
```

另外 VB.NET 的变量不分大小写，所以 `intIncome` 和 `intincome` 是一样的。不过为了容易阅读，我们会将型态简写后的第一个字母大写，例如 `shtAge`。变量名称若由两个以上的字所组成，例如纪录使用者名称的字符串型态变量，习惯上我们会取名为 `strUserName` 或是 `strUser_Name`，这样一来就容易阅读多了。接下来我们来做一个简单的例子，以下的程序代码为计算身高 173 公分的男生体重：

```
<%  
  
Dim shtHeight As Short  
  
Dim shtWeight As Short  
  
shtHeight=173  
  
shtWeight=(shtHeight-80)*0.7  
  
Response.Write(shtWeight)  
  
%>
```

首先我们先宣告了两个精简整数型态的变量，分别为 `shtHeight` 及 `shtWeight`。先将数值 173 存入变量 `shtHeight` 中，然后利用公式将男生的标准体重算出。男生的标准体重公式为：（身高一

80) ×0.7=标准体重，由于身高－80 这个叙述要先执行，所以我们用小括号括起来，VB.NET 会优先处理小括号内的叙述，所以执行的结果显示 65。

运算符的优先级

如果没有上述算式小括号的话，计算出来的结果是 117 而不是正确的 65。这是因为这些我们称为算数运算符的 +－×÷ 等符号有优先级，如下表所示：

优先级	运算符	名称	范例	结果
高	()	括号	A=(10+5)*3	A=45
	^	平方	B=3^2+10	B=19
	* /	乘除	C=12+3*2	C=18
	\	整数除法	D=11\3+5	D=8
	Mod	余数	E=11 Mod 3	E=2
低	+ -	加减法	F=18+4-10	F=12

VB.NET 对于优先级高的先执行运算，若优先级一样，则先执行左边的运算。不过要记忆这些运算符的顺序似乎不是很容易，所以我们可以使用小括号来强制指定运算叙述的优先级。VB.NET 遇到小括号先执行运算，倘若小括号中还有小括号，则最内层的小括号先执行运算。例如 (10×(3

+9)) / (8-2) 这个式子，其运算顺序为 3+9 先执行，得到 12 后再乘以 10，得到 120 后再除以 8 减 2 的结果 6，最后就得到正确的答案 20。

VB.Net 新增的运算符

VB.NET 支持下列的运算符：

运算符	范例	说明
+=	shtA + = 1	等于 shtA = shtA + 1
-=	shtB - = 1	等于 shtB = shtB - 1
*=	shtC * = 2	等于 shtC = shtC z* 2
/=	shtD / = 2	等于 shtD = shtD / 2
\=1	shtE \ = 3	等于 shtE = shtE \ 3
^=	shtF ^ = 2	等于 shtF = shtF ^ 2
&=	strG & = strH	等于 strG = strG & strH

这些运算符可以简化我们程序的输入。

常数

我们知道变量是用来暂时存放资料的地方，里面的数据随时可以改变。而常数是以有意义的名称代替特定的值，并且不允许改变常数内的数据。倘若使用者在宣告常数后要将常数以新值代替，此时会发生错误。假设我们在程序中使用圆周率 **3.14159** 来计算圆周长以及面积，如下范例所示：

```
<html>

<%

Dim shtR As Short=10

Response.Write("Round:")

Response.Write(2*3.14159*shtR)

Response.Write("<br>Area:")

Response.Write(2*3.14159*(shtR^2))

%>

</html>
```

如果我们的程序中有如上面程序中常常使用的数值 **3.14159** 时，我们就可以考虑用常数了。常常输入一长串的数值可能会发生错误，此时我们就可以利用常数。常数可以帮助我们的程序容易阅读，而且不容易发生输入的错误。下列是常数的宣告语法：

```
Const 常数名称 [As 数据类型] = 指定值
```

要使用常数必需以 **Const** 关键词宣告常数，并且可以指定常数的数据类型以及指定的值为何，

例如：

```
Const cnPI As Single=3.14159
```

以上宣告了单精浮点数型态的常数 **cnPI**，其值为圆周率 **3.14159**。其中数据型态的指定可以省略，例如：

```
Const cnPI=3.14159
```

所以可以将刚才的程序代码改成：

```
<html>

<%

Const cnPI=3.14159

Dim shtR As Short=10

Response.Write("Round:")

Response.Write(2*cnPI*shtR)

Response.Write("<br>Area:")

Response.Write(2*cnPI*(shtR^2))

%>

</html>
```

宣告了常数后，编译器在编译程序时会将常数以常数所代表的值代入常数中，并不影响程序执行的效率。

数组

数组是一种可以记录许多数据的一种特殊变量结构，这个结构里面是由相同数据型态的元素所组合而成。当我们想要将一些数据型态相同的数据，利用一个变量来管理的时候，数组是一个非常方便的变量结构。数组的宣告语法如下：

```
Dim 数组名(元素数量) [AS 数据型态]
```

数组的宣告语法和宣告一般变量差不多，一样都使用基本的数据型态（如 **Short**、**Integer**、**String**、**Single** 及 **Object** 等），只不过要在数组名后面加上小括号，并指定数组所要使用元素数量即可。假如我们想要纪录六个人的年龄，若没有使用数组的话就要宣告六个变量，不但不好管理而且容易出错。如果以数组来管理这些信息，程序代码将变的比较精简也比较好管理。我们来观察下面的叙述：

```
Dim shtAge(6) as Short
```

上面的叙述宣告了一个名为 **shtAge** 的精简整数型态数组，这个数组可以记录六个元素。宣告完毕后 **VB.NET** 会将数组先在内存里准备好，如下图所示：

shtAge(6)

index=0	index=1	index=2	index=3	index=4	index=5
0	0	0	0	0	0

这个数组宣告后在内存中被产生，六个元素分别都填入了初始值 **0**，并且将每个元素分配了一个索引值（**index**）。数组的索引值一律从 **0** 开始，若要存取指定元素中的数据，则要利用这个索引值；如下范例所示：

```
shtAge(0) = 20
```

```
shtAge(1) = 21
```

```
shtAge(2) = 22
```

```
shtAge(3) = 23
```

```
shtAge(4) = 24
```

```
shtAge(5) = 25
```

上述程序代码片段将元素 **0** 到 **5** 的内容分别填入 **20** 到 **25**，所以数组内的值就变成下列插图的状态：

shtAge(6)

shtAge(0)	shtAge(1)	shtAge(2)	shtAge(3)	shtAge(4)	shtAge(5)
20	21	22	23	24	25

如果想在宣告的时候顺便指定数组内元素的初始值，可以使用下列语法：

```
Dim 数组名() [AS 数据类型] = {值 1, 值 2, 值 3,...}
```

这里就不需要指定小括号里面的元素数量，所以之前所指定的元素值在宣告的时候即可用下列方式指定：

```
Dim shtAge() As Short = {20, 21, 22, 23, 24, 25}
```

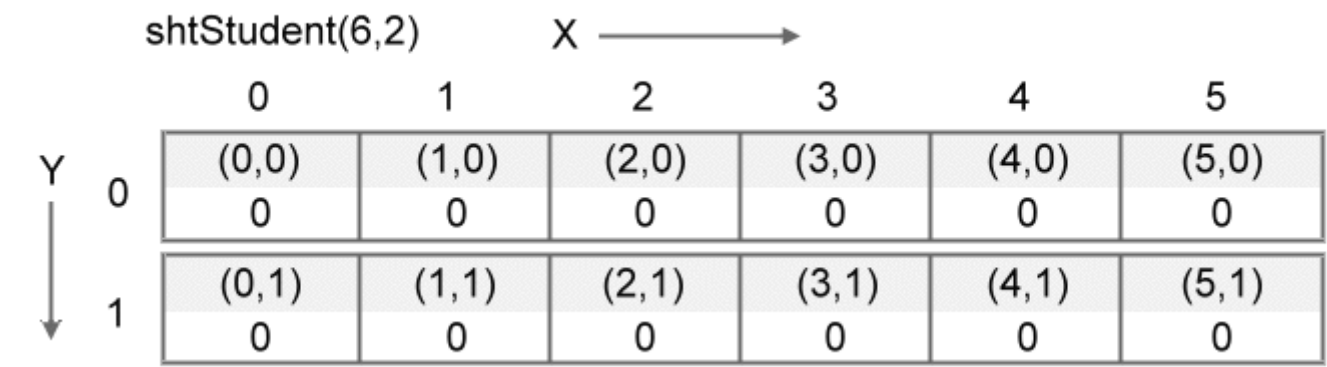
VB.NET 数组元素的数量最多可以宣告为 **264-1** 个元素（即是 **Long** 型态的范围）。数组虽然方便，可是它是需要付出代价的，这个代价就是越大的数组越占内存空间。由于刚才宣告的 **shtAge** 数组可储存 6 个精简整数型态的数据，所占的内存空间为 **6×2Bytes=12Bytes**。故对于数组，使用多少就宣告多少。

多维数组

想要在数组里面记载相关的数据，可以用多维数组。例如刚刚的一维数组记录了六个人的年龄，倘若要多纪录身高，则可以用下列方式宣告一个二维数组：

```
Dim shtStudent(6,2) As Short
```

上面叙述宣告了一个名为 **shtStudent** 精简整数型态的二维数组，这个数组为 **6×2** 的数组，可以记录 **12** 个元素。此时 **VB** 会将数组先在内存里准备好，如下图所示：



我们可以将这个二维数组的第一维想象成 **X** 坐标，第二维想象成 **Y** 坐标。倘若执行下列叙述：

```
shtStudent(0,0) = 20
```

```
shtStudent(1,0) = 21
```

```
shtStudent(0,1) = 170
```

```
shtStudent(1, 1) = 171
```

则 `shtStudent` 这个二维数组的内容变成：

shtStudent(6,2)		0	1	2	3	4	5
0	(0,0)	20	21	0	0	0	0
	(0,1)	170	171	0	0	0	0
1	(1,0)	20	21	0	0	0	0
	(1,1)	170	171	0	0	0	0

如果有需要，我们甚至可以宣告三维数组、四维数组等，**VB.NET** 最高支持 **64** 维的数组。不过

数组的维度越多就越复杂，也就越不容易管理及维护；所以适当的选择数组的元素数目以及维度

可以提升程序写作的效率。当你一开始无法精确的确定数组大小时，在宣告数组的时候空出元素

的数目即可；如下叙述所示：

```
Dim shtStudent() As Short
```

然后再程序代码中利用 **ReDim** 叙述来重新分配分配元素实际的数量：

```
ReDim shtStudent(6)
```

ReDim 叙述也可以用来重新配置数组的大小。假设数组原来在宣告的时候为 6 个元素，如有需要可以利用 **ReDim** 叙述将数组重新扩张或缩小，但数组内的所有数据会消失变回初始值。若希望在改变数组大小的时候可以保留元素的内容，可以在 **ReDim** 后面加上 **Preserve** 关键词。例如：

```
Dim shtStudent() As Short = {20, 21, 22, 23, 24, 25}

ReDim Preserve shtStudent(7)
```

以上范例是将 **shtStudent** 这个数组扩张 1 个元素，并保留其值。当然，针对多维数组也可改变其元素的数目，**但是数组的维度不可被改变，而且只有最后一维的元素数量可以被改变**，若是你改变了数组的维度或是其它维的元素数目，则会导致错误！

对象型态的数组

数组里面的数据型态，必需要和我们在宣告数组时所指定的数据型态一样。不过如果这个数组的数据型态为对象（**Object**）型态，则这个数组可以记载各种不同的数据型态，因为对象型态本来就可以容纳各种型态的数据，例如：

```
Dim objStudent(4) As Object

objStudent (0) = "Charles Lin"

objStudent (1) = "100 Civil Blvd"

objStudent (2) = 29

objStudent (3) = #10/03/1973#
```

这个范例宣告了一个名为 **objStudent** 并可纪录 4 个元素的对象型态数组，分别以姓名、住址、年龄以及生日填入数组中。请注意，**VB.Net** 对于日期型态的数据规定必须使用「#」号括起来。所以这个数组我们总共使用了三种数据型态，分别为字符串、数值以及日期。如果我们要将刚刚只能记一个数据的 **objStudent** 数组改成可以记录 6 个人，那么我们可以改成以下的二维数组：

```
Dim objStudent(6,4) As Object
```

所以要填入第一个学生的资料则变为：

```
objStudent (0,0) = "Charles Lin"  
objStudent (0,1) = "100 Civil Blvd"  
objStudent (0,2) = 29  
objStudent (0,3) = #10/03/1973#
```

而要填入第二个学生的资料则变为：

```
objStudent (1,0) = "Kevin Chen"  
objStudent (1,1) = "377 Jin-An Rd"  
objStudent (1,2) = 27  
objStudent (1,3) = #02/24/1975#
```

其它学生的资料依此类推。

数据的输入

本章前面已经介绍过将数据输出至浏览器的 **Response** 对象，接下来我们来看看如何取得使用者所输入的数据。欲取得使用者输入的数据，可以利用 **Request** 对象。**Request** 对象的语法如下：

```
变量 = Request("参数名称")
```

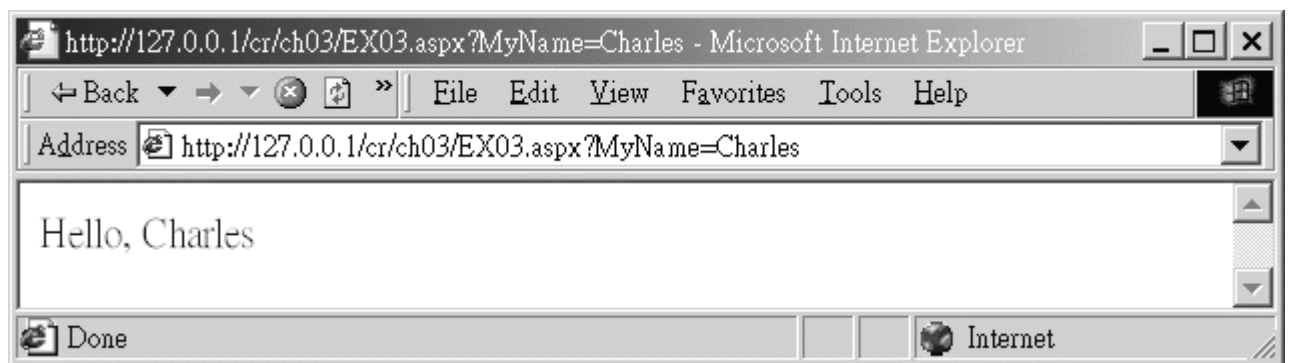
由于 **Request** 对象会传回使用者输入的数据，所以必需要使用变量来接收。我们来看看下面这个例子：

```
<%  
  
Dim strName As String  
  
strName=Request("MyName")  
  
Response.Write("Hello, ")  
  
Response.Write(strName)  
  
%>
```

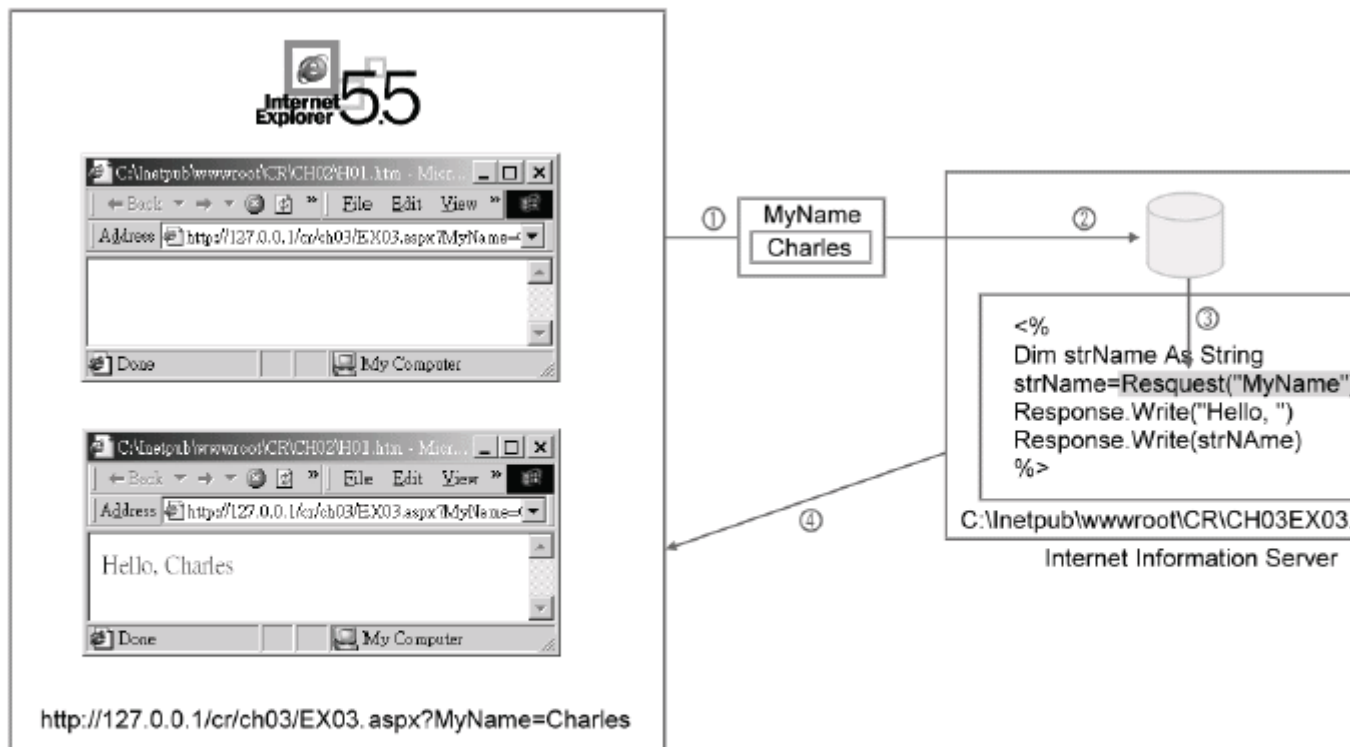
客户端要传送数据给网页服务器，只要在网址后面加上问号，并将数据名称以及指定的值填入即可，这个输入的数据我们称为「参数」；如下叙述所示：

```
http://127.0.0.1/cr/ch03/EX03.aspx?MyName=Charles
```

以上输入的参数名称为 **MyName**，参数内容为 **Charles**（注意，参数内容虽然是字符串，但是不用加双引号）。



我们利用浏览器向 IIS 要求执行 EX03.aspx 这个网页的数据时，**MyName** 这个参数名称会和参数内容 **Charles** 一并传送过去。我们来了解参数上传的实际情形：



使用者在要求浏览 EX03.aspx 这个网页时，会一并将参数传递至网站伺服器。此时伺服器将此参数先暂时存放于暂存区中，等 aspx 程序透过 Request 对象将指定的参数从暂存区取回。我们也可以一次传递多个参数，例如下列范例码接受两个参数：

```
<%  
  
Dim strName, strCity As String  
  
strName=Request("MyName")  
  
strCity=Request("MyCity")
```

```
Response.Write("Hello, " & strName & ", You live in " & strCity)  
%>
```

在呼叫这个网页时只要以「&」符号来连结第二个参数即可，如下所示：

```
http://127.0.0.1/cr/ch03/EX03.aspx?MyName=Charles&MyCity=Taipei
```

另外我们来看下列这段程序叙述：

```
Response.Write("Hello, " & strName & ", You live in " & strCity)
```

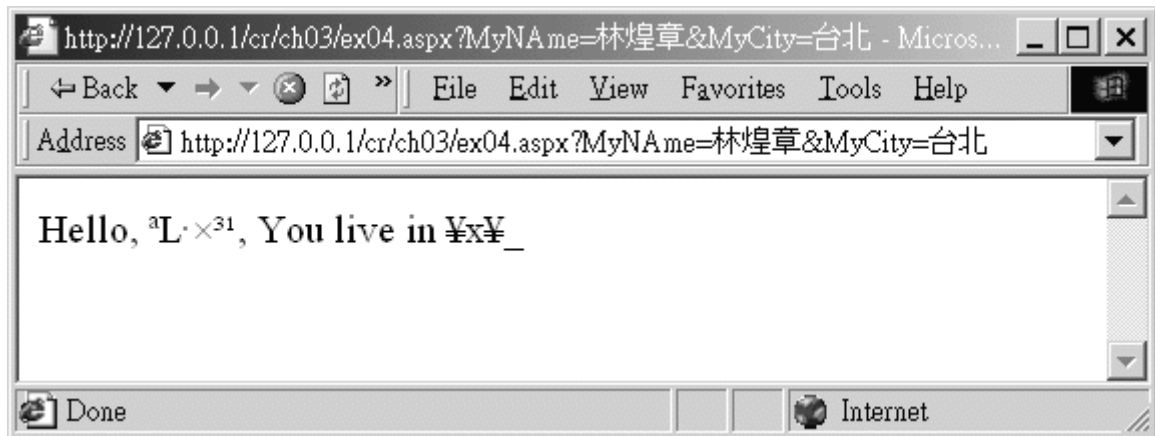
这里的「&」这个连结运算符作为连结字符串用，所以在一行叙述中可以输出这一串文字。

设定 IIS 的编码及译码语系

刚刚的 **aspx** 网页我们在输入参数值的时候都是输入英文，如果我们将参数值输入中文：

```
http://127.0.0.1/cr/ch03/EX03.aspx?MyName=林煌章&MyCity=台北
```

结果却变成如下的乱码：



这是因为我们要设定 IIS 服务器对于数据编码的语系，这个设定档为 **config.web**。我们利用

WordPad 来撰写这个档案，其内容如下：

```
<configuration>

<globalization requestencoding="big5" responseencoding="big5"/>

</configuration>
```

设定档的内容有分大小写，在编辑时要特别注意。编辑好之后，再将这个档案以纯文字文件的方式

储存于 **c:\inetpub\wwwroot** 这个路径中，则在这个路径之下的数据夹都会受影响；这样一来我

们的 **aspx** 网页就可以输入及输出中文了。

数据类型别的转换

型别转换函式

VB.NET 对于数据的处理是强型别，表示两种数据类型一样才可以执行运算。假设我们有 **strA** 字符串型态以及 **shtB** 精简整数型态这两个变量，并指定 **strA="100"**而 **shtB=10**。如果想要将 **shtB** 的值再加上 **strA** 的值，若直接执行 **shtB=shtB+strA** 这个叙述将会导致错误。这是因为 **shtB** 以及 **strA** 不是相同的数据型态，虽然 **strA** 的内容为 **10**，但是它是被双引号所括起来；被双引号括起来一律视为字符串。若想要执行加法运算，则必须将 **strA** 利用型态转换函式 **CShort()** 转换成数值型态后再执行加法计算。故上述式子要改成 **shtB=shtB+CShort(strA)**，才是正确的答案 **110**。

因为 VB.NET 是强型别，所以两个不同型态的数据要做处理，必需先转换成相同的数据型态才可以，VB.NET 已经不支持旧版 VB 的自动型别转换。VB.NET 提供了许多型态转换的函式：

函式	转换型态
Cbool	Boolean
Cbyte	Byte
Cchar	Char
CDate	Date
CDbl	Double
CDec	Decimal
CInt	Integer
CLng	Long
CObj	Object

CShort	Short
CSng	Single
CStr	String

这些函式很好记，**C** 是 **Convert** 转换的意思，然后除了 **Short** 之外都是加上要转换的型态简写。

下列例子是输入英呎及英吋，并将之转换成公制单位后输出：

```
<html>

<%

Dim sngFeet, sngInches, sngCentimeters As Single

sngFeet=CSng(Request("Feet"))

sngInches=CSng(Request("Inches"))

sngCentimeters=((sngFeet*12)+sngInches)*2.54

Response.Write(CStr(sngFeet) & " 英呎 " & CStr(sngInches) & " 英吋等于
")

Response.Write(CStr(sngCentimeters) & " 公分")

%>

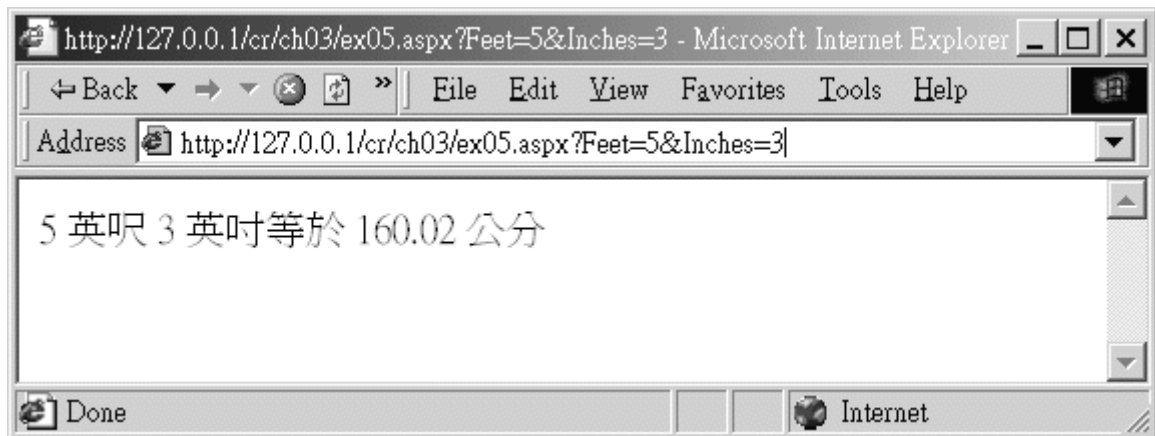
</html>
```

英制单位转公制单位为 1 英尺等于 12 英寸，1 英寸等于 2.54 公分。我们先将输入的数据都转成

Single 后再做运算，最后将输入的数据及运算结果转成字符串后，再利用字符串连结运算符 &

组合起来再做输出。以下是输入 5 呎 3 吋的执行结果：

```
http://127.0.0.1/cr/ch03/ex05.aspx?Feet=5&Inches=3
```



使用 To 进行转换

VB.NET 还有一个非常好用的数据转换用法，那就是在变量或者是叙述后面可以直接使用 To 方

法。下面为语法：

```
变量=变量.To 型态
```

或

变量=(叙述).To 型态

这是因为在 .NET 中 **所有的东西都是对象**，变量是对象、常数是对象，叙述也是对象；这些对象本身就提供了型别转换的方法供我们使用。例如下列范例将数值型态的变量转换成字符串型态后输出：

```
Dim shtNum As Short  
  
shtNum=12345  
  
Response.Write("shtNum 变数中的值是：" + shtNum.ToString())
```

我们将 **shtNum** 变量用 **ToString** 方法转成字符串后，就可以用字符串结合运算符「+」和字符串结合了（也可以使用「&」）。另外叙述也可以执行转换的方法，如下范例所示：

```
Response.Write("this is a string.".ToUpper()) ' 直接将字符串转大写  
  
Response.Write("计算结果为：" + (shtNum+123).ToString() ) ' 将叙述结果  
转字符串
```

下表是常用的 **To** 型别转换方法：

转换型态	使用方法
转字符串	ToString()
数值转字符	ToChar()

字符串转小写	ToLower()
字符串转大写	ToUpper()
转精简整数	ToInt16()
转整数	ToInt32()
转长整数	ToInt64()
转数值	ToDecimal()
转日期	DateTime()
转单精数	ToSingle()
转双精数	ToDouble()
转布尔	ToBoolean()
日期转精简日期	ToShortDateString()
时间转精简时间	ToShortTimeString()

程序的续行及批注

续行字符

我们在撰写程序时，可能遇到一行叙述太长超过一个画面的情形。这种冗长的程序代码在我们阅读程序的时候需要再卷动滚动条，并不是很方便。**VB.NET** 使用底线「_」作为续行字符，此时我们可以利用续行字符将太长的程序代码拆成比较容易阅读的方式，例如原来的程序代码：

```
Response.Write(CStr(sngFeet) & " 英尺 " & CStr(sngInches) & " 英寸等于  
" & CStr(sngCentimeters) & " 公分")
```

我们可以将它断成：

```
Response.Write(CStr(sngFeet) & " 英尺 " & _  
CStr(sngInches) & " 英寸等于 " & _  
CStr(sngCentimeters) & " 公分")
```

我们在断行时，必需在完整的叙述前面或后面空一个空格后再加上续行字符，然后再将程序叙述断到下一行。倘若所要断的是一串长字符串，例如下列叙述：

```
Response.Write("我们在断行时，必需在完整的叙述前面或后面空一个空格。")
```

则这一串长字符串应先分解成两个字符串后，再用连结运算符（&）连结后再断行：

```
Response.Write("我们在断行时，必需在完整的" & _  
"叙述前面或后面空一个空格。")
```

批注

批注是给程序设计师看的程序代码说明，**VB.NET** 遇到批注会予以忽略跳过，在批注符号后面的数据全部视为批注。批注并不会影响程序的执行，在程序中加入批注可以让别人比较容易阅读，也可以帮助我们日后在阅读自己的程序时更容易理解。**VB.NET** 有两种标注的方式，一是使用单引号「'」，另外一种则是使用 **REM** 关键词。我们可以在叙述后面使用单引号做批注，例如：

```
sngCentimeters=((sngFeet*12)+sngInches)*2.54 '将英制单位转成公制单位
```

或是在新的一行标注：

```
sngCentimeters=((sngFeet*12)+sngInches)*2.54  
  
'将英制单位转成公制单位
```

以 **REM** 作为批注的话则一定要在新的一行做批注：

```
sngCentimeters=((sngFeet*12)+sngInches)*2.54  
  
REM 以上为将英制单位转成公制单位
```

另外特别注意一点，就是在续行字符后面不可以加批注。因为在批注后面的叙述 **VB.NET** 会忽略，所以下面的批注会发生错误：

```
Response.Write(CStr(sngFeet) & " 英尺 " & _ '在这里批注是错的
```



```
CStr(sngInches) & " 英吋等于 " & _ '或是在这里批注皆错误  
CStr(sngCentimeters) & " 公分")
```

对于使用续行字符的批注，要再该叙述的最后一行批注才正确：

```
Response.Write(CStr(sngFeet) & " 英尺 " & _  
CStr(sngInches) & " 英吋等于 " & _  
CStr(sngCentimeters) & " 公分") '在这里批注就对了
```

程序

了解程序的种类

在 VB.NET 中有三种程序（**Procedure**），分别是一般程序、事件程序以及属性程序。属性程序最主要是在建立对象类别时定义对象的属性，超过我们所要讨论的范围，这里的重点放在一般程序以及事件程序。一般程序是指 **Sub** 或 **Function**，可以帮助我们将复杂的程序做成许多容易管理的单元，至于 **Sub** 和 **Function** 的差别我们后面再来了解。而事件程序会自动触发，用来响应系统或使用者所执行的动作。例如当网页被加载时会触发 **Init** 这个事件，并会自动执行 **Page_Init()** 这个事件程序。

一般程序

一般程序要被其它程序呼叫才能执行，它可以帮助我们复杂的程序代码分成容易管理的单元，并且可被不同的程序所叫用。一般程序有两种，分别为 **Sub** 以及 **Function**。

Sub

以 **Sub** 方式写成的程序没有传回值，也就是不会传回执行的结果。我们先看看 **Sub** 的语法：

```
Sub 程序名称(参数 1 As 型态, 参数 2 As 型态,...)

    程序代码...

End Sub
```

另外，**ASP.NET** 规定所有的程序必需另外放在 **script** 标注中，如下所示：

```
<Script Language="VB" Runat="server">

Sub 程序一()

    程序代码...

End Sub

Sub 程序二()

    程序代码...
```

```
End Sub

</Script>
```

被 **Script** 标注所围起来的部分表示为程序，**不管是一般程序或是事件程序都必需被 Script 标注所围起来**，程序只有被呼叫或是发生一事件的时候才会被执行。**Script** 标注的 **Language** 属性可以设定程序要使用的语言，这里我们所使用的语言为 **VB.NET**，所以设定值为 **VB**；**Runat** 属性值为 **Server**，表示这些程序会在 **Server** 端执行。我们要执行程序的时候，只要 **直接输入程序的名称并且要加上小括号** 即可；我们称使用程序的动作为「呼叫」。我们在呼叫程序时，主程序的执行会暂时停止，并跳到程序中继续执行，等遇到 **End Sub** 时表示程序执行完毕，程序的执行便跳回到刚刚呼叫的地方继续执行下一个叙述。例如：

```
<%

Response.Write("过程调用前<br>")

SayHello()

Response.Write("过程调用后")

%>

<Script Language="VB" Runat="server">

Sub SayHello()

    Response.Write("Hello<br>")

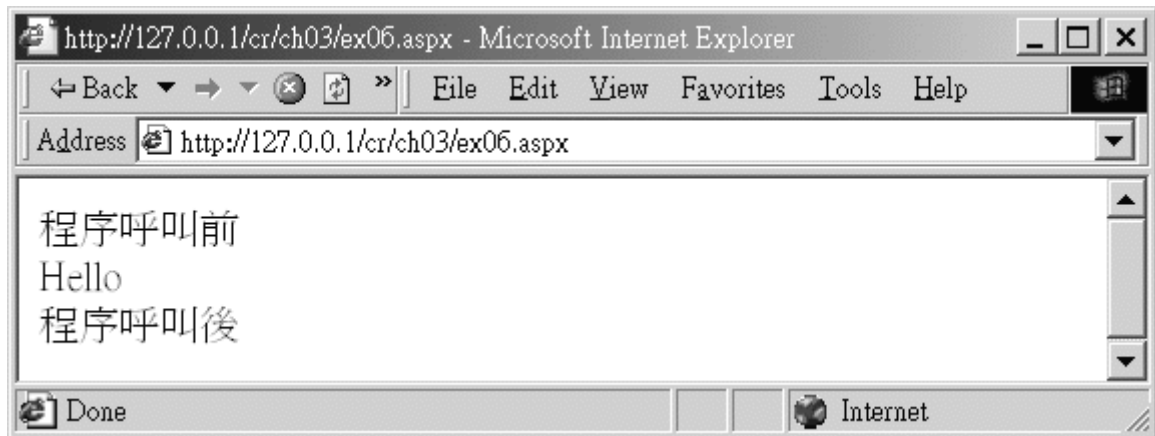
End Sub

</Script>
```



程序可以重复再利用，我们可以多次呼叫使用同一个程序，这样不但好管理而且比较简洁。上述

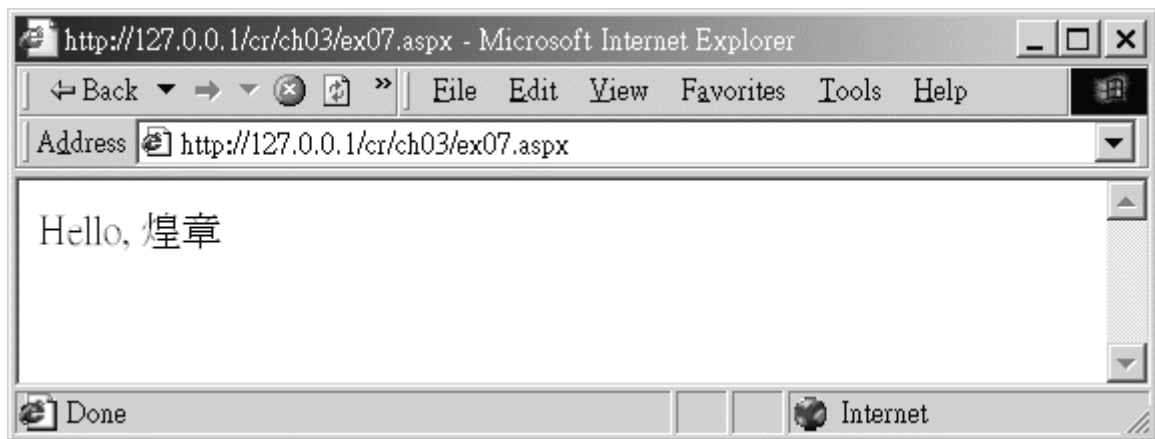
程序执行的结果为：



参数的传递


若有要一并传送的参数，则可以在小括号内输入。我们可以将变量的内容或是直接将数据传入程序内做处理，若参数为一个以上，则使用逗号来做分隔。

```
<%  
  
SayHello("煌章")  
  
%>  
  
<Script Language="VB" Runat="server">  
  
Sub SayHello(strName As String)  
  
    Response.Write("Hello, " & strName)  
  
End Sub  
  
</Script>
```



由于我们的程序要接收参数，所以就必需在宣告程序的时候，在小括号内宣告一个变量来接受参数。所以在主程序执行 **SayHello("煌章")**时，即将参数"煌章"传递给程序。这时候我们在宣告程序时，要在小括号内宣告变量来接收。在小括号内宣告变量不需要加 **Dim** 关键词，直接指定变量名称及数据型态即可。

```
<%  
SayHello("煌章")  
%>  
  
<Script Language="VB" Runat="Server">  
Sub SayHello(strName As String)  
    Response.Write("Hello, " & strName)  
End Sub  
</Script>
```



接下来我们来撰写传递两个参数的程序。参数若是有两个以上，在宣告的时候必需用逗号来做区隔，而且在传递参数时参数的顺序也要对应正确。下面的例子我们将刚刚的英制单位转公制单位的程序改成以呼叫程序的方式执行：

```
<html>  
  
<%  
  
Dim sngFeet, sngInches, sngCentimeters As Single  
  
sngFeet=CSng(Request("Feet"))  
  
sngInches=CSng(Request("Inches"))  
  
MyConvert(sngFeet, sngInches)  
  
%>  
  
<Script Language="VB" Runat="Server">  
  
Sub MyConvert(sngF as Single, sngI as Single)
```

```

Dim sngCM As Single

sngCM=((sngF*12)+sngI)*2.54

Response.Write(CStr(sngF) & " 英尺 " & CStr(sngI) & " 英寸等于 ")

Response.Write(CStr(sngCM) & " 公分")

End Sub

</Script>

</html>

```

```

<%
Dim sngFeet, sngInches, sngCentimeters As Single
sngFeet=CSng(Request("Feet"))
sngInches=CSng(Request("Inches"))
MyConvert(sngFeet,sngInches)
%>

<Script Language="VB" Runat="Server">
Sub MyConvert(sngF as Single, sngI As Single)
    Dim sngCM As Single
    sngCM=((sngF*12)+sngI)*2.54
    Response.Write(CStr(sngF) & " 英尺 " & CStr(sngI) & " 英寸等於 ")
    Response.Write(CStr(sngCM) & " 公分")
End Sub
</Script>

```

Function 程序

以 **Function** 的方式写成的程序有传回值，也就是会传回执行的结果，所以在呼叫 **Function** 的时候必需用变量或对象的属性来接收。**Function** 的语法：

```
Function 程序名称(参数 1 As 型态, 参数 2 As 型态,...)

    程序代码...

    Return 传回值

End Function
```

Function 程序也是一样必需被 **Script** 标注所括起来。以 **Function** 关键词所宣告的程序，表示是有传回值的 **Function** 程序。**Function** 程序一样可以接收参数，我们将处理完的结果利用 **Return** 关键词传回。下面的例子是将摄氏温度转换成华氏温度，我们知道其转换的公式为

$F=(C\times(9/5))+32$ ：

```
<html>

<%

dim sngC, sngF As Single

sngC=CSng(Request("C"))

sngF=CtoF(sngC)

Response.Write("您输入的摄氏温度为:" & CStr(sngC) & "<br>")

Response.Write("转换后的华氏温度为:" & CStr(sngF) & "<br>")

%>
```

```
<Script Language="VB" Runat="Server">
```

```
Function CtoF(sngTC As Single)
```

```
    Dim sngTempF As Single
```

```
    sngTempF=(sngTC*9/5)+32
```

```
    Return sngTempF
```

```
End Function
```

```
</Script>
```

```
</html>
```

```

sngF=CtoF(sngC)
③

<Script Language="VB" Runat="Server">
Function CtoF(sngC As Single)
Dim sngTempF As Single
SngTemp=(sngC*9/5)+32
Return sngTempF ②
End Function
</Script>

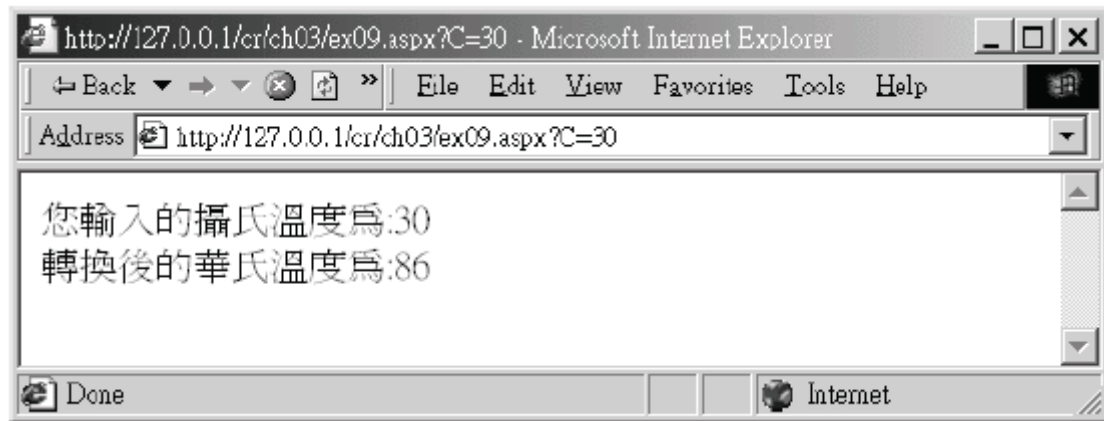
```

① 在呼叫Function時將所要處理的資料一併傳入Function內處理

② 將處理完的資料利用Return關鍵字傳回原來呼叫的地方

③ Function有傳回值，故需要用變數的屬性來接收

Function 一样可以接受参数，可以让我们在呼叫 Function 的时候顺便将要处理的数据或必需的参数一并传递给 Function 处理，等 Function 将数据处理完毕后，则利用 Return 关键词将结果传回至原来呼叫 Function 的地方。由于 Function 有传回值，所以我们在呼叫 Function 时必需用变量或对象的属性来接收其执行结果。另外要特别注意一点，那就是接收传回值的变量其数据型态要和传回值的数据型态相符合，如果不符合则可以使用转换函式进行数据型态的转换。以下为在浏览器输入 <http://127.0.0.1/cr/ch03/ex09.aspx?C=30> 的执行结果：



传值及传址

参数的传递有两种情形：一种是传递变量的值，另一种是传递变量的地址。上述两种情形称为传值（By Value）或传址（By Reference）。传递参数时要指定以传值或传址的方式来传递，只要在宣告程序时要在要接收参数的变量前加上关键词 **ByVal**（传值）或是 **ByRef**（传址）即可。VB.NET 的程序对于参数的传递预设是以传值的方式执行，所以如果要以传值的方式传递参数，则 **ByVal** 关键词可以省略。什么是传递值及传递地址呢？我们来看下列的例子：

```
<html>

<%

Dim shtA As Short

shtA=100

MyProcA(shtA)

Response.Write("传值后的 shtA 变数值:" & CStr(shtA) & "<br>")
```

```
MyProcB(shtA)
```

```
Response.Write("传址后的 shtA 变数值:" & CStr(shtA))
```

```
%>
```

```
<Script Language="VB" Runat="Server">
```

```
Sub MyProcA(ByVal shtB As Short)
```

```
    shtB=10
```

```
End Sub
```

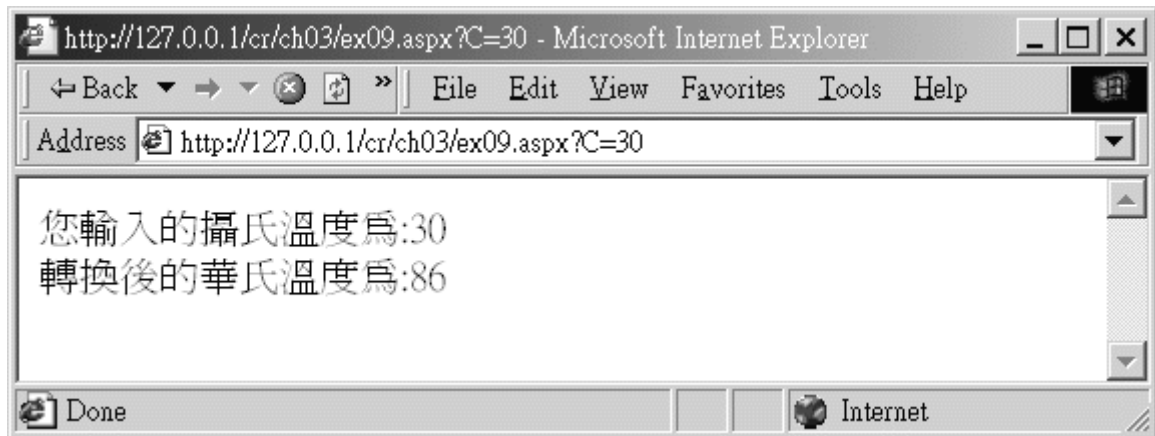
```
Sub MyProcB(ByRef shtB As Short)
```

```
    shtB=10
```

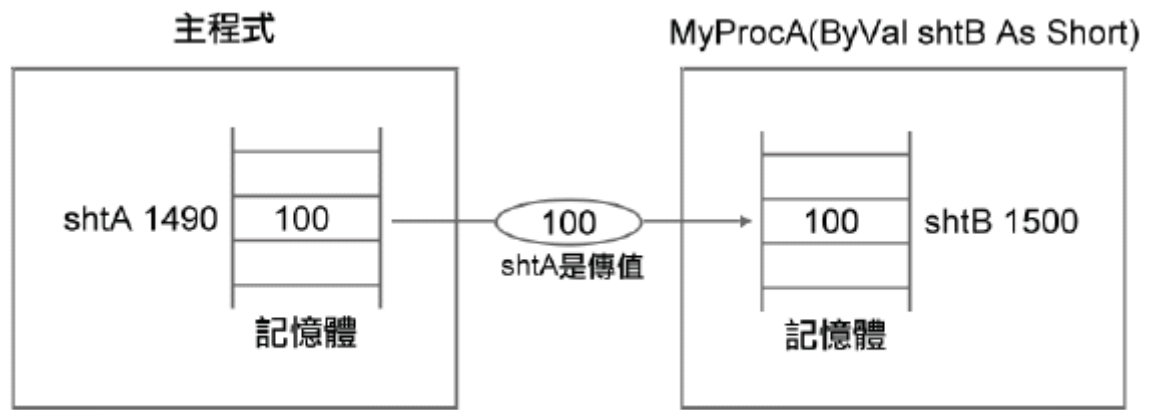
```
End Sub
```

```
</Script>
```

```
</html>
```

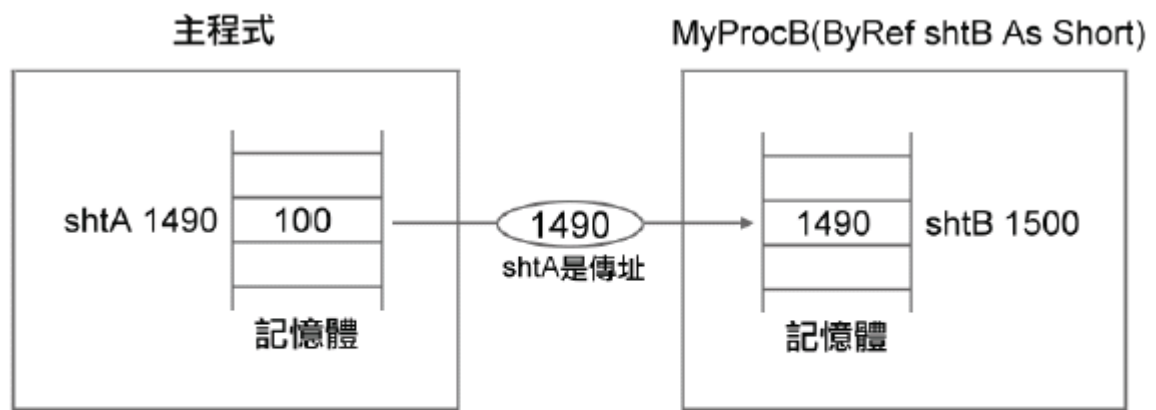


上述程序总共有两个 Sub 程序，分别为 MyProcA 以及 MyProcB。这两个程序都宣告了一个精简整数型态的变量 shtB，分别以传值以及传址的方式来接收参数，然后将 shtB 的值填入整数 10。而在主程序中有一个精简整数型态的变量 shtA，我们将 100 填入变量 shtA 之后分别呼叫 MyProcA 以及 MyProcB，并将变量 shtA 当作参数传进程序中。结果以传值的方式传递参数的 MyProcA 接收到的是 shtA 的值 100，并将接收到的 100 存入变数 shtB 中。所以我们将变量 shtB 变量内的值 100 改成 10，并不会影响传递进来的变量 shtA，shtA 的值还是 100。



shtA以及shtB為兩塊獨立的記憶體位址，不會相互干擾

执行完 MyProcA 后接下来我们呼叫 MyProcB，并以传址的方式将参数 shtA 传入程序中。此时我们所传递的不是值 100，而是 shtA 所在的内存位置。所以 shtB 所接收的信息为一个内存地址，任何程序叙述要存取变量 shtB 的值时，会读到 shtB 内所存放的内存地址，此时程序会将变量 shtB 所指到的内存地址进行存取。因为变量 shtA 以及变量 shtB 所参考到的内存地址是同一块，所以当我们改变 shtB 的值时，就是改变 shtA 的内容，变量 shtA 当然会受影响。故原来变量 shtA 的值为 100，后来变数 shtB 被填入 10 后，变量 shtA 的内容也改变了。



shtB的內容為記憶體位址，表示裡面的資料是參考記憶體位址1490內的內容，所以shtA以及shtB所參照到的記憶體是同一塊，故改變shtB的值將會影響shtA的值

程序的负载

VB.NET 也支持程序的负载。负载就是可以宣告许多名称一样的程序，但是接收不同的参数，并可以视使用者的使用情形决定由哪个程序动作。我们在呼叫程序时，程序会自动依不同的执行条件来呼叫不同的程序。支持负载的程序其宣告语法如下所示：

```
Overloads Sub|Function 程序名称(参数 1 As 型态, 参数 2 As 型态,...)
    叙述...
End Sub|Function
```

下列范例中我们负载了四个程序。这四个程序会依照使用者呼叫程序时，所输入的参数而自动执行相对应的程序。


```

<Html>

<%

Dim strRev As String

Test("字符串一")          ' 输入一个字符串数据，执行第一个对应的程序

Test(12345)                ' 输入一个数值数据，执行第二个对应的程序

Test(12345, 12345)         ' 输入两个数值数据，执行第三个对应的程序

Test("字符串一","字符串二") ' 输入两个字符串数据，执行第四个对应的
程序

strRev=Test("字符串一","字符串二") ' 输入两个字符串数据，并用指定运
算子将执行

        ' 结果存入指定变量，执行第四个对应的程序

%>

<Script Language="VB" Runat="Server">

' 输入一个字符串时启动下列程序

Overloads Sub Test(ByVal strA As String)

    Response.Write("这是第一个程序：您传来的数据是字符串<br>")

End Sub

' 输入一个数值时启动下列程序

Overloads Sub Test(ByVal decA As Decimal)

    Response.Write("这是第二个程序：您传来的资料是数值" & "<br>")

```

```
End Sub
```

```
' 输入两个字符串时启动下列程序
```

```
Overloads Sub Test(ByVal decA As Decimal, ByVal decB As Decimal)
```

```
    Response.Write("这是第三个程序：您传来的资料是两个数值<br>")
```

```
End Sub
```

```
' 输入两个数值，并用指定运算符接收执行结果时启动下列程序
```

```
Overloads Function Test(ByVal strA As String, ByVal strB As String)
```

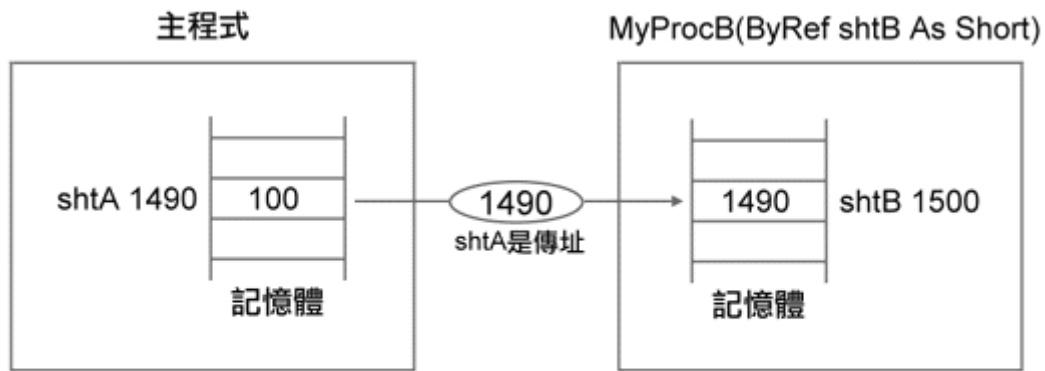
```
    Response.Write("这是第四个程序：您传来的数据是两个字符串" & "<br>")
```

```
    Return strA & strB ' 将字符串连结后传回
```

```
End Function
```

```
</Script>
```

```
</Html>
```



shtB的內容為記憶體位址，表示裡面的資料是參考記憶體位址1490內的內容，所以shtA以及shtB所參照到的記憶體是同一塊，故改變shtB的值將會影響shtA的值

Event 程序

前面我们提过，事件（Event）是对象所认识的动作。对动态网页来说，一页动态网页就是一个 Page 对象。网页在执行时会发生许多事件，这时候我们就可以撰写发生事件时所要执行的动作；例如网页每次在加载时会先发生 Init 事件，然后再发生 Load 事件，最后网页内容全部下载完毕时则会发生 Unload 事件。我们可以利用 Init 事件或是 Load 事件来设定对象的属性值，或是先将要处理的数据从数据库读取出来；最后利用 Unload 事件将使用完毕对象清除，或是将连到数据库的连结关闭，我们一般都是利用 Load 事件来执行一些初值设定的动作。每一种对象所认识的事件不一样，微软在设计这些对象的时候已经规划好对象所认识的事件有哪些。关于对象认识哪些事件，我们在后面章节有详细的介绍。以下为事件程序的语法：

```
Sub 对象名称_事件名(参数 1 As 型态, 参数 2 As 型态,...)
```

```
    程序代码...
```

```
End Sub
```

事件程序一样必需被 **Script** 标注括起来，事件程序是以 **Sub** 关键词所宣告的程序，必需指明希望响应哪个对象所发生的事件，而对象和事件之间要用底线来做分隔。事件程序一样可以接收参数，这些参数由系统所提供，表示发生某个事件时的一些状态。下面这个例子为响应网页执行时所触发的 **Load** 事件：

```
<html>

<%
Response.Write("事件发生后")
%>

<Script Language="VB" Runat="Server">

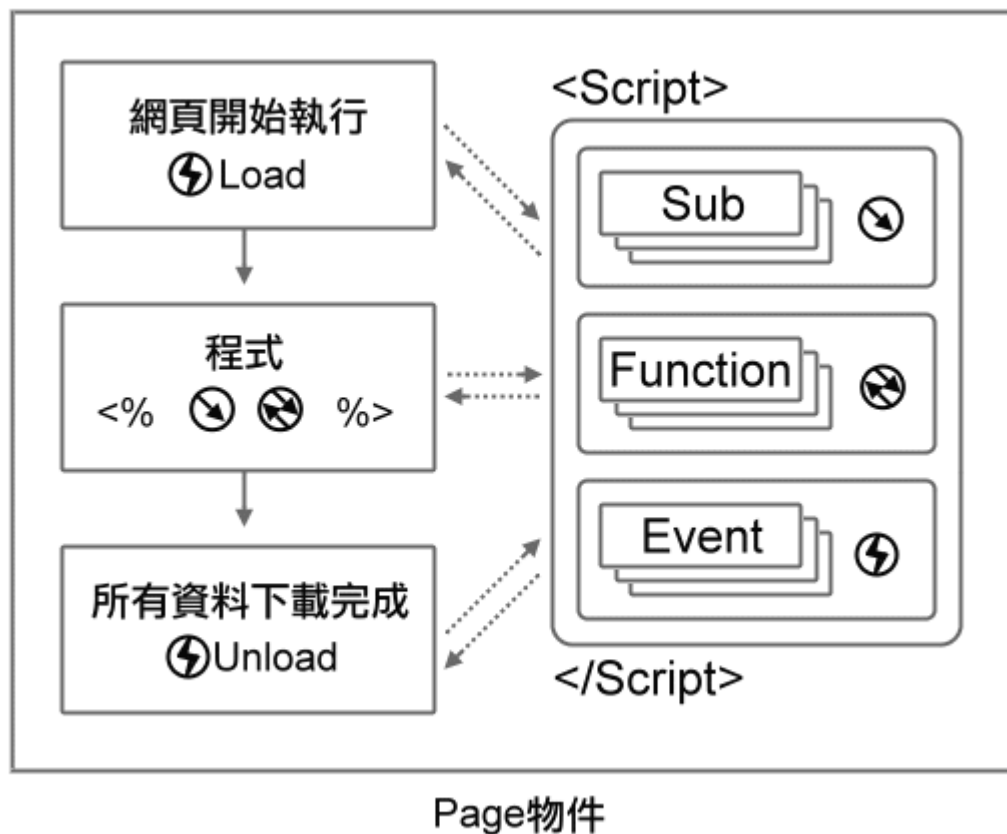
Sub Page_Load()

    Response.Write("网页在加载时发生 Load 事件，即自动执行本程序<br>")

End Sub

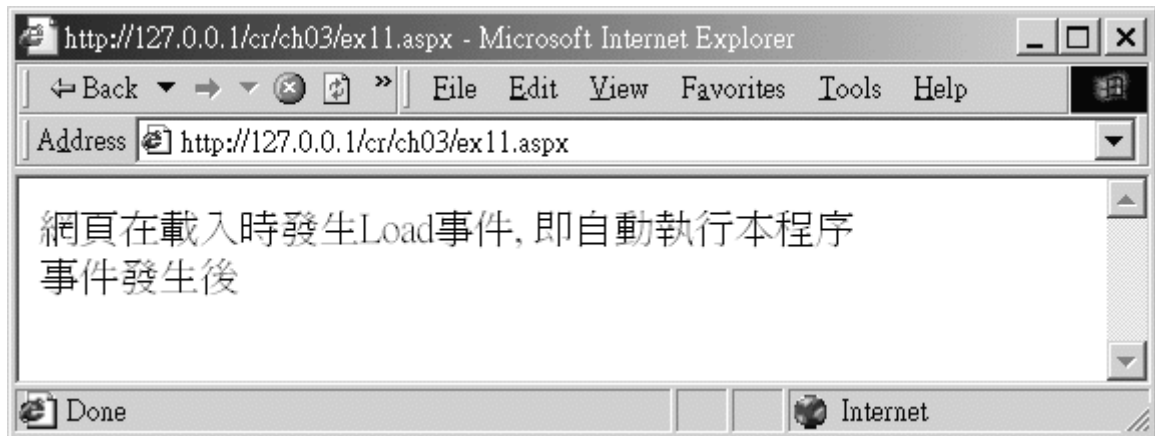
</Script>

</html>
```



网页在载入时会触发 **Load** 事件。由于网页是 **Page** 对象，所以在触发 **Load** 事件时程序会到 **Script** 标注中寻找名称为 **Page_Load** 的事件程序，若找到此事件程序则执行，若没有找到则不执行。

事件程序执行完毕后，程序会回到正常的程序中继续执行程序，等所有的数据下载完毕后即触发 **Unload** 事件。由于 **Unload** 事件最主要的工作是做一些收尾，并且是在所有数据完成下载后触发，所以不可以使用 **Response.Write()** 方法将数据再输出到浏览器，否则会发生错误。



显示时间及日期

基本函式

日期及时间是以数值的数据型态储存，日期可以表示的范围为公元 1 年 1 月 1 日到 9999 年 12 月 31 日，时间的部分为 0:00:00 到 23:59:59。要将指定的日期存入变量，使用如下语法：

```
dtVar=#mm/dd/yyyy#
```

或

```
dtVar=#mm-dd-yyyy#
```

我们要将指定的日期存入日期型态的变量中，该日期必需用井号「#」围起来，并且输入「月/日/年」，例如：

```

<html>

<%

Dim dtMyBday As date

dtMyBday=#10/03/1973#

Response.Write("我的生日是" & Cstr(dtMyBday))

%>

</html>

```

要将日期的某个部分取出，必需要使用一些函式。**.Net Framework** 提供了许多时间及日期的函

式，如下表所示。假设现在为 **2001 年 6 月 1 日星期五 9 点 10 分 11 秒**：

函式名称	说明	范例	传回值
Now()	传回今天的时间及日期	Now()	2001/6/1 上午 09:10:11
Year()	传回年的部分	Year(Now())	2001
Month()	传回月的部分	Month(Now())	6
Day()	传回日的部分	Day(Now())	1
Weekday()	传回一周的第几天	Weekday(Now())	6 (星期天为 1)
Hour()	传回时的部分	Hour(Now())	9
Minute()	传回分的部分	Minute(Now())	10

Second()	传回秒的部分	Second(Now)	11
----------	--------	-------------	----

下列范例是将 EX12.aspx 中生日的公元年份取出，并转换成民国的年份显示：

```
<html>

<%

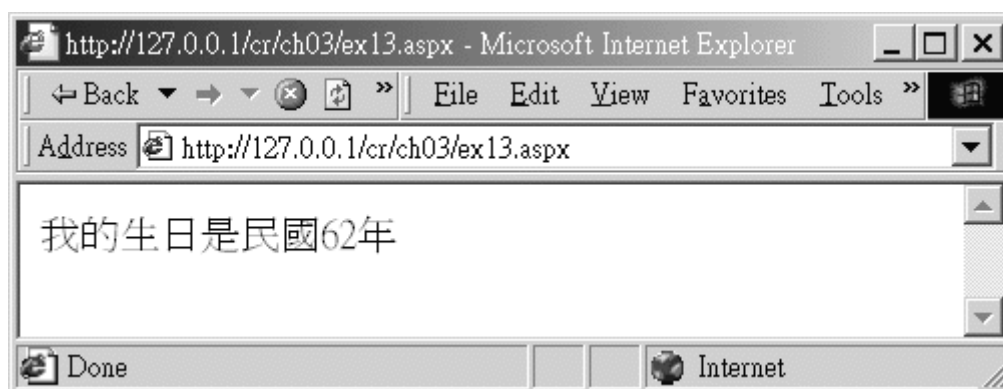
Dim dtMyBday As date

dtMyBday=#10/03/1973#

Response.Write("我的生日是民国" & Cstr(Year(dtMyBday)-1911) & "年")

%>

</html>
```



其它日期处理函式

DateTime 物件

DateTime 对象提供了许多属性及方法让我们来处理时间及日期，其中 **Now** 属性可以传回系统

现在的日期及时间，**Today** 则是传系统现在的日期。下列叙述分别利用 **Now** 及 **Today** 属性传回

系统现在时间日期：

```
Response.Write(DateTime.Now)
```

```
Response.Write(DateTime.Today)
```

另外 **Now** 以及 **Today** 其实是对象型态的属性， 分别有些自己的属性及方法， 如下表所示：

功能	语法	备注
传回系统现在的年	<code>DateTime.Today.Year()</code>	
传回系统现在的月	<code>DateTime.Today.Month()</code>	
传回系统现在的日	<code>DateTime.Today.Day()</code>	
传回系统现在星期几	<code>DateTime.Now.DayOfWeek()</code>	注意，星期一为 1
传回现在是一年的第几天	<code>DateTime.Now.DayOfYear()</code>	
传回系统现在的时	<code>DateTime.Now.Hour()</code>	
传回系统现在的分	<code>DateTime.Now.Minute()</code>	
传回系统现在的秒	<code>DateTime.Now.Second()</code>	

传回现在日期加上指定天数	DateTime.Now.AddDays()	加上指定的天
传回现在日期加上指定月数	DateTime.Now.AddMonths()	加上指定的月
传回现在日期加上指定天数	DateTime.Now.AddYears()	加上指定的年
传回现在日期加上指定秒数	DateTime.Now.AddSeconds()	加上指定的秒
传回现在日期加上指定分钟	DateTime.Now.AddMinutes()	加上指定的分钟
传回现在日期加上指定小时	DateTime.Now.AddHours()	加上指定的小时

DateSerial 函式

当我们需要对于日期做运算的时候，可以利用 **DateSerial** 函式。若是直接对日期型态做运算，则会发生错误。**DateSerial** 函式可以传回指定的日期，语法如下所示：

```
dtVar=DateSerial(Year, Month, Day)
```

下列范例是取回一个月的最后一天后，存入日期型态变量 **dtLastDay**：

```
<html>

<%

Dim dtLastDay As date

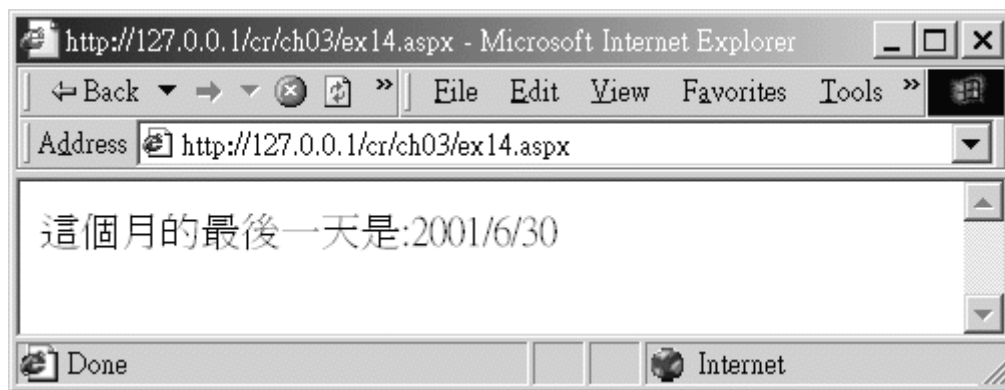
dtLastDay=DateSerial(Year(Now()), Month(Now)+1, 1-1)

Response.Write("这个月的最后一天是:" & Cstr(dtLastDay))
```

```
%>
```

```
</html>
```

我们将本月份取回后加上 1，就是次月。然后再将次月的第一天减去一天，就是本月的最后一天。



DateDiff

DateDiff 函式可以取得两个日期的间隔，并且可以用年、月、日等单位传回两个日期的差距，语法如下所示：

DateDiff(间隔参数，日期一，日期二)

日期相差的单位，必需使用下列表格的参数：

间隔参数	单 位
------	-----

DateInterval.Year	年
DateInterval.Quarter	季
DateInterval.Month	月
DateInterval.Day	日
DateInterval.Week	周
DateInterval.Hour	时
DateInterval.Minute	分
DateInterval.Second	秒

下列范例由使用者输入一日期，并和今天比较相差几天：

```
<html>

<%

Dim dtDate As date,intDiffDay as Integer

dtDate=Cdate(Request("Date"))

intDiffDay=DateDiff(DateInterval.Day,Now(),dtDate)

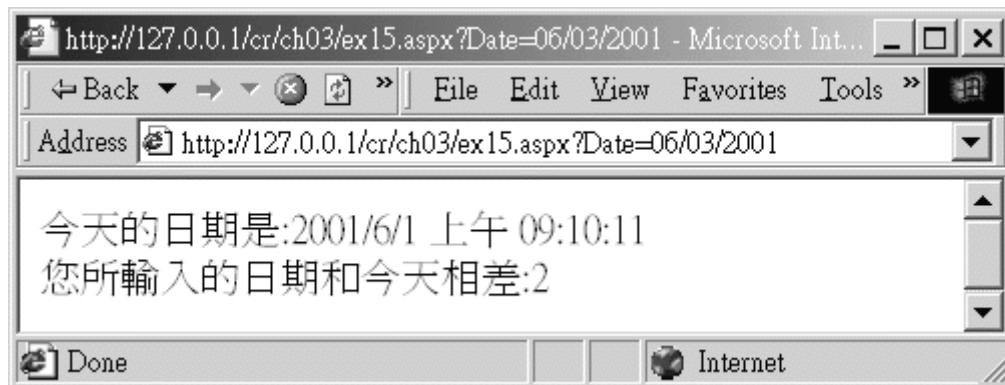
Response.Write("今天的日期是:" & CStr(Now()))

Response.Write("<br>您所输入的日期和今天相差:")

Response.Write(CStr(intDiffDay))

%>
```

```
</html>
```



资料的修饰

要将数据呈现出来之前，我们可以稍作修饰。修饰数据意味着我们要让信息以最适当以及容易阅读的方式呈现出来，所以数据呈现前的修饰是必要的。

使用 **Format** 函数

Format 函数可以修饰日期、数值以及字符串型态的数据，其传回值的数据型态为字符串。以下为使用语法：

```
Format (要修饰的数据[, 要修饰的格式[, 一周的第一天[, 一年的第一周]])
```

格式的参数是一些有意义的符号，这些符号的意义如下表所示：

符号	意义
0	数值配置符号，如果所指定的位置没有数值则印出 0
#	数值配置符号，如果本符号前面为 0 则不印出
.	小数点配置符号
,	千分符号
-\$() 与空格符	文字字符则一五一十的印出

下列程序将数值 50000 格式化成 \$50,000.00：

```
<html>

<%

Response.Write("应付帐款为" & Format(50000,"$##,###.00"))

%>

</html>
```

打印指定的时间及日期格式

Format 函数也可以用来修饰时间及日期，假设现在时间仍为 2001 年 6 月 1 日星期五 9 点 10 分

11 秒，如下表所示：

语法	结果
<code>Format (Now(), "M/d/yy")</code>	6/1/01
<code>Format (Now(), "MM 月 dd 日, dddd, yyyy 年")</code>	06 月 01 日, 星期五, 2001 年
<code>Format (Now(), "MMM-d")</code>	六月-1
<code>Format (Now(), "h:m:s")</code>	9:10:11
<code>Format (Now(), "hh:mm:ss")</code>	09:10:11

字符串处理

有时候我们需要将字符串做一些处理，例如计算字符串的长度、转换字符串的大小写，或是从字符串的某个地址取出指定长度的字符串等，这时候我们就可以使用字符串操作函式来处理字符串。

UCase 及 LCase

想要转换字符串的大小写，可以使用 **UCase** 以及 **LCase** 这两个函式。**UCase** 是将字符串全部转成大写，而 **LCase** 则是全部转成小写，如下范例所示：

```
<html>
```

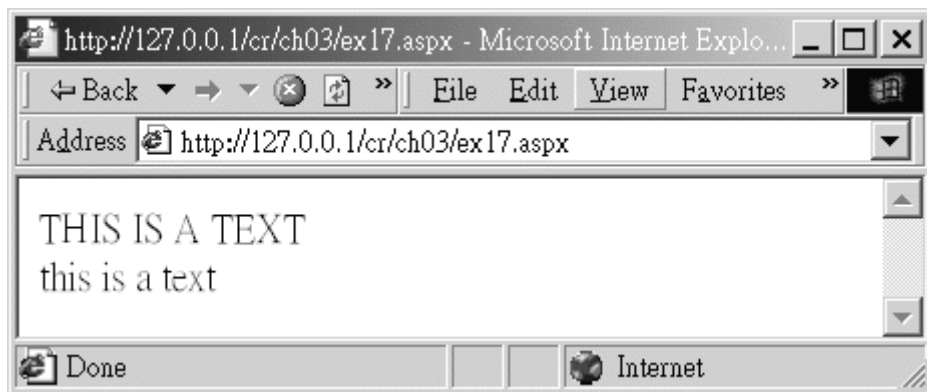
```
<%
```

```
Response.Write(UCASE("this is a text") & "<br>")

Response.Write(LCASE("THIS IS A TEXT") & "<br>")

%>

</html>
```



StrComp

想要比较两个字符串，可以使用 **StrComp** 函式。**StrComp** 的语法如下所示：

```
StrComp(字符串一, 字符串二[, 比较方式])
```

其中比较方式有两种，如下表所示：

比较方式	参 数 值
------	-------

以二进制内容比较（分大小写）	0
以文字内容比较（不分大小写）	1

由于比较方式的参数被中括号括起来，表示为非必要参数，若不指定比较方式则预设**为 0**。字符串的比较结果，其传回值如下表所示：

传回值	说明
-1	字符串一小于字符串二
0	字符串一等于字符串二
1	字符串一大于字符串二

美国国家标准局（**ANSI**）对每个字符建立了一个指定码，例如大写**A**被指定为**65**，而大写**B**被指定为**66**依此类推，所以字符可以做比较。这边要注意的是小写的定义在大写的定义之后，所以小写的编码比大写大，比较的时候要注意。对于字符串的比较则是一个一个字符做比较，倘若再不分大小写的状态下，两个字符串的长度和排列顺序都相等时，则这两个字符串相等。倘若遇到不同的字符时，则比较不同字符的大小。例如 **different** 和 **difference** 这两个字做比较，则是 **different** 大；因为 **t** 比 **c** 来的大。下面范例比较字符串「test」以及字符串「TEST」：

```
<html>

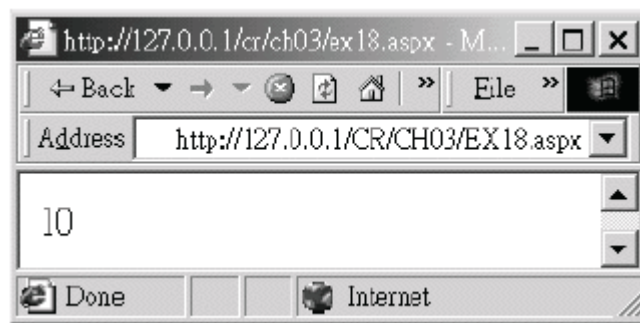
<%

Response.Write(StrComp("test","TEST",0))
```

```
Response.Write(StrComp("test", "TEST", 1))

%>

</html>
```



Len、Left、Right、Mid、Trim

Len 函式可以传回字符串的长度，**Left**、**Right** 函式可以分别从左边及右边取回指定长度的字符串，**Mid** 函式可以指定从字符串的第几个字符取回指定长度的字符串，最后 **Trim** 函式可以去掉字符串左右两边的空格符。我们来看看下面的综合范例：

```
<html>

<%

Dim strA As String="Hello, ASP.Net!"

Dim strB As String="你好，新一代的动态网页!"
```

```
Response.Write("英文字符串的长度为:" & Len(strA))

Response.Write("<br>中文字符串的长度为:" & Len(strB))

Response.Write("<br>从英文字符串左边取五个字符:" & Left(strA, 5))

Response.Write("<br>从英文字符串右边取八个字符:" & Right(strA, 8))

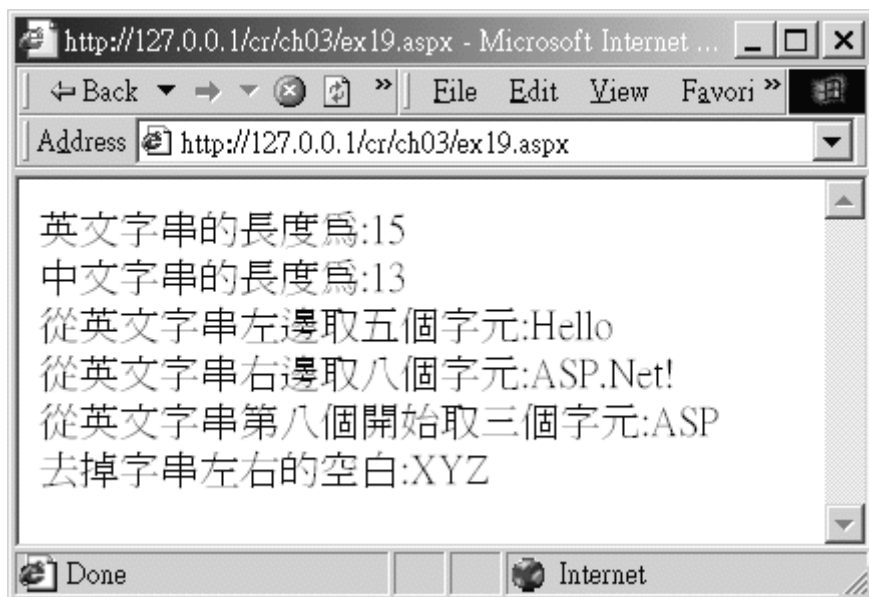
Response.Write("<br>从英文字符串第八个开始取三个字符:" &

Mid(strA, 8, 3))

Response.Write("<br>去掉字符串左右的空白:" & "X" & Trim(" Y ") & "Z")

%>

</html>
```



流程控制

程序在执行的时候，我们可以视情况决定要执行哪些程序，不执行哪些程序，或决定让某段程序代码可以重复执行；这些动作我们称为程序的流程控制。要控制程序的流程，首先我们要了解两种运算符，这两种运算符为比较运算符以及逻辑运算符。这两种运算符的传回值型态都是为布尔，不是代表真的 **True** 就是代表伪的 **False**。

比较运算符

要比较数值、字符串或日期，可以利用比较运算符。例如假设我们的程序要求使用者输入年龄，如果未满十八岁的话就不允许执行一些程序。这时候我们就要来检查使用者所输入的年龄，

VB.NET 提供了六种用来测试数据的比较运算符，如下图所示：

运算符	说明	范例	结果
<	小于	A=8<7	A=False
<=	小于等于	B=5<=5	B=True
>	大于	C=6>5	C=True
>=	大于等于	D=7>=3	D=True
=	等于	E=4=5	E=False

<>	不等于	F=8<>9	F=True
----	-----	--------	--------

比较运算符会将其比较测试结果以 **True** 或 **False** 的布尔型态传回，上表的范例利用指定运算符将比较的结果放到指定运算符左边的变量 **A~F**。

逻辑运算符

除了比较运算符外，还有逻辑运算符。最常用到的逻辑运算符是 **And**、**Or** 及 **Not**。

And

And 运算符用来组合两个叙述，**And** 运算符只有在这 **A** 以及 **B** 两个叙述同时为 **True** 时传回 **True**，其余皆传回 **False**。以下为 **And** 运算符的真值表，其中 **F** 代表 **False**，**T** 代表 **Ture**，而 **Y** 代表输出：

A	B	Y
F	F	F
F	T	F
T	F	F
T	T	T

Or

Or 运算子用来组合两个叙述，只要 Or 运算子在这 A 或 B 两个叙述有一个为 True 时传回 True，

只有在两个叙述都为 False 时传回 False。以下为 Or 运算子的真值表，其中 F 代表 False，T 代

表 True，而 Y 代表输出：

A	B	Y
F	F	F
F	T	F
T	F	F
T	T	T

其中 F 代表 False，T 代表 True，而 Y 代表输出。

Not

Not 运算子用倒置单一叙述，Not 运算子在叙述为 False 时传回 True，叙述为 True 时传回 False。

以下为 Not 运算子的真值表，其中 F 代表 False，T 代表 True，而 Y 代表输出：

A	Y
F	T
T	F

If...Then 叙述

If...Then 叙述可以依条件式的检查结果决定程序代码的执行。If...Then 的结构有三种，分别为

If...Then、If...Then...Else 以及 If...Then...Elseif 这三种。

If...Then 叙述

我们利用 If...Then 叙述来决定程序是否要或是不要执行某段程序代码，其语法如下所示：

If 条件判断 Then 叙述

或

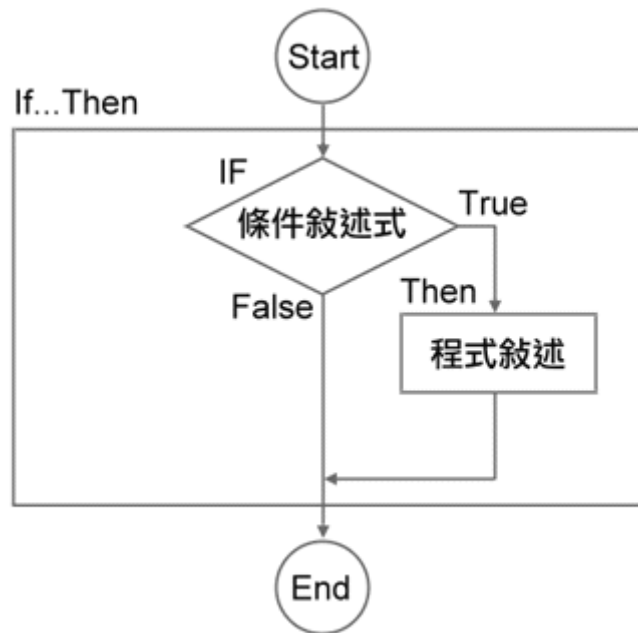
If 条件判断 Then

叙述一

叙述二

叙述 N...

End If



If 判断句会检查条件判断式的测试结果。其结果若传回 **True** 或是非零的数值，则表示条件成立，便执行 **Then** 后面的叙述。倘若测试结果传回 **False**，则检查结果不成立，那们就不会执行 **Then** 后面的程序叙述。如果叙述只有一行，则可以接在 **Then** 后面撰写程序叙述：

```
IF shtCPUSpeed < 500 Then Response.Write("你的计算机速度太慢了!")
```

以上的程序代码叙述为 **shtCPUSpeed** 变量和 **500** 做小于比较，倘若成立的话则传回 **True** 成立，并执行 **Then** 后面的叙述。由于叙述只有一行，故直接写在 **Then** 后面并且不需要加上 **End If** 叙

述。倘若要执行的叙述多于一行，则要在新的五行加入程序代码叙述，叙述结束后必需要加上

End If 关键词：

```
<html>

<%

Dim intIncome As Integer

intIncome=Cint(Request("Income"))

If intIncome>990000 Then

    Response.Write("您的年输入超过 99 万，您应该缴的税为:")

    Response.Write(Format((intIncome*0.21)-105100,"$##,###,###.00"))

End If

%>

</html>
```

我们在呼叫上述 **aspx** 网页时如果将 **Income** 参数以 **990000** 传入，则 **If** 后面的条件判断式传回 **False** 代表不成立，所以 **Then** 至 **End If** 之间的程序代码将不被执行；如果将 **Income** 参数以 **1000000** 传入，则 **If** 后面的条件判断式传回 **True** 代表成立，所以 **Then** 至 **End If** 之间的程序代码将被执行。

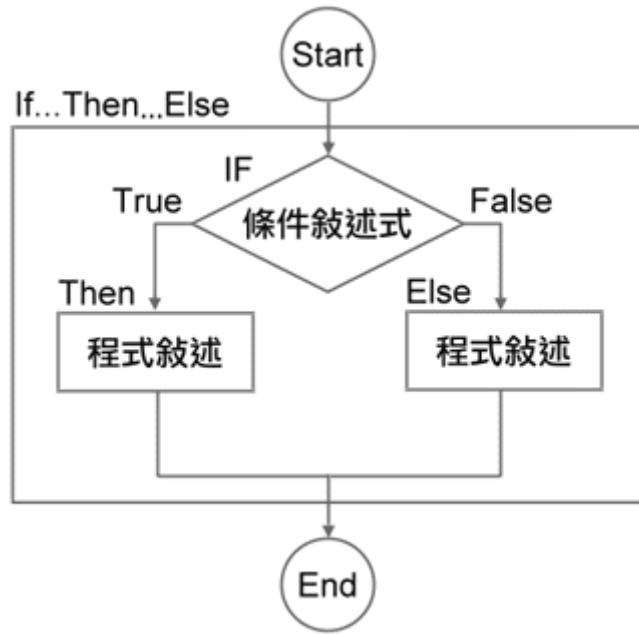
Note

上列程序我们将 **If** 和 **End If** 之间的程序代码往内缩两个字符，这个程序代码的编排方式称做缩排。对于有结构的程序代码叙述，在撰写时将程序代码缩排可以让程序代码更易阅读。

If...Then...Else 叙述

我们利用 **If...Then...Else** 叙述来决定程序要执行的程序代码区块，其语法如下所示：

```
If 条件判断 Then
    程序区块一
Else
    程序区块二
End If
```



If...Then...Else 结构在条件判断式的测试结果为 True 成立时，执行 Then 以及 Else 之间的程序，执行完毕后则直接跳出 If 判断结构继续执行程序；倘若条件判断式的结果为 False 不成立，则执行 Else 以及 End If 之间的程序代码，执行完毕后一样直接跳出 If 判断结构继续执行程序。

```

<html>

<%

Dim intIncome As integer

intIncome=CInt(Request("Income"))

If intIncome>990000 Then

    Response.Write("您的年收入超过 99 万，您应该缴的税为:")

    Response.Write(Format((intIncome*0.21)-105100,"$##,###,###.00"))

Else

    Response.Write("您的年收入没超过 99 万，您应该缴的税为:")
  
```

```
Response.Write(Format((intIncome*0.13)-25900, "$##, ###, ###.00"))  
  
End If  
  
%>  
  
</html>
```

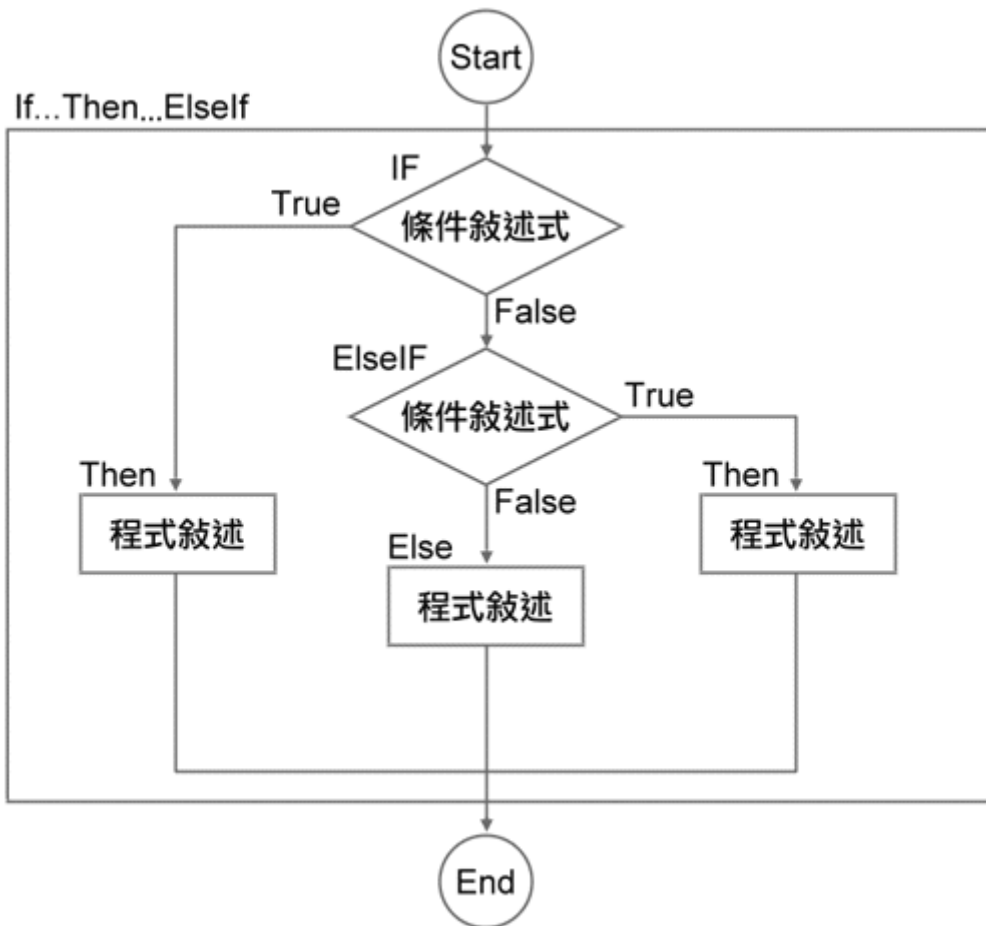
我们在呼叫上述 **aspx** 网页时如果将 **Income** 参数以 1000000 传入，则 **If** 后面的条件判断式传回 **True** 代表成立，所以 **Then** 至 **Else** 之间的程序代码将被执行，执行完毕后即略过 **Else** 后面的程序代码并直接跳出 **If** 判断结构继续执行程序；如果将 **Income** 参数以 990000 传入，则 **If** 后面的条件判断式传回 **False** 代表不成立，所以 **Then** 至 **Else** 之间的程序代码将被略过不被执行，直接无条件执行 **Else** 至 **End If** 之间的程序区块。

If...Then...ElseIf 叙述

我们利用 **If...Then...ElseIf** 叙述来执行多个条件式的判断，其语法如下所示：

```
If 条件判断一 Then  
    程序区块一  
ElseIf 条件判断二  
    程序区块二  
[Else  
    程序区块三]
```

End If



If...Then...Elseif 结构在条件判断式一的测试结果为 **True** 成立时，执行程序区块一的程序，执行完毕后不再做其它的条件判断，直接跳出 **If** 判断结构继续执行程序；倘若条件判断式一的结果为 **False** 不成立，则执行 **Elseif** 叙述的条件判断式二。倘若测试结果为 **True** 成立时，则执行程序区块二；倘若条件表达式二的检查结果依然为 **False** 不成立，则无条件执行 **Else** 及 **End If** 之间

的程序区块三，执行完毕后一样跳出 If 结构继续执行程序；倘若没有 **Else** 叙述，则不执行任何动作。

```
<html>

<%

Dim shtGrade As Short=CShort(Request("Grade"))

If shtGrade>=80 Then

    Response.Write("您的等级为 A")

ElseIf shtGrade>=70 Then

    Response.Write("您的等级为 B")

ElseIf shtGrade>=60 Then

    Response.Write("您的等级为 C")

Else

    Response.Write("您的等级为 D")

End If

%>

</html>
```

上述程序代码为将学生的成绩转换成 A、B、C、D 的等级。这里的程序代码我们不再多做说明，

唯一要注意的是下面这行用法：

```
Dim intGrade As Integer=CInt(Request("Grade"))
```

上列叙述用 **Resquest** 对象从暂存区将参数 **Grade** 取出后，为变量 **intGrade** 的初始值，这样一来就可以取代原来的两行叙述：

```
Dim intGrade As Integer  
  
IntGrade=CInt(Request("Grade"))
```

巢状结构

如果 **If** 条件判断式所要执行的程序区块里还有 **If** 条件判断式，这种样式的循环我们称为巢状结构，如下所示：

```
If 条件判断 Then  
    If 条件判断 Then  
        程序区块  
    Else  
        程序区块  
    End If  
Else  
    程序区块  
End If
```

透过巢状结构的运用，程序的撰写就可以更灵活更有弹性。例如我们将上述程序多加个巢状 If 判断，六十分以下除印出等级为 D 外，倘若不低于五十分则印出可以补考，低于五十分则印出不能补考。

```
<html>

<%

Dim shtGrade As Short=CShort(Request("Grade"))

If shtGrade>=80 Then

    Response.Write("您的等级为 A")

ElseIf shtGrade>=70 Then

    Response.Write("您的等级为 B")

ElseIf shtGrade>=60 Then

    Response.Write("您的等级为 C")

Else

    Response.Write("您的等级为 D")

    If shtGrade>=50 Then

        Response.Write("可以补考")

    Else

        Response.Write("不能补考")

    End If

End If
```



```
End If

%>

</html>
```

条件判断式和逻辑运算子的配合

上述所做的范例都是使用比较运算子进行比较运算，而本部分我们要利用逻辑运算子和条件判断式搭配使用。在条件判断式中，最主要是检查最后的结果来决定程序的执行。只要我们所使用的条件判断式最后传回 **True** 或是 **False**，就可以决定程序如何执行。在条件判断式中使用逻辑运算子，可以让我们做多个条件的判断，让我们撰写程序时更具弹性。

And

And 运算子用于我们想要多个同时条件都成立时，才执行某段程序代码时使用。例如下列程序为检查使用者的名称及密码：

```
<html>

<%

Dim strID As String=CStr(Request("ID"))

Dim strPWD As String=CStr(Request("PWD"))

If strID="Charles" AND strPWD="1234" Then
```

```
Response.Write("使用者名称及密码正确!")

Else

Response.Write("使用者名称及密码错误!")

End If

%>

</html>
```

If 陈述最主要是检查 If 和 Then 间条件判断式的结果，在上述程序中条件判断式为：

```
strID="Charles" AND strPWD="1234"
```

VB.NET 执行这行条件判断式时，先检查左边的比较运算；如 `strID="Charles"` 的比较结果为 `True`，则再进行右边 `strPWD="1234"` 的比较运算；倘若右边比较结果为 `True`，最后才进行 `And` 运算。由于是执行 `And` 运算，`And` 运算子左右两边的运算结果为 `True` 时，`And` 运算才传回 `True`。所以只要执行时，`And` 运算子左右两边的运算结果只要传回 `False`，VB.NET 即不继续执行下一步的检查并传回 `False`。

快捷方式回路 **short-curruited**

上述表达式若是在之前的 VB 版本，则不管 `And` 运算子左右两边表达式的运算结果，一定先依序将两个表达式运算完毕后再执行 `And` 运算；而 VB.NET 对于一些表达式可先行判断，并不见得会执行全部的运算。例如 VB.NET 在执行 `Or` 运算时，只要运算到 `True` 值后即停止后续运

算，并直接传回 **True**。这是因为目前的条件传回 **True** 已经成事实，后面的结果并不会影响整个结果。这样一来就可以少执行一些运算，所以能有效提升执行的效率；这个新的功能我们称为快捷方式回路（**short-curuted**）。

Or

Or 运算符则是用在多个条件中，只要有一个条件成立时，就执行某段程序代码时使用。例如下列程序只要使用者小于 **6** 岁或是大于 **60** 岁，就将门票售价打五折；其余年龄则打八折：

```
<html>

<%

Dim shtAge As Short=CShort(Request("Age"))

Dim sngDiscount As Single

Const cnTicketPrice=200

If shtAge<6 Or shtAge>60 Then

    sngDiscount=0.5

Else

    sngDiscount=0.8

End If
```

```
Response.Write("您的票价为:" & CStr(cnTicketPrice * sngDiscount) & "
元")

%>

</html>
```

Not

Not 值可以将布尔值做倒置，也就是将 **False** 变成 **True**，将 **True** 变成 **False**。例如下列程序只要使用者的年龄不小于 **18**，即印出欢迎语：

```
<html>

<%

Dim shtAge As Short=CShort(Request("Age"))

If Not shtAge<18 Then

    Response.Write("欢迎您的浏览")

Else

    Response.Write("本站不接受未成年之青少年浏览，请离开")

End If

%>

</html>
```

Select Case 叙述

Select Case 和 **If...Then...Elseif** 的结构很相似，都是让程序检查叙述值后，再决定所要执行的程序代码。不过 **Select Case** 执行起来比 **If...Then...Elseif** 更有效率，这是因为 **Select Casse** 只需将要做比较的变量取出一次。我们来看看 **Select Case** 的语法：

```
Select Case 测试叙述
```

```
Case 条件叙述一
```

```
    程序区块一
```

```
Case 条件叙述二
```

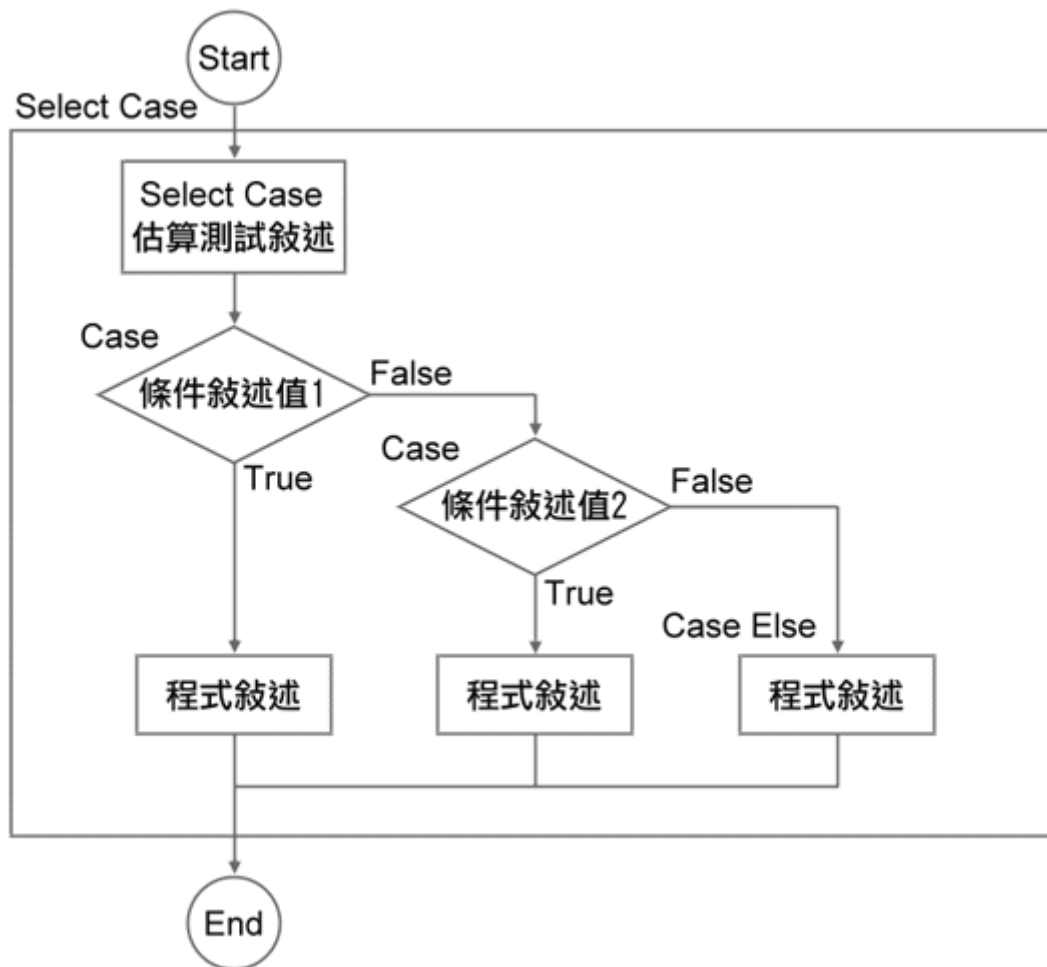
```
    程序区块二
```

```
Case 条件叙述 N
```

```
    程序区块 N
```

```
[Case Else]
```

```
End Select
```



其中测试叙述可以是任何变量、数值或字符串，VB.NET 将叙述直接取出或运算后，再将结果和 **Case** 后面的条件叙述所执行完的结果做比较。和 **If...Then...Elseif** 一样，倘若测试叙述的结果和条件叙述的结果相等，则执行相对应的程序区块，执行完毕后即跳出架构外继续程序的执行，其中 **Case Else** 也可以省略。我们来看看下列这个范例：

```

<html>

<%

Dim strUserName As String=CStr(Request("UserName"))

Dim strMSG As String
  
```

```
Select Case strUserName

Case "Administrator"

    strMSG="Hello, Administrator."

Case "User"

    strMSG="Hello, User."

Case Else

    strMSG="Hello, Guest."

End Select

Response.Write(strMSG)

%>

</html>
```

如果上述程序以 **If...Then...Elseif** 来写也可以，但是 **Select Case** 的结构比较易于阅读，执行起来效率也好多了。

测试数值的范围

Select Case 的架构用于范围的测试非常理想，例如下列范例：

```
<html>

<%
```

```
Dim shtAns As Short=CShort(Request("Ans"))

Dim shtGrade As Short

Select Case shtAns

Case 1 ' 等于 1

    shtGrad=1

Case 2 to 4 ' 2 到 4(包含)

    shtGrad=2

Case 5,6 ' 5 以及 6

    shtGrad=3

Case Is >= 7 ' 大于等于七

    shtGrad=4

End Select

Response.Write("您得到的积分为:" & CStr(shtGrad))

%>

</html>
```

循环

循环结构可以让我们的程序在我们所指定的条件下反复的执行一段程序代码，循环结构的执行可以利用检查条件值的结果为 **False** 或 **True** 来决定是否继续执行。在 **VB.NET** 中有两种型态的循环，分别为 **Do...Loop** 循环以及 **For...Next** 循环。

Do...Loop 循环

Do...Loop 循环结构可以让我们依据某个条件的传回值为 **False** 或 **True**，决定是否要反复执行某段程序代码区块。**Do...Loop** 循环有四种，分别为 **Do...Loop While**、**Do...Loop Until**、**Do While...Loop** 以及 **Do Until...Loop**。

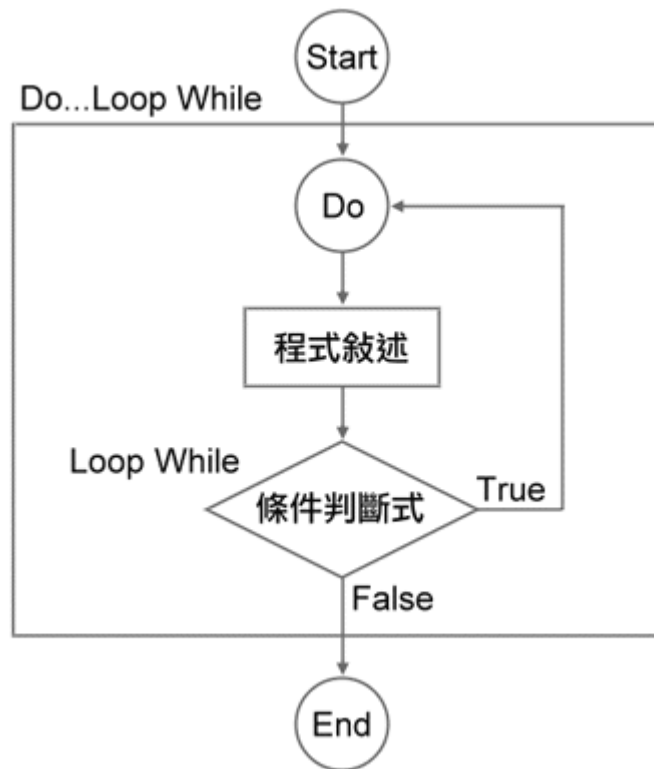
Do...Loop While 循环

这种 **Do...Loop While** 结构的循环，不管条件判断式是否成立，至少执行一次。**Loop While** 则表示如果 **While** 后面的条件判断式成立的话，即跳回 **Do** 继续执行循环，其语法如下：

```
Do
```

```
    程序代码叙述
```

```
Loop While 条件判断式
```



Do...Loop While 循环在执行时，会先跳进循环内执行程序代码叙述；执行完毕遇到 Loop While 即检查条件判断式的结果。倘若条件判断式的结果为 **True**，程序的执行就会跳到 Do 重新执行循环，一直执行到条件判断式的结果为 **False** 为止。我们来看看下列的范例：

```
<html>

<%

Dim shtLoop As Short=CShort(Request("Loop"))

Dim shtX As Short=1

Do

    Response.Write("循环执行了:" & CStr(shtX) & "次<br>")
```

```
shtX+=1

Loop While shtX<=shtLoop

%>

</html>
```

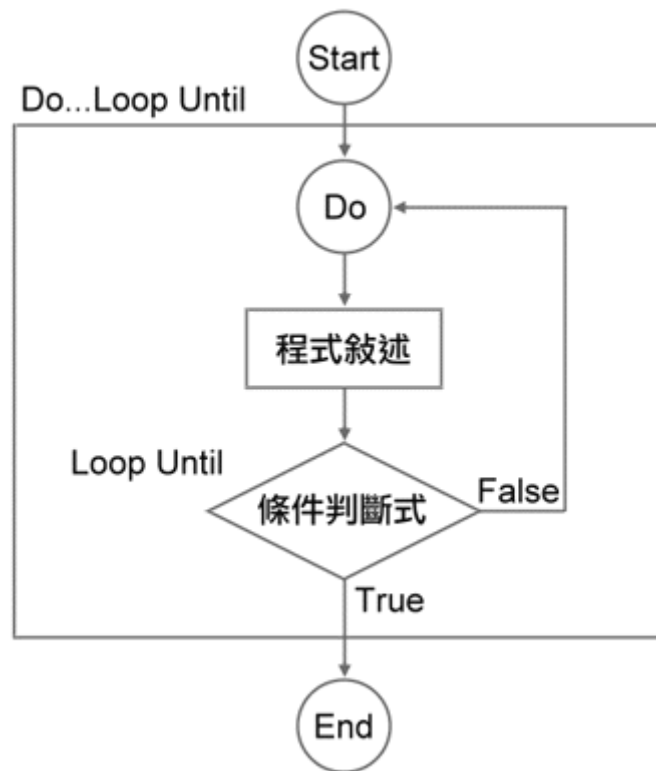
Do...Loop Until 循环

这种 Do...Loop Until 结构的循环，不管条件判断式是否成立，一样至少执行一次。Loop Until 则表示如果 Until 后面的条件判断式传回 **False** 不成立的话，即跳回 Do 继续执行循环，做到 Until 后面的条件判断式传回 **True** 成立为止，其语法如下：

```
Do

    程序代码叙述

Loop Until 条件判断式
```



Do...Loop Until 循环在执行时，会先跳进循环内执行程序代码叙述；执行完毕遇到 Loop Until 即检查条件判断式的结果。倘若条件判断式的结果为 **False**，表示目前的条件判断式还未满足，程序的执行就会跳到 Do 重新执行循环，一直执行到条件判断式的结果为 **True** 满足为止。我们将上列的范例改成以 Do...Loop Until 的方式撰写：

```
<html>

<%

Dim shtLoop As Short=CShort(Request("Loop"))

Dim shtX As Short=1

Do
```

```
Response.Write("循环执行了:" & CStr(shtX) & "次<br>")

shtX+=1

Loop Until shtX>shtLoop

%>

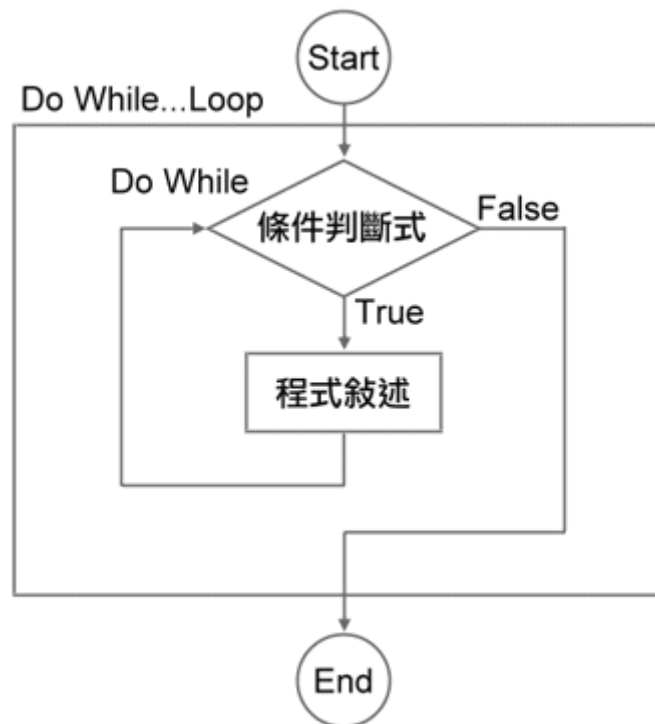
</html>
```

Do While...Loop 循环

这种 Do While...Loop 结构的循环，则是先判断条件判断式是否成立，才会跳入循环内执行。其

语法如下：

```
Do While 条件判断式
    程序代码叙述
Loop
```



Do While...Loop 循环在执行时，会先判断条件是否成立，再进入循环。循环内的程序叙述执行到 Loop 时，则表示跳回 Do While 叙述执行后面的条件判断式，倘若为 True 成立即跳入循环继续执行；倘若为 False 则跳出循环，继续程序的执行，我们来观察下列的范例：

```
<html>

<%

Dim shtLoop As Short=CShort(Request("Loop"))

Dim shtX As Short=1

Do While shtX<=shtLoop

    Response.Write("循环执行了:" & CStr(shtX) & "次<br>")
```

```
shtX+=1
```

```
Loop
```

```
%>
```

```
</html>
```

另外这种 **Do While...Loop** 循环还有另一种写法，那就是 **While...End While**。其语法如下所示：

```
While 条件判断式
```

```
    程序代码叙述
```

```
End While
```

上述程序代码范例可以改成：

```
<html>
```

```
<%
```

```
Dim shtLoop As Short=CShort(Request("Loop"))
```

```
Dim shtX As Short=1
```

```
While shtX<=shtLoop
```

```
    Response.Write("循环执行了:" & CStr(shtX) & "次<br>")
```

```
    shtX+=1
```

```
End While
```

```
%>
```

```
</html>
```

Do Until...Loop 循环

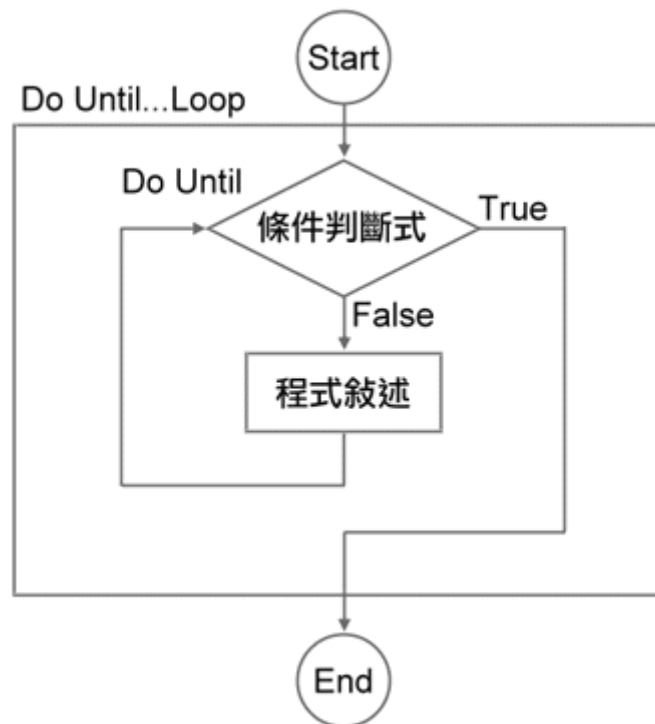
这种 Do Until...Loop 结构的循环，则是先判断条件判断式是否满足，才会跳入循环内执行。其

语法如下：

```
Do Until 条件判断式
```

```
    程序代码叙述
```

```
Loop
```

Do Until...Loop 循环在执行时，会先判断条件是否已经为 **True**，倘若为 **False** 则进入循环。循环内的程序叙述执行到 **Loop** 时，则表示跳回 Do Until 叙述执行后面的条件判断式，倘若为 **False** 即跳入循环继续执行；倘若为 **True** 则跳出循环，继续程序的执行，我们来观察下列的范例：

```
<html>

<%

Dim shtLoop As Short=CShort(Request("Loop"))

Dim shtX As Short=1

Do Until shtX>shtLoop

    Response.Write("循环执行了:" & CStr(shtX) & "次<br>")
```

```
shtX+=1
```

```
Loop
```

```
%>
```

```
</html>
```

无穷循环

在使用循环结构时，要撰写可以让程序跳出循环的程序叙述。也就是说要可以让循环判断式的结果核定为不继续执行循环。倘若循环中没有可以改变条件叙述之结果的程序，则永远无法跳出循环，这样的循环我们称为无穷循环。例如上面的 **Do Until...Loop** 循环，我们将改变 **shtX** 的叙述 **shtX+=1** 拿掉，这样的循环就变成无穷循环了，如下范例所示：

```
Do Until shtX>shtLoop
```

```
    Response.Write("循环执行了:" & CStr(shtX) & "次<br>")
```

```
Loop
```

VB.NET 提供了可以强迫跳出循环的叙述 **Exit Do**，但是在条件叙述中如果可以指定跳出循环的叙述，则最好不要用 **Exit Do** 叙述来强迫跳出循环。**Exit Do** 叙述在循环内利用 **If** 判断式判断是否要跳出循环，用法如下范例：

```
If 条件叙述式 Then
```

```
    Exit Do
```

```
End If
```

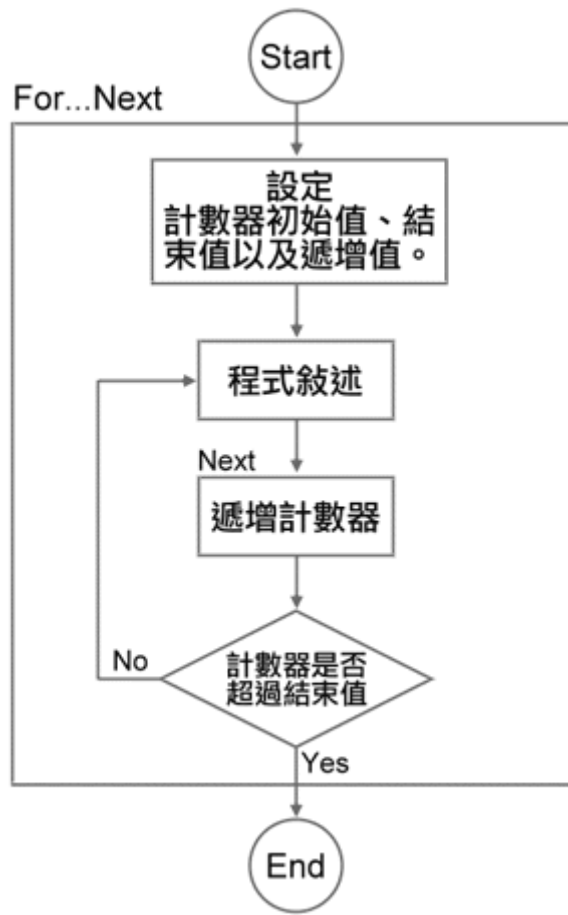
For...Next 循环

当我们知道一段程序代码所要执行的条件以及次数时，就可以使用 For...Next 循环。For...Next 循环比 Do...Loop 循环容易使用及维护，以下为其语法：

```
For 计数器=起始值 to 结束值 [Step 递增值]
```

```
    程序代码叙述
```

```
Next [计数器]
```



循环执行时，要先设定计数器的初始值、结束值与执行一次的递增值为多少。当我们开始执行 For...Next 循环时，不管当作计数器的变量其值原来为何，都会重新被填入初始值；当循环执行到 Next 时，计数器会被先加上递增值，然后再检查是否超过结束值；如果没有超过结束值则继续执行循环，超过结束值则跳出循环。我们看看下面的程序代码：

```
<html>

<%

Dim shtLoop As Short=CShort(Request("Loop"))
```

```
For shtCount=1 to shtLoop Step 1

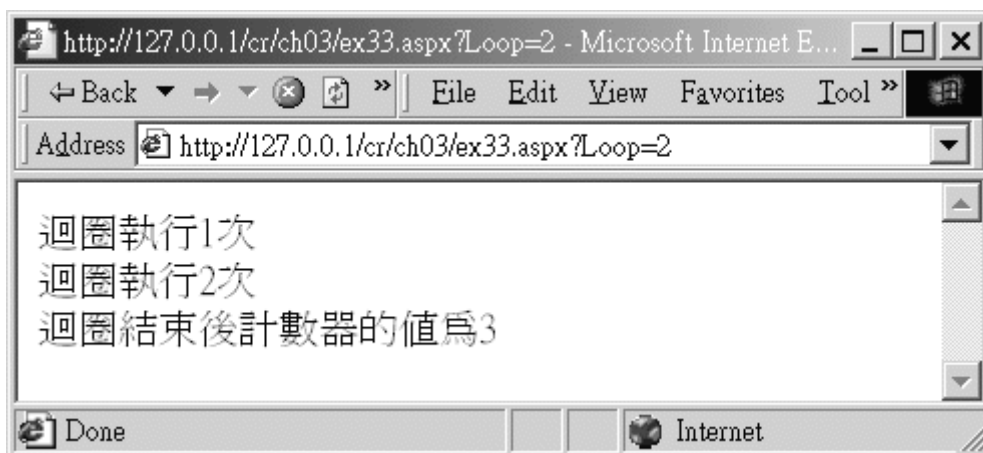
    Response.Write("循环执行" & CStr(shtCount) & "次<br>")

Next shtCount

Response.Write("循环结束后计数器的值为" & CStr(shtCount))

%>

</html>
```



上述范例码利用 **Request** 对象输入循环所要执行次数至变量 **shtLoop** 中，当作是循环的结束值；然后指定计数器 **shtCount** 的初始值为 **1**，并且指定执行完一次循环所的递增值为 **1**。我们假设使用者所输入的 **Loop** 参数为 **2**，所以循环执行完第一次时，遇到 **Next** 叙述将计数器递增 **1**，并检查是否超过结束值；此时 **2** 并没有超过结束值，所以在跳进循环中继续执行程序，遇到 **Next** 叙述又将计数器递增 **1**，并检查是否超过结束值。此时计数加 **1** 后的值为 **3** 超过结束值，所以便

跳出循环。这里要注意的是跳出循环时，计数器的值总是比结束值多了递增值。另外如果递增值为 1，那么 **Step 1** 的叙述可以省略。

巢状 For...Next 循环

For...Next 循环也可以写成巢状结构，如下所示：

```
For 计数器 1=起始值 to 结束值 [Step 递增值]

    For 计数器 2=起始值 to 结束值 [Step 递增值]

        程序代码叙述

    Next [计数器 1]

Next [计数器 2]
```

这种巢状的 **For...Next** 循环执行时先设定外层的条件，然后跳进内层回圈内执行内层循环。待内层循环执行超过结束值后，便跳至外层循环的 **Next** 叙述，将外层的计数器递增。外层循环计数器递增后若没超过结束值，则再从新进入循环内重新执行内层循环。此时内层循环的计数器会重新以初始值设定，重头执行。这样执行的次数要等外层循环结束后才会结束，如下面范例所示：

```
<html>

<%

Dim shtX, shtY As Short
```

```
For shtX=1 To 2 Step 1

    For shtY =1 To 2 Step 1

        Response.Write("shtX=" & CStr(shtX))

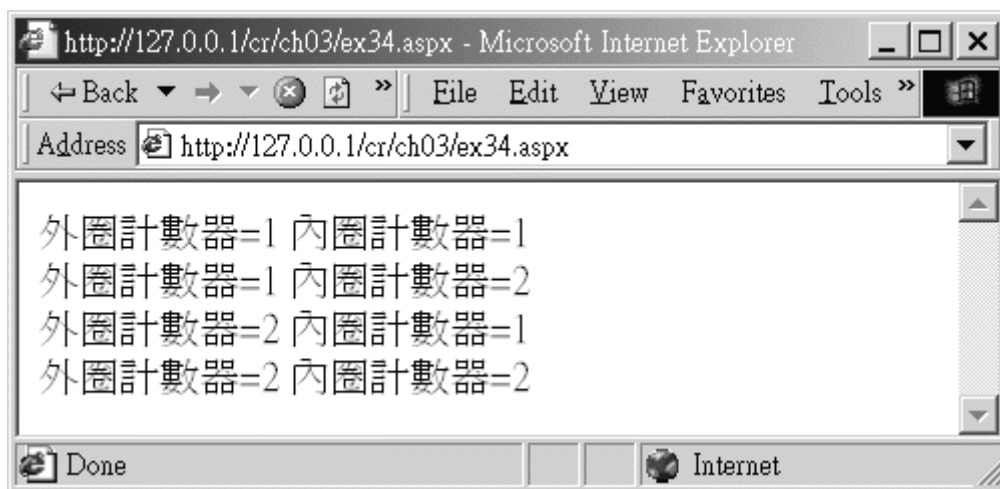
        Response.Write("  shtY=" & CStr(shtY) & "<br>")

    Next

Next shtX

%>

</html>
```



我们观察外层循环以及内层循环计数器的状态值便可知道，外层循环跳进内层循环执行时，外层等到内层执行完毕后再跳至外层将计数器加一，而每次跳进内层时内层计数器的值都会被重新设

定。内层的循环会一直反复重新执行，直到外层超过结束值为止。接下来的例子我们利用巢状循

环画出九九表：

```
<html>

<table border="1">

<%

Dim shtX, shtY As Short

For shtX=1 To 9 Step 1

    If shtX=1 OR shtX=4 OR shtX=7 Then Response.Write("<tr>")

    Response.Write("<td>")

    For shtY =1 To 9 Step 1

        Response.Write(CStr(shtX) & "X" & CStr(shtY) & "=" & _

                        CStr(shtX * shtY) & "<br>")

    Next shtY

    Response.Write("</td>")

    If shtX=3 OR shtX=6 OR shtX=9 Then Response.Write("</tr>")

Next shtX

%>

</table>
```


1X1=1	2X1=2	3X1=3
1X2=2	2X2=4	3X2=6
1X3=3	2X3=6	3X3=9
1X4=4	2X4=8	3X4=12
1X5=5	2X5=10	3X5=15
1X6=6	2X6=12	3X6=18
1X7=7	2X7=14	3X7=21
1X8=8	2X8=16	3X8=24
1X9=9	2X9=18	3X9=27
4X1=4	5X1=5	6X1=6
4X2=8	5X2=10	6X2=12
4X3=12	5X3=15	6X3=18
4X4=16	5X4=20	6X4=24
4X5=20	5X5=25	6X5=30
4X6=24	5X6=30	6X6=36
4X7=28	5X7=35	6X7=42
4X8=32	5X8=40	6X8=48
4X9=36	5X9=45	6X9=54
7X1=7	8X1=8	9X1=9
7X2=14	8X2=16	9X2=18
7X3=21	8X3=24	9X3=27
7X4=28	8X4=32	9X4=36
7X5=35	8X5=40	9X5=45
7X6=42	8X6=48	9X6=54
7X7=49	8X7=56	9X7=63
7X8=56	8X8=64	9X8=72
7X9=63	8X9=72	9X9=81

4. HTML 控件

HTML 控件基本观念

HTML 控件的优点

在第二章里我们介绍了一些基本的 HTML 标注, 这些 HTML 标注在以往的静态网页或 ASP 动态网页里即可满足我们的需求。但是标准的 HTML 标注并没有办法利用程序直接来控制它们的属性、使用方法和接收事件, 程序设计师必须另外学习其它如 JavaScript 等程序语言才得以控制这些 HTML 标注。ASP.NET 为动态网页程序设计带来了许多新的技术, 这些技术其中之一就是将所有的 HTML 标注对象化, 让程序可以直接控制; 对象化之后的 HTML 标注我们称为 HTML 控件。我们可以使用如 VB.NET 或 C# 等语言来撰写控制 HTML 控件的程序, ASP.NET 把 HTML 标注对象化, 可以让网页对象的互动、程序的写作及维护变的更轻松容易, 也让执行的效率明显的改善不少。ASP.NET 将 HTML 标注对象化的好处, 我们观察以下程序便了解:

```
<!--传统的 HTML 静态网页写作方式-->

<Html>

<Body>

<A Href="http://www.microsoft.com">请按这里</A>
```

```
</Body>
```

```
</Html>
```

传统的 HTML 标注无法利用程序直接控制，这是因为 HTML 标注当初设计时并没有彻底对象化；

所以如果要动态的利用程序设定标注的属性，必需要插入 ASP 程序才可以，如下所示：

```
<!--为了动态的设定标注的属性，必需在标注中插入许多程序-->
```

```
<html>
```

```
<%
```

```
    strAddress="http://www.microsoft.com"
```

```
%>
```

```
<A href=<%=strAddress%>>请按这里</A>
```

```
</html>
```

ASP 网页设计师没有办法直接利用程控对象，所以必需在标注后面插入一些 ASP 程序代码。这

就是为什么以前的 ASP 程序代码非常杂乱，常常会看到标注中插入许多叙述的程序，这样会导

致程序代码在维护以及阅读上的困难。ASP.NET 为了解决这种杂乱无章的程序写作风格，便

将 HTML 标注对象化而产生出 HTML 控件。HTML 控件可以让程序直接控制并设定其属性，如

下范例所示：

```
<!--ASP.NET 的 HTML 控件可以利用程序直接控制-->
```

```
<html>
```

```
<A Id="Anchor1" Runat="Server">请按这里</A>
```

```
</html>
```

```
<Script Language="VB" Runat="Server">
```

```
Sub Page_Load()
```

```
    Anchor1.Href="http://www.microsoft.com"
```

```
End Sub
```

```
</Script>
```

HTML 控件比 HTML 标注多了 ID 以及 Runat 这两种属性。ID 属性表示程序是以本属性来控制对象的，所以任何对象的名称不可重复，不管它们是否为同一种类。而 Runat 属性表示这个对象是在 Server 端执行，所有的 HTML 控件都必须加上这个属性设定值；倘若该对象在程序执行时不需要被程控，则可以忽略 ID 属性的设定。我们知道网页在被加载时会先触发 Page_Load 事件，此时我们就可以利用这个事件进行对象的初值化，以及从数据库抓数据回来等工作，所以我们在 Page_Load 事件程序中利用程序指定超级链接控件 Anchor1 的 Href 属性。这样一来程序代码和 HTML 控件分开，程序的架构就不会显的杂乱无章而不好管理。了解 HTML 控件可以直接被程序所控制后，我们再了解 HTML 控件对事件的支持：

```
<html>
```

```
<Form Runat="Server">
```

```
<Button Id="Button1" Runat="Server" OnServerClick="Button1_Click">
```

```
改变字体</Button><BR>

</Form>

<Span Id="Sp1" Runat="Server">原来的字体</Span>

<Script Language="VB" Runat="Server">

    Sub Button1_Click(Sender As Object, e As EventArgs)

        Sp1.InnerHtml="<B>按下 Button1 后出现的字体</B>"

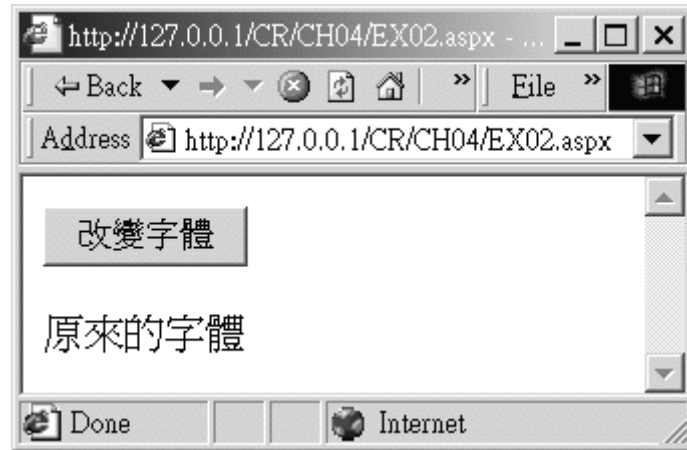
    End Sub

</Script>

</html>
```

上述程序代码中我们宣告了名为 **Button1** 的 **Button** 控件，除指定 **Runat** 属性值为 **Server** 外还指定了 **OnServerClick** 属性为 **Button1_Click**。**OnServerClick** 是 **Button** 对象所支持的事件，本事件在使用者按下按钮时便会触发。设定本属性表示发生 **OnServerClick** 事件时要执行哪个事件程序，我们将属性值填入 **Button1_Click** 则表示当使用者按下按钮时，便会执行 **Button1_Click** 这个事件程序。

Button1_Click 这个事件程序中宣告了对象型态的变量 **Sender** 及事件参数 **e**，分别表示是由哪个对象发出事件，以及发生事件时的相关信息；**每个事件程序中都要加入「Sender As Object, e As EventArgs」这两个参数宣告**。另外 **Button** 控件必需被 **<Form Runat="Server">** 以及 **</Form** 所包围，我们在介绍 **Form** 的时候会详细说明。程序的执行结果为：

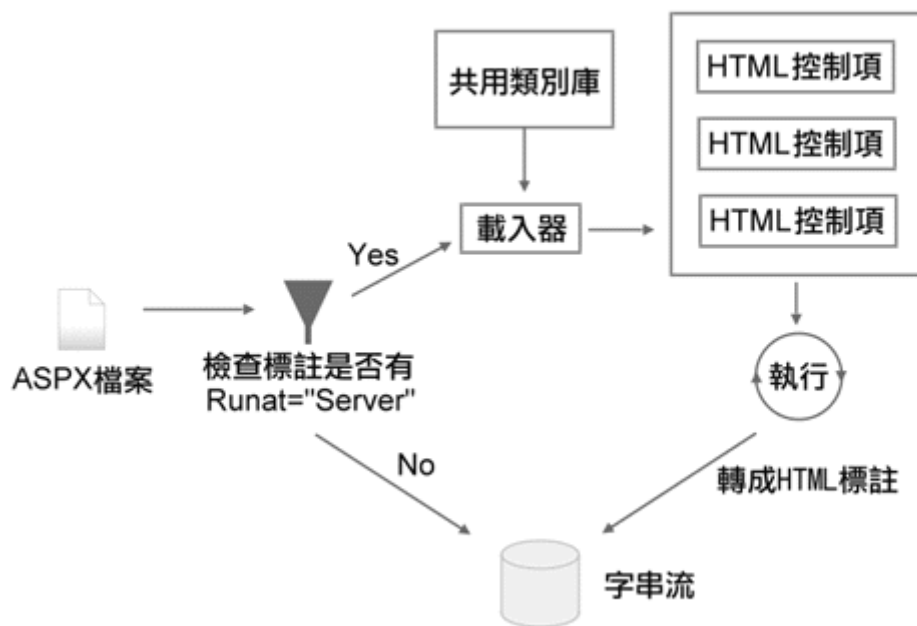


Span 控件最主要的功能是用来显示文字，设定 Span 控件的 InnerHtml 属性可以决定 Span 控件所显示的文字。由下面的执行结果我们看到了 Button1 按下后改变了 Sp1 的 InnerHtml 属性，所以 Sp1 所显示出来的文字变成"按下 Button1 后出现的字体"。InnerHtml 属性内容中若含有 HTML 标注，这个标注将会被解译并生效；所以按下按钮后不但文字内容变了，而且 标注也被解译并生效，所以出现的文字变成粗体。



HTML 控件架构

当 ASP.NET 网页执行时，会检查标注有无 **Runat** 属性。如果标注没有设定这个属性，那么该标注就会被视为字符串，并被送到字符串流等待送到客户端的浏览器进行解译。如果标注有设定 **Runat="Server"** 属性，那么就会依照该标注所对应的 HTML 控件来产生对象，所以 ASP.NET 对象的产生是由 **Runat** 属性值所决定的。当程序在执行时解析到有指定 **Runat="Server"** 属性的标注时，**Page** 对象会将该控件从 .NET 共享类别库加载并列入控制架构中，表示这个控件可以被程序所控制。等到程序执行完毕后再 将 HTML 控件的执行结果转换成 HTML 标注，然后送到字符串流和一般标注一起下载至客户端的浏览器进行解译。



了解 HTML 控件的动作原理后，下表列出指定 HTML 标注的 Runat 属性时，所对应的 HTML 控件：

HTML 标注	对应的 HTML 控件
<A>	HtmlAnchor
<Input>	HtmlInputButton, HtmlInputCheckBox, HtmlInputRadioButton, HtmlInputFile, HtmlInputHidden, HtmlInputImage, HtmlInputText
<Form>	HtmlForm
	HtmlImage

<Table>	HtmlTable
<Tr>	HtmlTableRow
<Td>	HtmlTableCell
<Select>	HtmlSelect

HTML 控件的使用和 **HTML** 标注使用的方法差不多，只要在使用的時候加上 **Id** 以及 **Runat** 这两个属性即可。我们可以选择下列两种风格来使用 **HTML** 控件：

```
<标注 Id=控件名称 Runat="Server" 属性 1="值" 属性 2...>所要显示的文字
</标注>

或

<标注 Id=控件名称 Runat="Server" 属性 1="值" 属性 2... />
```

以 **HtmlButton** 为例，第二种用法如果没有撰写所要显示的文字，则可以在网页被加载发生

Page_Load 事件时来设定初始值，如下范例所示：

```
<html>

<Form Runat="Server">

<Button Id="Button1" Runat="Server"/>

</Form>
```

```
<Script Language="VB" Runat="Server">

    Sub Page_Load(Sender As Object, e As EventArgs)

        Button1.InnerText="按钮一"

    End Sub

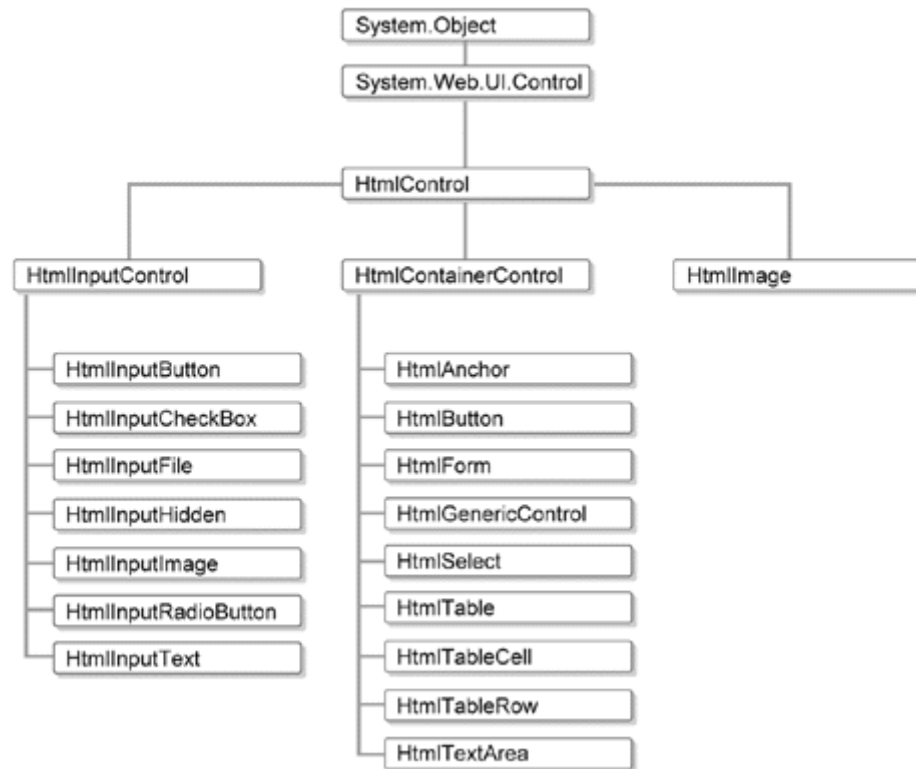
</Script>

</html>
```

若使用其它如 <DIV>、 或 没有显示于上述列表中的标注，ASP.NET 则以

HtmlGenericControl 类别来支持。以下为 HTML 控件的架构图：

System.Web.UI.HtmlControls



HTML 控件常用属性

了解 HTML 控件的原理及架构后，接下来我们先来介绍常在许多 HTML 控件中出现的属性。

InnerHtml 属性、InnerText 属性

InnerHtml 以及 InnerText 这两个属性主要是用来设定控件所要显示的文字。这两个控件的属性

假设都为「 试验 」，对于 InnerHtml 属性而言会将其中的 标注加以解译，所以

显示出粗体的文字;而对于 **InnerText** 属性而言不会将其中的 `` 标注加以解译,所以会将「``

试验 ``」一五一十的显示出来,如下列范例所示:

```
<Html>

<Form Runat="Server">

<Button Id="Button1" Runat="Server" OnServerClick="Button1_Click"/><P>

<Span Id="Sp1" Runat="Server"/> <br>

<Span Id="Sp2" Runat="Server"/>

</Form>

<Script Language="VB" Runat="Server">

Sub Page_Load(Sender As Object, e As EventArgs)

    Sp1.InnerHtml="InnerHtml 测试"

    Sp2.InnerText="InnerText 测试"

    Button1.InnerText="请按此处"

End Sub

Sub Button1_Click(Sender As Object, e As EventArgs)

    Sp1.InnerHtml="<b>测试</b>"

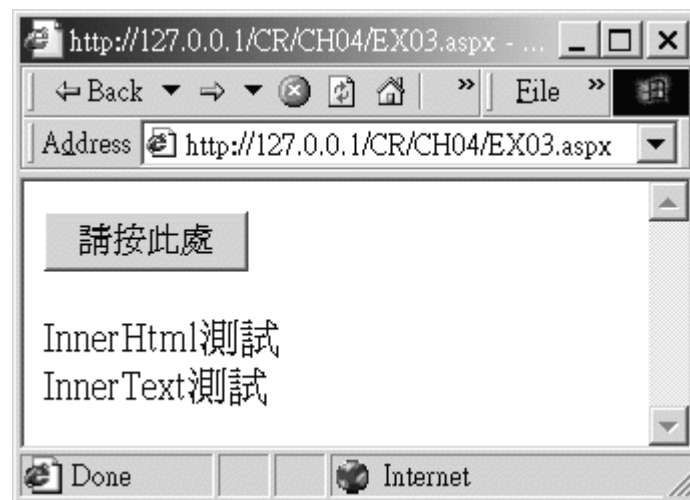
    Sp2.InnerText="<b>测试</b>"
```

```
End Sub
```

```
</Script>
```

```
</Html>
```

程序执行结果:



按钮按下后:



Disabled 属性

Disabled 属性我们称做禁能。禁能就是将一个对象的功能关闭，让对象暂时无法执行工作。所以如果将对象的 **Disabled** 属性设为 **True** 时，该对象会显示为灰色并且停止工作；然而若我们将 **Disabled** 属性设回 **False**，该控件即可正常工作。以 **Button** 对象为例，若该对象的 **Disabled** 属性被设定为 **True**，则按钮无法被按下，如下范例所示：

```
<Html>

<Button Id="Button1" Runat="Server">Disable 状态</Button><p>

<Button Id="Button2" Runat="Server">Enable 状态</Button>

<Script Language="VB" Runat="Server">

    Sub Page_Load(Sender As Object, E As EventArgs)

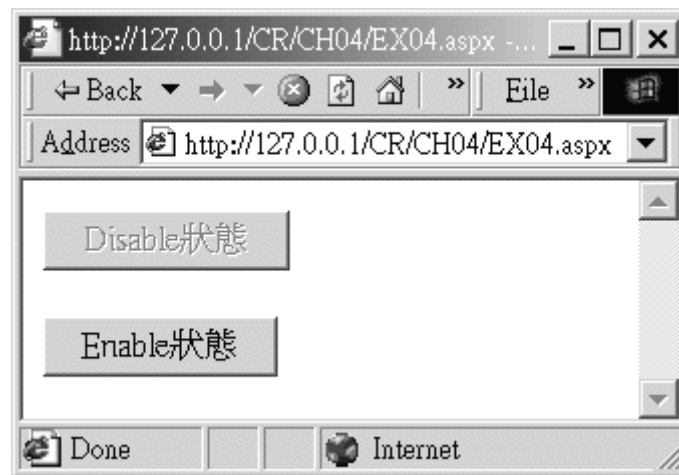
        Button1.Disabled=True

    End Sub
```

```
</Script>
```

```
</Html>
```

程序执行结果：



Visible 属性

Visible 属性可以让一个对象的视觉元素消失，换句话说就是将对象隐藏起来让使用者看不到。

下面范例在 **Page_Load** 事件中将名为 **Anchor1** 的超级链接控件隐藏起来，待使用者按下 **Button1**

按钮后再将其 **Visible** 属性设为 **True**：

```
<Html>
```



```
<A Id="Anchor1" Runat="Server" Href="http://127.0.0.1">出现的 Anchor 控件</A>
```

```
<Form Runat="Server">
```

```
    <Button Id="Button1" Runat="Server"
```

```
    OnServerClick="Button1_Click">Click!!
```

```
    </Button>
```

```
</Form>
```

```
<Script Language="VB" Runat="Server">
```

```
    Sub Page_Load(Sender As Object, e As EventArgs)
```

```
        Anchor1.Visible=False
```

```
    End Sub
```

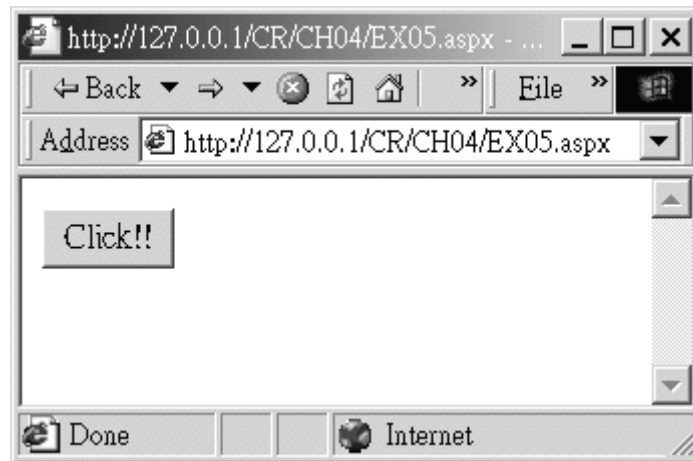
```
    Sub Button1_Click(Sender as Object, e As EventArgs)
```

```
        Anchor1.Visible=True
```

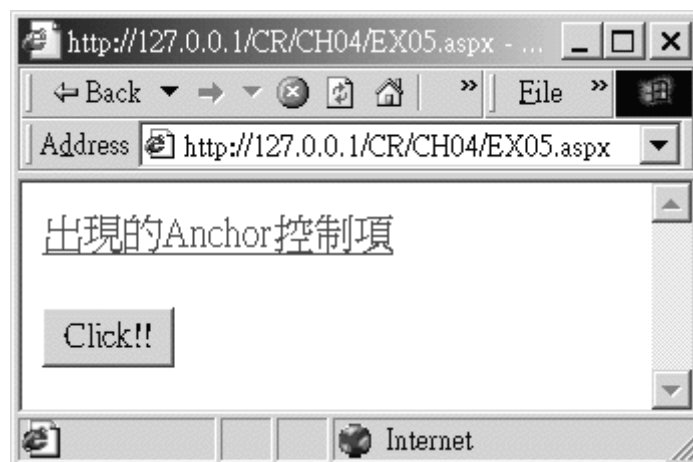
```
    End Sub
```

```
</SCRIPT>
```

```
</Html>
```



按下 Button1 后:



Attributes 属性

有两个方法可以指定对象的属性，第一种是我们前面常用的 **对象.属性**，而另外一种就是 **对**

象.Attributes("属性名称")。下列范例利用 Attributes 属性指定 Anchor1 的 Href 属性：

```
<Html>
```

```
<Script Language="VB" Runat="Server"> Sub Page_Load(Sender As Object, E  
As EventArgs) Anchor1.Attributes("Href")="http://msdn.microsoft.com"  
End Sub </SCRIPT> <A Id="Anchor1" Runat="Server">按这里</A> </Html>
```

Style 属性

本属性可以用来设定控件的样式。我们以 **Button** 控件为例子，标准 **Button** 控件的底色为灰色，而文字为黑色。对于只使用 **HTML** 标注来说，除非我们配合 **CSS**（**Cascading Style Sheet**，串接样式表。这是为了补强 **HTML** 的不足，由全球信息网联盟所提出的规格）的使用，否则无法更改按钮的颜色。为了让开发人员可以方便的设定对象样式，**ASP.NET** 便为控件设计了 **Style** 属性。下表列出 **Style** 属性可以设定的样式：

样式名称	说明	设定值
Background-Color	背景 色	RGB 值或指定颜色
Color	前景 色	RGB 值或指定颜色
Font-Family	字型	标楷体

Font-Size	字体 大小	20pt
Font-Style	斜体	Italic（斜体）或 Normal（一般）
Font-Weight	粗体	Bold（粗体）或 Normal（一般）
Text-Decoration	效果	Underline（底线）、Strikethrough（穿越线）、Overline（顶线） 或是 None（无）
Text-Transform	转大 小写	Uppercase（全转大写）、Lowercase（全转小写）、Initial Cap（前 缀大写）或是 None（无）

下列范例改变了 **Button** 控件的样式：

```
<Html>

<Button Id="Button1" Runat="Server">button 按钮</Button>

<Script Language="VB" Runat="Server">

    Sub Page_Load(Sender As Object, e As EventArgs)

        Button1.Style("Background-Color")="#FFFF00" ' 以 RGB 设定颜色

        Button1.Style("Color")="Blue"

        Button1.Style("Font-Family")="标楷体" ' 设定字型

        Button1.Style("Font-Size")="20pt" ' 设定字体大小

        Button1.Style("Font-Style")="italic" ' 设定为斜体字
```

```
Button1.Style("Font-Weight")="bold" ' 设定为粗体字

Button1.Style("Text-Decoration")="Underline" ' 设定为底线字

Button1.Style("Text-Transform")="UpperCase" ' 小写转大写

End Sub

</Script>

</Html>
```



基础 HTML 控件

了解 HTML 控件的原理及一些基本的属性后，接下来我们就来介绍一些基础 HTML 控件的实际应用。

HtmlAnchor 控件

HtmlAnchor 控件可以用来指定超级链接，其使用语法为：

```
<A
    Id="被程序代码所控制的名称"
    Runat="Server"
    Href="欲连结的 URL 地址"
    Name="欲前往的书签名称"
    OnServerClick="事件程序名"
    Target="要将连结开启至哪个框架"
    Title="小提示"
>
超级链接文字
</A>
```

这个控件有如下的属性：

属性	说明	设定值
Href	欲连结的 WWW、E-Mail、Ftp 服务器、Gopher 服务器、News 服务器、或是 Telnet 服务器的地址。	URL 地址

Target	如果有设定网页框架，则可将欲开启的连结开启至指定的框架	_blank(无)、_self(自己)、_parent(父框架) 及 _top(上层)
Title	本属性可以决定当鼠标移到连结文字上时会出现小提示	字符串
Name	书签名称	书签名称

使用范例：

下面范例当使用者将鼠标移至「这是超级链接」时，会应显示文字「小提示」；按下「这是超级链接」时，会将网页重新导向微软的网站：

```
<Html>

<A Id="Anchor1" Runat="Server">这是超级链接</A>

<Script Language="VB" Runat="Server">

    Sub Page_Load(Sender As Object, e As EventArgs)

        Anchor1.Href="http://www.microsoft.com"

        Anchor1.Target="_blank"

        Anchor1.Title="小提示"

    End Sub

</Script>
```

</Html>



HtmlImage 控件

HtmlImage 控件对应于 HTML 元素中的 元素，是用来显示图片于网页上的控件。它的使用方法和 HTML 的 标注很类似，只是在 ASP.NET 里变为一个可以随程序来动态改变其属性的 HTML 控件。其使用语法为：

<img

Id="被程序代码所控制的名称"

Runat="Server"

Alt="无法显示图形时所显示的小提示"

Align="Top | Middle | Bottom | Left | Right"


```
Border="边框宽度"

Height="影像高度"

Src="影像所在的地址"

Width="影像宽度"

>
```

其中语法中有「|」符号表示或的意思。下表为 **HtmlImage** 控件的常用属性：

属性	说明	设定值
Src	图形文件的地址	URL 或檔名
Width	图片宽	像素
Height	图片高	像素
Border	图片显示时的外框大小	数值
Align	设定和图片旁边文字的排列方式	top、middle、bottom、left、right
Alt	当图片无法下载时显示的文字或当鼠标光标移至图片上时显示小提示内的文字	字符串

使用范例：

下面的程序利用 **Page_Load** 事件来设定 **Image1** 对象的属性:

```
<Html>

<Img Id="Image1" Runat="Server"/>

<Script Language="VB" Runat="Server">

    Sub Page_Load(Sender As Object, e As EventArgs)

        Image1.Src="Board.jpg"

        Image1.Alt="This is a picture"

        Image1.Width="512"

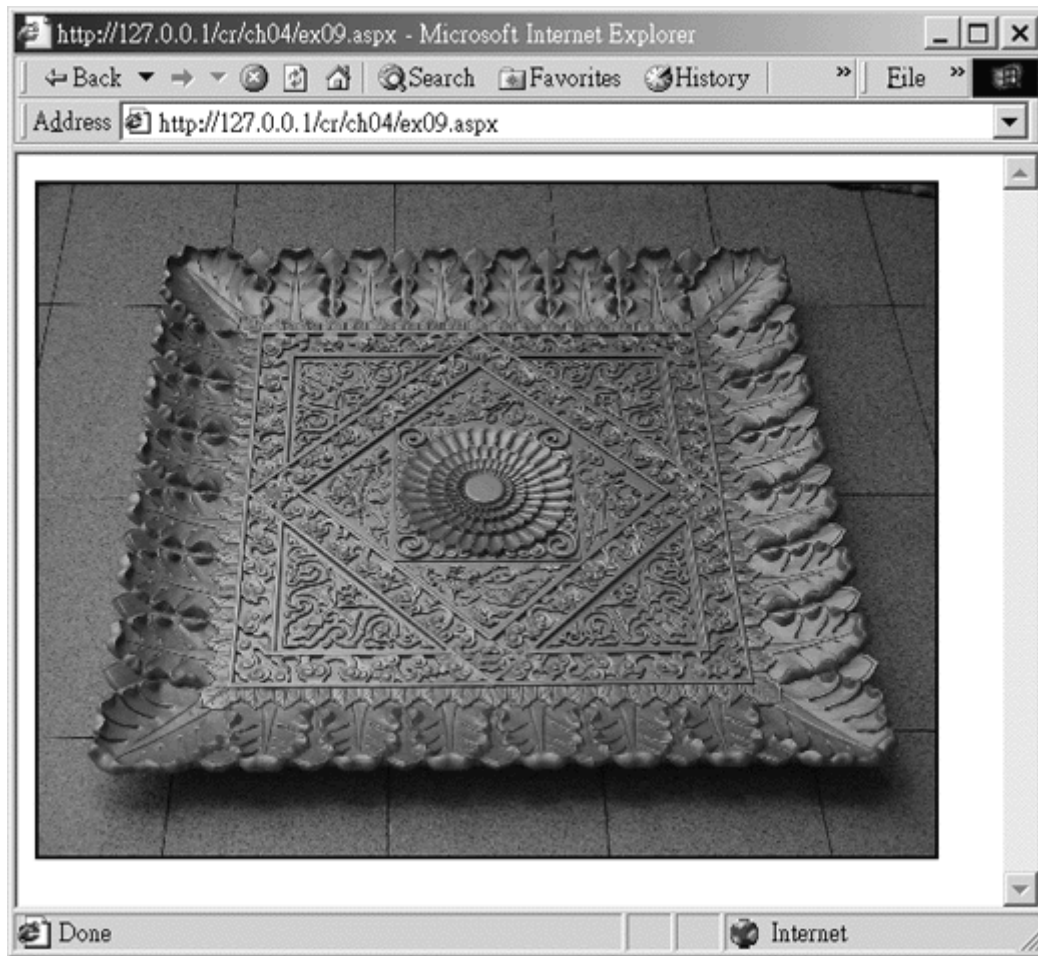
        Image1.Height="384"

        Image1.Border="2"

    End Sub

</Script>

</Html>
```



HtmlButton 控件

HtmlButton 控件最主要是让使用者透过按钮执行命令或动作，所以最重要的就是 OnServerClick 事件 OnServerClick 事件当使用者按下按钮时便会触发。要指定发生 OnServerClick 事件时所要执行的程序，设定 OnServerClick 属性即可。例如指定 OnServerClick="Button1_Click" 时，即表示使用者按下按钮触发事件时，会呼叫 Button1_Click 这个事件程序，我们就可以在 Button1_Click 这个事件程序内撰写我们所要执行的程序代码。另外 <Button> 控件必须写在窗体控件 <Form Runat="Server"></Form> 之内，这是因为 Button 控件可以决定数据的上传，

而只有被 **<Form>** 控件所包围起来的数据输入控件，其数据才会被上传。关于这个部分我们后面会详细说明，其使用语法为：

```
<Button  
  
    Id="被程序代码所控制的名称"  
  
    Runat="Server"  
  
    OnServerClick="事件程序名"  
  
>  
  
按钮上的文字、图形或控件  
  
</Button>
```

使用范例：

下面的范例码中 **Button1** 开始时并没有显示任何文字，但是当按下 **Button1** 时因为触发了

Button1_Click 事件，使得 **Button1** 内的文字、背景色都发生了变化：

```
<Html>  
  
<Form Runat="Server">  
  
<Button Id="Button1" Runat="Server" InnerText="请按这里" OnServerClick  
="Button1_Click"/>  
  
</Form>
```

```
<Script Language="VB" Runat="Server">

    Sub Button1_Click(Sender As Object, e As EventArgs)

        Button1.Style("Background-Color")="Blue"

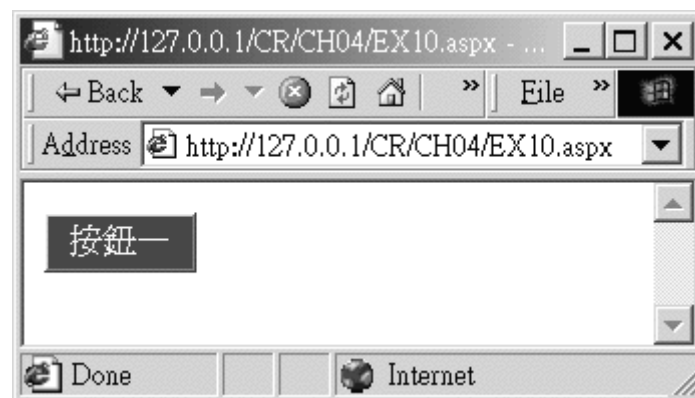
        Button1.Style("Color")="Yellow"

        Button1.InnerText="按钮一"

    End Sub

</Script>

</Html>
```



按下按钮后程序执行的结果

数据的输入

还记得我们第三章教大家上传数据的方法：

网址?参数名称=值

第三章所使用的这种用法只不过先配合程序范例来输入数据，实际不可能让使用者这样输入。要

让使用者输入数据，还是要用如按钮、文字输入盒以及选单等比较友善的视觉接口，所以接下来

我们就要介绍这些数据输入的控件。

HtmlForm 控件

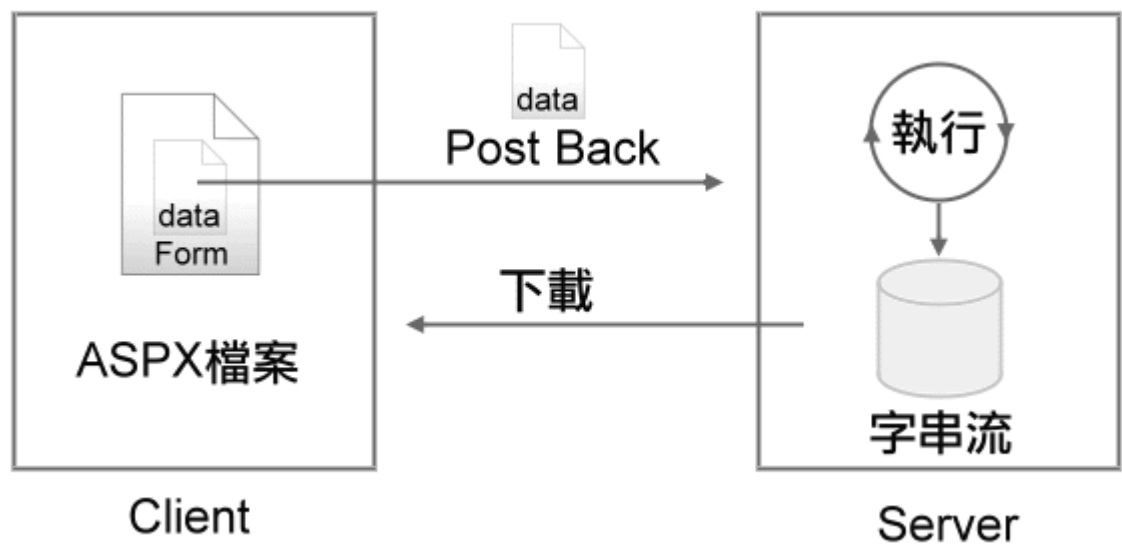
HtmlForm（窗体）控件是设计动态网页一个相当重要的组件，它可以让我们将 Client 端的数据

传送至 Server 端作处理。在窗体内的确认按钮被按下去后，只要被 Form 控件所包起来的数据

输入控件都会被一并送到 Server 端，这个动作称为回贴（Post Back）。这时 Server 端收到这

些数据及 OnServerClick 事件后会执行指定的事件程序，并且将执行结果重新下载到 Client 端浏

览器，如下图所示：



其使用语法为：

```
<Form  
    Id="被程序代码所控制的名称"  
    Runat="Server"  
    Method="Post | Get"  
    Action="要执行程序的地址"  
>  
其它控件  
</Form>
```

HtmlForm 控件有两个主要的属性：

属性	内容	设定值
Method	传递数据的方法	Post（Server 抓取资料）或 Get（上传）
Action	呼叫的网址	URL 或檔名

如果 **Method** 属性为 **Post**（默认值）则表示由 **Server** 端来抓取资料，如为 **Get** 则表示由浏览器主动上传资料至 **Server** 端。其中的差别为 **Get** 是立即传送，其执行效率较快，不过所传送的数据不能太大；而 **Post** 则表示等待 **Server** 来抓取数据，数据的传送虽然不是那么立即，不过可传送的数据量则没什么限制。而 **Action** 属性则表示数据要送至哪个网址，预设是自己。假设 **Button** 触发 **OnServerClick** 事件时所呼叫的程序是 **Button1_Click** 事件程序，所以如果触发 **OnServerClick** 事件时，预设会呼叫自己所在网页的 **Button1_Click** 事件程序。有一点要特别注意，那就是 **窗体中的按钮被按下时，会呼叫原网页中的 **Button1_Click** 事件，此时原网页会重新被加载，所以先触发 **Page_Load** 事件后才会再触发 **Button1_Click** 事件。**

使用范例：

下列范例在 **Form** 控件中配置了一个 **Button** 对象，并指定按下按钮时所要呼叫的事件程序为

Button_Click 事件：

```
<Html>
```

```
<Form Runat="Server"><!--其 Method 属性预设为 Post, Action 预设是自己-->
```



```
<Button Id="Button1" Runat="Server" OnServerClick="Button1_Click"
InnerText="请按这里"/>

</Form>

<Script Language="VB" Runat="Server">

    Sub Page_Load(Sender As Object , e As EventArgs)

        Response.Write("这是 Page_Load 事件<br>")

    End Sub

    Sub Button1_Click(Sender As Object, e As EventArgs)

        Button1.Style("BackGround-Color")="Blue"

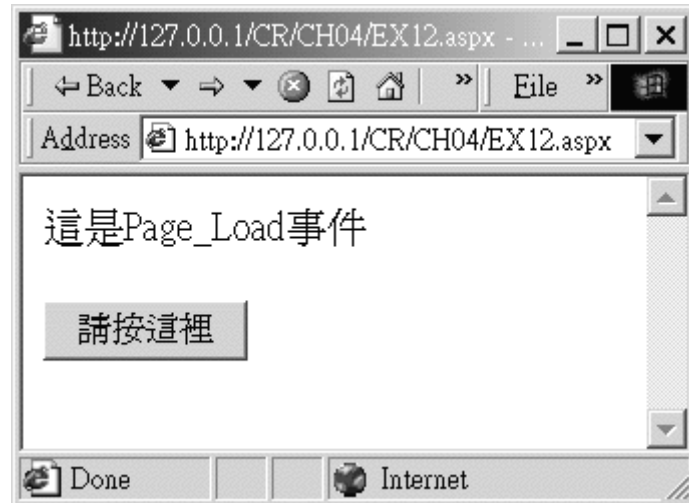
        Response.Write("这是 Button1_Click 事件")

    End Sub

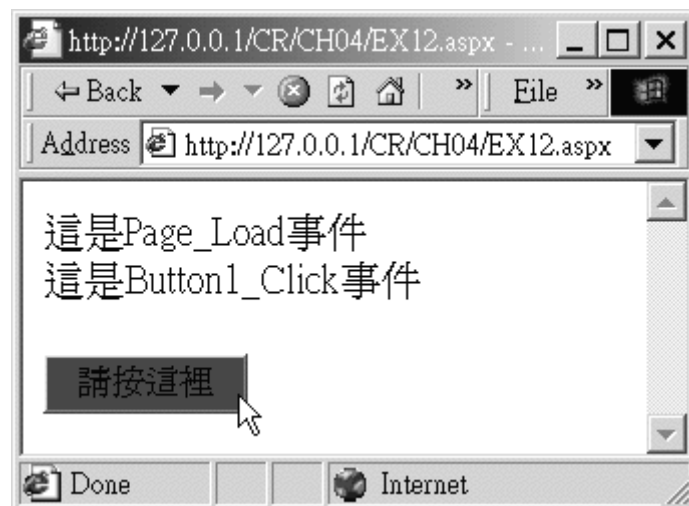
</Script>

</Html>
```

上述程序执行的结果，先触发 **Page_Load** 事件，如下所示：



使用者按下按钮后:



观察上面的执行结果，我们看到 **Page_Load** 事件先发生后才触发 **Button1_Click** 事件。倘若

Action 所指到的地址是其它的网页或档案，则呼叫其它网页；此时 **原网页的 Button1_Click 事**

件程序便不触发，直接会将所指定的网页加载。另外一个网页中 **只允许有一个窗体控件** 存在。

HtmlInput 控件

要让使用者输入数据，则使用 **HtmlInput** 控件。而要让程序抓到这些数据，则必需利用 **HtmlForm**

控件将这个输入控件包围起来。**HtmlInput** 控件会因为 **Type** 属性的设定而产生不同种类的控件：

輸入種類	Type 屬性值	常用的屬性及事件	視覺元素
按鈕	Button	事件：OnServerClick	
確定按鈕	Submit	屬性：Value（按鈕所顯示的文字）	
重置按鈕	Reset		
文字輸入盒	Text	屬性：Value（輸入盒內所顯示的文字） MaxLength（可接受的文字長度） Size（輸入盒的大小）	
密碼輸入盒	Password		
檢核盒	CheckBox	屬性：Checked（是否有被核取，布林） Value（檢核盒的相關資料）	
收音機按鈕	Radio	屬性：Checked（是否有被核取，布林） Value（收音機按鈕的相關資料）	
隱藏欄位	Hidden	屬性：Value（隱藏欄位的内容）	無

上述的种类都是使用者输入数据的基本元素，另外要注意 **HtmlInput 控件不需要相对应的结束**

结构，也就是只需要撰写 `<Input 属性="设定值">` 即可，不需要相对应的 `</Input>` 或是以

`<Input 属性="设定值"/>` 的方式来写作。以下我们来对各个输入的种类说明。

HtmlInputButton 按钮控件

按钮最主要的功用为执行一个指令或动作。对于窗体来说是将填好的数据传送出去。它的 **Type**

属性有三种型态：当为 **Submit** 时是传送数据，等于 **Button** 时可以用来触发事件程序，而 **Reset**

是用来重置窗体成为初始状态；指定 **Type="Reset"** 时，并不需要指定任何程序代码就可以重设

窗体内的输入控件。在 **ASP.NET** 里我们大多使用 **Type=Button**，因为我们可以利用

OnClick 事件程序中撰写我们所要执行的程序代码。其使用语法为：

```
<Input  
  
    Id="被程序代码所控制的名称"  
  
    Runat="Server"  
  
    Type="Button | Submit | Reset"  
  
    OnClick="事件程序名"  
  
>
```

使用范例：

下列范例在 **Form** 控件中配置了三种型态的 **HTMLInputButton** 控件，并指定按下按钮时所要呼

叫的事件程序。当按下 **Button** 或 **Submit** 时，会显示按了哪个按钮；而 **Reset** 按钮可以将文字

输入盒内的文字重设回原值：

```
<Html>  
  
<Form Runat="Server">  
  
    <Input Type="Text" Id="Text1" Runat="Server" Value="这是文字输入盒">
```

```
<Input Type="Button" Id="Button1" Runat="Server"
OnServerClick="Button1_Click" Value="这是
按钮">

<Input Type="Submit" Id="Submit1" Runat="Server"
OnServerClick="Submit1_Click" Value="这是
确定">

<Input Type="Reset" Runat="Server" Value="这是重置">

</Form>

<Span ID="Sp1" Runat="Server"/>

<Script Language="VB" Runat="Server">

    Sub Button1_Click(Sender As Object, e As EventArgs)

        Sp1.InnerText="您按了 Button"

    End Sub

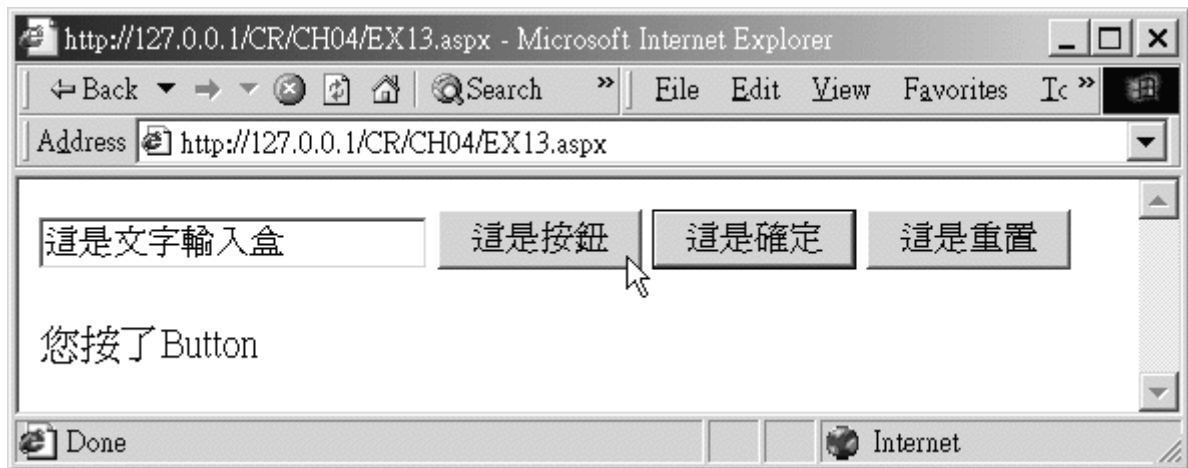
    Sub Submit1_Click(Sender As Object, e As EventArgs)

        Sp1.InnerText="您按了 Submit"

    End Sub

</Script>

</Html>
```



HtmlInputText 文字輸入盒

文字輸入盒就是讓使用者輸入數據的地方。它有两种型态：当为 **Text** 时是输入的一般数据，所输入的字符串会显示在文字輸入盒内；而 **Password** 是密码输入的文字輸入盒，输入的字符会以「*」来显示。其使用语法为：

```
<Input
```

Id="被程序代码所控制的名称"

Runat="Server"

Type="Text | Password"

MaxLength="可接受的字符串长度"

Size="文字输入盒的宽度"

Value="显示在文字输入盒的默认值"

>

使用范例：

下列程序代码利用文字输入盒取得使用者的身分验证信息，使用者可以按下 **Button** 或是 **Submit** 来确定资料的输入，**Reset** 则可以重设文字输入盒的内容：

```
<Html>

<Form Runat="Server">

姓名： <Input Type="Text" Id="Text1" Runat="Server"><br>

密码： <Input Type="Password" Id="Text2" Runat="Server"><br>

    <Input Type="Button" Id="Button1" Runat="Server"
OnServerClick="Button1_Click"                                Value="执行
程序">

    <Input Type="Submit" Id="Submit1" Runat="Server"
OnServerClick="Submit1_Click"                                Value="确定
">

    <Input Type="Reset" Runat="Server" Value="重置">
```

```
</Form>

<Script Language="VB" Runat="Server">

    Sub Button1_Click(Sender As Object, e As EventArgs)

        IDPWDchk()

    End Sub

    Sub Submit1_Click(Sender As Object, e As EventArgs)

        IDPWDchk()

    End Sub

    Sub IDPWDchk()

        If Text1.Value="Charles" And Text2.Value="Pass" Then

            Response.Write("使用者名称及密码正确, 你好!")

        Else

            Response.Write("使用者名称及密码错误, 请重新输入!")

            Text1.Value=""

            Text2.Value=""

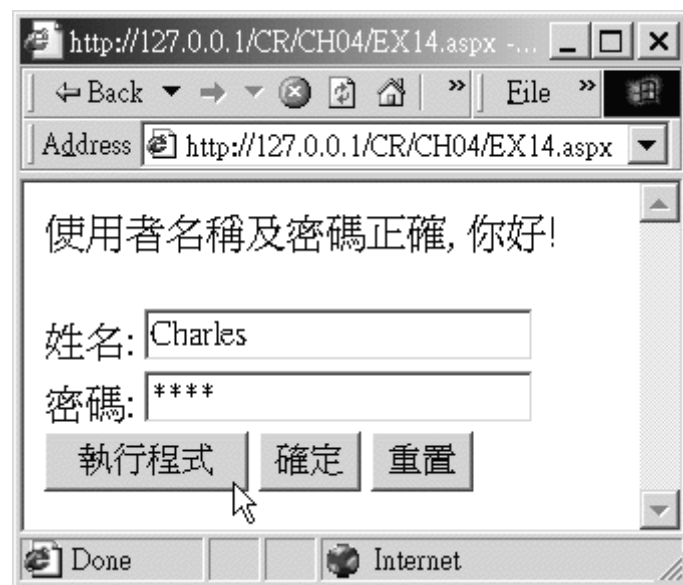
        End If

    End Sub

</Script>

</Html>
```


使用者在文字输入盒中所输入的数据会被存在 **Value** 属性里面，使用者输入完数据后，按下 **Button** 或是 **Submit** 则会触发相对应的 **OnServerClick** 事件程序。我们在事件程序中呼叫了检查使用者名称及密码是否正确的子程序 **IDPWDchk()**，如果使用者输入正确的使用者名称及密码，则会出现输入正确的讯息，如下图所示：



倘若输入错误的使用者名称或密码，则会显示输入错误，并将使用者所输入的使用者名称及密码清除，如下图所示：



HtmlInputRadio 收音机按钮控件

当我们要限制使用者的选择为单选，并只能够在我们所提供的项目中选择一个答案时，可以使用 **HtmlInputRadio**。以输入使用者性别数据为例，我们提供使用者「男」及「女」的选项让使用者选择，利用收音机按钮可以限制他只能选择一个答案。其中 **HtmlInputRadio** 控件最重要的属性为 **Name** 属性，用来设定收音机按钮的群组。收音机按钮有一个规则，那就是同一个群组的收音机按钮同一时间内只能有一个按钮被选择，所以它可以用在单选的选项上。此时被选取的收音机按钮其 **Checked** 属性则为 **True**，没有被选取则为 **False**。其使用语法为：

```
<Input
```

```
    Id="被程序代码所控制的名称"
```

```
    Runat="Server"
```

```
    Type="Radio"
```

```
Checked="True | False"
```

```
Name="收音机按钮所属群组"
```

```
>
```

使用范例：

下列程序代码利用收音机按钮取得使用者的性别信息，使用者可以选择「男」或是选择「女」，

按下按钮后可以显示使用者所选择的内容：

```
<Html>
```

```
<Form Runat="Server">
```

```
<Input Type="Radio" Id="Radio1" Name="G1" Runat="Server"
```

```
Checked="True">男<br>
```

```
<Input Type="Radio" Id="Radio2" Name="G1" Runat="Server">女<br>
```

```
<Input Type="Button" ID="Button1" Runat="Server"
```

```
OnServerClick="Button1_Click" Value="确定">
```

```
</Form>
```

```
<Span ID="Sp1" Runat="Server"/>
```

```
<Script Language="VB" Runat="Server">
```

```
Sub Button1_Click(Sender As Object, e As EventArgs)

Dim strMsg As String="您的性别为: "

IF Radio1.Checked=True Then

    strMsg+="男"

Else

    strMsg+="女"

End If

Sp1.InnerText=strMsg

End Sub

</Script>

</Html>
```



HtmlInputCheckBox 检核盒控件

当我们要让使用者可以复选多个项目，不过只能够在我们所提供的项目中选择答案时，可以使用 `HtmlInputCheckBox`。以输入使用者专长数据为例，我们提供使用者多种专长的选项让使用者选择，利用检核盒控件可以提供多种选项让使用者选取。如果检核盒被使用者选取，其 `Checked` 属性则为 `True`，没有被选取则为 `False`。其使用语法为：

```
<Input  
    Id="被程序代码所控制的名称"  
    Runat="Server"  
    Type="CheckBox"  
    Checked="True | False"  
>
```

使用范例：

下列程序代码利用利用检核盒控件取得使用者的专长信息，使用者可以从多个选项中复选，按下按钮后可以显示使用者所选择的内容：

```
<Html>  
  
<Form Runat="Server">
```

请选择您的专长:

<Input Type="CheckBox" Id="Check1" Runat="Server">ASP.Net

<Input Type="CheckBox" Id="Check2" Runat="Server">C#

<Input Type="CheckBox" Id="Check3" Runat="Server">MS SQL Server

<Input Type="CheckBox" Id="Check4" Runat="Server">VB.Net

<Input Type="Button" ID="Button1" Runat="Server"

OnServerClick="Button1_Click" Value="确定">

</Form>

<Script Language="VB" Runat="Server">

Sub Button1_Click(Sender As Object, e As EventArgs)

Dim strMsg As String="您的专长为: "

If Check1.Checked Then strMsg+="ASP.Net "

If Check2.Checked Then strMsg+="C# "

If Check3.Checked Then strMsg+="MS SQL Server "

If Check4.Checked Then strMsg+="VB.Net"

Sp1.InnerText=strMsg

End Sub

</Script>

</Html>



HTMLInputHidden 隐藏输入控件

当我们要在使用者传送所输入的数据时，顺便传送不需要使用者输入的数据时，可以使用隐藏输入控件。其使用语法为：

<Input

Id="被程序代码所控制的名称"

Runat="Server"

```
        Type="Hidden"

        Value="所要传送的数据"

    >
```

使用范例：

下列程序可以记载使用者开始填写表格的时间：

```
<Html>

<Form Runat="Server" ID=Form1></head>

请输入您的住址：

    <Input Type="Text" Id="Text1" Runat="Server">

    <Input Type="Hidden" Id=Hidden1 Runat="Server">

    <Input Type="button" ID="Button1" Runat="Server"

OnServerClick="Button1_Click" Value="确定">

</FORM>

<Span ID="Sp1" Runat="Server"/>

<Script Language="VB" Runat="Server" ID=Script1>

Sub Page_Load(Sender As Object, e As EventArgs)
```



```
        If Page.IsPostBack=False then

            Hidden1.Value=CStr(DateTime.Now())

        End If

    End Sub

Sub Button1_Click(Sender As Object, e As EventArgs)

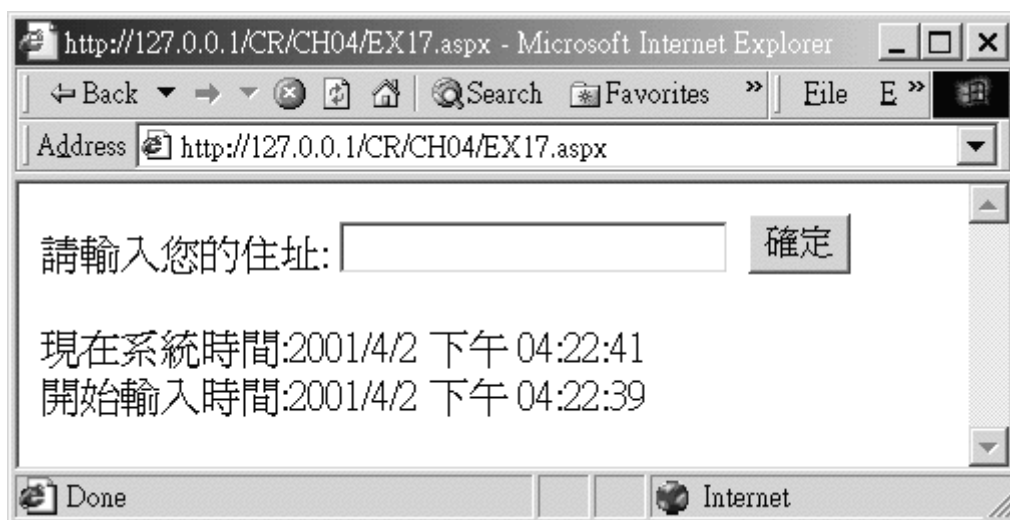
    Sp1.InnerHtml="现在系统时间:" & CStr(DateTime.Now()) & _

        "<br>开始输入时间:" & CStr(Hidden1.Value)

End Sub

</SCRIPT>

</Html>
```



若使用者是第一次浏览这个网页，那么网页第一次被加载的时间会存入 **Hidden1** 这个隐藏字段的 **Value** 属性中。要得知网页是否为第一次加载，可以使用 **Page** 对象的 **IsPostBack** 属性。如网页是第一次加载，那么 **IsPostBack** 属性为 **False**；若使用者是因为触发 **OnServerClick** 事件而让网页重新加载执行，那么 **IsPostBack** 属性则为 **True**。这样一来我们就可以利用 **IsPostBack** 属性来保留最初网页被加载时的时间。

HtmlTextArea 文字输入区控件

HtmlTextArea 控件的功能和前述的 **Text** 对象类似，只是 **HtmlTextArea** 控件可以多设定长度和高度，可以用来输入一小段文字；网站上讨论区的内容都是利用 **HtmlTextArea** 输入的，使用者输入的内容会存在 **Value** 属性中。**HtmlTextArea** 的写法和 **Input** 对象不同，必须要加上 **</TextArea>** 结束结构或以 **<TextArea .../>** 的风格来撰写，以下是 **TextArea** 的使用语法：

```
<TextArea  
  
    Id="被程序代码所控制的名称"  
  
    Runat="Server"  
  
    Cols="单行的长度"  
  
    Rows="文字输入区的列数"  
  
>  
  
文字区内容
```

```
</TextArea>
```

使用范例：

下列范例利用 **HtmlTextArea** 控件输入使用者的建议，然后使用 **Span** 控件显示出来。以下为范

例码：

```
<Html>

<Form Runat="Server" ID=Form1></head>

请输入你的建议:<br>

    <TextArea Id="TextArea1" Runat="Server" Cols="20" Rows="4">文字输入区

</TextArea>

    <Input Type="button" ID="Button1" Runat="Server"
OnServerClick="Button1_Click"                                Value="确定"
">

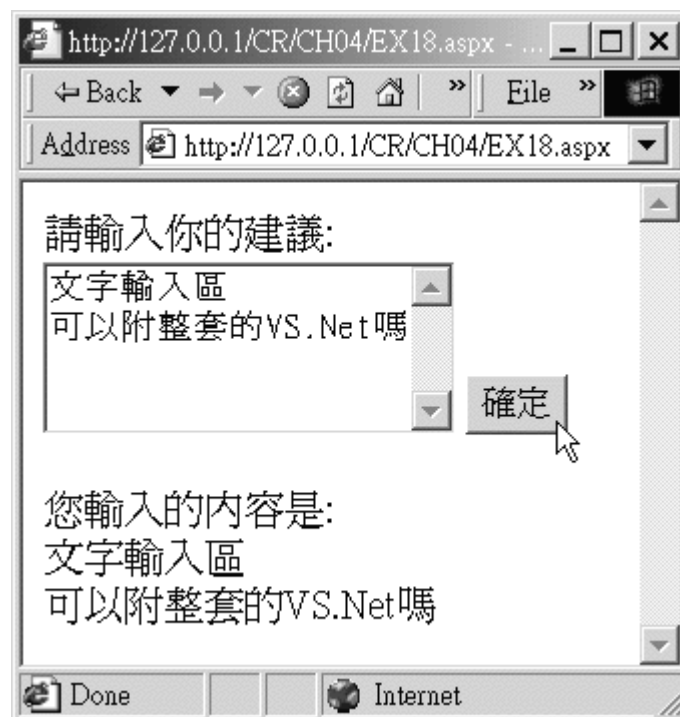
</FORM>

<Span ID="Sp1" Runat="Server"/>

<Script Language="VB" Runat="Server" ID=Script1>

Sub Button1_Click(Sender As Object, e As EventArgs)
```

```
Sp1.InnerHtml="您輸入的內容是:<br>" & _  
        Replace(TextArea1.Value, Chr(13), "<br>")  
  
End Sub  
  
</SCRIPT>  
  
</Html>
```



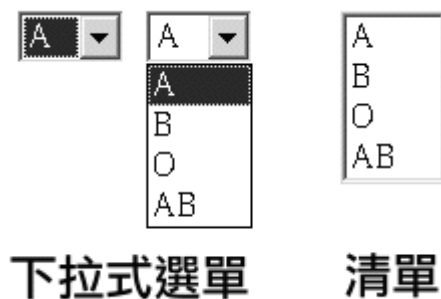
由于 Chr(13) 换行字符在 HTML 中没有作用，所以我们利用 **Replace** 函式将换行字符转换成

**
** 标注，其语法如下：

```
Replace("字符串", "原数据中的字符串或字符", "取代的字符串或字符")
```

HtmlSelect 选项控件

HtmlSelect 控件就是选单。选单控件有两种风格，一种是下拉式选单，另一种是清单：



要决定选项的风格是选单还是清单，由 **Size** 属性决定。倘若有指定 **Size** 属性，则出现固定大小之清单；若没有指定 **Size** 属性，则为下拉式选单。另外选项可以动态的加入项目，只要利用 **Items** 集合的 **Add** 方法即可；如果要取得使用者选择哪个项目，可以使用 **Value** 属性传回。以下为语法：

```
<Select  
    Id="被程序代码所控制的名称"  
    Runat="Server"  
    Items="选项集合"  
    Size="选单长度"
```

```
>  
  
<Option>选项</Option>  
  
<Option>选项...</Option>  
  
</Select>
```

使用范例：

以下范例配置了两个选项控件，分别为 **Select1** 以及 **Select2**。**Select1** 利用指定 **Option** 标注的方式将项目配置好，注意 **Select1** 的结束结构 **</Select>**；而 **Select2** 则是利用 **Items** 集合的 **Add** 方法在 **Page_Load** 事件中动态的加入项目。为了不让项目重复被加入，所以我们检查 **Page** 对象的 **IsPostBack** 属性是否为 **False**，是第一次加载才加入项目；这样才不会加入重复的选项。

以下为范例码：

```
<Html>  
  
<Form Runat="Server" ID=Form1></head>  
  
  血型:<Select ID="Select1" Runat="Server">  
  
    <Option>A</Option>  
  
    <Option>B</Option>  
  
    <Option>O</Option>  
  
    <Option>AB</Option>
```

```

</Select>

性别:<Select ID="Select2" Runat="Server" Size="2"/>

<Input Type="button" ID="Button1" Runat="Server"
OnServerClick="Button1_Click"                               Value="确定
">

</Form>

<Span Id="Sp1" Runat="Server"/>

<Script Language="VB" Runat="Server" ID=Script1>

    Sub Page_Load(Sender As Object, e As EventArgs)

        If Page.IsPostBack=False then

            Select2.Items.Add("男")

            Select2.Items.Add("女")

        End If

    End Sub

    Sub Button1_Click(Sender As Object, e As EventArgs)

        Sp1.InnerText= "您的血型是: " & Select1.Value & _

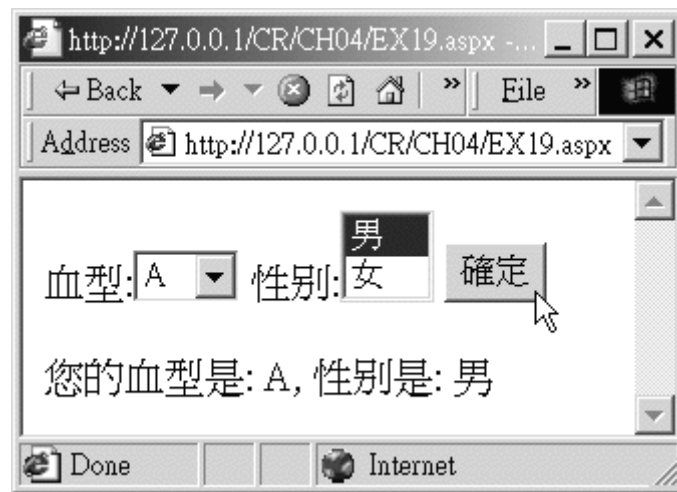
            ", 性别是: " & Select2.Value

    End Sub

```

```
</SCRIPT>
```

```
</Html>
```



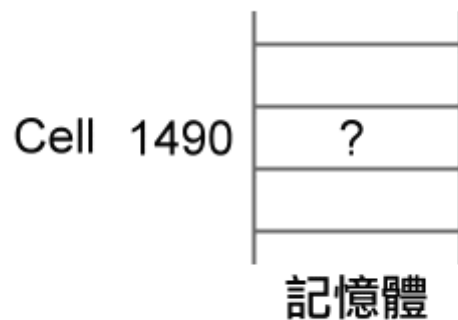
对象、变量与集合

对象变量

所谓对象变量,指的是变量里面所存放的数据是某个对象存放于内存的哪个地方。我们举个例子:

```
Dim Cell As HtmlTableCell
```

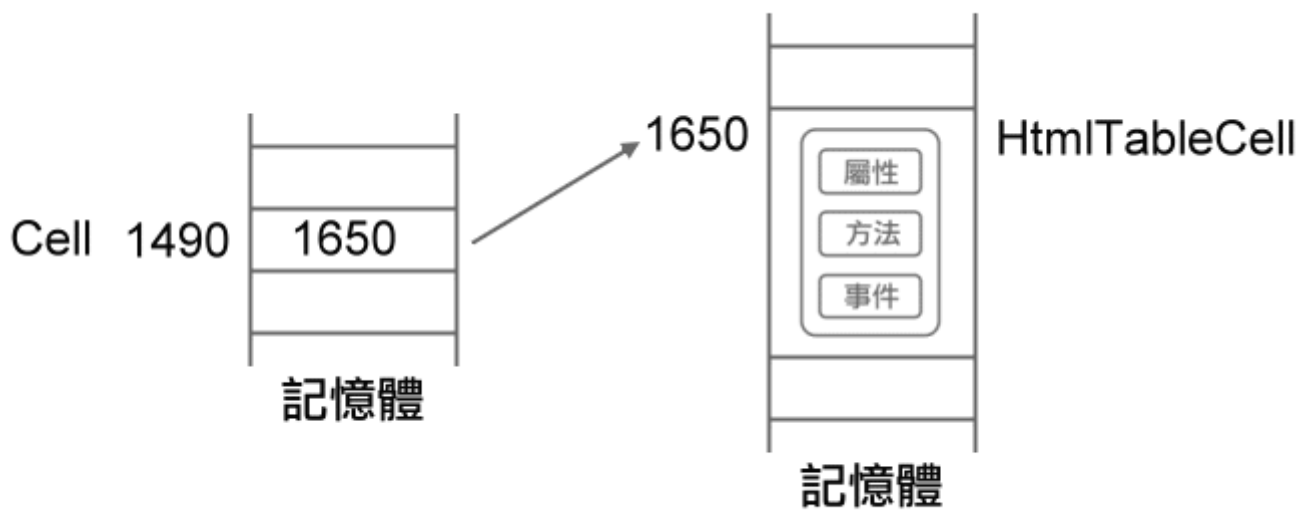

上述叙述宣告了一个 **Cell** 变量，里面可以存放 **HtmlTableCell** 控件所在的内存地址。所以我们执行这个叙述时，内存会为我们准备好一个空间，用来等待存放 **HtmlTableCell** 控件的内存地址：



这时候 **Cell** 变量内的内容是空的，因为我们还没有指定一个对象的内存地址给这个变量。要产生一个对象，要使用 **New** 运算符。**New** 运算符会依照后面所指定的对象类别来产生对象，例如：

```
Cell=New HtmlTableCell
```

这个叙述执行完后，会在内存中产生一个 **HtmlTableCell** 对象，然后将对象所在的内存地址传给对象变量 **Cell**：



将新产生对象的内存地址传给变量时，就是宣告要以这个变量名为这个对象的 ID 属性，所以我们可以 在程序代码中以指定变量名称的方式控制对象。这就是我们说的对象的参考，所以我们就 可以利用下列的方式来控制 `HtmlTableCell` 对象：

```
Cell. 属性="Value"      ' 设定对象的属性
```

```
变数=Cell. 属性  ' 取出对象的属性
```

```
Cell. 方法()      ' 执行对象的方法
```

所以我们要使用指到一个对象型态的变量时，倘若没有将这个变量指到实际产生出来的对象，

那么便不能使用这个对象的属性、方法以及事件。另外以下这两行叙述：

```
Dim Cell As HtmlTableCell
```

```
Cell=New HtmlTableCell
```

可以合并成一行：

```
Dim Cell As HtmlTableCell = New HtmlTableCell
```

Collection 集合对象

集合对象可以被我们用来管理许多对象。它可以取得对象的地址，并将对象有条理的收纳管理，

让我们以一个集合对象就可以取得其它对象的地址，进而控制对象。我们刚刚了解产生对象的方法，我们再来看看如何将这些对象收纳到集合中进行管理。首先我们观察下列的叙述：

```
Dim Cell As HtmlTableCell = New HtmlTableCell

Dim Row As HtmlTableRow = New HtmlTableRow

Cell.InnerText="Cell1"

Row.Cells.Add(Cell)

Cell = New HtmlTableCell    ' 再产生一个新的 HtmlTableCell 对象

Cell.InnerText="Cell2"

Row.Cells.Add(Cell)
```

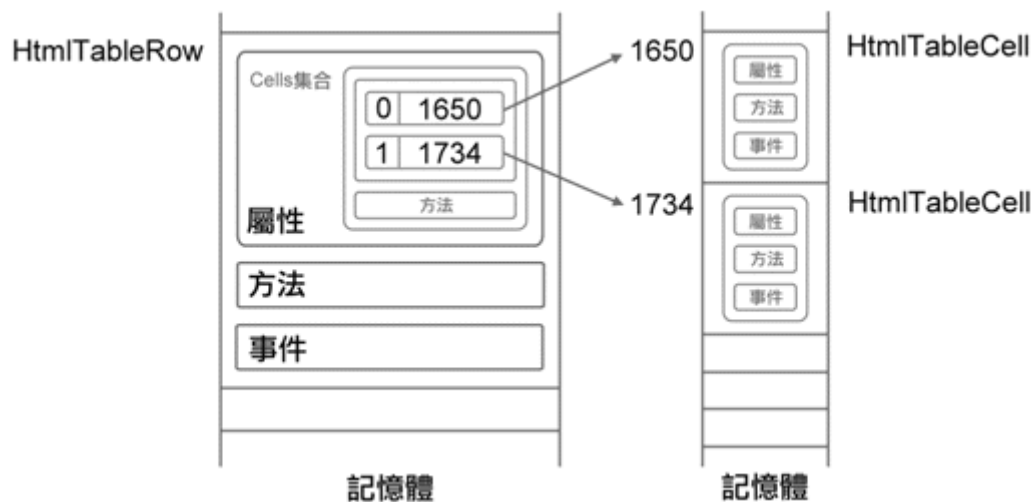
程序执行完前两行时，会分别产生 **HtmlTableCell** 对象及 **HtmlTableRow** 对象，并将其内存地址

指定给 **Cell** 以及 **Row** 这两个对象变量。当程序执行到第四行时，会利用 **Cells** 集合对象之 **Add**

方法，将 **Cell** 对象所指到的内存地址存入 **Row** 对象的 **Cells** 集合中；并且指定一个 **Index** 值 **0**，

好让我们日后可以指定集合中的这个对象。而当程序执行到第五行时，又产生一个新的

HtmlTableCell 对象，并将这个新对象的内存地址给 Cell 变量，所以此时 Cell 变量所指到的对象已经是另外一个新的 HtmlTableCell 对象。程序执行到最后一行时，又再利用 Row 对象中的 Cells 集合之 Add 方法，将 Cell 变量新指到的内存地址加入集合中，并指定 Index 值为 1。上列程序代码执行后的内存状况，如下图所示：



此时我们可以利用集合对象的 **Count** 属性，传回集合中总共记录了多少对象的内存地址，例如：

```
Response.Write(Row.Cells.Count)
```

这时候浏览器就会印出 **2**，表示有两个对象被集合所管理。我们将对象的地址收纳进集合时，指定了一个 **Index** 值。这个 **Index** 值可以让我们把集合中的项目取出来，只要透过集合对象的 **Item** 方法取出即可。如下范例所示：

```
Dim TmpCell As HtmlTableCell
```

```
TmpCell = Row.Cells.Item(0)
```

上述程序代码宣告了名为 **TmpCell** 指到 **HtmlTableCell** 型态对象的对象变量,然后再利用集合对

象的 **Item** 方法将集合中 **Index** 为 **0** 的内存地址取出,并指定给变量 **TmpCell**;所以我们就可以

透过 **TmpCell** 变量来控制 **HtmlTableCell** 对象。我们将这个观念做个简单的验证:

```
Dim Cell As HtmlTableCell = New HtmlTableCell
```

```
Dim Row As HtmlTableRow = New HtmlTableRow
```

```
Dim TmpCell As HtmlTableCell
```

```
Cell.InnerText="Cell1"
```

```
Row.Cells.Add(Cell)
```

```
Cell = New HtmlTableCell
```

```
Cell.InnerText="Cell2"
```

```
Row.Cells.Add(Cell)
```

```
TmpCell = Row.Cells.Item(0)    ' 传回 Index 为 0 所记载的内存地址
```

```
Response.Write(TmpCell.InnerText)    ' 印出"Cell1"
```

```
TmpCell = Row.Cells.Item(1)    ' 传回 Index 为 1 所记载的内存地址
```

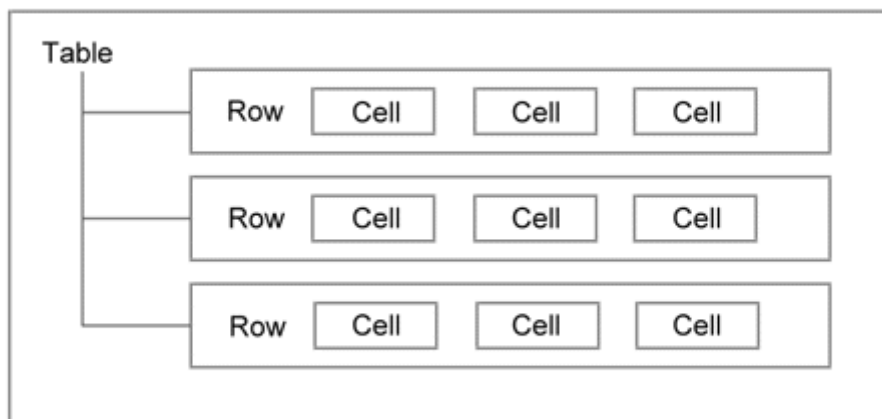
```
Response.Write(TmpCell.InnerText)    ' 印出"Cell2"
```



上述程序执行结果印出 "Cell1"及"Cell2"，验证了集合对象可以管理对象的内存地址，并可以利用 Item 方法将内存地址取出。

HtmlTable、HtmlTableRow、HtmlTableCell 控件

HtmlTable 控件可以配合 HtmlTableRow 以及 HtmlTableCell 控件来动态的产生表格。其关系为：



HtmlTable 控件是由许多列（Row）所组成，而每一列中是由许多储存格（Cell）所组成。所以

HtmlTable 控件中有 Rows 集合，HtmlTableRow 控件中有 Cells 集合。我们依秩序利用

HtmlTableRow 中 Cells 集合的 Add 方法，将 HtmlTableCell 控件串成一列（Row）后，再将这

一列加到 HtmlTable 的 Rows 集合中，这样一来表格就大功告成了。这些组成表格的控件都可以

设定一些外观属性，我们先来看看 HtmlTableCell 控件的语法：

<Td 或 Th

Id="被程序代码所控制的名称"

Runat="Server"

Align="Left | Center | Right"

BGColor="背景色"

BorderClolr="边框颜色"

ColSpan="跨栏数"

Hight="表格高度"

NoWarp="True | False"

RowSpan="跨列数"

Valign="垂直对齐方式"

Width="表格宽度"

>储存格内容

</Td 或 /Th>

一般来说我们会利用程序来产生 **HtmlTableCell** 对象，设定好属性之后，我们再加入

HtmlTableRow 对象中的 **Cells** 集合中。接下来我们来看看 **HtmlTableRow** 控件的语法：

<Tr

Id="被程序代码所控制的名称"

Runat="Server"

Align="Left | Center | Right"

BGColor="背景色"

BorderColor="边框颜色"

Hight="表格高度"

Cells="Cell 集合"

Valign="垂直对齐方式"

>

<Td>字段内容</Td>


```
<Td>字段内容</Td>
```

```
</Tr>
```

利用程序来产生 **HtmlTableCell** 对象后，我们再加入 **HtmlTableRow** 对象中的 **Cells** 集合中。等

表格的一列定义好之后，再利用 **HtmlTable** 对象的 **Rows** 集合，将表格的列加入集合中。我们来

看看 **HtmlTable** 控件的语法：

```
<Table
```

```
    Id="被程序代码所控制的名称"
```

```
    Runat="Server"
```

```
    Align="Left | Center | Right"
```

```
    BGColor="背景色"
```

```
    BorderColor="边框颜色"
```

```
    CellPadding="像素"
```

```
    CellSpacing="像素"
```

```
    Hight="表格高度"
```

```
    Rows="Row 集合"
```

```
    Width="表格宽度"
```

```
>
```

```
<Tr><Td><Td></Tr>
```

```
<Tr><Td><Td></Tr>
```

```
</Table>
```

使用范例：

下列范例利用表格控件印出九九表：

```
<Html>

<Table Id="Table1" Runat="Server" Border="1"/>

<Form Runat="Server">

    <Button Id="Button1" Runat="Server" OnServerClick="Button1_Click"

        InnerText="请按我"/>

</Form >

<Script Language="VB" Runat="Server" ID=Script1>

Sub Button1_Click(Sender As Object, e As EventArgs)

    Dim Cell As HtmlTableCell

    Dim Row As HtmlTableRow

    Dim X, Y As Short

    For X=1 To 9 Step 1
```

```
        Cell=New HtmlTableCell

For Y=1 To 9 Step 1

    Cell.InnerHtml+=CStr(X) & " * " & CStr(Y) & " = " & CStr(X * Y)

    If Y<>9 Then Cell.InnerHtml+="<br>"

Next Y

    If X=1 Or X=4 Or X=7 Then Row=New HtmlTableRow

    Row.Cells.Add(Cell)

    If X=3 Or X=6 Or X=7 Then Table1.Rows.Add(Row)

Next X

End Sub

</SCRIPT>

</Html>
```


5. ADO.NET

关系型数据库简介

什么是关系型数据库

数据表的组成

数据库就是储存数据的地方。关系型数据库是由许多数据表（Table）所组成，资料表又是由许多笔记录（Row 或 Record）所组成，而纪录又是由许多的字段（Column 或 Filed）所组成。假设我们是一个电子商务网站，现在要纪录使用者的数据，你可能会想到要纪录使用者的账号、密码、姓名、电话、住址以及 E-mail 等数据；这一些你所要纪录的项目，每一个项目就是一个字段。所以我们将这些字段做个整理，分析出这些字段的长度、数据型态、是否必须要有数据后，得到下列数据表的规格表：

编号	字段用途	英文名称	数据型态	长度	必须要有数据	备考
1	使用者账号	UserId	字符	10	是	
2	使用者密码	UserPwd	字符	10	是	

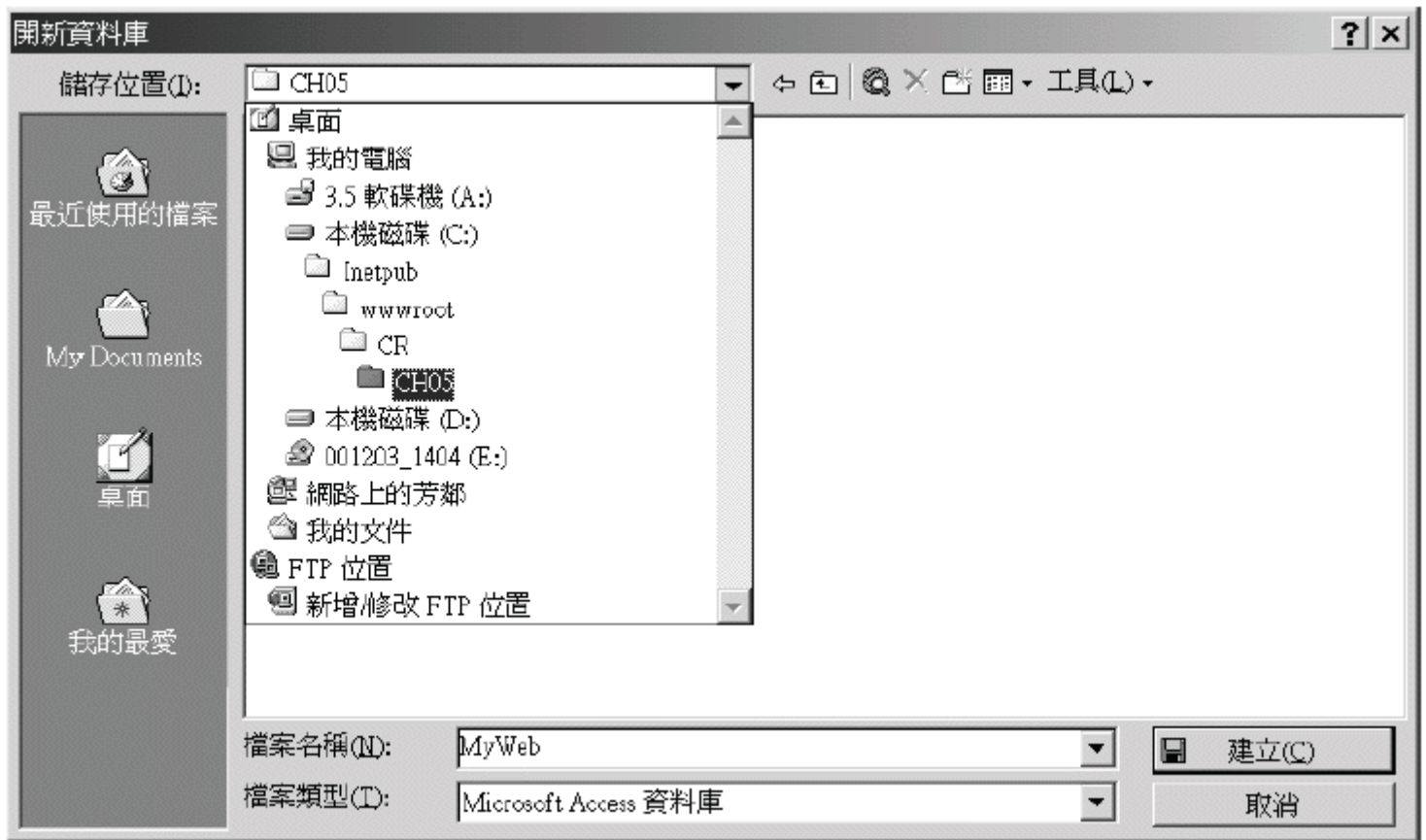
3	使用者姓名	UserName	字符	10	是	
4	使用者电话	UserTel	字符	10	是	
5	使用者住址	UserAdd	字符	50	是	
6	使用者 E-mail	UserEmail	字符	50	是	

我们先利用 Access 2000 先将这个数据表建好，首先先打开 Access 2000，则出现下列对话框：

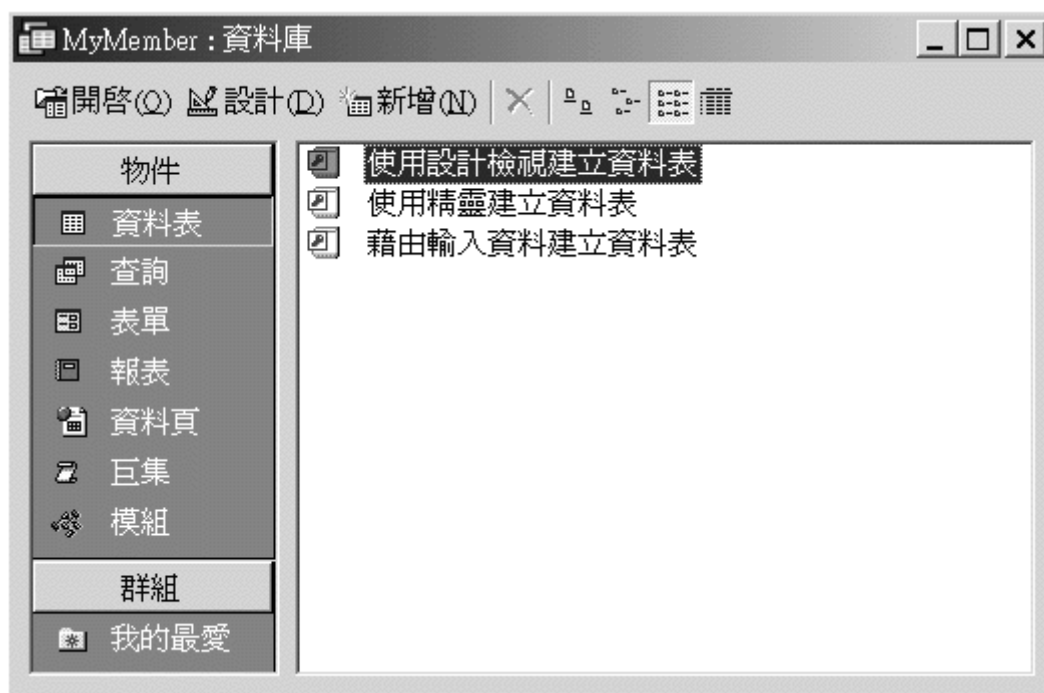


选择「Access 空白数据库」选项来建立新的数据库，并将这个档案取名 MyWeb 后，储存于

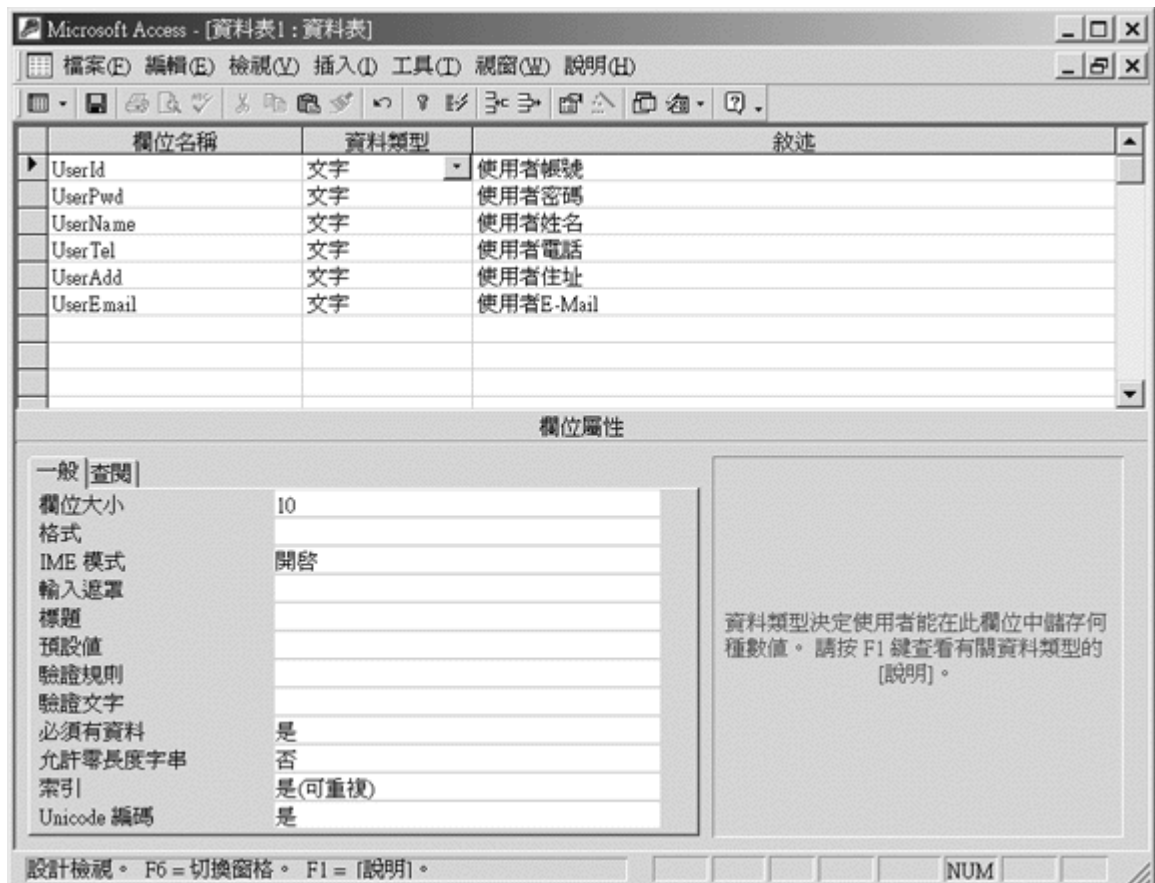
C:\inetpub\wwwroot\CR\CH05 的目录下：



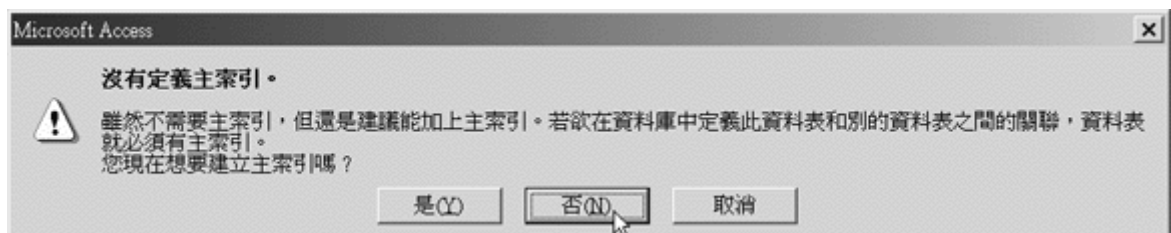
Access 2000 就会将这个数据库存在我们指定的目录之下，并自动将扩展名命名为 **mdb**，表示这个档案是 **Access** 数据库档案。现在这个数据库就建立好了，但是这个数据库里面是空的，里面还没有任何数据表。所以我们开始将字段一个个的填上去，我们在「使用设计检视建立数据表」上按两下或选择开启数据表设计画面：



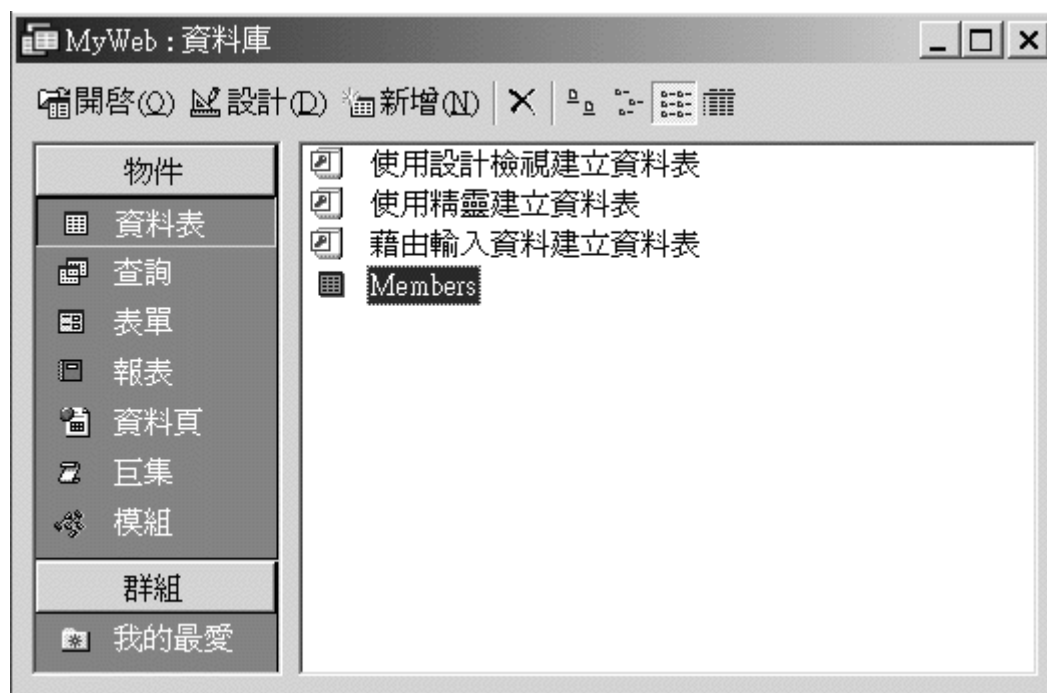
接下来我们依照数据规格表所定义的规格，设定每一个字段的名称、数据型态以及长度。另外要特别注意，由于我们强迫使用者不可以输入空白的字段，所以我们在「必须有数据」这一栏要选择「是」：



数据输入完毕后，我们再将数据表存盘，命名为 **Members**。但是我们在存盘的时候，系统会出现没有定义主索引的警告，这里我们先选择「否」：



这样我们的第一个数据表就定义完成了。



当然，数据表除了定义好字段外，还要实际拥有数据；这些数据就是我们所说的纪录。我们将每一个使用者的数据完整的输入后，每一个横列所组合起来的就是一个完整的数据，也就是我们所说的一笔记录。数据表就是由直的字段以及横的纪录所组合的。完成之后的数据表内并没有任何数据数据，所以我们先简略的输入下列三笔数据，让数据表完成。请在「Members」数据表上按两下后输入：

UserId	UserPwd	UserName	UserTel	UserAdd	UserMail
tina	1234	黄淑媛	0920086000	台北县三重市	tina@hotmail.com

jacky	5678	曾志远	0933455000	台北市中山区	jacky@hotmail.com
jolin	abcd	林久祥	0939507000	桃园县桃园市	jolin@hotmail.com

输入完毕后就是一个数据表了。



这样一来我们就可以在数据表内作数据的新增、编辑、查询以及删除等工作了。

主键

不过这个数据表严格说起来还有瑕疵，那就是使用者一但增加，一定会有重复的使用者账号。如

果使用者账号有重复的话，使用在者登入的时候就无法确认到底是哪个使用者。所以我们必须将

使用者账号设定成不允许重复，只要将这个字段设定成主键（Primary Key）就可以了：

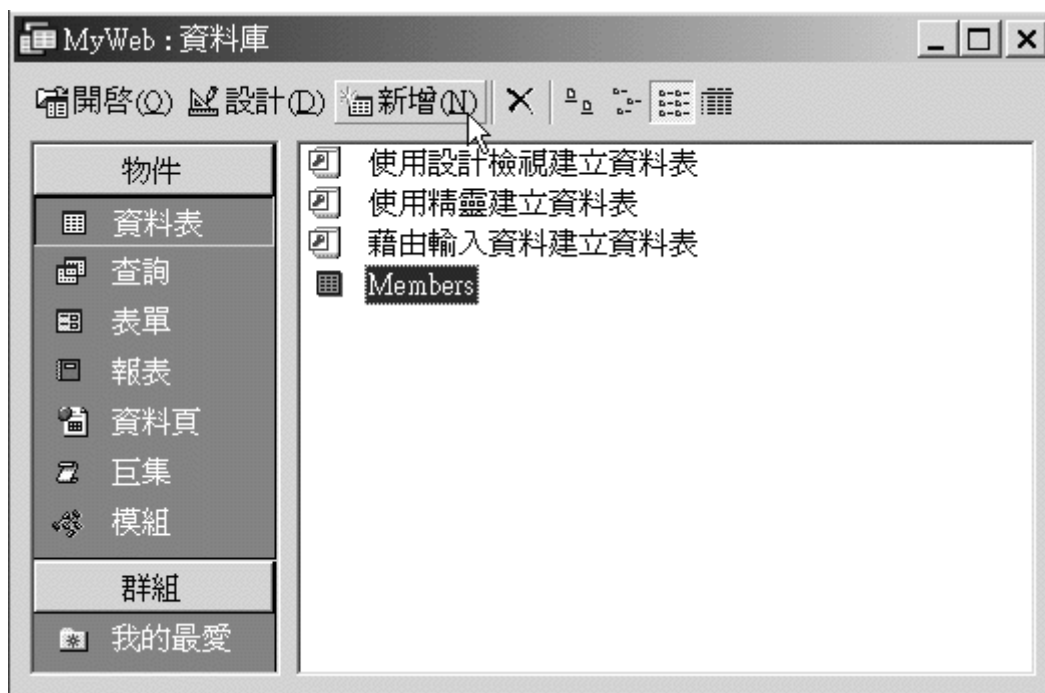


外来键

字段设定成主键最主要的功能除了不允许重复外，另外还可以建立两个数据表之间的关联。将数据库的结构作一个合理的分析，适当的将数据分散到各个数据表以及建立数据表之间的关联，这个动作叫做数据库的正规化。关系型数据库有什么好处呢？我们来举一个实际的例子，假设我们的网站除了使用者的数据外，还要记载使用者的订单状况。这样一来数据表就会多了订单日期、产品名称、单价、订购数量以及小计等项目，如果这些数据都和使用者基本数据记录在一起，我们的数据表就会变成（因为会重复的输入使用者的基本数据，所以我们将使用者数据表主键的设定关掉）：

UserId	UserPwd	UserName	UserTel	UserAdd	UserE-mail	OrderDat	ProductName	UnitPri	Quar	Total
tina	1234	黃淑媛	0920086000	台北縣三重市	tina@hotmail.com					
jacky	5678	曾志遠	0933455000	台北市中山區	jacky@hotmail.com					
jolin	abcd	林久祥	0939507000	桃園縣桃園市	jolin@hotmail.com					
tina	1234	黃淑媛	0920086000	台北縣三重市	tina@hotmail.com	2001/5/3	MP3隨身聽	4000	1	4000
jacky	5678	曾志遠	0933455000	台北市中山區	jacky@hotmail.com	2001/5/3	CD音響	6000	1	6000
jolin	abcd	林久祥	0939507000	桃園縣桃園市	jolin@hotmail.com	2001/5/1	CD空白片	700	1	700
								0	0	0

我们在上列数据表中新增了三笔资料。但是除了使用者的订单数据外，我们发现使用者的基本数据重复输入了。数据重复输入有许多坏处，第一是增加数据库的储存空间，第二是可能会导致数据输入错误，第三是不容易维护及管理。假设使用者的电子邮件信箱改变了，在这种数据库里面需要将使用者所有纪录的电子邮件信箱字段全部作更新，不但浪费数据库空间而且不好管理；所以我们应该把这些重复出现的数据独立出来，再建一个订单数据表。我们在数据库对话框中选择新建，如下图所示：



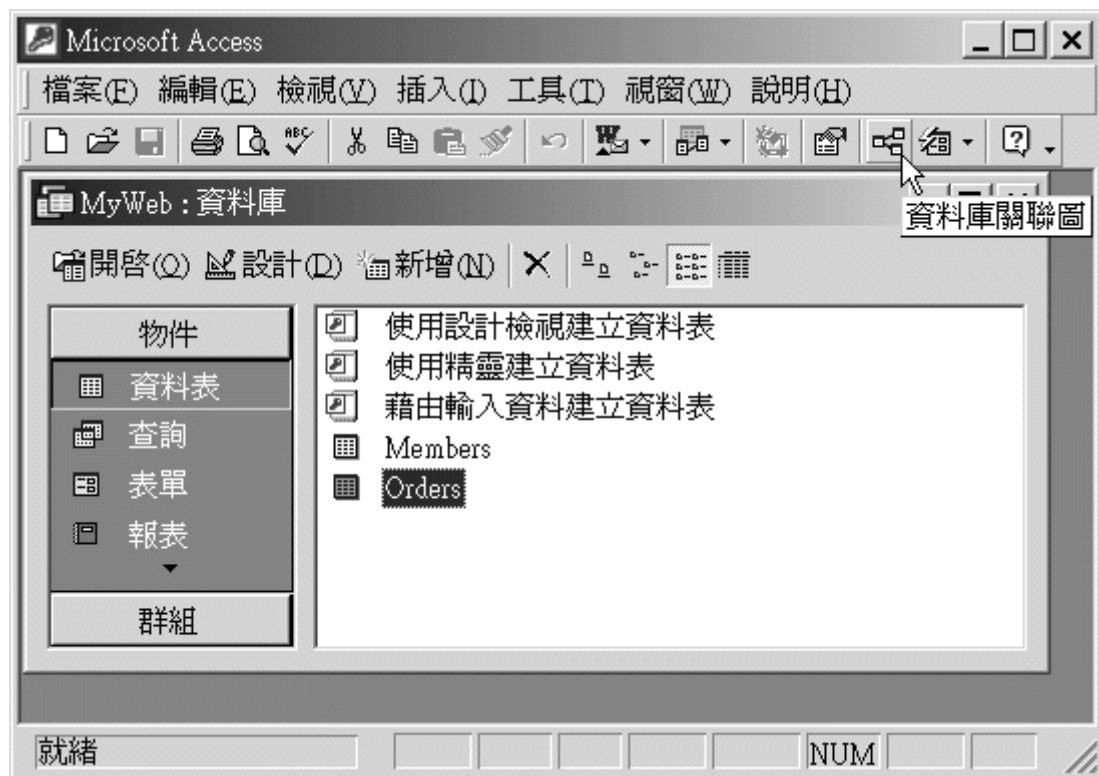
然后输入数据表架构，输入完毕后将数据表命名为 **Orders**，如下所示：

資料表1: 資料表			
	欄位名稱	資料類型	敘述
	UserId	文字	使用者帳號
	OrderDate	日期/時間	訂單日期
	ProductName	文字	產品名稱
	UnitPrice	數字	單價
	Quantity	數字	數量
	Total	數字	總計

由于订单数据表中要纪录订单是哪个使用者所下的，所以我们在这里产生了和 **Members** 数据表中定义一样的 **UserId** 字段，这个字段表示是要参考 **Members** 数据表中的 **UserId** 字段，所以一个字段若要参考其它字段的主键，则这个字段就叫做外来键（Foreign Key），这个字段我们在

下个建立数据表的关联时会用到。数据表结构在存盘的时候依然发出没有主键的警告，在这里我们不需要建立主键，所以选择「否」存档后离开。接下来我们要设定这两个数据库之间的关联了。

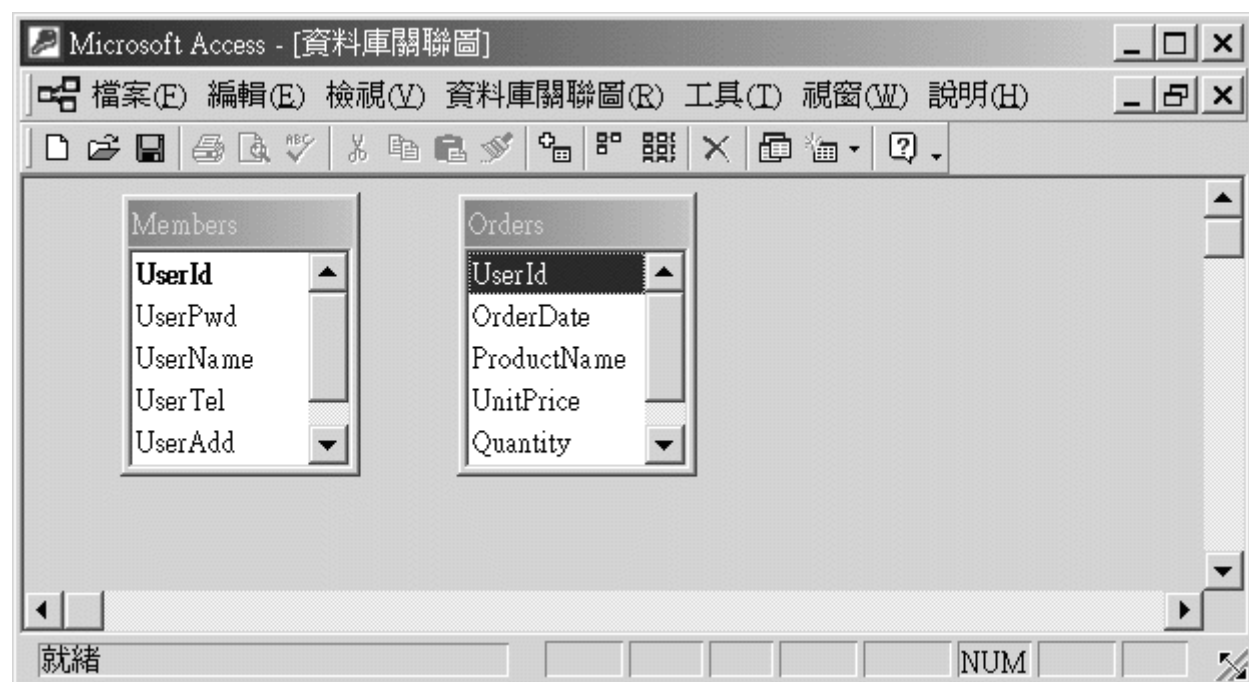
在设定关联之前请先确定 **Members** 数据表是否有设定主键为 **UserId**，倘若没有请设先定完后再选择「数据库关联图」选项：



出现「显示资料表」对话框后，我们将 **Members** 以及 **Orders** 分别点选新增：



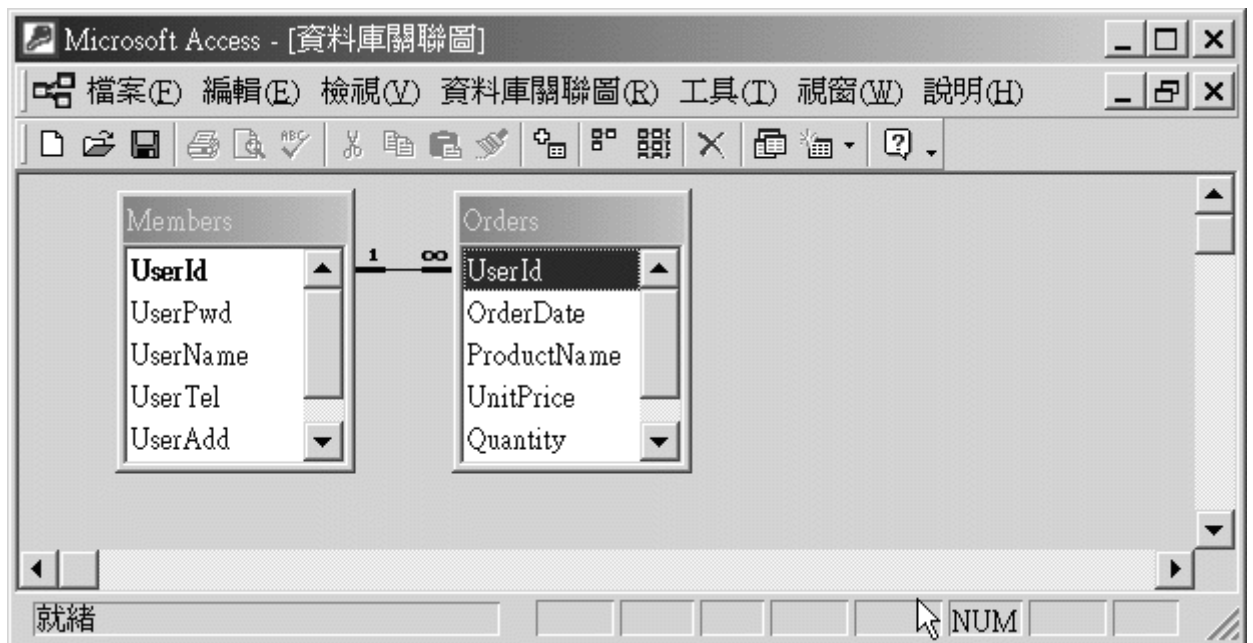
新增完毕后这两个数据表就被加到「数据关联图」窗口中了，如下图所示：



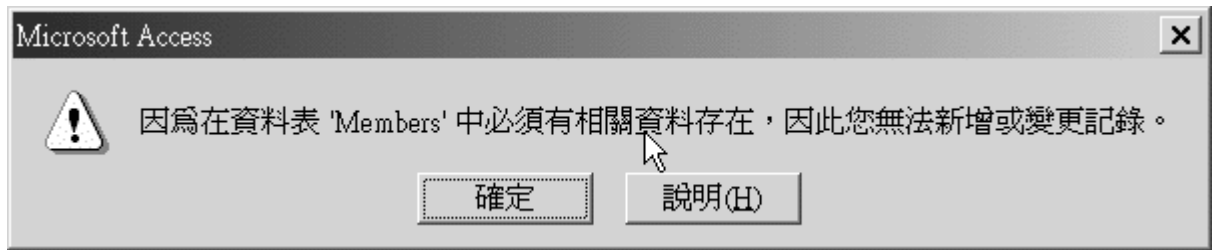
接下来我们可以利用拖放的方式，将 **Orders** 数据表中的 **UserId** 字段拖到 **Members** 窗体中，就会出现「编辑关联」对话框。我们将对话框中的「强迫参考完整性」、「串接更新相关数据」以及「串接删除相关纪录」的选项打勾，如下所示：



选择这些选项表示在输入订单数据表时其 **UserId** 字段会参考 **Members** 数据表的 **UserId** 字段，倘若所输入的数据在 **Members** 数据表中没有这个使用者账号，则所输入的使用者是错误的数据，所以不会被接收；这一来可以确保所输入的订单都可以找到是哪个使用者所下的。另外要删除某个使用者数据的时候，假设订单数据中还有参考到这笔要删除的纪录，则也是不允许删除；因为如果把被参考到的纪录删除，那么会造成只有订单数据却不知道是哪一个使用者下单的情形。这样一来就可以确保资料的完整性，不会造成错误的数据或是有孤儿数据等等的问题。选项设定好后，则出现下列的关联图：



在关联图中我们看到两个资料表有一条线连起来，表示这两个数据表之间有关联的关系。到在 **Members** 资料表旁显示数值 1，表示 **Members** 资料表的 **UserId** 字段是主键，而且字段内的值不允许重复；而 **Orders** 数据表旁显示 ∞ 符号，表示 **Orders** 数据表的 **UserId** 字段是外来键，表示字段的数据参考到别的字段的主键，而且允许输入重复的值（因为使用者可以下多笔订单）。所以倘若我们再输入订单数据时，随便输入一个没有在 **Members** 数据表中存在的 **UserId**，我们所输入的数据不会被接受外还会出现如下的警告讯息：



所以其实关联就是一种限制（**Constraint**），限制使用者所输入的数据必须是正确的，以及限制数据的完整性。善用关系型数据库的一些特点，不但可以帮助我们避免输入错误的数据外，也可以让日后的维护更轻松容易。

结构化查询语言 SQL

SQL 概述

要操作数据源中的数据，可以使用结构化查询语言（**Structured Query Language**）。SQL 的正确念法是 **S · Q · L**，不过大家都习惯念成 **Sequel**，SQL 几乎是所有大型数据服务器都支持的数据操作语言，它提供一些可以帮我们快速的执行数据查询、更新、删除等数据操作的叙述；要撰写操作数据的应用程序，SQL 语言是个非常重要的课题。SQL 的用法非常灵活，在这里我们不深入讨论，我们只介绍比较常用的 SQL 叙述。

Select 陈述

Select 陈述可以从资料源传回我们所指定的字段，其语法如下所示：

```
Select 字段 1 [, 字段 N] From 数据表名称
```

例如我们想要传回 **Members** 数据表中 **UserId** 以及 **UserPwd** 这两个字段的数据，可以使用如下叙述：

```
Select UserId, UserPwd From Members
```

如果要将所有的字段传回，则可以使用「*」来代表。例如我们要将 **Members** 数据表中的所有字段传回，则使用下列叙述：

```
Select * From Members
```

利用 **Where** 子句来过滤数据

利用 **Where** 子句可以限制我们所要过滤的纪录，其语法如下所示：

```
Select 字段一 [, 字段 N] From 数据表名称 Where 条件
```

条件可以是 **=**、**>**、**<**、**>=**、**<=** 比较运算符，其中如果所要判断的数据是日期或是字符串，必须用单引号「'」刮起来。例如我们要将会员数据中 **UserId** 字段为 **tina** 的数据全部传回，使用下列叙述：

```
Select * From Members Where UserId = 'tina'
```

另外我们也可以搭配逻辑运算子来过滤两个字段的条件。例如我们要将 **UserId** 字段为 **tina** 以及 **UserPwd** 字段为 **1234** 的数据传回来，可以使用下列叙述：

```
Select * From Members Where UserId = 'tina' And UserPwd = '1234'
```

Where In

如果只要符合某些条件的资料我们都要找出来，则可以使用 **Where In**。例如下列范例传回使用者名称为 **tina** 或是 **jacky** 的纪录：

```
Select * From Members Where UserId In ('tina', 'jacky')
```

Where Like

如果我们想搜寻住台北市的顾客，则可以使用 **Like** 比对。**Like** 比对要配合「%」符号来操作，可以找出以特定字符串为开头或是结尾的字段。例如下列 **SQL** 叙述将住台北市的使用者列出：

```
Select * From Members Where UserAdd Like '台北市%'
```

Order By

若要将查询回的资料表进行排序的工作，则可以利用 **Order By** 子句。**Order By** 子句是依照字段内数据的顺序进行排序，其语法如下所示：

```
Select 字段一 [, 字段 N] From 数据表 [Where 子句] [Order By 子句]
```

Order By 依照字段顺序排序的方式有升幂以及降序，如过要由小排到大，则在最后指定 **Asc**；

倘若是由大排到小，则是 **Desc**。例如下列叙述将所有使用者数据传回，并依 **UserId** 字段作升幂排列：

```
Select * From Members Order By UserId ASC
```

Insert 陈述

Insert 陈述可以将新的纪录加入数据源中，其语法如下所示：

```
Insert Into 数据表名称[(字段 1, 字段 2, ... 字段 N)] Values(字段 1, 字段 2, ... 字段 N)
```

数据表后面的字段可以省略。如果省略表示全部的字段都要输入，并且必需按照字段的顺序来输入。例如下列叙述增加一位新的使用者：

```
Insert Into Members Values('elvira', 'wxyz', '邓宜玲', '0935123000', _  
'台北县中和市', 'elvira@hotmail.com')
```

Update 陈述

Update 陈述可以更新数据源中纪录的数据，其语法如下所示：

```
Update 数据表名称 Set 字段一 = 叙述 [, ... 字段 N=叙述] [Where 子句]
```

例如下列叙述将使用者数据表中 **UserId** 字段为 **elvira** 的纪录，将其 **UserPwd** 更改为 **zyxw**：

```
Update Members Set UserPwd = 'zyxw' Where UserId = 'elvira'
```

Delete 陈述

Delete 陈述可以删除数据源中的纪录，其语法如下所示：

```
Delete 数据表名称 [Where 子句]
```

特别注意如果没有设定 **Where** 子句的条件，则会将所有数据表中的纪录全部删除。

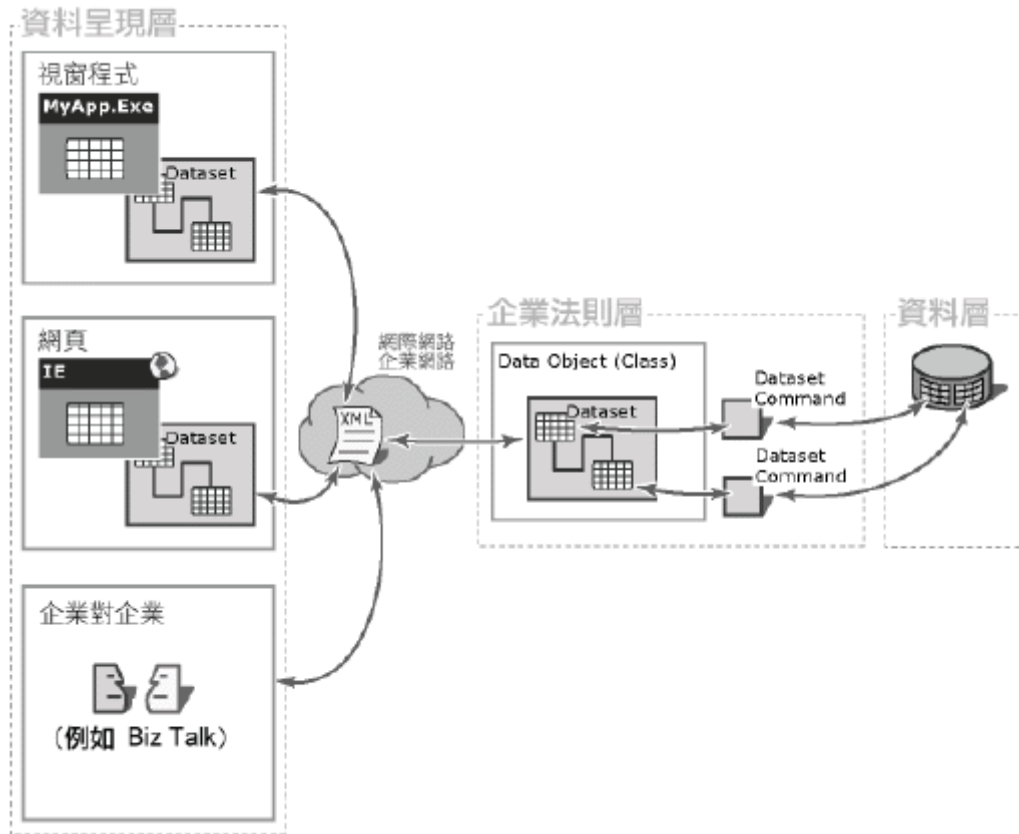
ADO.NET 基本概念

ADO.NET 的特色

ADO (ActiveX Data Object) 对象是继 ODBC (Open Database Connectivity, 开放数据库连接架构。微软所制定的架构, 可以让透过这种架构和数据库连结。) 之后微软主推存取数据的最新技术, ADO 对象是程序开发平台用来和 OLE DB 沟通的媒介, ADO 目前的最新版本为 ADO.NET。

ADO.NET 不像以前的 ADO 版本是站在为了存取数据库的观点而设计的, ADO.NET 是为了因应广泛的数据控制而设计, 所以使用起来比以前的 ADO 更灵活有弹性, 也提供了更多的功能。

ADO.NET 的出现并不是要来取代 ADO, 而是要提供更有效率的数据存取。微软透过最新的 .NET 技术提供了可以满足众多需求的架构, 这个架构就是 .NET 共享对象类别库。这个共享对象类别库不但涵盖了 Windows API (Windows Application Programming Interface, Windows 应用程序设计界面。提供许多撰写 Windows 程序所需要使用的对象以及基本函式等。) 的所有功能, 并且还提供更多的功能及技术; 另外它还将以前放在不同 COM 组件上, 我们常常使用的对象及功能一并含括进来。除此之外 ADO.NET 还将 XML 整合进来, 这样一来数据的交换就变的非常轻松容易了。所以 ADO.NET 的架构及新功能是为了能满足广泛的数据交换需求所产生出来的新技术, 这个就是 ADO.NET。



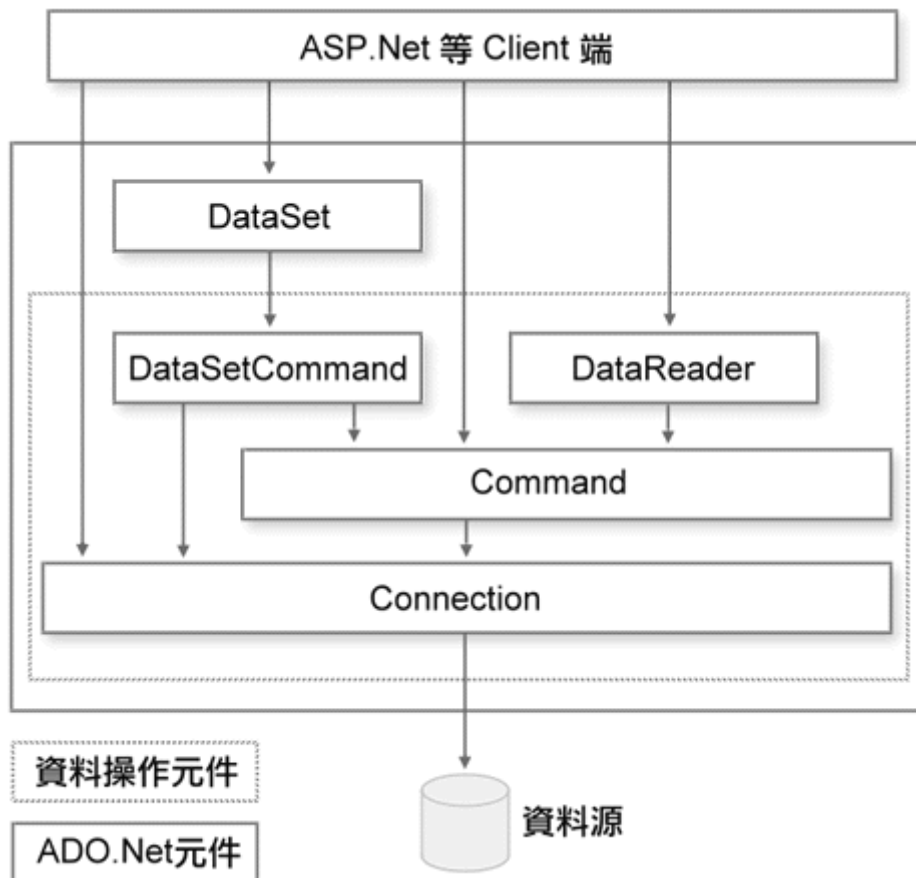
ADO.NET 架构

ADO.NET 对象可以让我们快速简单的来存取各种数据。传统的主从式应用程序在执行时，都会保持和数据源的联机。但是在某些状况下和数据库一直保持联机是不需要的，而且一直保持和数据源的联机会浪费系统资源。有些时候我们只需要很单纯的将数据取回，这时候就不需要保持对数据源的联机。ADO.NET 被设计成对于数据处理不一直保持联机的架构，应用程序只有在要取得数据或是更新数据的时候才对数据源进行联机的工作，所以应用程序所要管理的连结减少；数据源就不用一直和应用程序保持联机，负载减轻了效能自然也就提升。不过我们的应用程序也有

些情况需要和数据源一直保持联机，例如在线订位系统；此时我们还是可以使用 ADO 对象和数据源随时保持联机的状态。

ADO.NET 对象模型

ADO.NET 对象模型中有五个主要的组件，分别是 Connection 对象、Command 对象、DataSetCommand、DataSet 以及 DataReader。这些组件中 **负责建立联机和数据操作的部分我们称为数据操作组件（Managed Providers）**，分别由 Connection 对象、Command 对象、DataSetCommand 对象以及 DataReader 对象所组成。数据操作组件最主要是当作 DataSet 对象以及数据源之间的桥梁，负责将数据源中的数据取出后植入 DataSet 对象中，以及将数据存回数据源的工作。下图是显示这些对象关系的 ADO.NET 对象模型：



Connection 物件

Connection 对象主要是开启程序和数据库之间的连结。没有利用连结对象将数据库打开，是无法从数据库中取得数据的。这个物件在 ADO.NET 的最底层，我们可以自己产生这个对象，或是由其它的对象自动产生。

Command 物件

Command 对象主要可以用来对数据库发出一些指令，例如可以对数据库下达查询、新增、修改、删除数据等指令，以及呼叫存在数据库中的预存程序等。这个对象是架构在 **Connection** 对象上，也就是 **Command** 对象是透过连结到数据源的 **Connection** 对象来下命令的；所以 **Connection** 连结到哪个数据库，**Command** 对象的命令就下到哪里。

DataSetCommand 物件

DataSetCommand 对象主要是在数据源以及 **DataSet** 之间执行数据传输的工作，它可以透过 **Command** 对象下达命令后，并将取得的数据放入 **DataSet** 对象中。这个对象是架构在 **Command** 对象上，并提供了许多配合 **DataSet** 使用的功能。在 Beta 2 版中 **DataSetCommand** 物件会更名为 **DataAdapter**。

DataSet 物件

DataSet 这个对象可以视为一个暂存区（**Cache**），可以把从数据库中所查询到的数据保留起来，甚至可以将整个数据库显示出来。**DataSet** 的能力不只是可以储存多个 **Table** 而已，还可以透过 **DataSetCommand** 对象取得一些例如主键等的数据库表结构，并可以记录数据库表间的关联。

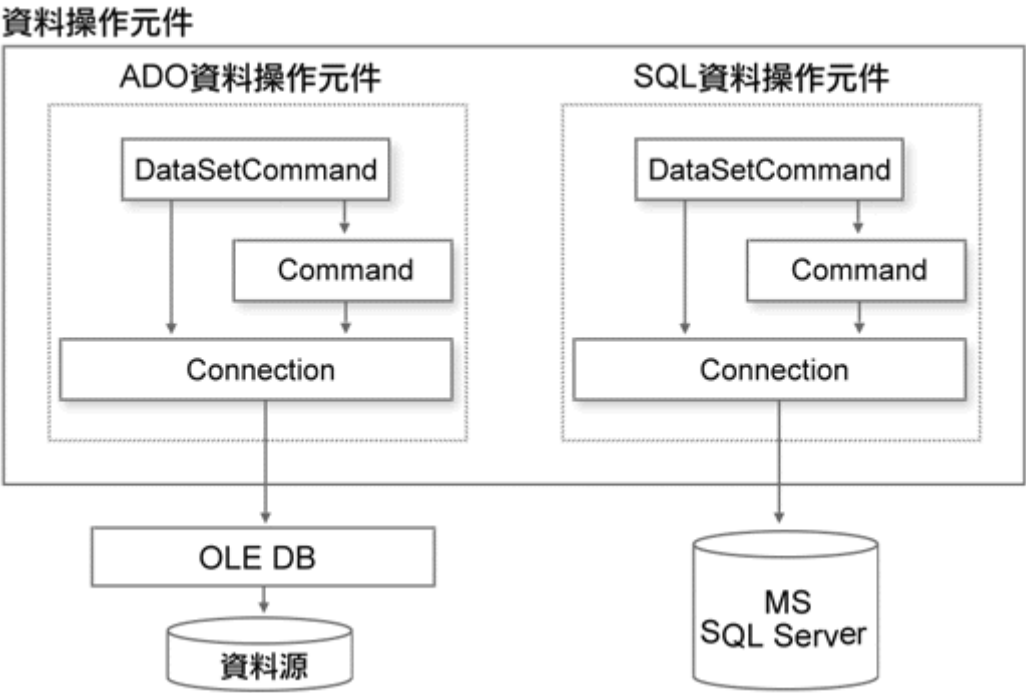
DataSet 对象可以说是 ADO.NET 中重量级的对象，这个对象架构在 **DataSetCommand** 对象上，本身 **不具备和数据源沟通的能力**；也就是说我们是将 **DataSetCommand** 对象当做 **DataSet** 对象以及数据源间传输数据的桥梁。

DataReader 物件

当我们只需要循序的读取数据而不需要其它操作时，可以使用 **DataReader** 对象。**DataReader** 对象只是一次一笔向下循序的读取数据源中的数据，而且这些数据是只读的，并不允许作其它的操作。因为 **DataReader** 在读取数据的时候限制了每次只读取一笔，而且只能只读，所以使用起来不但节省资源而且效率很好。使用 **DataReader** 对象除了效率较好之外，因为不用把数据全部传回，故可以降低网络的负载。

ADO.NET 的数据操作组件（Managed Providers）

ADO.NET 的数据存取和之前的版本不一样。前版的 ADO 存取数据的方式只有一种，那就是透过 OLE DB 来存取数据；而现在的 ADO.NET 则分为两种，一种是直接存取 MS SQL Server 中的数据，另一种是透过 OLE DB 来存取其它数据库中的数据。前面我们提过：[要存取数据源中的数据，要透过数据操控组件](#)。这个数据操作组件就是 **Connection** 对象、**Command** 对象、**DataSetCommand** 对象以及 **DataReader** 对象。由于我们可以选择透过 OLE DB 和资料源联机，或是和 MS SQL Server 直接联机；所以 **ASP.NET** 提供了两组数据操作组件，分别为 **ADO 数据操作组件** 以及 **SQL 数据操作组件**。



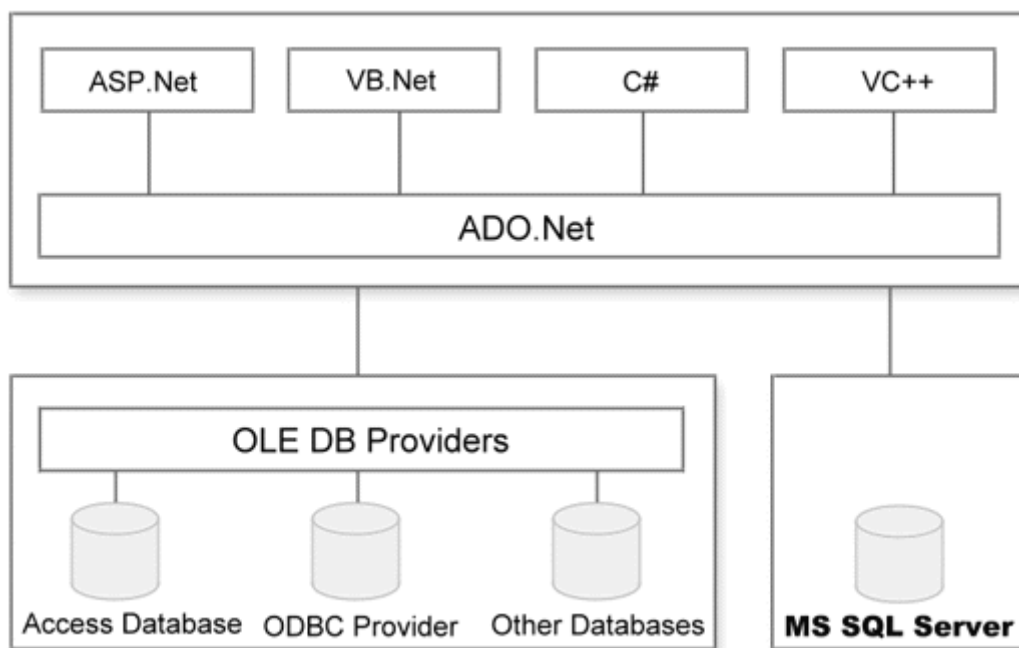
每组数据操作组件内都有 **Connection** 对象、**Command** 对象、**DataSetCommand** 对象及 **DataReader** 对象。为了容易分别这两组数据控制对象，我们将这四个对象分别加上前缀 **ADO** 以及 **SQL**，如下表所示：

ADO 数据操作组件	SQL 数据操作组件
ADOConnection	SQLConnection
ADOCommand	SQLCommand
ADODataSetCommand	SQLDataSetCommand
ADODataReader	SQLDataReader

这两种数据操作组件虽然针对的数据源不一样，但是这些对象的架构都一样。例如

ADOConnection 和 **SQLConnection** 对象虽然一个是针对 **OLE DB**，而另一个是针对 **MS SQL**

Server，但是这两个对象都有一样的属性、事件及方法，所以使用起来并不会造成困扰；只要针对所要建立的数据源种类来选择 ADO 数据操作组件，或是 SQL 数据操作组件就可以了。虽然我们也可以透过 OLE DB 来存取 MS SQL Server 中的资料，但是透过 SQL 类别对象来存取 MS SQL Server 中的数据效率最好；这是因为 SQL 类别不经过 OLE DB 这一层，而是直接呼叫 MS SQL Server 中的 API，所以效率比较好。ADO.NET 对于这两种数据存取方式使用的对象完全不一样，在使用的时候必须要特别注意。



OLE DB 简介

我们要开启如 **Access** 数据库中的数据，必须用 **ADO.NET** 透过 **OLE DB** 来开启。**ADO.NET** 利用 **OLE DB** 来取得数据，这是因为 **OLE DB** 了解如何和许多种数据源作沟通，所以对 **OLE DB** 有相当程度的了解是很重要的。**OLE DB** 为一种开放式的标准，并且设计成 **COM**（**Component Object Model**，一种对象的格式。凡是依照 **COM** 的规格所制作出来的组件，皆可以提供功能让其它程序或组件所使用。）组件。**OLE DB** 最主要是由三个部分组合而成：

Data Providers 数据提供者

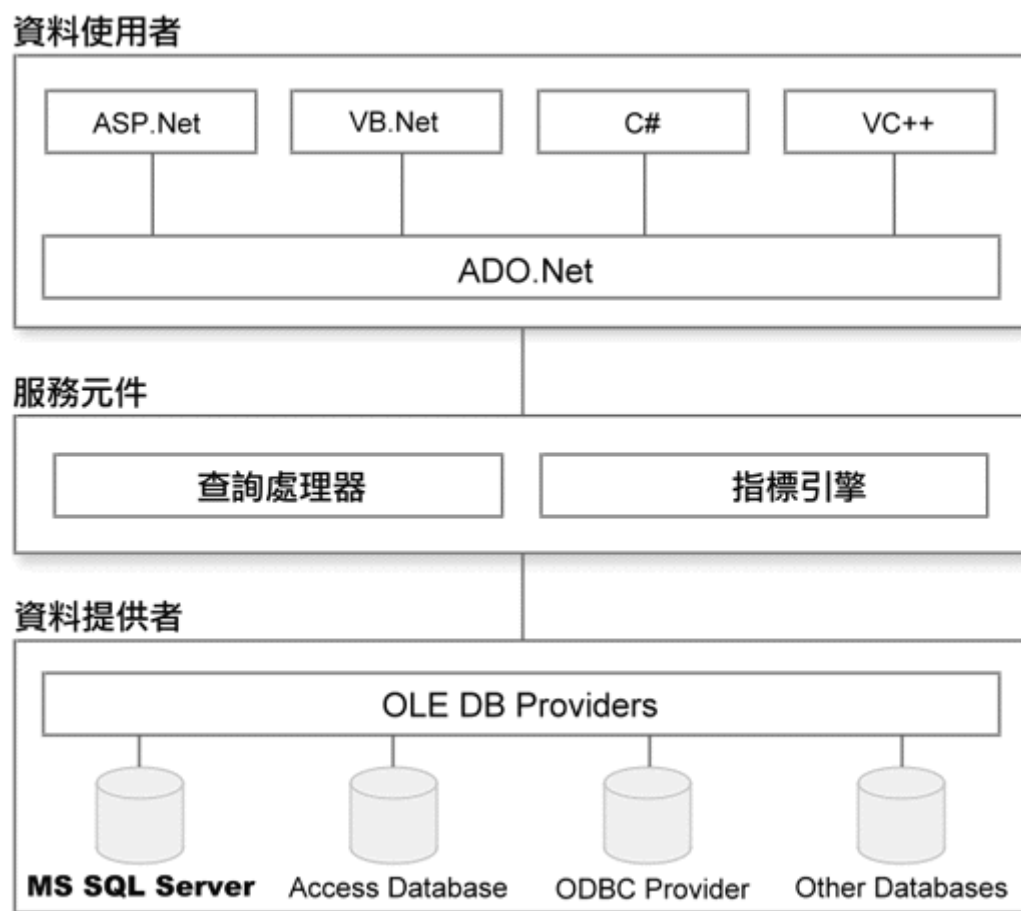
凡是透过 **OLE DB** 将数据提供出来的，就是数据提供者。例如 **SQL Server** 数据库中的数据表，或是附文件名为 **mdb** 的 **Access** 数据库档案等，都是 **Data Provider**。

Data Consumers 数据使用者

凡是使用 **OLE DB** 提供数据的程序或组件，都是 **OLE DB** 的数据使用者。换句话说，凡是使用 **ADO** 的应用程序或网页都是 **OLE DB** 的数据使用者。

Service Components 服务组件

数据服务组件可以执行数据提供者以及数据使用者之间数据传递的工作，数据使用者要向数据提供者要求数据时，是透过 OLE DB 服务组件的查询处理器执行查询的工作，而查询到的结果则由指针引擎来管理。



名称地址 (Namespace)

我们要使用 ADO.NET 中的对象，必需先宣告 ADO.NET 的名称地址（Namespace）。因为微软为 .NET 做了数量相当可观的类别对象，所以宣告名称地址是必要的。名称地址中记录了对象的名称以及所在，这样编译器在编译我们程序的时候才知道这些对象要到哪里去加载，和 VB 6 的设定 Reference 很类似。以下为宣告名称地址的语法：

```
<%@Import Namespace="对象类别的名称地址"%>
```

System.Data 名称地址

我们要使用 ADO.NET，必须要先宣告 System.Data 这个名称地址。因为 System.Data 这个名称地址中含括大部分组成 ADO.NET 架构的基础对象类别，例如 DataSet 对象、数据表、字段、关联等，所以要使用 ADO.NET 就一定要宣告 System.Data 这个名称地址。所以我们先在 ASP.NET 网页前面输入下列宣告：

```
<%@Import Namespace="System.Data"%>
```

System.Data.ADO 名称地址

我们要使用 ADO 数据操作组件来存取数据，必须宣告 System.Data.ADO 这个名称地址。ADO 数据操作组件是透过 OLE DB 和数据源联机，故 System.Data.ADO 这个名称地址定义了 ADO 数据操作组件的对象类别，例如 ADOConnection 对象、ADOCommand 对象、

ADODataSetCommand 对象及 ADODataReader 对象。要使用 ADO 数据操作组件就一定要宣告 System.Data.ADO 这个名称地址。所以我们在 ASP.NET 网页前面加入下列宣告：

```
<%@Import Namespace="System.Data.ADO"%>
```

System.Data.SQL 名称地址

我们要使用 SQL 数据操作组件来存取数据，必须宣告 System.Data.SQL 这个名称地址。SQL 数据操作组件是直接和 MS SQL Server 联机，故 System.Data.SQL 这个名称地址定义了 SQL 数据操作组件的对象类别，例如 SqlConnection 对象、SqlCommand 对象、

SQLDataSetCommand 对象及 SqlDataReader 对象。要使用 SQL 数据操作组件就一定要宣告 System.Data.SQL 这个名称地址。所以我们在 ASP.NET 网页前面加入下列宣告：

```
<%@Import Namespace="System.Data.SQL"%>
```

和数据源联机

Connection 物件简介

要存取数据源内的数据，首先要建立程序和数据源之间的联机，这个工作可以藉由 **Connection** 对象帮我们完成。首先我们先以和 **Access 2000** 数据库联机为例，首先在 **ASP.NET** 网页前面宣告名称地址：

```
<%Import Namespace="System.Data"%>  
<%Import Namespace="System.Data.ADO"%>
```

由于我们要使用 **ADO.NET**，所以一定要宣告 **System.Data** 的名称空间。而要和 **Access 2000** 数据库联机，必须要透过 **OLE DB**；所以我们使用 **ADO** 数据控制组件来和 **OLE DB** 沟通。宣告好名称地址后，我们就可以使用 **Connection** 对象了。**Connection** 对象可以使用下列语法来产生：

```
Dim 变数 As ADOConnection  
变数=New ADOConnection[("ConnectionString")]
```

或是直接在宣告的时候直接用 **New** 关键词产生：

```
Dim 变数 As ADOConnection=New ADOConnection[("ConnectionString")]
```

接下来我们在 **Page_Load** 事件中宣告一个指到 **ADOConnection** 对象的变量，并且利用 **New** 运算子实际产生 **ADOConnection** 对象后，再将对象的地址传给 **cnA**：

```
Dim cnA As ADOConnection=New ADOConnection
```

接下来就可以开始设定这个 **Connection** 对象的属性了，下表列出 **Connection** 对象常用的属性：

属性	说明
ConnectionString	指明要如何连接至数据源
ConnectionTimeout	联机逾时时间
Database	开启连结时所要开启的数据库，或目前开启的数据库
DataSource	要连结的数据库
UserID	登入数据库的账号
Password	登入数据库的密码
Provider	要联机数据库的种类

ConnectionString 属性

要开启一个数据库，必需指明要开启数据库的种类、数据库服务器名称、要开启数据库名称、登入使用者名称以及密码等信息，这些信息可以直接宣告在这个属性里面。我们在开启 **Connection** 对象之前要先设定 **ConnectionString** 属性才可以开启，这个属性有下列参数：

联机参数	说明
Provider	所要连结的资料源种类
User ID	登入数据源的使用者账号
Password	登入数据源的使用者密码

Data Source	数据库档案：数据库档案所在地址 数据服务器：指定数据服务器中所要连结的数据库名称
Initial Catalog	指定数据服务器中所要连结的数据库名称

以我们之前所建立的 **Access 2000** 数据库 **MyWeb.mdb** 为例，该档案是 **Access 2000** 所建的数据库，并和我们的网页存放于同目录，数据库文件名称为 **MyWeb.mdb**，使用者名称没指定即为默认值 **Admin**，并且没有密码：

```
cnA.ConnectionString="Provider= Microsoft.Jet.OLEDB.4.0;" & _
"Data Source=C:\Inetpub\wwwroot\cr\ch05\MyWeb.mdb;" & _
"User ID=Admin"
```

或者是在建立 **Connection** 对象时直接指定 **ConnectionString** 的值，如下范例所示：

```
Dim cnA As ADOConnection = New ADOConnection("连结字符串")
```

首先 **ADOConnection** 对象所宣告的 **Provider** 等于 **"Microsoft.Jet.OLEDB.4.0"**，由于 **Access 2000** 的 **mdb** 数据库档案是透过 **Jet 4.0** 引擎来驱动的，所以设定这个值是要告诉 **OLE DB** 我们所要开启数据库是透过 **Jet 4.0** 引擎所驱动的。**Provider** 参数可以支持许多数据源的设定值，如下表所示：

资料来源种类	说明
SQLOLEDB	MS SQL Server（建议不要用 OLE DB 操作数据库）
MSDASQL	ODBC
Microsoft.Jet.OLEDB.4.0	MS Jet 引擎 4.0（连结 Access 的 mdb 档）
MSIDXS	MS Index Server
ADSDS00object	MS Active Directory Services
MSDAORA	Oracle

另外参数和参数之间要用分号「;」作分隔，其中密码没有则可以省略。连结字符串设定好后，

我们就可以用 **Open** 方法和数据源联机了。以下为开启 **Connection** 对象的完整程序代码：

```

<%@Import Namespace=System.Data%>

<%@Import Namespace=System.Data.ADO%>

<Script Language="VB" Runat="Server">

Sub Page_Load(sender As Object, E As EventArgs)

    Dim cnA as ADOConnection=New ADOConnection

    cnA.ConnectionString= _

        "Provider=Microsoft.Jet.OLEDB.4.0;" & _

        "Data Source=C:\Inetpub\wwwroot\cr\ch05\MyWeb.mdb;" & _

        "User ID=Admin"

```

```
        cnA.Open

End Sub

</SCRIPT>
```

除了设定 **ConnectionString** 属性可以指定 **Connection** 对象的联机行为外，我们也可以直接指定

Connection 对象的属性，所以上述程序可以改写成：

```
<%@Import Namespace=System.Data%>

<%@Import Namespace=System.Data.ADO%>

<Script Language="VB" Runat="Server">

Sub Page_Load(sender As Object, e As EventArgs)

    Dim cnA As ADOConnection=New ADOConnection

    cnA.Provider="Microsoft.Jet.OLEDB.4.0"

    cnA.DataSource="C:\Inetpub\wwwroot\cr\ch05\MyWeb.mdb"

    cnA.UserID="Admin"

    cnA.Open()

End Sub

</SCRIPT>
```

从数据源取回数据

Command 物件简介

虽然我们刚刚已经和数据源联机了，但是还必须透过 **Command** 对象来对数据源进行数据操作的工作。**Command** 对象最主要的工作是透过 **Connection** 对象对数据源下达操作数据库的命令。

我们以下列语法产生 **Command** 对象：

```
Dim cmA As Command=New ADODCommand
```

或是在产生对象的时候顺便指定属性：

```
Dim cmA As Command=New ADODCommand("CommandText",ActiveConnection)
```

当我们将 **Command** 对象建立好之后，就可以设定 **Command** 对象的属性了。首先我们先来了

解 **Command** 对象有哪些常用的属性：

属性	说明
ActiveConnection	设定要透过哪个连结对象下命令
CommandBehavior	设定 Command 对象的动作模式
CommandType	命令型态
CommandText	要下达至数据源的命令
CommandTimeout	指令逾时时间

Parameters	参数集合
RecordsAffected	受影响的纪录笔数

ActiveConnection

ActiveConnection 属性是设定 **Command** 对象对数据源的操作要透过哪个 **Connection** 对象，例

如我们想透过 **cnA** 这个 **Connection** 对象对数据源进行数据操作：

```
cmA.ActiveConnection=cnA
```

CommandType

CommandType 属性可以用来指定 **CommandText** 属性中的内容是 **SQL** 陈述、数据表名称还是

预存程序，如下表所示：

参数名称	说明
CommandType.TableDirect	数据表名称
CommandType.Text	SQL 陈述或资料源了解之命令
CommandType.StoredProcedure	预存程序名称

如果本属性没有指定，则为默认值 **CommandType.Text**。例如我们指定要以数据表名称将数据表内的数据全部传回来，则设定为：

```
cmA.CommandType=CommandType.TableDirect
```

CommandText

视 **CommandType** 属性设定，表示要下达到数据源的内容是 **SQL** 陈述、数据表名称或预存程序名。例如下列范例中直接输入数据表名称，将数据表中的所有数据传回：

```
cmA.CommandType=CommandType.TableDirect  
  
cmA.CommandText="Members"
```

设定好 **Command** 对象的属性后，接下来是 **Command** 对象常用的方法：

方法	说明
Execute	透过 Connection 对象下达命令至数据源
Cancel	放弃命令的执行
ExecuteNonQuery	使用本方法表示所下达的命令不会传回任何纪录
Prepare	将命令以预存程序储存于数据源，以加快后续执行效率

Execute

Command 物件最常用的方法是 **Execute**，**Execute** 方法可以将 **CommandText** 属性中的数据传到数据源。例如下面范例中，使用者可以在文字输入盒中输入 **SQL** 陈述，并会显示所输入的叙述影响资料源的笔数：

```
<%@Import Namespace=System.Data.ADO%>

<%@Import Namespace=System.Data%>

<Html>

<Form id="F1" Runat="Server">

    请输入 SQL 陈述: <Input Type="Text" Id="Text1" Runat="Server" >

    <Button ID="Button1" Runat="Server" OnServerClick="Button1_Click">
执行

    </Button>

</form>

<Span ID="Sp1" Runat="Server"/>

<Script Language="VB" Runat="Server">

Sub Button1_Click(sender As Object, E As EventArgs)

    Dim cnA As ADOConnection=New ADOConnection

    Dim cmA As ADOCommand=New ADOCommand
```

```
cnA.Provider="Microsoft.Jet.OLEDB.4.0"

cnA.DataSource="C:\Inetpub\wwwroot\cr\ch05\MyWeb.mdb"

cnA.UserID="Admin"

cnA.Open()

cmA.ActiveConnection=cnA

cmA.CommandType=CommandType.Text

cmA.CommandText=Text1.Value

cmA.Execute()

Sp1.InnerText="这个叙述影响了" & cmA.RecordsAffected & "笔资料"

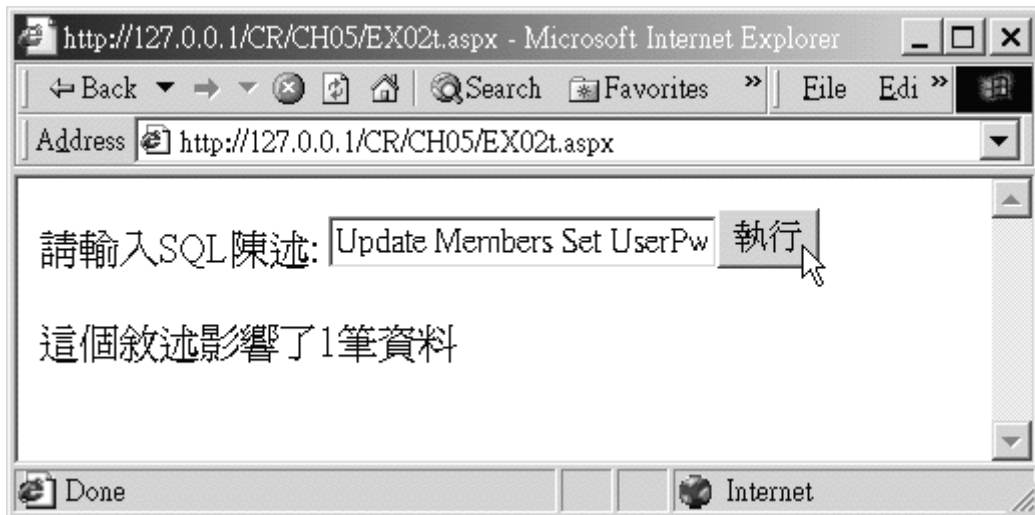
End Sub

</SCRIPT>

</Html>
```

例如我们将使用者 **tina** 密码修改成 **9876**，我们在文字输入盒输入下列叙述：

```
Update Members Set UserPwd= '9876' Where UserId='tina'
```



产生独立的 Command 对象

要使用 **Command** 对象，不一定要明确的宣告一个 **Connection** 对象。我们只要在产生 **Command** 对象的时候，将 **ActiveConnection** 参数所要指定的连结对象改成 **ConnectionString** 即可。不过独立的 **Command** 对象并不代表不需要 **Connection** 对象，而是 **Command** 对象会自动产生。在使用独立的 **Command** 对象之前要明确的将连结打开，我们可以利用 **ActiveConnection.Open()** 方法来开启连结，如下所示：

```
Dim cmA As ADOCommand = New _  
ADOCommand("SQL 陈述", "Provider=Microsoft.Jet.OLEDB.4.0;" & _  
    "Data Source=C:\Inetpub\wwwroot\cr\ch05\MyWeb.mdb;" & _  
    "User ID=Admin"  
cmA.ActiveConnection.Open() ' 将连结打开
```

下列范例我们利用控件将数据输入数据源：

```
<%@Import Namespace=System.Data.ADO%>

<%@Import Namespace=System.Data%>

<Form Id="Form1" Runat="Server">

    使用者账号： <Input type=text id="Text1" runat=server><br>

    使用者密码： <Input type=text id="Text2" runat=server><br>

    使用者姓名： <Input type=text id="Text3" runat=server><br>

    使用者电话： <Input type=text id="Text4" runat=server><br>

    使用者住址： <Input type=text id="Text5" runat=server><br>

    E-Mail 信箱： <Input type=text id="Text6" runat=server><br>

    <Button Id=Button1 Runat="Server"

OnServerClick="Button1_Click">确定

</Button>

</FORM>

<Span Id="Sp1" Runat="Server"/>

<Script Language="VB" Runat="Server">

Sub Button1_Click(Sender As Object, e As EventArgs)

    Dim strConStr As String = "Provider=Microsoft.Jet.OLEDB.4.0;" & _

        "Data Source=C:\InetPub\wwwroot\CR\CH05\MyWeb.mdb"
```

```
Dim cmA As ADOCommand = New ADOCommand("",strConStr)

CmA.ActiveConnection.Open()

cmA.CommandText="Insert Into Members Values(' " & _

    Text1.Value & "',' " & Text2.Value & "',' " & _

    Text3.Value & "',' " & Text4.Value & "',' " & _

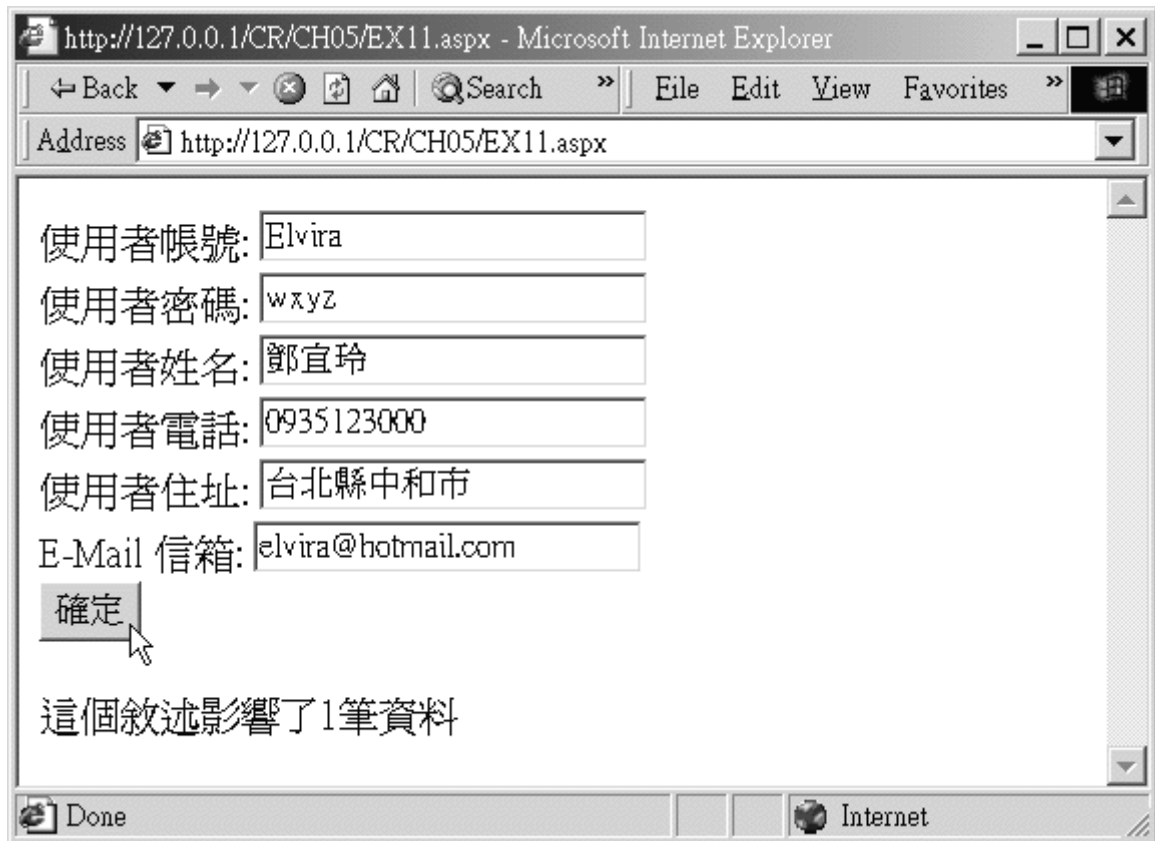
    Text5.Value & "',' " & Text6.Value & "')"

cmA.Execute()

Sp1.InnerText="这个叙述影响了" & cmA.RecordsAffected & "笔资料"

End Sub

</SCRIPT>
```

DataReader 物件

如果我们利用 **Command** 对象所执行的命令是有传回数据的 **Select** 叙述，此时 **Command** 对象会自动产生一个 **DataReader** 对象。**DataReader** 是我们写 **ASP.NET** 网页的好朋友，因为我们常常会将数据源的数据取出后显示给使用者，这时候我们就可以使用 **DataReader** 对象。我们就可以在执行 **Execute** 方法时传入一个 **DataReader** 型态的变量来接收。**DataReader** 对象很单纯的一次只读取一笔纪录，而且只能只读，所以效率很好而且可以降低网络负载。由于 **Command** 对象自动会产生 **DataReader** 对象，所以我们只要宣告一个指到 **DataReader** 对象的变量来接收

即可，并不需要使用 **New** 运算子来产生；另外要注意的是 **DataReader 对象只能配合 Command**

对象使用，而且 **DataReader** 对象在操作的时候 **Connection** 对象是保持联机的状态。下列程序

代码片段传回可以读取 **Members** 数据表中所有的纪录的 **DataReader** 对象：

```
Dim cmA As ADOCommand=New _  
    ADOCommand("命令字符串", "Provider=Microsoft.Jet.OLEDB.4.0;"  
& _  
    "Data Source=C:\Inetpub\wwwroot\cr\ch05\MyWeb.mdb")  
  
Dim drA as ADODataReader  
  
cmA.ActiveConnection.Open()  
  
cmA.CommandText="Select * From Members"  
  
cmA.Execute(drA)
```

当我们将 **DataReader** 对象传入 **Execute** 方法后，就可以使用 **DataReader** 对象来读取数据了。

以下为 **DataReader** 常用的属性：

属性	说明
FieldCount	只读，表示纪录中有多少字段
HasMoreResults	表示是否有多个结果，本属性和 SQL Script 搭配使用。
HasMoreRows	只读，表示是否还有资料未读取
IsClosed	只读，表示 DataReader 是否关闭

Item	只读，本对象是集合对象，以键值（Key）或索引值（Index）的方式取得纪录中某个字段的数据
RowFetchCount	用来设定一次取回多少笔记录，预设值为 1 笔

了解 **DataReader** 对象有什么属性后,我们就可以利用 **DataReader** 所提供的方法来取回资料了。

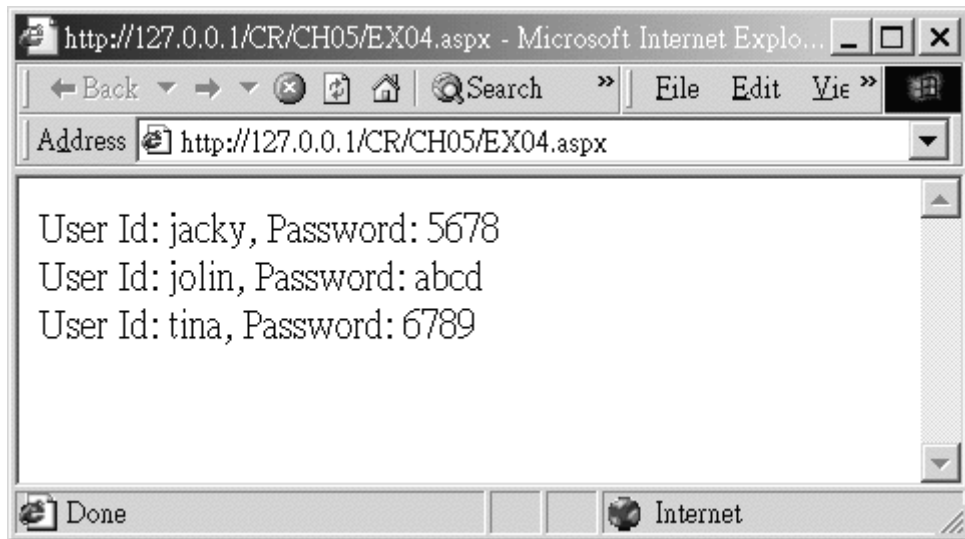
下表为 **DataReader** 常用的方法：

方法	说明
Close	将 DataReader 对象关闭
GetDataTypeName	取得指定字段的数据型态
GetName	取得指定字段的字段名称
GetOrdinal	取得指定字段名称在纪录中的顺序
GetValue	取得指定字段的数据
GetValues	取得全部字段的数据
IsNull	用来判断字段内是否为 Null 值
NextResult	用来和 SQL Script 搭配使用，表示取得下一个结果
Read	让 DataReader 读取下一笔记录，如果有读到数据则传回 True，若没有纪录则传回 False

Read 方法

在取得 **Command** 对象执行 **Execute** 方法所产生的 **DataReader** 对象后，我们就可以将纪录中的数据取出使用。**DataReader** 一开始并没有取回任何数据，所以我们要先使用 **Read** 方法让 **DataReader** 先读取一笔数据回来。如果 **DataReader** 对象成功取得数据则传回 **True**，若没有取得资料则传回 **False**。这样一来我们就可以利用 **Do While...Loop** 循环来取得所有的数据，如下程序所示：

```
Do While drA.Read()  
  
    Response.Write("User Id: " & drA.Item("UserId") & ", Password: ")  
  
    Response.Write(drA.Item(1) & "<br>")  
  
Loop
```



上述程序代码片段利用 **Read** 方法将数据取回后，再利用 **Item** 集合以键值（**Key**）的方式取出 **UserId** 字段的数据，以及利用索引值（**Index**）取得使用者 **UserPwd** 字段的数据；索引值是由 0 开始计数，故第一个字段的索引值为 0，依此类推。当数据读取完毕后 **Read** 方法会传回 **False**，所以就跳出循环。

GetValue 方法

我们也可以使用 **GetValue** 方法取得指定字段内的记录，这个方法和 **Item** 属性很像；不过 **GetValue** 方法的参数只接收索引值，并不接收键值为参数。我们改用 **GetValue** 取得所有字段内的数据，如下程序所示：

```
Do While drA.Read()  
  
    Response.Write("User Id: " & drA.GetValue(0) & ", Password: ")  
  
    Response.Write(drA.GetValue(1) & "<br>")
```

Loop

GetValues 方法

GetValues 方法是取得字段内所有的记录。这个方法接收一个数组，并且将所有字段填入数组中，

如下程序所示：

```
Dim arValue(drA.FieldCount)

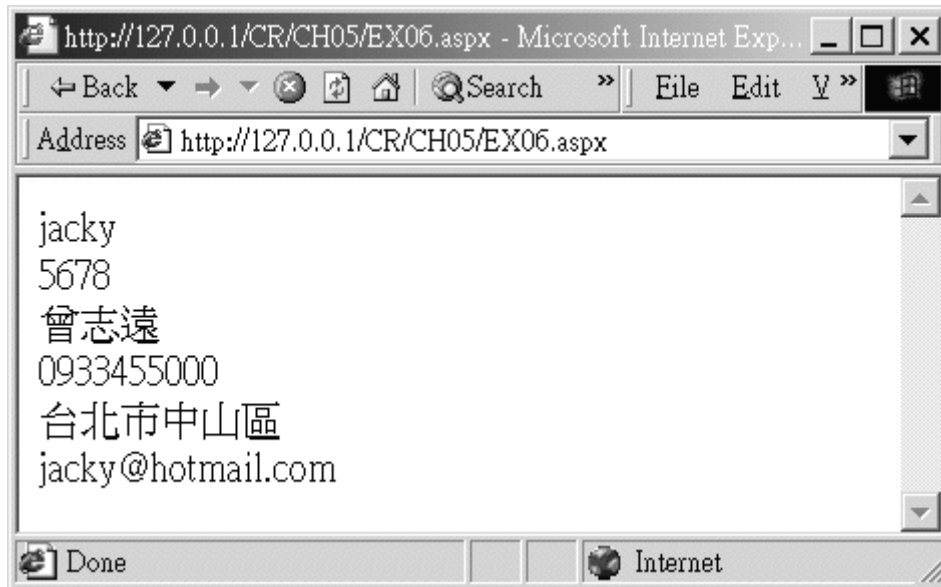
drA.Read() ' 先抓取一笔记录

drA.GetValues(arValue) ' 将记录填入数组中

For shtI=0 To drA.FieldCount - 1

    Response.Write(drA.GetValue(shtI) & "<br>")

Next
```



因为索引值是由零开始算，所以我们在使用 **For...Next** 循环的时候记得将结束值减一。

GetDataTypeName 以及 GetName 方法

GetDataTypeName 方法可以传回指定字段的数据型态，而 **GetName** 方法则是传回指定字段的字段名称（就是键值）。这两个方法一样以键值或是索引的方式来指定字段。下列程序代码片段显示每个字段的名称以及数据型态：

```
Dim shtI As Short

For shtI = 0 To drA.FieldCount - 1

    Response.Write("索引值为 " & shtI.ToString & " 的字段, 名称为: " & _
        DrA.GetName(shtI) & ", 数据型态: " & DrA.GetDataTypeName(shtI) &
        "<br>")
```

Next



Close 方法

Close 方法可以关闭 **DataReader** 对象和数据源之间的联机。除非把 **DataReader** 对象关闭，否则当 **DataReader** 对象尚未关闭时，**DataReader** 所使用的 **Connection** 对象就无法执行其它的动作。

综合范例

下列范例在文字输入盒内输入数据表名称，按下确定后程序会将数据表的索引、字段名称、字段型态以及字段内容全部显示出来：

```
<%@Import Namespace=System.Data.ADO%>

<%@Import Namespace=System.Data%>

<Html>

<Form Id="F1" Runat="Server">

    <Input Id="Text1" Runat="Server" Value="Members"/>

    <Button Id="Button1" OnServerClick="Button1_Click"

Runat="Server">执行

    </Button>

</Form>

<Script Language="VB" Runat="Server">

Sub Button1_Click(sender As Object, E As EventArgs)

Dim cmA As ADOCommand=New _

    ADOCommand("", "Provider=Microsoft.Jet.OLEDB.4.0;" & _

    "Data Source=C:\Inetpub\wwwroot\cr\ch05\MyWeb.mdb")

Dim drA As ADODataReader

Dim shtI As Short

cmA.ActiveConnection.Open()
```

```
cmA.CommandType=CommandType.TableDirect

cmA.CommandText=Text1.Value

cmA.Execute(drA)


For shtI=0 To drA.FieldCount - 1

    Response.Write("字段索引值: " & shtI.ToString & _

        " 字段名称: " & drA.GetName(shtI) & _

        " 数据型态: " & drA.GetDataTypeName(shtI) & "<br>")

Next

Do While drA.Read()

    For shtI=0 To drA.FieldCount-1

        Response.Write(drA.GetValue(shtI) & " / ")

    Next

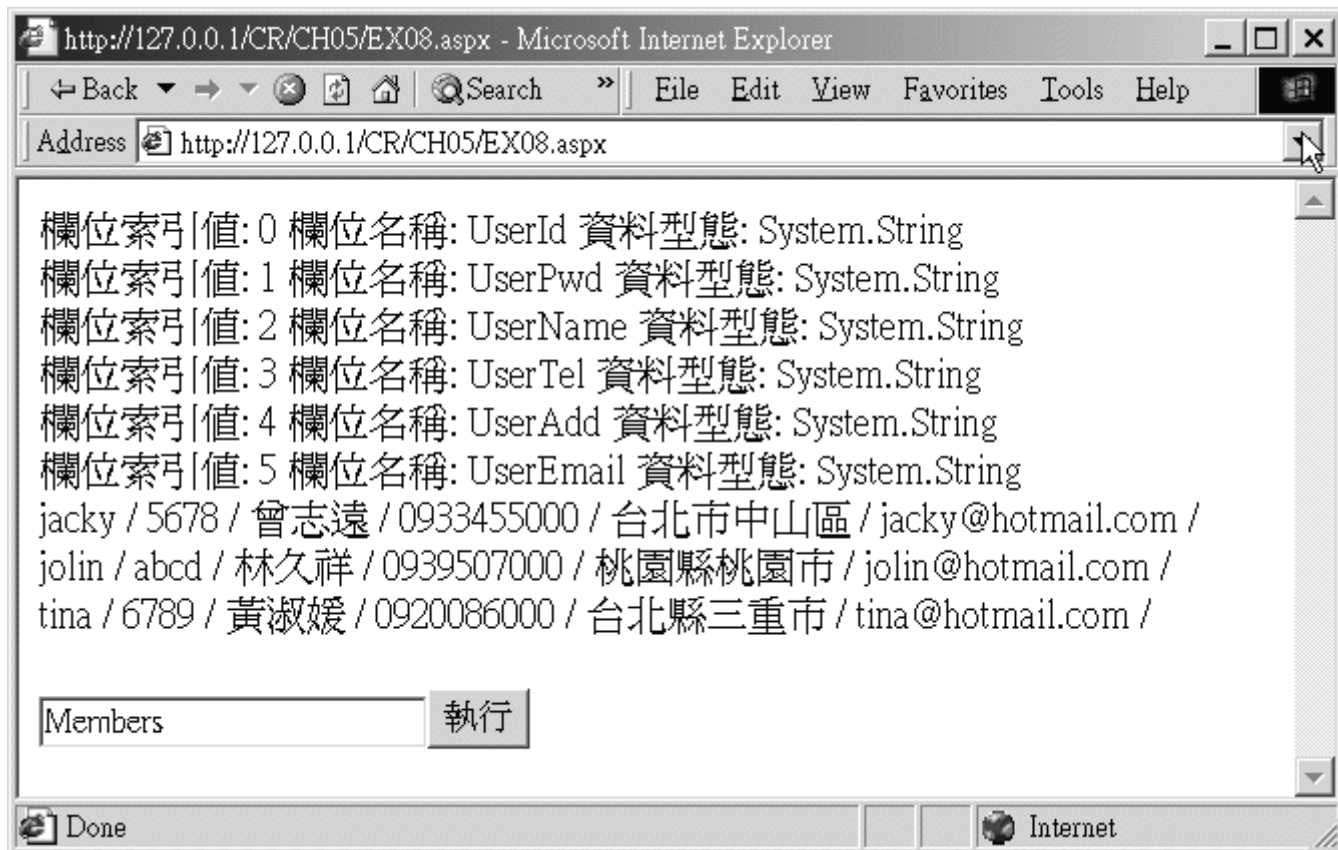
    Response.Write("<br>")

Loop

End Sub

</SCRIPT>

</Html>
```



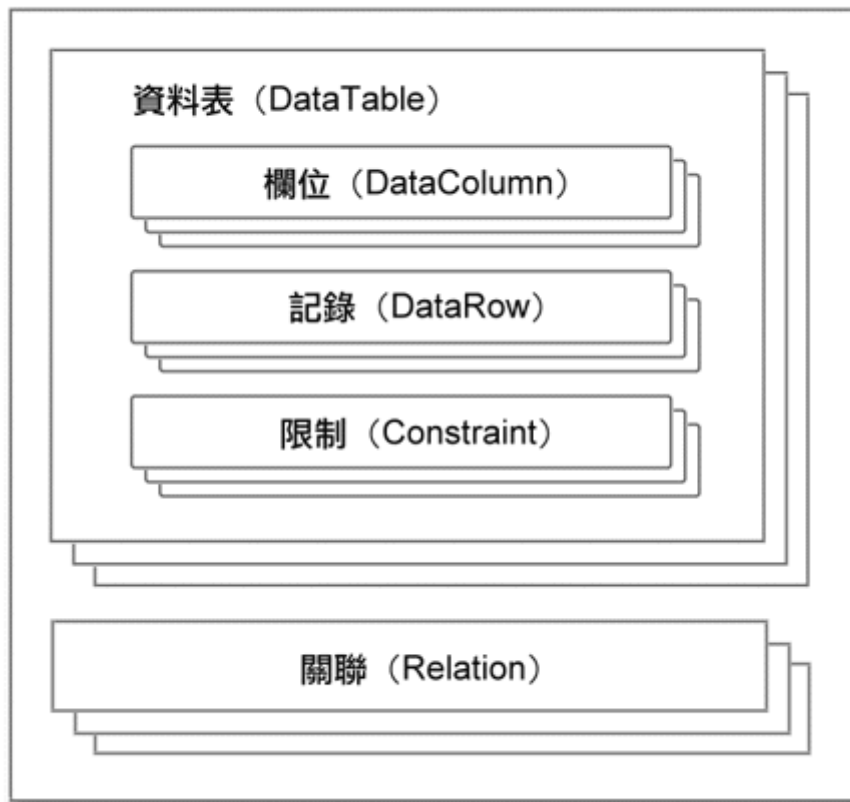
DataSet 物件

DataSet 物件是 ADO.NET 架构中非常重要的对象。我们可以把 DataSet 对象想象成是一个保留

从数据库取回数据的内存暂存区，这个暂存区可以用来群组以及管理资料表。DataSet 对象让我

们可以很灵活的操作数据表内的数据，它的架构如下图所示：

DataSet



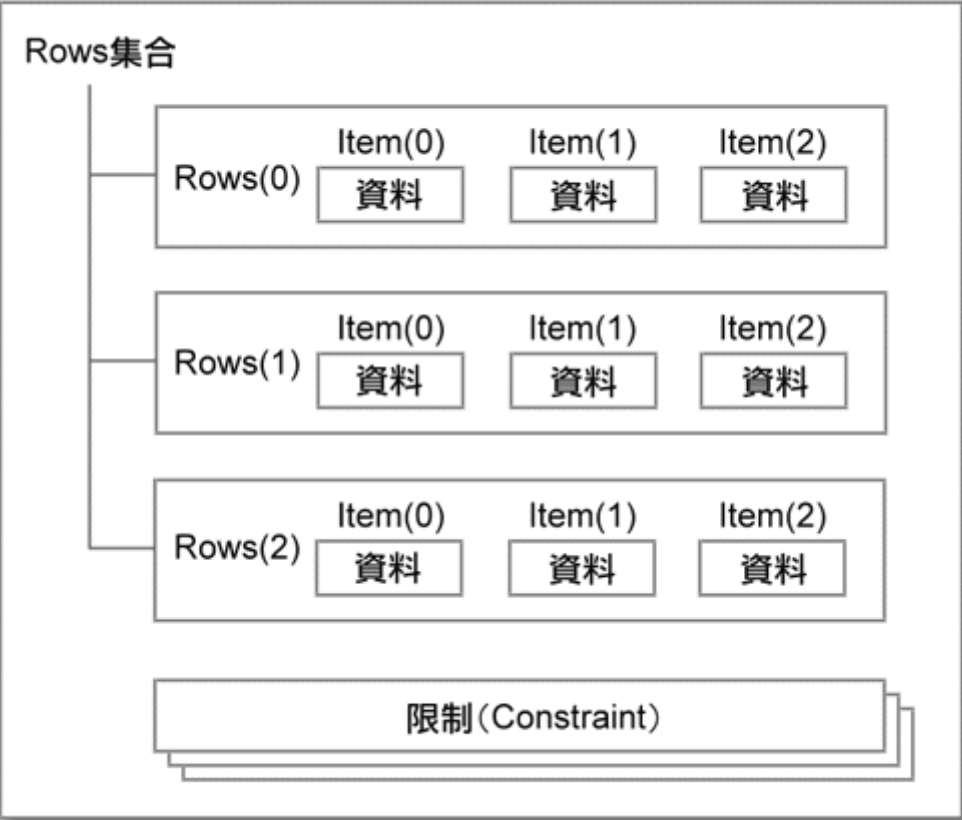
DataSet 对象是由许多数据表、数据表关联（Relation）、限制（Constraint）、记录（Row）以及字段（Column）对象的集合所组成；这意味着 DataSet 架构内所有的成员都非常对象化，可以让我们更有弹性的来操作这些对象。DataSet 对象本身没有和数据源联机的能力，它只是一个暂时存放数据的容器，数据的存取都是透过数据操作组件（Managed Providers）来执行；所以数据操作组件可以说是 DataSet 对象和数据源资间的沟通桥梁，没有数据作组件就无法从数据源取回数据。不过 DataSet 的数据可以不用透过数据操作组件对象从数据源取得，而可以利用程序自行设计产生，或是来自一般档案以及 XML 檔；这样一来 DataSet 对象的运用就更灵活

了。**DataSet** 对象基本上被设计成不和数据源一直保持联机的架构，也就是说和数据源的联机发生的很短暂，我们在透过 **DataSetCommand** 对象取得数据后就立即和数据源立即断线，等到数据修改完毕或是要操作数据源内的数据时才会再建立连结。这意味着程序和数据源要管理的连结就会变少，网络频宽不但可以得到舒缓外，服务器的负载也会减轻；所以多出来的网络频宽以及服务器资源，就可以额外服务其它需要服务的客户端。**DataSet** 内的数据可以从数据源取回，也可以自己产生。要使用 **DataSet** 对象，对 **DataSet** 内部的对象有相当程度的了解是非常重要的。

DataTable 物件

DataTable 是构成 **DataSet** 最主要的对象。**DataTable** 对象是由 **DataColumns** 集合以及 **DataRow** 集合所组成，我们透过数据控制组件将数据从数据源取回后，被取回的数据就是存放在 **DataTable** 对象中。我们也可以产生自订的资料表，只要先将数据表的字段定义好，就可以利用 **DataTable** 中 **DataRow** 集合对象的 **Add** 方法加入新的数据。**DataTable** 的对象模型如下图所示：

Table



以下是 **DataTable** 对象的常用属性：

属性	说明
CaseSensitive	表示执行字符串比较、搜寻以及过滤时是否分大小写
Columns	传回 DataTable 内的字段集合
Constraints	传回 DataTable 的限制集合
DataSet	传回 DataTable 对象所属 DataSet 名称
DefaultView	传回 DataTable 对象的视图，可用来排序、过滤及搜寻数据
Name	传回或设定数据表的 Name 属性

ParentRelations	传回 DataTable 对象的父关联集合
PrimaryKey	设定或传回字段在 DataTable 对象中的功能是否如主键
Rows	传回 DataTable 内的记录集合
TableName	设定或传回 DataTable 的名称

要产生 **DataTable** 对象，使用如下语法：

```
Dim 变数 As DataTable = New DataTable(["DataTable 名"])
```

我们在可以产生一个自订的 **DataTable** 对象，只要把数据表的 **DataColumn** 的属性设定好后，

就可以新增数据了；所以我们接下来介绍 **DataColumn** 对象。

DataColumn 物件

DataColumn 对象就是字段对象，是组成数据表的最基本单位。 **DataColumn** 有些属性可以帮我

们取得或设定 **DataTable** 中的 **DataColumn** 属性，如下表所示：

属性	说明
AllNull	传回或设定 DataColumn 是否接受 Null 值
AutoIncrement	传回或设定当加入 DataRow 时，否要自动增加字段值
AutoIncrementSeed	传回或设定 DataColumn 的递增种子

AutoIncrementSupported	传回 DataColumn 是否支持自动递增
Caption	传回或设定 DataColumn 的标题
ColumnName	传回或设定在 DataColumns 集合中字段的名称
DataType	传回或设定 DataColumn 的数据型态
DefaultValue	传回或设定 DataColumn 的默认值
Ordinal	传回字段集合中 DataColumn 的顺序
ReadOnly	传回或设定 DataColumn 是否为只读
Table	传回 DataColumn 所属 DataTable 对象的参考
Unique	传回或设定 DataColumn 是否不允许重复的数据

了解 **DataColumn** 有哪些属性后，我们先来产生一个 **DataTable**：

```
Dim dtTable As DataTable = New DataTable()

Dim dcColumn As DataColumn = New DataColumn()

dcColumn.ColumnName = "UserId" ' 设定字段名称

dcColumn.DataType = System.Type.GetType("System.String") ' 设定字段型态

dcColumn.AllowNull = False      ' 不允许空白

dtTable.Columns.Add(dcColumn) ' 将字段的定义加入 DataTable

' 对象的 Column 集合里
```



```

dcColumn = New DataColumn()      ' 再产生一个新的 DataColumn 对象

dcColumn.ColumnName = "UserPwd"    ' 设定字段名称

dcColumn.DataType = System.Type.GetType("System.String") ' 设定字段型态

dcColumn.AllowNull = False        ' 不允许空白

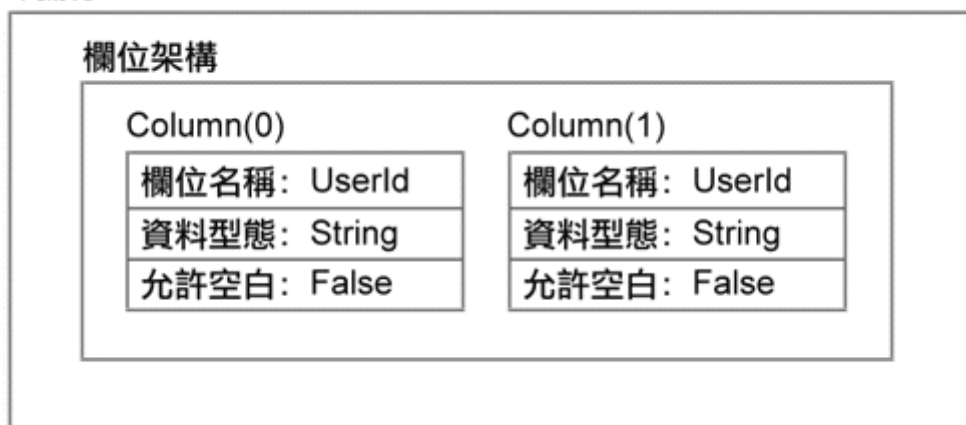

dtTable.Columns.Add(dcColumn) ' 将字段的定义加入 DataTable

' 对象的 Column 集合里

```

上述程序产生了一个自订的 **DataTable** 对象，分别有两个字段字符串型态，并且不允许不输入数据的 **UserId** 及 **UserPwd** 字段。这个 **DataTable** 的架构如下图所示：

Table



了解 **DataTable** 的属性之后，以下是 **DataTable** 对象的常用方法：

方法	说明
AcceptChanges	确定 DataTable 所作的改变
Clear	清除 DataTable 内所有的数据
NewRow	增加一笔新的记录

DataRow 物件

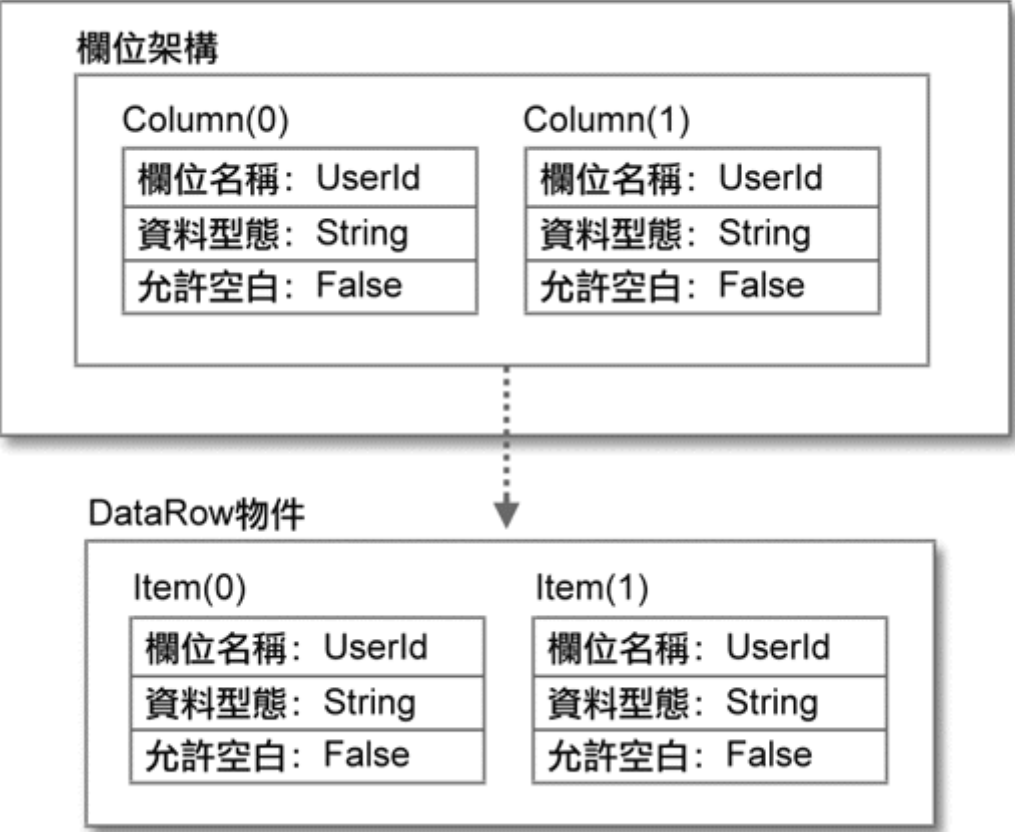
字段定义好之后，接下来我们就可以加入记录了。要为 **DataTable** 加入记录首先要先产生

DataRow 对象，这个对象是由 **DataTable** 的 **NewRow** 方法所产生，例如下列程序代码片段所示：

```
...  
  
Dim drRow As DataRow 'DataRow 是由 Table 产生，所以不需要使用 New 运算  
子  
  
drRow = dtTable.NewRow() 'dtTable 依照 Columnm 的架构产生 DataRow
```

我们利用 **DataTable** 产生 **DataRow** 时，**DataTable** 会依照 **Columns** 集合中的字段架构的定义来产生一个独立的 **DataRow** 对象。因为 **DataTable** 是依照字段的架构来产生 **DataRow** 对象，所以新产生的 **DataRow** 对象中会有一个和 **DataTable** 内的 **Columns** 集合架构一样的 **Columns** 集合：

Table



以下为 DataRow 对象常用的属性:

属性	说明
Item	传回或设定 DataRow 内 DataColumn 的数据
ItemArray	以数组的方式传回或设定所有 DataColumn 内容
RowState	传回或设定 DataRow 的状态

我们可以利用 **DataRow** 对象的 **Item** 属性来设定或传回纪录中字段的数据, 如下程序代码片段所示:

```
drRow.Item("UserId") = "Charles" ' 以传入 Key 来指定
```

或

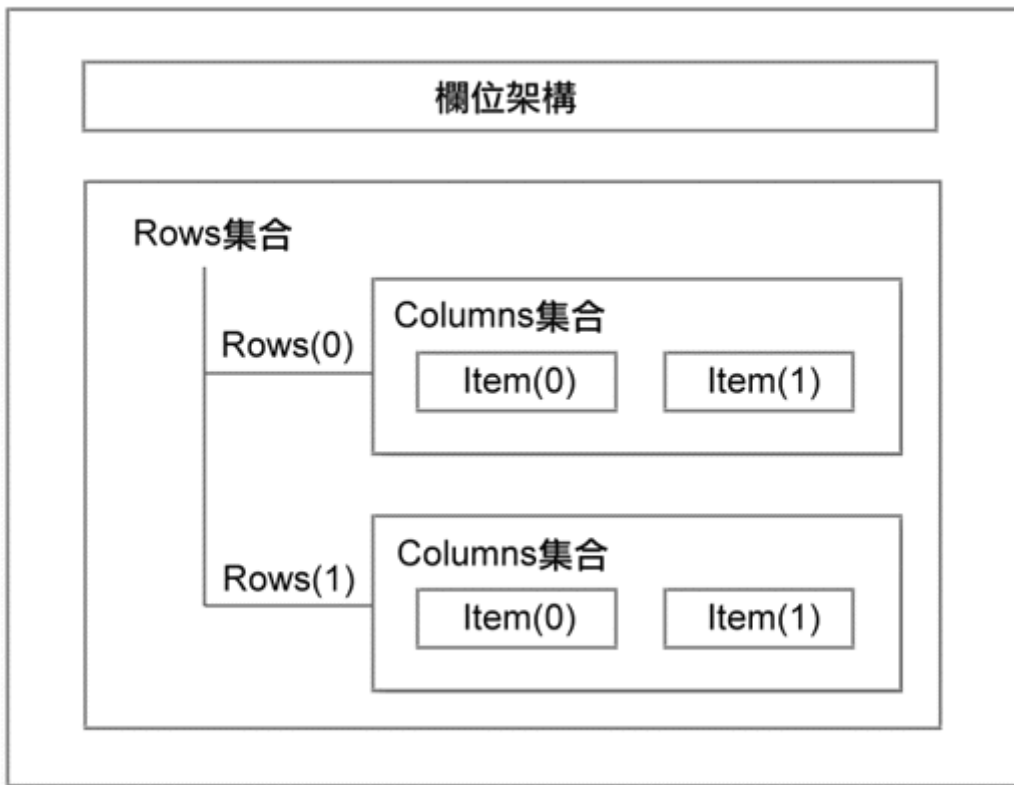
```
drRow(1) = "1234" ' 省略 Item 的简略写法, 并传入 Index 来指定
```

由于这个 **DataRow** 对象是独立的对象, **DataTable** 在产生 **DataRow** 时并没有将它加入自己的 **Rows** 集合内; 所以我们设定完 **DataRow** 对象中的数据后, 还必须使用 **DataTable** 对象中 **Rows** 集合的 **Add** 方法将 **DataRow** 加入到我们的 **DataTable** 内, 如下程序代码片段所示:

```
dtTable.Rows.Add(drRow) ' 将 DataRow 对象加入 DataTable 中
```

例如下图在 **DataTable** 中加入了两笔记录:

Table



在 **DataTable** 对象中有许多笔记录，每一笔记录中都有许多字段。要取得指定的记录可以利用

DataTable 对象中 **Rows** 集合的 **Item** 属性来指定。例如下列程序代码片段将第一笔记录的第一

个栏为值取回：

```
Dim strFiled As String
```

```
strField = dtTable.Rows.Item(0).Item(0) ' 将第一笔数据的第一个字段取回
```

或

```
strFiled = dtTable.Rows(0).Item(0) ' 省略 Rows 的 Item 属性也可以
```

或

```
strFiled = dtTable.Rows(0)(0) ' 省略全部的 Item 属性也接受
```

下列范例产生了上述架构的使用者自订 **DataTable** 对象，并填入三笔记录后，再将 **DataTable** 的内容显示出来：

```
<%@Import Namespace=System.Data.ADO%>

<%@Import Namespace=System.Data%>

<Script Language="VB" Runat="Server">

Sub Page_Load(Sender As Object, e As EventArgs)

    Dim dtTable As DataTable = New DataTable()

    Dim dcColumn As DataColumn = New DataColumn()

    Dim drRow As DataRow 'DataRow 是由 Table 产生，所以不需要使用
New 运算符

    Dim shtI As Short

    dcColumn.ColumnName = "UserId" ' 设定字段名称

    dcColumn.DataType = System.Type.GetType("System.String") ' 设定
字段型态

    dcColumn.AllowNull = False ' 不允许空白

    dtTable.Columns.Add(dcColumn) ' 将字段的定义加入 DataTable
```

’对象的 Column 集合里

dcColumn = New DataColumn() ’再产生一个新的 DataColumn 对象

dcColumn.ColumnName = "UserPwd" ’设定字段名称

dcColumn.DataType = System.Type.GetType("System.String") ’设

定字段型态

dcColumn.AllowNull = False ’不允许空白

dtTable.Columns.Add(dcColumn) ’将字段的定义加入 DataTable

’对象的 Column 集合里

For shtI = 0 To 2

drRow = dtTable.NewRow() ’dtTable 依照 Column 的架构产生

DataRow

drRow("UserId") = "账号 " & (shtI+1).ToString ’以 Key 来取得

drRow(1) = (shtI+1).ToString ’以 Index 来取得

dtTable.Rows.Add(drRow)

Next

For shtI = 0 To 2

Response.Write(dtTable.Rows(shtI)("UserId") & _

", 密码: " & dtTable.Rows(shtI)("UserPwd") & "
")

Next

End Sub

</SCRIPT>

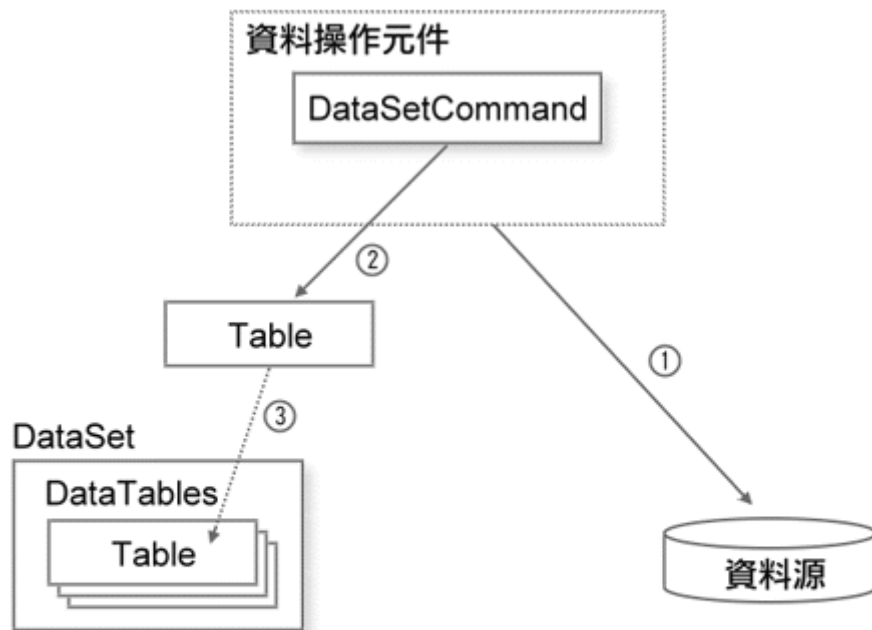


DataSet 对象与 DataSetCommand 对象

DataSet 对象与 DataSetCommand 对象合作的基本原理

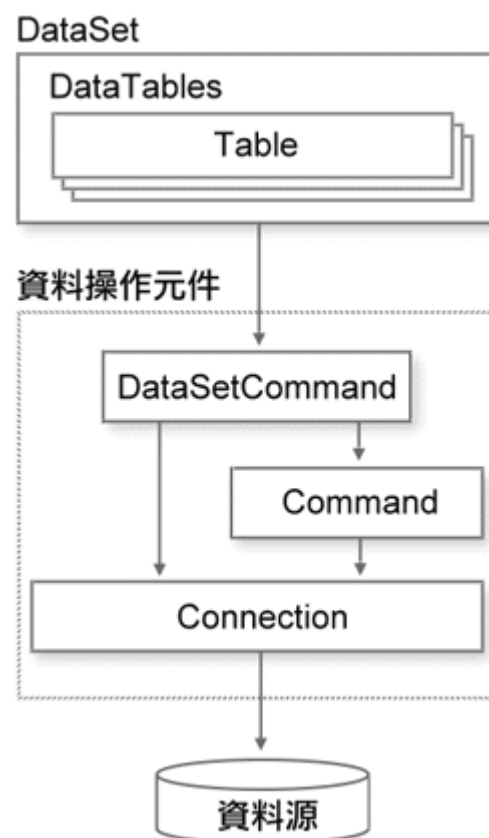
DataSet 对象可以让我们群组以管理 DataTable 对象，我们可以把它看成是暂时存放资料的地方，它本身并不具备和数据源联机以及操作数据源的能力。如果想要将数据源的数据取回并存放在 DataSet 里面的 DataTable 中，要透过数据操作组件才办的到。数据操作组件（Managed Provider）就是 Connection、Command、DataSetCommand 以及 DataReader 对象，其中 DataSet

对象和 **DataSetCommand** 对象的关系最密切，因为 **DataSetCommand** 对象是帮助 **DataSet** 对象和数据源沟通的桥梁；我们透过 **DataSetCommand** 对象来取得数据源的数据时，它会帮我们先依照数据在数据源中的架构产生一个 **DataTable** 对象，然后将数据源中的数据取回后填入 **DataRow** 对象，再将 **DataRow** 对象填入 **DataTable** 的 **Rows** 集合，直到数据源中的数据取完为止。**DataSetCommand** 对象将数据源中的数据取出，并将这些数据都填入自己所产生的 **DataTable** 对象后，立即将这个 **DataTable** 对象加入 **DataSet** 对象的 **DataTables** 集合，并结束和数据源的联机。**DataSetCommand** 对象的执行流程如下图所示：



- ① 由資料操作元件建立和資料源的連結
- ② **DataSetCommand** 依原資料結構建立 **DataTable** 物件，並將取回的資料填入
- ③ 資料取回完畢後，將 **DataTable** 加入 **DataSet** 的 **DataTables** 集合

我们来详细了解数据操作组件的实际动作：首先由 **Connection** 建立和数据源的联机，然后由 **DataSetCommand** 对象透过 **Command** 对象下达将数据取回的命令。这些命令透过 **Connection** 对象送至数据源后，数据源会将我们所要取得的数据透过 **Connection** 对象传回给 **DataSetCommand**，**DataSetCommand** 将这些数据填入自己产生的 **DataTable** 对象。全部的数据取回后，再把这个由 **DataSetCommand** 对象所产生的 **DataTable** 对象，加入 **DataSet** 中的 **DataTables** 集合对象来统一管理。**DataSet** 对象从到尾都没有主动和数据源有任何互动，这些命令的下达以及数据的传递都是透过数据操作组件而完成的；如下图所示：



DataSetCommand 对象读取以及更新数据的方式

我们可以利用 **DataSetCommand** 对象来执行下列的工作：

1. 将数据源的记录取回，并植入 **DataSet** 对象作管理。

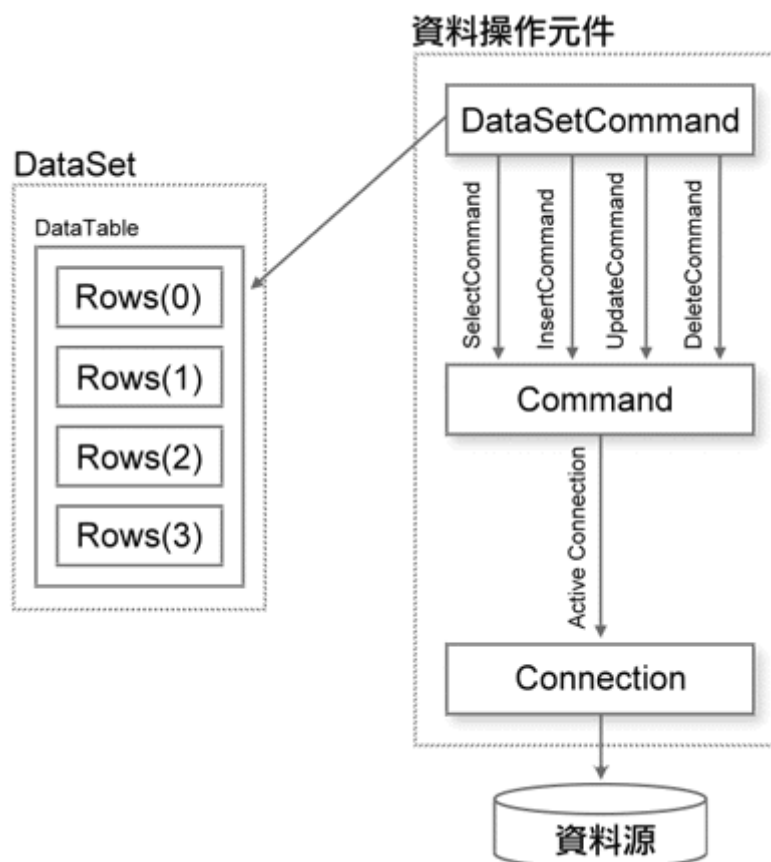
我们可以利用 **DataSetCommand** 对象的 **FillDataSet** 方法来将取得的数据填入 **DataSet** 对象中。当我们执行这个方法的时候，它会将 **SQL Select** 的叙述送至数据源。

2. 将 **DataTable** 的内容传回数据源。

要将 **DataSet** 中的 **DataTable** 对象所作的变更传回数据源作更新，我们可以使用 **DataSetCommand** 对象的 **Update** 方法。当我们使用这个方法时，它会将所需要的 **SQL Insert**、**Update** 或是 **Delete** 传回数据源。**Update** 这个方法会检查每一个 **DataRow** 的状态，若 **DataRow** 是新增加的，该方法就下达 **Insert** 的 **SQL** 命令；若 **DataRow** 有被修改过，该方法就下达 **Update** 的 **SQL** 叙述；若 **DataRow** 被删除，则下达 **Delete** 的 **SQL** 叙述。

DataSetCommand 操作数据源的属性

所以 DataSetCommand 中有四个属性，而这四个属性其实都是 Command 对象；分别是 SelectCommand、InsertCommand、UpdateCommand 以及 DeleteCommand 属性。虽然我们可以明确宣告 DataSetCommand 中这些对资料源执行更新动作的 Command 对象，并设定好该 Command 对象的 CommandText 属性，并指定适当的 SQL 叙述来达到对数据源的 Insert、Update 以及 Delete 等目的；但是实际上 DataSetCommand 对象会自动产生它所需要的 SQL 陈述，并不需要我们特别指定。



例如我们将数据从数据源取回，放到 **DataSet** 对象中的 **DataTable** 对象，其数据表内容如下表所示：

记录状态 (RowState)	UserId	UserPwd	UserName	UserTel
未改变 (Unchanged)	tina	6789	黄淑媛	0920086000
未改变 (Unchanged)	jacky	5678	曾志远	0933455000
未改变 (Unchanged)	jolin	abcd	林久祥	0939507000

其中 **DataRow** 对象中有一个用来表示记录内的数据有无改变的 **RowState** 属性，预设都是未改变（**Unchanged**）。假设程序将 **jolin** 的 **UserTel** 字段内容改掉，其字段状态就会变成已改变（**Modified**），如下表所示：

记录状态 (RowState)	UserId	UserPwd	UserName	UserTel
未改变 (Unchanged)	tina	6789	黄淑媛	0920086000
未改变 (Unchanged)	jacky	5678	曾志远	0933455000
已改变 (Modified)	jolin	abcd	林久祥	0955520000

当我们使用 **DataSetCommand** 对象的 **Update** 方法，将 **DataSet** 的状态更新回数据源时，
DataSetCommand 对象会去检查 **DataTable** 中每一笔记录的 **RowState**。当 **DataSet** 对象检查
第一笔和第二笔时，并不会产生任何 **SQL** 陈述，因为 **RowState** 属性标示为未改变 (**Unchanged**)；

当检查到第三笔时，因为 **RowState** 标示为已改变（**Modified**），**Update** 方法会自动产生适当的 **SQL** 叙述并且传送到数据源。

使用 **DataSetCommand** 对象

DataSetCommand 对象可以说是 **DataSet** 对象的工作引擎，**DataSet** 和数据源的互动都是由 **DataSetCommand** 对象来执行；而 **DataSetCommand** 则是控制 **Command** 对象透过 **Connection** 对象对数据源下命令，和数据源进行互动的工作。以下为 **DataSetCommand** 的宣告语法：

```
Dim 变数 As DataSetCommand = New DataSetCommand()
```

我们先来了解 **DataSetCommand** 对象和其它数据操作对象如何搭配使用：

```
Dim strConStr As String = "Provider=Microsoft.Jet.OLEDB.4.0;" & _  
    "Data Source=C:\InetPub\wwwroot\CR\CH05\MyWeb.mdb"  
  
Dim strComStr As String = "Select * From Members"  
  
Dim cnA As ADOConnection = New ADOConnection(strConStr) '宣告及产生  
Connection 对象  
  
Dim cmA As ADOCommand = New ADOCommand(strComStr) '宣告及产生 Command  
对象  
  
Dim dscA As ADODataSetCommand = New ADODataSetCommand() '产生  
DataSetCommand
```

```
cmA.ActiveConnection = cnA    ' 指定 Command 对象 cmA 要透过 cnA 这个
Connection 对象下命令

dscA.SelectCommand = cmA      ' 指定 DataSetCommand 要从数据源取回数据
要透过 cmA 这个

    ' Command 物件来对 Connection 下达命令
```

其中在使用 **New** 运算符建构 **DataSetCommand** 时，也可以顺便作初值设定的工作，如下语法所示：

```
Dim 变量 As DataSetCommand = New DataSetCommand(Command 对象名)
```

例如下列范例于宣告 **Command** 对象时，直接指定 **Command** 对象所要执行的命令，以及要透过哪个 **Connection** 对象；并在宣告 **DataSetCommand** 时，直接指定所要使用的 **Command** 对象名称：

```
Dim strConStr As String = "Provider=Microsoft.Jet.OLEDB.4.0;" & _
    "Data Source=C:\InetPub\wwwroot\CR\CH05\MyWeb.mdb"

Dim strComStr As String = "Select * From Members"

Dim cnA As ADONConnection = New ADONConnection(strConStr)

Dim cmA As ADONCommand = New ADONCommand(strComStr, cnA)

Dim dscA As ADONDataSetCommand = New ADONDataSetCommand(cmA)
```

甚至还可以不需要明确宣告 **Connection** 对象以及 **Command** 对象，直接以命令文字以及联机字符串来代替，如下语法所示：

```
Dim 变量 As DataSetCommand = New DataSetCommand("命令字符串", "联机字符串")
```

例如下列范例于宣告 **DataSetCommand** 对象时，直接指定 **DataSetCommand** 对象所要执行的命令，以及如何建立 **Connection** 对象和数据源连结。我们在使用这个 **DataSetCommand** 对象时，它会自动建立并管理 **Command** 对象以及 **Connection** 对象：

```
Dim strConStr As String = "Provider=Microsoft.Jet.OLEDB.4.0;" & _  
    "Data Source=C:\InetPub\wwwroot\CR\CH05\MyWeb.mdb"  
  
Dim strComStr As String = "Select * From Members"  
  
Dim dscA As ADODatasetCommand = New ADODatasetCommand(strComStr,  
strConStr)
```

这样程序就清楚多了。接下来我们就要利用 **DataSetCommand** 从数据源取回数据，并填入 **DataSet** 对象。以下为宣告的语法：

```
Dim 变量 As DataSet = New DataSet(["DataSet 名称"])
```

要从数据源取回数据并填入 **DataSet** 对象，我们利用 **DataSetCommand** 对象的 **FillDataSet** 方法。以下为 **FillDataSet** 方法的语法：


```
DataSetCommand.FillDataSet(DataSet, "DataTable 名称")
```

我们利用 **DataSetCommand** 和数据源联机，它会自动管理 **Connection** 对象以及 **Command** 对象，所以 **DataSetCommand** 使用的 **Connection** 对象并不需先用 **Open** 方法打开。我们在呼叫 **DataSetCommand** 对象的 **FillDataSet** 时，如果 **Connection** 对象没有开启和数据源的连结，**DataSetCommand** 对象会自动呼叫 **Connection** 对象的 **Open** 方法将对数据源的连结打开；**DataSetCommand** 对数据源的操作执行完毕后，会自动将 **Connection** 对象和数据源的连结利用 **Connection** 对象的 **Close** 方法关闭。如果 **DataSetCommand** 在执行 **FillDataSet** 方法时 **Connection** 对象已经开启连结，在执行完毕后 **DataSetCommand** 会维持 **Connection** 对象原来开启连结的状况。下列范例使用 **DataSetCommand** 对象从数据源撷取数据回来，并填入 **DataSet** 对象中：

```
<%@Import Namespace=System.Data.ADO%>

<%@Import Namespace=System.Data%>

<Script Language="VB" Runat="Server">

Sub Page_Load(Sender As Object, e As EventArgs)

    Dim strConStr As String = "Provider=Microsoft.Jet.OLEDB.4.0;" & _
        "Data Source=C:\InetPub\wwwroot\CR\CH05\MyWeb.mdb"

    Dim strComStr As String = "Select * From Members"

    Dim dscA As ADODataSetCommand = New
ADODataSetCommand(strComStr, strConStr)
```

```

Dim dsDataSet As DataSet = New dataset()

dscA.FillDataSet(dsDataSet, "Members") '将数据填入数据表内，并取
名为 Members

Dim shtI As Short

For shtI=1 To (dsDataSet.Tables("Members").Rows.Count).ToInt16

    Response.Write(dsDataSet.Tables(0).Rows(shtI-1)("UserName") &
"<BR>")

Next

End Sub
</SCRIPT>

```

上述范例将数据将所取回的 **DataTable** 对象填入 **DataSet** 对象中的 **Tables** 集合，我们可就可以利用 **Index** 或是 **DataTable** 名称的方式来取出集合中的对象。取出 **DataTable** 对象后，我们可以利用 **DataTable** 中 **Rows** 集合的 **Count** 属性取得总共有几笔记录，并将这些记录全部显示出来。由于 **Rows** 集合是由 0 开始计算，所以我们最后一个 **DataRow** 对象的 **Index** 值总是比 **Count** 属性少 1。

从资料源取回第二个 **DataTable**

取回第一个数据表后要从数据源再取回第二个数据表，要修改 **DataSetCommand** 对象中

SelectCommand 属性的 **CommandText** 属性，将 **CommandText** 的内容改成我们要从数据源取

回数据的叙述。因为 **SelectCommand** 属性本身就是 **Command** 对象，所以设定 **SelectCommand**

的方式和设定 **Command** 对象的方法一样；修改完毕后就可以再利用 **DataSetCommand** 对象的

FillDataSet 方法将数据取回。下列范例将 **MyWeb** 数据库中的 **Members** 以及 **Orders** 数据表取

回：

```
<%@Import Namespace=System.Data.ADO%>

<%@Import Namespace=System.Data%>

<Script Language="VB" Runat="Server">

Sub Page_Load(Sender As Object, e As EventArgs)

    Dim strConStr As String = "Provider=Microsoft.Jet.OLEDB.4.0;" & _
        "Data Source=C:\InetPub\wwwroot\CR\CH05\MyWeb.mdb"

    Dim strComStr As String = "Select * From Members"

    Dim dscA As ADODatasetCommand = New
ADODatasetCommand(strComStr, strConStr)

    Dim dsDataSet As DataSet = New dataset()

    dscA.FillDataSet(dsDataSet, "Members")

    dscA.SelectCommand.CommandText = "Select * From Orders"

    dscA.FillDataSet(dsDataSet, "Orders")
```

```
End Sub
```

```
</SCRIPT>
```

上述程序代码将数据从数据源取回后，填入名为 **dsDataSet** 的 **DataSet** 物件中。第一个 **DataTable**

在 **Tables** 集合中的 **Index** 值为 0，而第二个填入的 **DataTable** 在 **Tables** 集合中的 **Index** 值为 1。

DataSet



修改数据并更新回数据源

在处理数据的时候，**DataRow** 对象会自动记录目前记录的状况。只要记录一有改变便作笔记，等呼叫 **DataSetCommand** 对象的 **Update** 方法时，**DataSetCommand** 会自动产生适当的 SQL 叙述将修改更新至数据源。以下为 **Update** 方法的语法：

```
DataSetCommand.Update(DataSet, "DataTable 名称")
```

下列程序代码片段，显示如何更改 **DataSet** 对象中 **DataTable** 对象里的数据：

```
Dim dtTable As DataTable

dtTable=dsDataSet.Tables("Members")    ' 方便程序写作起见，先将
DataTable

        ' 的参考取回并由 dtTable 对象管理

dtTable.Rows.Item(0).Item(0) = 资料    ' 或 dtTable.Rows(0).Item(0) =
资料

        ' 或 dtTable.Rows(0)(0) = 资料

dscA.Update(dsDataSet, "Members")    ' 利用 DataSetCommand 对象的

        ' Update 方法将改变更新至数据源
```

上述程序代码叙述在 **DataSetCommand** 对象呼叫 **Update** 方法时，会自动产生适当的 SQL 叙述来执行更新的动作。接下来我们来作一个完整的范例，这个范例一开始会先将所有会员数据显示出来，并准备一些文字输入盒。我们可以在文字输入盒内输入所要修改的记录编号、字段名称以及要替代的新值，输入完毕后可按确定按钮执行更新的工作：

```

<%@Import Namespace=System.Data%>

<%@Import Namespace=System.Data.ADO%>

<html>

<FORM id=Form1 Runat="Server" Action=""Ex15.aspx"">

    要修改的序号: <INPUT Type="Text" Id=Text1 Runat="Server"><br>

    要修改的字段: <INPUT Type="Text" Id=Text2 Runat="Server"><br>

    要修改的新值: <INPUT Type="Text" Id=Text3 Runat="Server">

    <INPUT Type="Submit" Value="确定">

</FORM>

<Table Id="Table1" Runat="Server" Border="1"/>

<Script Language="VB" Runat="Server">

Sub Page_Load(Sender As Object, e As EventArgs)

    Dim strConStr As String = "Provider=Microsoft.Jet.OLEDB.4.0;" & _
        "Data Source=C:\InetPub\wwwroot\CR\CH05\MyWeb.mdb"

    Dim strComStr As String = "Select * From Members"

    Dim dscA As ADODatasetCommand = New ADODatasetCommand(strComStr,
strConStr)

    Dim dsDataSet As DataSet = New DataSet()

    dscA.FillDataSet(dsDataSet, "Members")      ' 将使用者数据填入
DataSet 对象中

```

```

If IsPostBack Then

    If Text1.Value<>Nothing And Text2.Value<>Nothing _
        And Text3.Value<>Nothing Then ' 确定文字输入盒都
有数据

        dsDataSet.Tables("Members").Rows((Text1.Value).ToInt16 -
1) (Text2.Value) _
            = Text3.Value

        dscA.Update(dsDataSet, "Members")

    End If

End If

AddTableToTable(dsDataSet) ' 呼叫将 DataTable 填入 HtmlTable 的副程序
End Sub

Sub AddTableToTable(ByRef dsDataSet As DataSet)

    Dim Cell As HtmlTableCell

    Dim Row As HtmlTableRow

    Dim X,Y As Short

    Row=New HtmlTableRow

    Cell=New HtmlTableCell ' 将 序号 这一列的

```

```

Cell.InnerText="序号"      ' 名称加入 Row 对象中

Row.Cells.Add(Cell)

For Y=0 To dsDataSet.Tables("Members").Columns.Count-1      ' 将各字
段

    Cell=New HtmlTableCell      ' 名称加入

    Cell.InnerText=dsDataSet.Tables(0).Columns(Y).ColumnName

    ' Row 物件

    Row.Cells.Add(Cell)

Next

Table1.Rows.Add(Row) ' 将显示字段名称的第一列加入表格中


For X=0 To dsDataSet.Tables("Members").Rows.Count-1 ' 将所有 Members
的记录取出

    Row=New HtmlTableRow

    Cell=New HtmlTableCell

    Cell.InnerText=X+1      ' 将记录的顺序加入第一栏

    Row.Cells.Add(Cell)      ' 并加入 Row 对象的 Cells 集合


    For Y=0 To dsDataSet.Tables("Members").Columns.Count-1      ' 将记录
中所有

```



```
Cell=New HtmlTableCell          ' 字段的
数据取

Cell.InnerText=dsDataSet.Tables(0).Rows(X)(Y) ' 出,并
放入 Row

Row.Cells.Add(Cell)             ' 物件中

Next

Table1.Rows.Add(Row)            ' 将完成的一笔记录加入 Table 对象的 Rows
集合中

Next

End Sub

</SCRIPT>

</html>
```

http://127.0.0.1/CR/CH05/EX11t.aspx - Microsoft Internet Explorer

Back Forward Stop Home Search File Edit View Favorites Tools Help

Address http://127.0.0.1/CR/CH05/EX11t.aspx

要修改的序號:

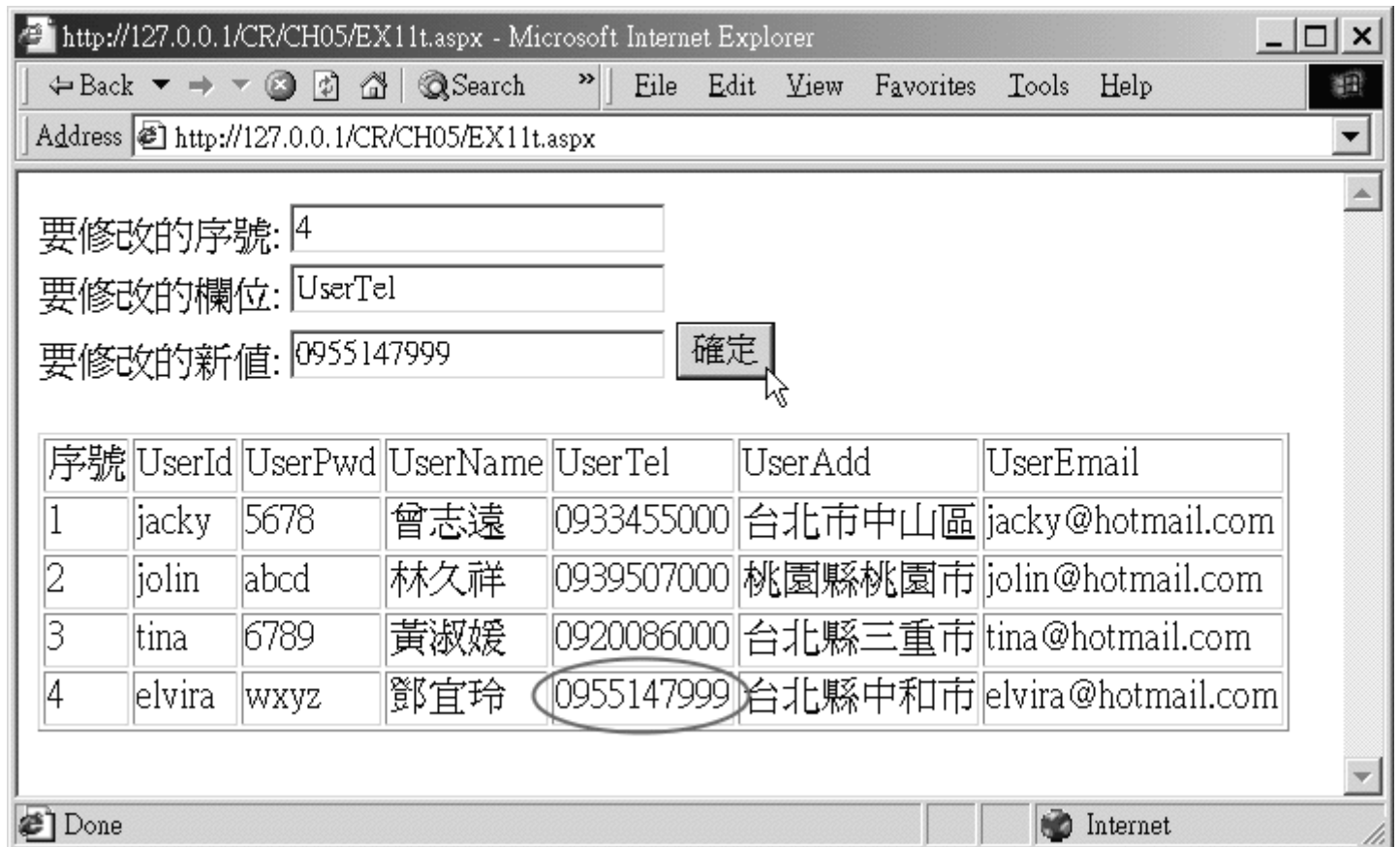
要修改的欄位:

要修改的新值:

序號	UserId	UserPwd	UserName	UserTel	UserAdd	UserEmail
1	jacky	5678	曾志遠	0933455000	台北市中山區	jacky@hotmail.com
2	jolin	abcd	林久祥	0939507000	桃園縣桃園市	jolin@hotmail.com
3	tina	6789	黃淑媛	0920086000	台北縣三重市	tina@hotmail.com
4	elvira	wxyz	鄧宜玲	0935123000	台北縣中和市	elvira@hotmail.com

Done Internet

数据尚未修改前



数据修改后

定义数据表之间的关联

我们之前在产生 **Members** 数据表以及 **Orders** 数据表时，也定义了这两个数据表之间的关联。

如果我们希望维持资料表之间的关系，在数据表取回后可以将数据表的关联加入 **DataSet** 对象

的 **Relations** 集合中。以下为 **Relation** 集合加入关联的语法：

```
DataSet.Relations.Add("关联名称", 父关联主键字段, 子关联外来键字段)
```

其中两个数据表的关联关系，是由父数据表的主键以及子数据表的外来键字段所组成的，这两个字段分别是 **Members** 数据表的主键 **UserId** 以及 **Orders** 数据表的外来键 **UserID**；所以这两个取回来的 **DataTable** 要建立关联，也是要透过这两个字段。下列范例建立了 **Members** 以及 **Orders** 数据表之间的关联：

```
<%@Import Namespace=System.Data.ADO%>

<%@Import Namespace=System.Data%>

<Script Language="VB" Runat="Server">

Sub Page_Load(Sender As Object, e As EventArgs)

    Dim strConStr As String = "Provider=Microsoft.Jet.OLEDB.4.0;" & _
        "Data Source=C:\InetPub\wwwroot\CR\CH05\MyWeb.mdb"

    Dim strComStr As String = "Select * From Members"

    Dim dscA As ADODataSetCommand = New ADODataSetCommand(strComStr,
strConStr)

    Dim dsDataSet As DataSet = New DataSet()

    dscA.FillDataSet(dsDataSet, "Members")

    dscA.SelectCommand.CommandText = "Select * from Orders"

    dscA.FillDataSet(dsDataSet, "Orders")

    dsDataSet.Relations.Add("MO",
dsDataSet.Tables("Members").Columns("UserId"), _
```

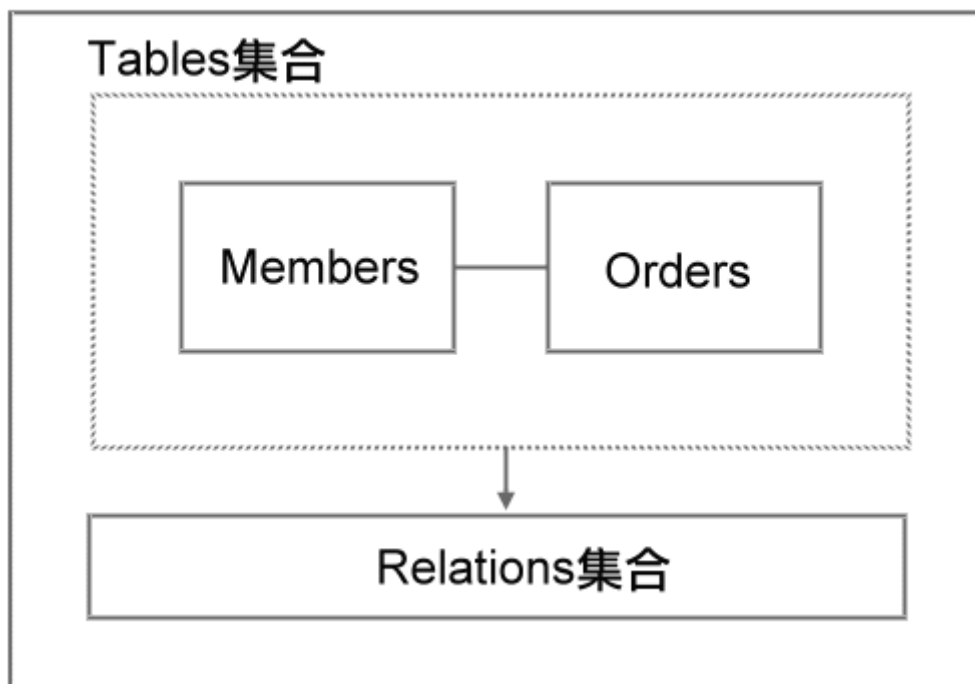
```
dsDataSet.Tables("Orders").Columns("UserId"))
```

```
End Sub
```

```
</SCRIPT>
```

这样这两个 **DataTable** 的关联就建立完成了，并且由 **DataSet** 中的 **Relations** 集合对象所管理。

DataSet



利用关联将子关联记录取出

关联建立之后，我们就可以享受到关联的好处了。我们可以透过 **DataRow** 对象的 **GetChildRows**

以及 **GetParentRow** 方法取得记录的子记录或是父记录；这个方法接收的一个参数，就是我们加

到关联集合的关联名称。下列范例将每一个使用者所下过的订单列出：

```
<%@Import Namespace=System.Data.ADO%>

<%@Import Namespace=System.Data%>

<Script Language="VB" Runat="Server">

Sub Page_Load(Sender As Object, e As EventArgs)

    Dim strConStr As String = "Provider=Microsoft.Jet.OLEDB.4.0;" & _
        "Data Source=C:\InetPub\wwwroot\CR\CH05\MyWeb.mdb"

    Dim strComStr As String = "Select * From Members"

    Dim dscA As ADODatasetCommand = New ADODatasetCommand(strComStr,
strConStr)

    Dim dsDataSet As DataSet = New DataSet()

    dscA.FillDataSet(dsDataSet, "Members")

    dscA.SelectCommand.CommandText = "Select * from Orders"

    dscA.FillDataSet(dsDataSet, "Orders")

    dsDataSet.Relations.Add("MO",
dsDataSet.Tables(0).Columns("UserId"), _
```

```

dsDataSet.Tables(1).Columns("UserId"))

    Dim shtI As Short

    Dim rowTemp As DataRow

    For shtI=0 To dsDataSet.Tables("Members").Rows.Count-1

        Response.Write("使用者: " &

dsDataSet.Tables("Members").Rows(shtI)("UserId") & _

            " 所下过的订单有:")

        For Each rowTemp In

dsDataSet.Tables("Members").Rows(shtI).GetChildRows("MO")

            Response.Write("<br>订单日期: " & rowTemp("OrderDate") & _

                " / 订购产品: " & rowTemp("ProductName") & _

                " / 产品单价: " & rowTemp("UnitPrice") & _

                " / 订购数量: " & rowTemp("Quantity") & _

                " / 小    计: " & rowTemp("Total"))

        Next

        Response.Write("<p>")

    Next

End Sub

```

</SCRIPT>



For Each...In 循环结构

上述范例使用了 For Each...In 的循环结构，如下所示：

```
For Each rowTemp In
```

```
dsDataSet.Tables("Members").Rows(shtI).GetChildRows("MO")
```

```
Response.Write("<br>订单日期: " & rowTemp("OrderDate") & _
```

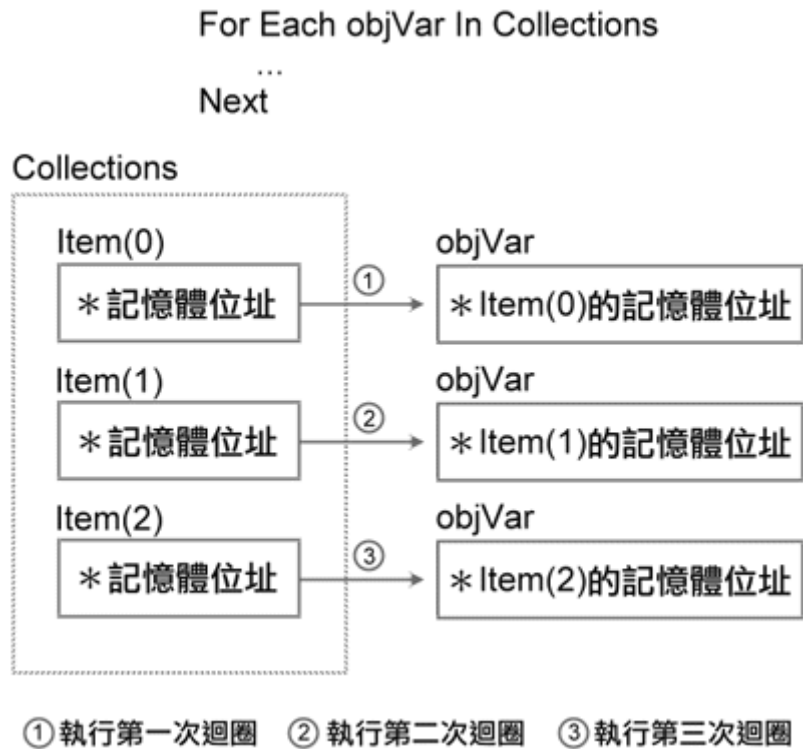
```
" / 订购产品: " & rowTemp("ProductName") & _
```



```
" / 产品单价: " & rowTemp("UnitPrice") & _  
" / 订购数量: " & rowTemp("Quantity") & _  
" / 小    计: " & rowTemp("Total"))
```

Next

在执行 **For Each...In** 循环结构时,会将 **In** 后面的集合对象中第一个对象的内存地址从内存取出,并由 **For Each** 后面的变量接收;遇到 **Next** 叙述时则取出下一个在集合对象中的项目,并将内存地址给 **For Each** 后面的变量接收,直到集合对象中的所有对象全部被取出完毕为止。利用 **For Each...In** 这种循环结构的好处是可以让我们以单一的变量名称,就可以参考到所有在集合中的对象。下列插图中假设有一个 **Colletions** 集合,并群组了三个对象;所以我们可以使用 **For Each...In** 的循环结构将集合对象中,每一个项目的内存地址一一取出,并存入 **objA** 对象型态的变量中来控制这些对象:



故上述程序执行时将 `dsDataSet.Tables("Members").Rows(shtl).GetChildRows("MO")` 集合中的第一个子关联的记录取回，并放入 `rowTemp` 这个存放 `DataRow` 型态的对象变量中，所以我们可以利用 `rowTemp` 这个变量来控制集合中的第一个对象；遇到 `Next` 叙述后再取出集合中的下一个对象的内存地址，直到集合中的项目被取完为止。

利用关联将父关联记录取出

下列范例利用 `DataRow` 对象的 `GetParentRow` 方法取得 `Orders` 订单记录的父关联记录，并指定所要取出的字段为 `UserId`：

```
<%@Import Namespace=System.Data.ADO%>
```

```

<%@Import Namespace=System.Data%>

<Script Language="VB" Runat="Server">

Sub Page_Load(Sender As Object, e As EventArgs)

    Dim strConStr As String = "Provider=Microsoft.Jet.OLEDB.4.0;" & _
        "Data Source=C:\InetPub\wwwroot\CR\CH05\MyWeb.mdb"

    Dim strComStr As String = "Select * From Members"

    Dim dscA As ADODatasetCommand = New ADODatasetCommand(strComStr,
strConStr)

    Dim dsDataSet As DataSet = New DataSet()

    dscA.FillDataSet(dsDataSet, "Members")

    dscA.SelectCommand.CommandText = "Select * from Orders"

    dscA.FillDataSet(dsDataSet, "Orders")

    dsDataSet.Relations.Add("MO",
dsDataSet.Tables(0).Columns("UserId"), _

dsDataSet.Tables(1).Columns("UserId"))

    Dim rowTemp As DataRow

    For Each rowTemp In dsDataSet.Tables("Orders").Rows

        Response.Write("订单: 订单日期: " & rowTemp("OrderDate") & _
            " / 订购产品: " & rowTemp("ProductName") & _

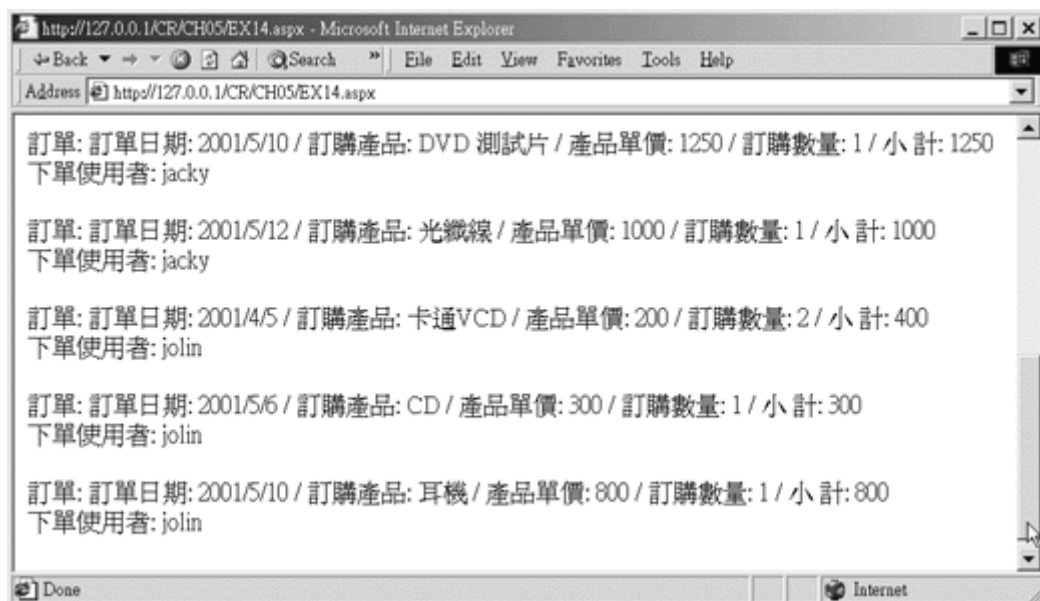
```

```
" / 产品单价: " & rowTemp("UnitPrice") & _  
" / 订购数量: " & rowTemp("Quantity") & _  
" / 小 计: " & rowTemp("Total") & "<br>" & _  
"下单使用者: " & rowTemp.GetParentRow("MO")("UserId") & "<p>"
```

Next

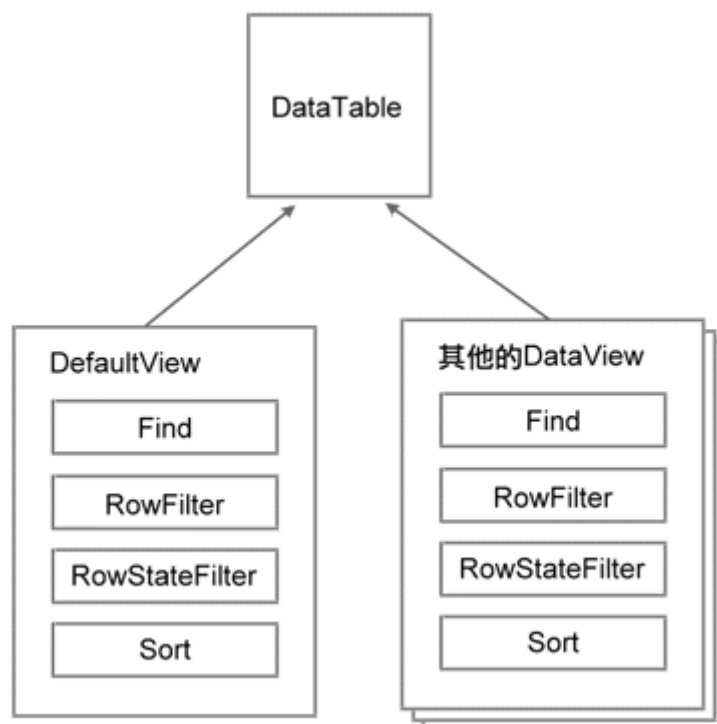
End Sub

</SCRIPT>



DataView 物件

ADO.NET 提供了一个可以自订数据外观的 **DataView** 对象。它是一种用来帮我们设定 **DataTable** 中的资料要如何显示出来的对象，本身并不包含 **DataTable** 中的数据。**DataView** 对象可以被当作是一种经过排序或是条件过滤过的 **DataTable** 对象来使用，可以让我们取得原 **DataTable** 中记录的状态，也可以用来指定记录的排列顺序或是指定所要过滤的数据，甚至是搜寻数据。这样一来我们就可以透过 **DataView** 将同一个 **DataTable** 以不同的方式呈现出来，让 **DataTable** 的运用变的更灵活。**DataTable** 已经准备了一个 **DataView** 对象来让我们使用，这个对象就是 **DataTable** 的 **DefaultView** 属性，这个属性就是 **DataView** 对象。以下为 **DataTable** 对象以及 **DataView** 对象的关系图：



DefaultView 属性

DataTable 的原始输出格式可以透过 DefaultView 属性来取得，DefaultView 属性本身就是

DataView 对象；我们可以设定 DefaultView 的属性来指定 DataTable 的显示格式。如果预设的

一个 DefaultView 这个 DataView 对象无法满足你的需求，我们还可以产生多个 DataView 对象

来制定多个数据显示外观。下表列出了 DataView 对象的常用的属性及方法：

属性或方法	说明
Count 属性	传回 DataView 中的纪录笔数
RowFilter 属性	传回或设定过滤记录的条件
RowStateFilter 属性	传回或设定以记录的状态来设定过滤条件
Sort 属性	传回或设定记录的显示顺序
Find 方法	搜寻指定的记录

排序数据

要排序数据，可以使用 DataView 对象的 Sort 属性。Sort 属性是以字段做为排序的依据，其设

定语法如下所示：

```
DataView.Sort="字段一 ASC|DESC [, ... 字段 N ASC|DESC]"
```

下列范例指定了 **Members** 这个 **DataTable** 的 **DefaultView** 的 **Sort** 属性以 **UserId** 做升幂排序:

```
<%@Import Namespace=System.Data.ADO%>

<%@Import Namespace=System.Data%>

<Script Language="VB" Runat="Server">

Sub Page_Load(Sender As Object, e As EventArgs)

    Dim strConStr As String = "Provider=Microsoft.Jet.OLEDB.4.0;" & _
        "Data Source=C:\InetPub\wwwroot\CR\CH05\MyWeb.mdb"

    Dim strComStr As String = "Select * From Members"

    Dim dscA As ADODataSetCommand = New ADODataSetCommand(strComStr,
strConStr)

    Dim dsDataSet As DataSet = New DataSet()

    dscA.FillDataSet(dsDataSet, "Members")

    Dim dtTable As DataTable = dsDataSet.Tables("Members") ' 为了方
便使用而宣告的别名

    Dim shtR As Short

    Response.Write("DataTable 中的原始资料:<br>")

    For shtR=0 To dtTable.Rows.Count-1 ' 直接抓出 DataTable 中的数
据

        Response.Write("顺序 " & (shtR+1).ToString & " " & _
```

```

        dtTable.Rows(shtR)("UserId") & " ")

    Next

    dtTable.DefaultView.Sort="UserId Asc" ' 设定 DefaultView 的排序
条件

    Response.Write("<p>DataTable.DefaultView 的资料:<br>")

    For shtR=0 To dtTable.DefaultView.Count-1 ' 显示 DataView 整
理的资料

        Response.Write("顺序 " & (shtR+1).ToString & " " & _

            dtTable.DefaultView(shtR)("UserId") & " ")

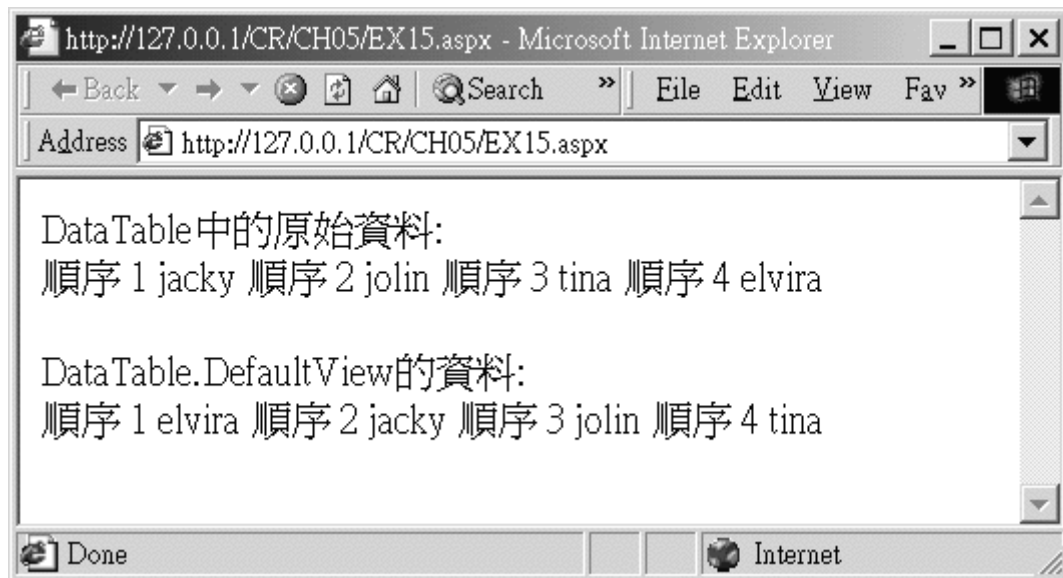
    Next

End Sub

</SCRIPT>

```

我们可以利用 **DataView** 的 **Rows** 集合取回被 **DataView** 对象整理过的数据，所以上述范例的执行结果如下所示：



筛选记录

要筛选记录，可以使用 **DataView** 对象的 **RowFilter** 属性以及 **RowStateFilter** 属性。**RowFilter**

属性可以利用比较运算符「<」、「>」、「<=」、「>=」以及「Like」来过滤记录中的数据，

其语法如下所示：

```
DataView.RowFilter = "条件叙述"
```

下列范例设定 **DefaultView** 的 **RowFilter** 属性，只有台北县的记录会被显示出来：

```
<%@Import Namespace=System.Data.ADO%>

<%@Import Namespace=System.Data%>

<Script Language="VB" Runat="Server">

Sub Page_Load(Sender As Object, e As EventArgs)
```

```

Dim strConStr As String = "Provider=Microsoft.Jet.OLEDB.4.0;" & _
    "Data Source=C:\InetPub\wwwroot\CR\CH05\MyWeb.mdb"

Dim strComStr As String = "Select * From Members"

Dim dscA As ADODatasetCommand = New ADODatasetCommand(strComStr,
strConStr)

Dim dsDataSet As DataSet = New DataSet()

dscA.FillDataSet(dsDataSet, "Members")

Dim dtTable As DataTable = dsDataSet.Tables("Members")

Dim shtR As Short

dtTable.DefaultView.RowFilter = "UserAdd Like ' 台北县%'"

Response.Write("住台北县的总共有" & _
    dtTable.DefaultView.Count.ToString & " 人， 分别是：")

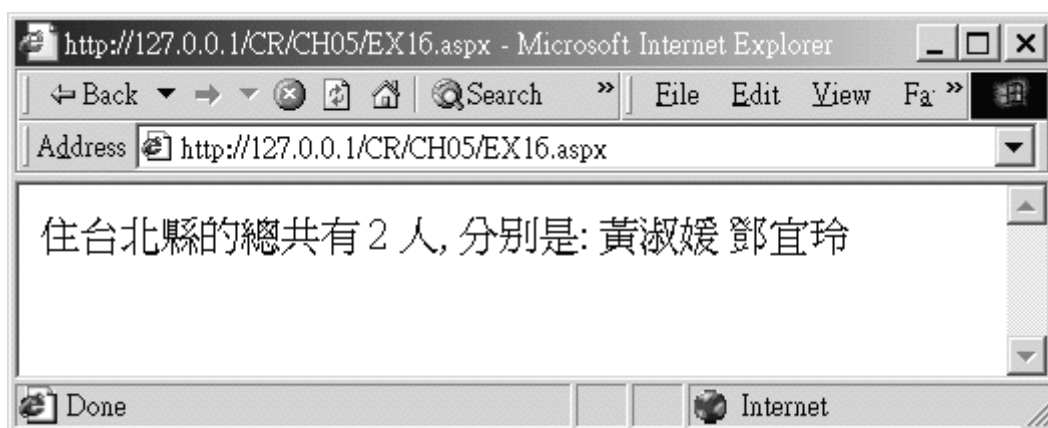
For shtR=0 To dtTable.DefaultView.Count-1    ' 显示 DataView 整理的
资料

    Response.Write(dtTable.DefaultView(shtR) ("UserName") & " ")

Next

End Sub
</SCRIPT>

```



而 **RowStateFilter** 属性则是以记录的状态来作筛选的条件。其使用语法如下所视：

```
DataView.RowStateFilter = DataViewState. 状态
```

可过滤的字段的状态如下表所示：

字段状态	说明
CurrentRows	所有的记录，包括没被修改、新增加、与被修改的字段
Deleted	被删除掉的记录
ModifiedCurrent	被修改后的记录
ModifiedOriginal	被修改前的记录
New	新增的记录
None	无

OriginalRows	包括未被修改以及删除掉的原始记录
Unchanged	没有被修改的记录

下列范例中删除了一笔记录，并且利用 **RowStateFilter** 属性显示这笔被删除的记录：

```
<%@Import Namespace=System.Data.ADO%>

<%@Import Namespace=System.Data%>

<Script Language="VB" Runat="Server">

Sub Page_Load(Sender As Object, e As EventArgs)

    Dim strConStr As String = "Provider=Microsoft.Jet.OLEDB.4.0;" & _
        "Data Source=C:\InetPub\wwwroot\CR\CH05\MyWeb.mdb"

    Dim strComStr As String = "Select * From Members"

    Dim dscA As ADODatasetCommand = New ADODatasetCommand(strComStr,
strConStr)

    Dim dsDataSet As DataSet = New DataSet()

    dscA.FillDataSet(dsDataSet, "Members")

    Dim dtTable As DataTable = dsDataSet.Tables("Members") ' 为了方便使用

    dtTable.Rows(0).Delete() ' 删除 DataTable 中的第一笔记录

    Dim shtR As Short

    dtTable.DefaultView.RowStateFilter = DataViewRowState.Deleted

    Response.Write("被删除的使用者总共有 " & _
```

```
dtTable.DefaultView.Count.ToString & " 人, 分别是:")

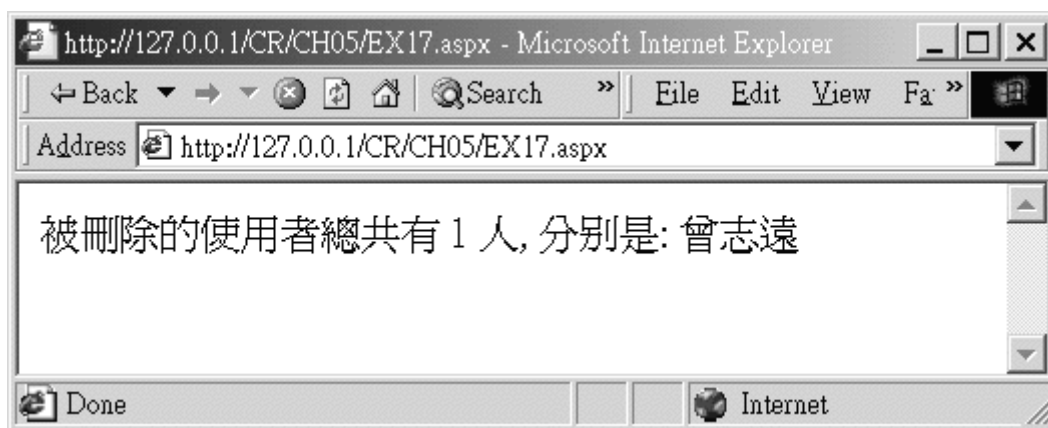
For shtR=0 To dtTable.DefaultView.Count-1    ' 显示 DataView 整理的资料

    Response.Write(dtTable.DefaultView(shtR)("UserName") & " ")

Next

End Sub

</SCRIPT>
```



上述程序虽然删除了 **DataTable** 中的第一笔记录, 除非使用 **DataSetCommand** 对象的 **Update** 方法将所作的更动更新回数据源, 否则是不会影响数据源的数据状态。

搜寻数据

要搜寻 **DataTable** 里面的数据，可以利用 **DataView** 的 **Find** 方法。如果 **Find** 方法有找到符合的数据，则传回数据所在记录的 **Index** 值；倘若没找到则传回 **-1**。其使用语法如下所示：

```
变数 = DataView.Find("要搜寻的字符串")
```

下列范例可以在文字输入盒中输入数据，按下确定后利用 **Find** 方法找寻使用者的数据，并显示数据所在记录的 **Index** 值：

```
<%@Import Namespace=System.Data.ADO%>

<%@Import Namespace=System.Data%>

<Form Id="Form1" Runat="Server">

请输入所要查询的数据：

<Input Type="Text" Runat="Server" Id="Text1">

<Button Id="Button1" Runat="Server" OnServerClick="Button1_Click">
查询</Button>

</Form>

<Span Id="Sp1" Runat="Server"/>

<Script Language="VB" Runat="Server">

Sub Button1_Click(Sender As Object, e As EventArgs)
```

```

Dim strConStr As String = "Provider=Microsoft.Jet.OLEDB.4.0;" &
—
    "Data Source=C:\InetPub\wwwroot\CR\CH05\MyWeb.mdb"

Dim strComStr As String = "Select * From Members"

Dim dscA As ADODatasetCommand = New ADODatasetCommand(strComStr,
strConStr)

Dim dsDataSet As DataSet = New DataSet()

dscA.FillDataSet(dsDataSet, "Members")

Dim dtTable As DataTable = dsDataSet.Tables("Members")

dtTable.DefaultView.Sort="UserId" ' 必须要指定 Sort 属性才可以搜
寻

Dim shtR As Short

shtR=dtTable.DefaultView.Find(Text1.Value) ' 传回符合记录的
Index 值

If shtR = -1 Then ' 如果没找到就传回 -1

    Sp1.InnerText="没找到您所输入的数据"

Else

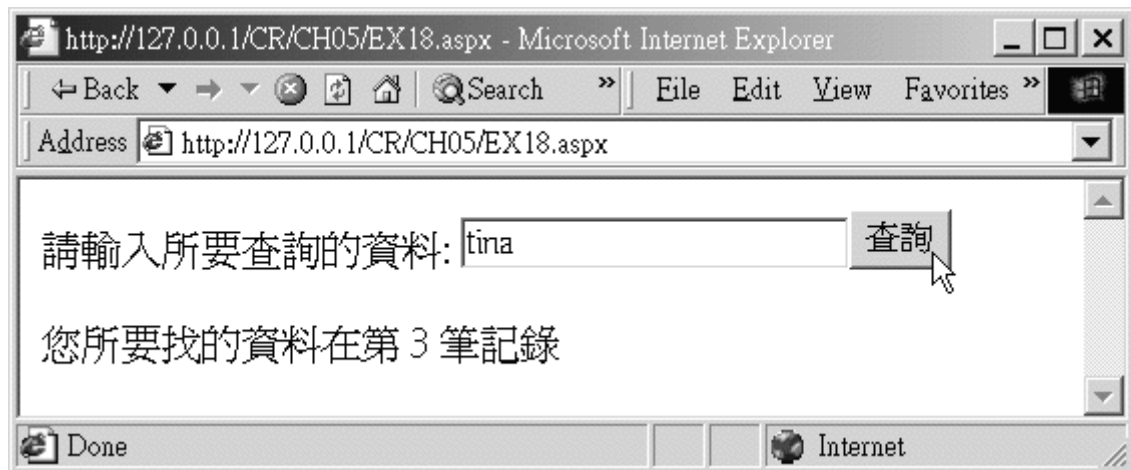
    Sp1.InnerText="您所要找的资料在第 " & shtR.ToString & " 笔记录"

End if

End Sub

```

</SCRIPT>



产生自订的 **DataView** 对象

如果一个 **DataView** 对象不能满足我们的需求，我们还可以建立多个 **DataView** 对象来制定数据的显示格式。其宣告语法如下所示：

```
Dim 变量 As DataView = New DataView(数据表)
```

下列范例中从同一个 **DataTable** 中产生了两个自订的 **DataView** 对象，并指定不同的数据排序方式：


```

<%@Import Namespace=System.Data.ADO%>

<%@Import Namespace=System.Data%>

<Script Language="VB" Runat="Server">

Sub Page_Load(Sender As Object, e As EventArgs)

    Dim strConStr As String = "Provider=Microsoft.Jet.OLEDB.4.0;" & _
        "Data Source=C:\InetPub\wwwroot\CR\CH05\MyWeb.mdb"

    Dim strComStr As String = "Select * From Members"

    Dim dscA As ADODatasetCommand = New ADODatasetCommand(strComStr,
strConStr)

    Dim dsDataSet As DataSet = New DataSet()

    dscA.FillDataSet(dsDataSet, "Members")

    Dim dv1st As DataView = New DataView(dsDataSet.Tables(0))

    Dim dv2nd As DataView = New DataView(dsDataSet.Tables(0))

    dv1st.Sort = "UserId ASC"      ' 指定依 UserId 由小排到大的配置

    dv2nd.Sort = "UserId DESC"    ' 指定依 UserId 由大排到小的配置

    Dim shtR As Short

    Response.Write("dv1st 指定依 UserId 由小排到大的配置:<br>")

    For shtR=0 To dv1st.Count-1      ' 显示 DataView 整理的资料

```

```

Response.Write("顺序 " & (shtR+1).ToString & " " & _
               " -> " & dv1st(shtR)("UserId") & " ")

Next

Response.Write("<p>dv2nd 指定依 UserId 由大排到小的配置:<br>")

For shtR=0 To dv2nd.Count-1    ' 显示 DataView 整理的资料

Response.Write("顺序 " & (shtR+1).ToString & " " & _
               " -> " & dv2nd(shtR)("UserId") & " ")

Next

End Sub

</SCRIPT>

```



我们可以将 **DataView** 直接当成 **DataTable** 来使用，因为 **DataView** 的数据来源还是 **DataTable**

对象，所以我们对 **DataView** 对象所执行的任何操作都会影响原来的 **DataTable** 中的数据。下列

范例显示修改 **DataView** 中的数据对原来 **DataTable** 对象的影响：

```
<%@Import Namespace=System.Data.ADO%>

<%@Import Namespace=System.Data%>

<Script Language="VB" Runat="Server">

Sub Page_Load(Sender As Object, e As EventArgs)

    Dim strConStr As String = "Provider=Microsoft.Jet.OLEDB.4.0;" & _
        "Data Source=C:\InetPub\wwwroot\CR\CH05\MyWeb.mdb"

    Dim strComStr As String = "Select * From Members"

    Dim dscA As ADODatasetCommand = New ADODatasetCommand(strComStr,
strConStr)

    Dim dsDataSet As DataSet = New DataSet()

    Dim shtR As Short

    dscA.FillDataSet(dsDataSet, "Members")

    Dim dv1st As DataView = New DataView(dsDataSet.Tables(0))

    Dim dv2nd As DataView = New DataView(dsDataSet.Tables(0))

    dv1st.Sort = "UserId ASC"      ' 指定依 UserId 由小排到大的配置

    dv2nd.Sort = "UserId DESC"    ' 指定依 UserId 由大排到小的配置
```

```

dv1st(0)("UserTEL")="1234567890" ' 修改其中一个 DataView 的内容

Response.Write("观察原来 DataTable 中的数据是否有改变:<br>")

For shtR=0 To dsDataSet.Tables(0).Rows.Count-1 ' DataTable 中的原来数
据

Response.Write("顺序 " & (shtR+1).ToString & " " & _
                " -> " & dsDataSet.Tables(0).Rows(shtR)("UserId") & "
")

Next

For shtR=0 To dsDataSet.Tables(0).Rows.Count-1 ' 显示 UserTel 字段的资
料

Response.Write("<br>内容 " & (shtR+1).ToString & " " & _
                " -> " & dsDataSet.Tables(0).Rows(shtR)("UserTel"))

Next

Response.Write("<p>dv1st 指定依 UserId 由小排到大的配置:<br>")

For shtR=0 To dv1st.Count-1 ' 显示 DataView 整理的资料

Response.Write("顺序 " & (shtR+1).ToString & " " & _
                " -> " & dv1st(shtR)("UserId") & " ")

Next

For shtR=0 To dv1st.Count-1 ' 显示 UserTel 字段的资料

```

```

Response.Write("<br>内容 " & (shtR+1).ToString & " " & _
                " -> " & dv1st(shtR) ("UserTel"))

Next

Response.Write("<p>dv2nd 指定依 UserId 由大排到小的配置:<br>")

For shtR=0 To dv2nd.Count-1      ' 显示 DataView 整理的资料

Response.Write("顺序 " & (shtR+1).ToString & " " & _
                " -> " & dv2nd(shtR) ("UserId") & " ")

Next

For shtR=0 To dv2nd.Count-1      ' 显示 UserTEL 字段的资料

Response.Write("<br>内容 " & (shtR+1).ToString & " " & _
                " -> " & dv2nd(shtR) ("UserTEL"))

Next

End Sub
</SCRIPT>

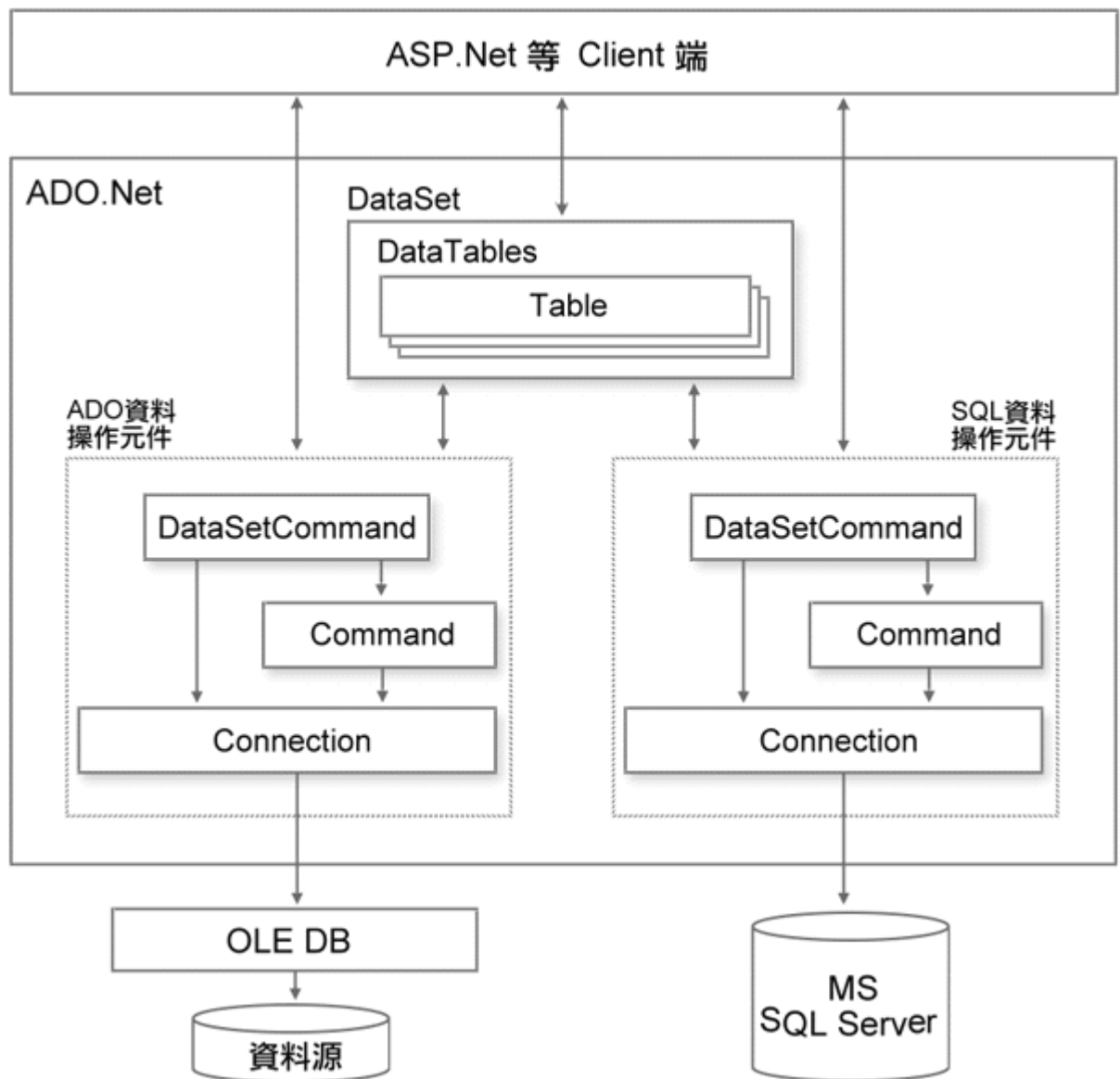
```



上列范例码修改了 dv1st 这个 DataView 对象中第一笔记录的数据，其修改会影响原数据源；由于 dv2nd 这个 DataView 的资料来源也是从原 DataTable 中取得，故 dv2nd 的资料显示也受影响。

利用 SQL 数据操作组件和 MS SQL Server 联机

如果操作 MS SQL Server 内的数据，要透过 SQL 数据操作组件。SQL 数据操作组件最主要是针对 MS SQL Server 来进行数据操作，它直接呼叫 MS SQL Server 中的 API 而不透过 OLE DB，所以效率比较好。ADO 数据操作组件与 SQL 数据操作组件的对象模型以及使用方法都一样，如下图所示：

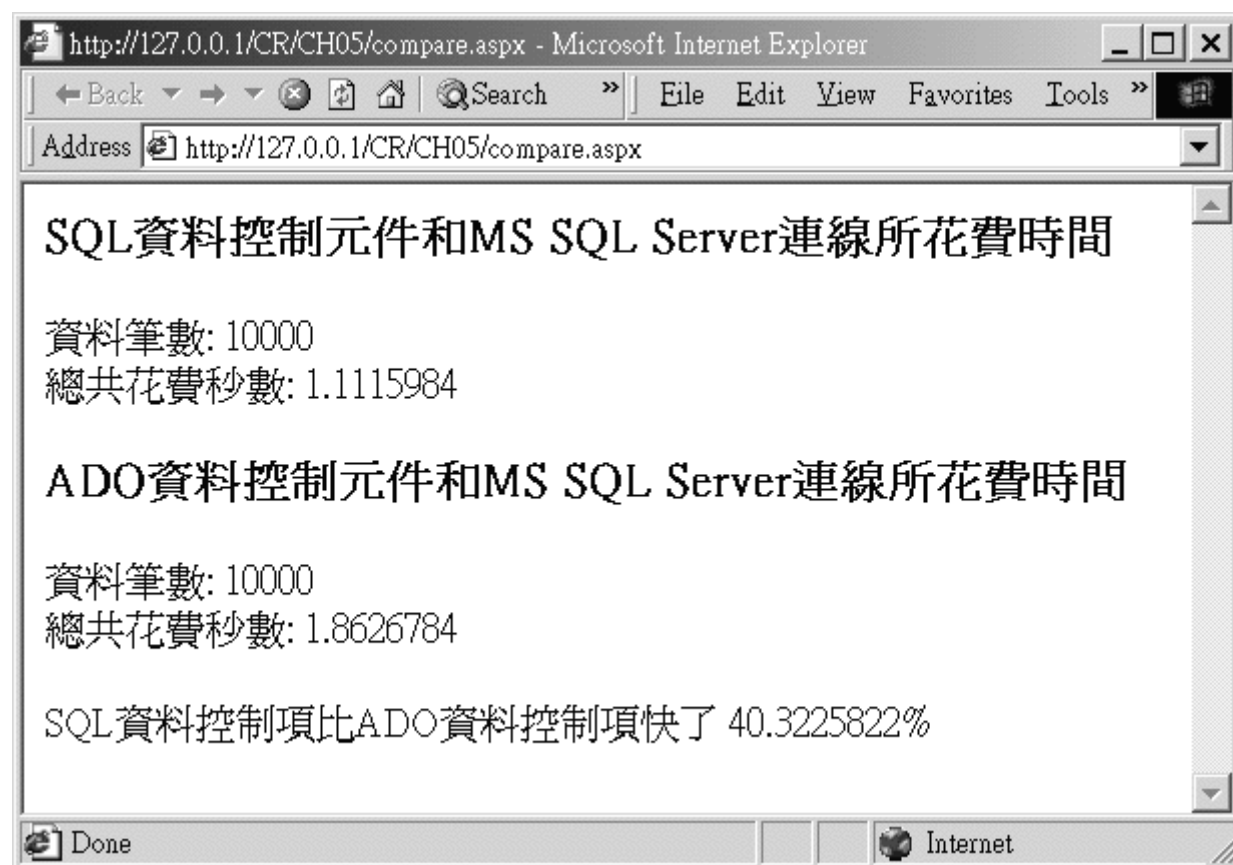


为了证明使用 SQL 数据操作组件和 MS SQL Server 联机比使用 ADO 数据操作组件的效率好，

我们写了一个测试程序。这个程序从 MS SQL Server 中取回一万笔数据，并计算使用 SQL 数据

操作组件比 ADO 数据操作组件的效能快多少（对测试程序有兴趣可以到本章目录中检视

Compare.aspx 这个程序）：



我们看到 SQL 数据操作组件比 ADO 数据操作组件大约快了 35% 的时间，所以针对 MS SQL

Server 的数据操作当然使用 SQL 数据操作组件。

宣告 **SQL** 数据操作组件的名称地址

在使用 SQL 数据操作组件的时候，除了要宣告 **System.Data** 的名称地址外，还要宣告

System.Data.SQL 名称地址；如下程序代码片段所示：

```
<%@Import Namespace="System.Data"%>  
  
<%@Import Namespace="System.Data.SQL"%>
```

宣告 SQL 数据操作组件

我们在宣告 **Connection** 对象、**Command** 对象、**DataSetCommand** 对象及 **DataReader** 对象的

时候，记得加上 **SQL** 三个前缀。如下列范例所示：

```
Dim cnA As SqlConnection  
  
Dim cmA As SqlCommand  
  
Dim dscA As SQLDataSetCommand  
  
Dim drA As SqlDataReader
```

从 MS SQL Server 取回资料

另外在建立 **Connection** 对象的时候，由于已经知道要和 **MS SQL Server** 联机，所以不需要指

定 **Connection** 对象的 **Provider** 属性。另外因为 **DataSet** 对象不是数据操作组件，不负责执行数

据源的数据操作，故 **DataSet** 的宣告及使用方法不变。下列范例利用 **SQL** 数据控制组件将 **MS**

SQL Server 中北风数据库的 **Employees** 数据表取回：

```
<%@Import Namespace=System.Data%>

<%@Import Namespace=System.Data.SQL%>

<Script Language="VB" Runat="Server">

Sub Page_Load(Sender As Object, e As EventArgs)

    Dim strConStr As String = "Data Source=Charles;" & _
        "Initial Catalog=Northwind;" & _
        "User Id=sa;Password="

    Dim strComStr As String = "Select * From Employees"

    Dim dscA As SQLDataSetCommand = New SQLDataSetCommand(strComStr,
strConStr)

    Dim dsDataSet As DataSet = New DataSet()

    dscA.FillDataSet(dsDataSet, "Employees")

    Dim dtTable As DataTable = dsDataSet.Tables("Employees")

    Response.Write("<h3>MS SQL Server 中北风数据库的 Employees 数据
表:</h3>")
```

```
Dim shtR As Short

For shtR=0 To dtTable.Rows.Count-1

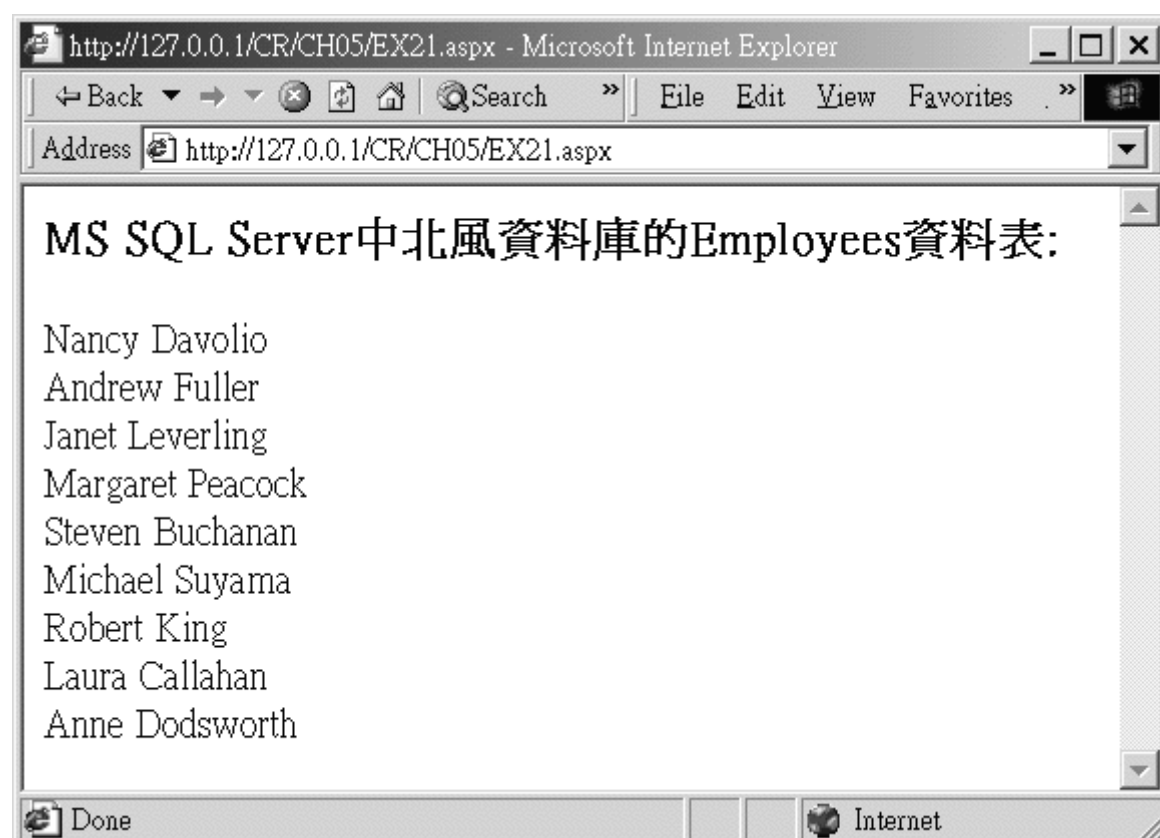
    Response.Write(dtTable.Rows(shtR)("FirstName") & " " & _

                    dtTable.Rows(shtR)("LastName") & "<br>")

Next

End Sub

</SCRIPT>
```



6. Web 控件

Web 控件简介

Web 控件基本概念

我们在第四章已经介绍过 HTML 控件，接下来要介绍的是 Web 控件（或称为 ASP.NET Server 控件）。Web 控件和 HTML 控件不一样，HTML 控件是将 HTML 标注对象化，让我们的程序代码比较好控制以及管理这些控件；不过基本上它还是转成相对应的 HTML 标注。而 Web 控件的功能比较强，它会依 Client 端的状况产生一个或多个适当的 HTML 控件，它可以自动侦测 Client 端浏览器的种类，并自动调整成适合浏览器的输出。Web 控件还拥有一个非常重要的功能，那就是支持数据系结（Data Binding）；这种能力可以和资料源连结，用来显示或修改数据源的数据。

Web 控件的基本架构

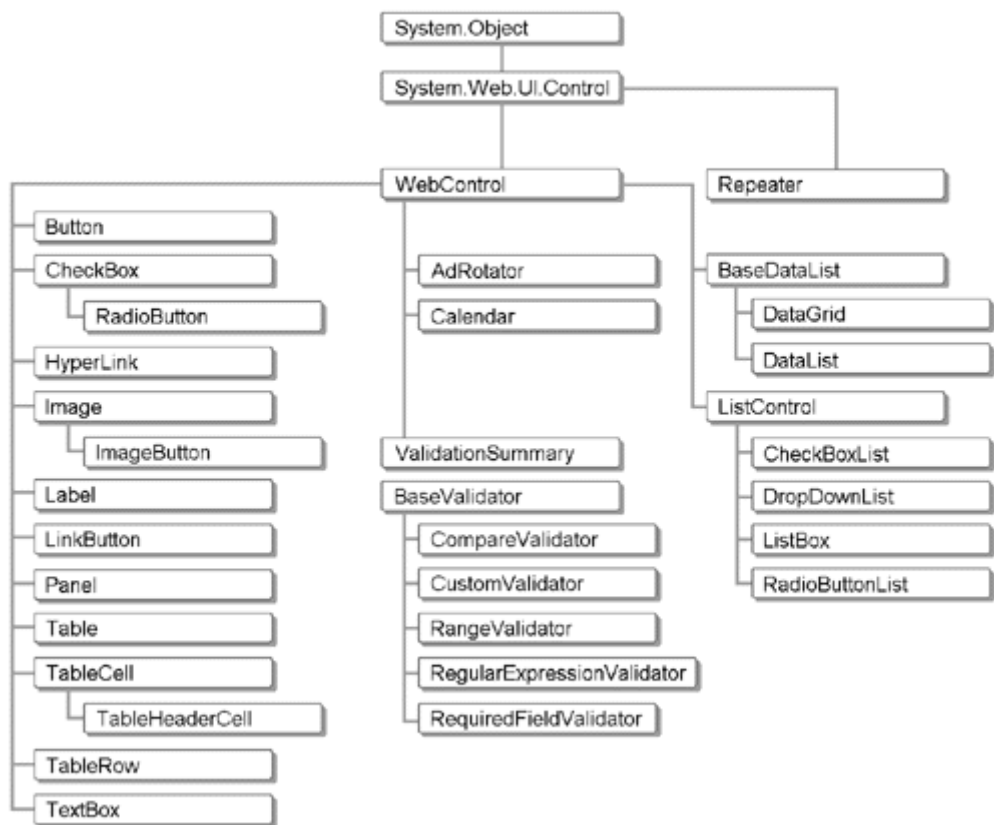
Web 控件的对象模型比较有弹性，有些对象的特质甚至和一般的窗口控件很像，行为也非常一致。这是因为微软希望我们在开发软件解决方案的时候，可以不用因为一种执行环境就要学习不同的控件。这样的好处在于发展分布式，并混合标准窗口应用程序以及因特网应用程序的解决方

案时，可以因为这些控件的行为及架构非常相似，可以让我们转移已知的知识到新的环境上；因为这些控件的行为及架构非常相似，所以不用浪费时间在学习功能一样，但是架构不一样的对象上。我们将这些 Web 控件分为四种类型，如下表所示：

名称	内容
内建控件 (Intrinsic Control)	对应一般的 HTML 元素
清单控件 (List Control)	提供选项给使用者选择
丰富控件 (Rich Control)	提供许多以前没有的控件
验证控件 (Validation Control)	提供数据验证控件

其架构图如下图所示：

System.Web.UI.WebControls



Web 控件的使用

基本概念

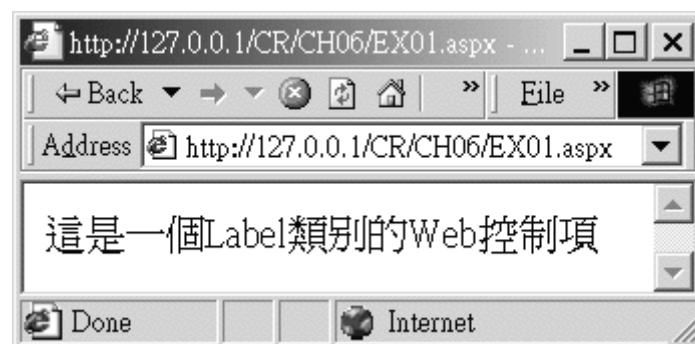
要使用 Web 控件，只要在标注中先加上 ASP:，并指定 Web 控件的类别名称即可。下表列出所有的 Web 控件：

AdRotator	Button	Calendar
-----------	--------	----------

CheckBox	CheckBoxList	DataGrid
DataList	DropDownList	Hyperlink
Image	ImageButton	Label
ListBox	LinkButton	Panel
RadioButton	RadioButtonList	Repeater
Table	TableCell	TableRow
TextBox		

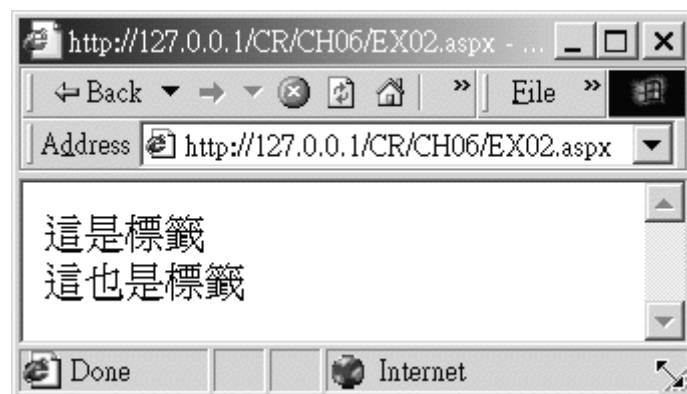
下列范例宣告了一个 **Label Web 控件**：

```
<ASP:Label Id="Label1" Text="这是一个 Label 类别的 Web 控件"
Runat="Server" />
```



如同 HTML 控件一样，我们必须赋予每个控件一个 **Id** 属性，并且指定 **Runat** 属性为 **Server**，表示控件是在 **Server** 端执行。**Web** 控件设定属性的方式有两种，一种是开始在页面布置对象时便将属性设定好；另一种是由程序来设定。下列程序代码范例分别利用两种方式来设定 **Web** 控件的属性：

```
<ASP:Label Id="Label1" Text="这是标签" Runat="Server"/><br>  
<ASP:Label Id="Label2" Runat="Server"/>  
  
<Script Language="VB" Runat="Server" ID=Script1>  
  
Sub Page_Load(Sender As Object, e As EventArgs)  
  
    Label2.Text="这也是标签"  
  
End Sub  
  
</Script>
```



以上两种方法虽然都会使 **Button Web** 控件上出现文字，不过一般来说我们习惯将不会更动的属性直接就设定在标注中，而会更动的属性就用程序代码来设定。

Web 控件的基础属性

接着我们来介绍 **Web** 控件的基础属性，所谓基础属性就是所有的 **Web** 控件共同都有的属性。

这些属性有：

AccessKey	Attributes	BackColor
BorderWidth	BorderStyle	CSSClass
CSSStyle	Enabled	Font-Bold
Font-Italic	Font-Name	Font-Names
Font-Overline	Font-Size	Font-Strikeout
Font-Underline	ForeColor	Height
TabIndex	ToolTip	Width

AccessKey 属性

这个属性可以用来指定键盘的快速键。我们可以指定这个属性的内容为数字或是英文字母，当使用者按下键盘上的「Alt」再加上我们所指定的值时，表示选择该控件。例如下列范例指定 Web 控件 **Button** 的 **AccessKey** 属性为「A」，当使用者按下「Alt」+「A」时即表示按下了按钮：

```
<Form Id="Form1" Runat="Server">

<ASP:Button Id="Button1" Text="请按我" Runat="Server" AccessKey="A"

    OnClick="Button1_Click"/>

或是按 Alt+A 也可以

</Form>

<ASP:Label Id="Label1" Runat="Server"/>

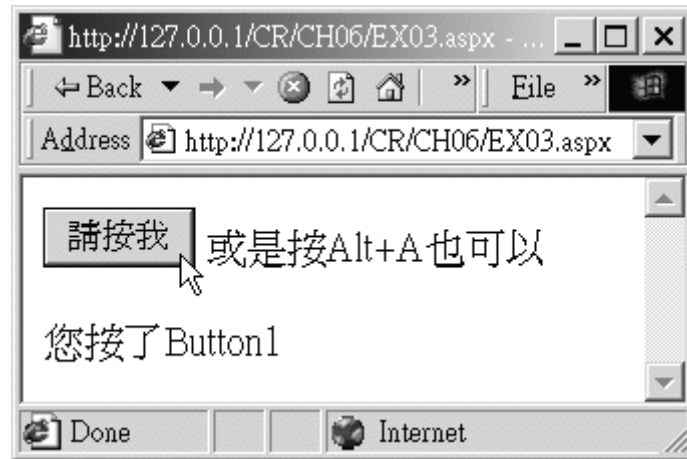

<Script Language="VB" Runat="Server" ID=Script1>

Sub Button1_Click(Sender As Object, e As EventArgs)

    Label1.Text="您按了 Button1"

End Sub

</Script>
```



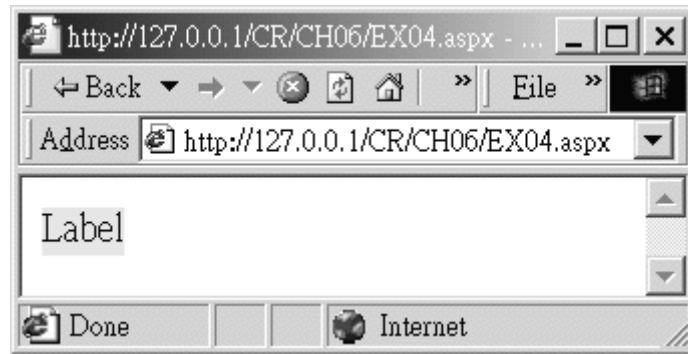
上述程序无论是直接按了按钮或是按下「Alt」+「A」，都会触发 `Button1_Click` 这个事件程序。

这里要特别注意一点，那就是 `Button Web` 控件的按钮事件驱动属性不是 `OnServerClick`，而是 `OnClick` 属性。

BackColor 属性

设定对象的背景色，其属性的设定值为颜色名称或是 `#RRGGBB` 的格式。如果用 RGB 来调色，利用影像软件或 `FrontPage` 来查询颜色的值较为方便。下列程序代码设定了 `Label Web` 控件的背景色为灰色：

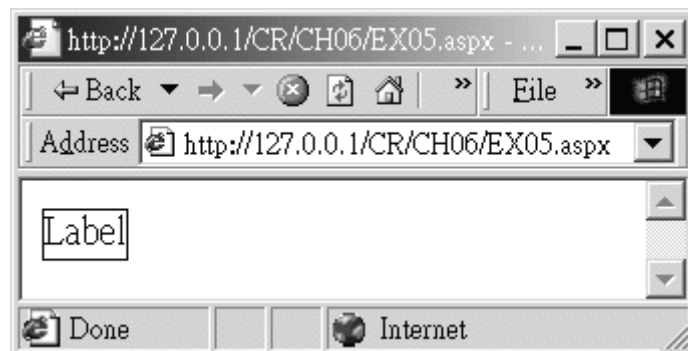
```
<ASP:Label Id="Label1" Text="Label" BackColor="#E0E0E0" Runat="Server"
/>
```



BorderWidth 属性

本属性可以用像素来设定 **Web** 控件的边框宽度,不过在有些能力较差的浏览器上可能无法显示。

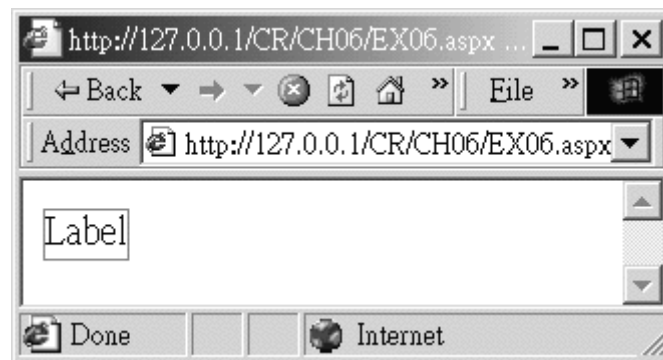
```
<ASP:Label Id="Label1" Text="Label" BorderWidth=1 Runat="Server"/>
```



Bordercolor 属性

本属性可以用来设定外框的颜色。程序及执行结果如下：

```
<ASP:Label Id="Label1" Text="Label" BorderWidth=1 BorderColor="Red"
Runat="Server" />
```



BorderStyle 属性

本属性可用来设定对象的外框样式，总共有十种设定，如下表所示：

设定值	说明
Notset	默认值。
None	没有外框。
Dotted	外框为虚线，点较小。
Dashed	外框为虚线，点较大。
Solid	外框为实线。

Double	外框为实线，但厚度是 Solid 的两倍。
Groove	在对象四周出现 3D 凹陷式的外框。
Ridge	在对象四周出现 3D 突起式的外框。
Inset	物件呈陷入状。
Outset	物件成突起状。

下列范例为 **BorderStyle** 属性的效果测试：

```

<ASP:Button Id="B1" Text="Notset" Runat="Server"/>

<ASP:Button Id="B2" Text="None" Borderstyle="None" Runat="Server"/>

<ASP:Button Id="B3" Text="Dotted" Borderstyle="Dotted" Runat="Server"/>

<ASP:Button Id="B4" Text="Dashed" Borderstyle="Dashed" Runat="Server"/>

<ASP:Button Id="B5" Text="Solid" Borderstyle="Solid"
Runat="Server"/><p>

<ASP:Button Id="B6" Text="Double" Borderstyle="Double" Runat="Server"/>

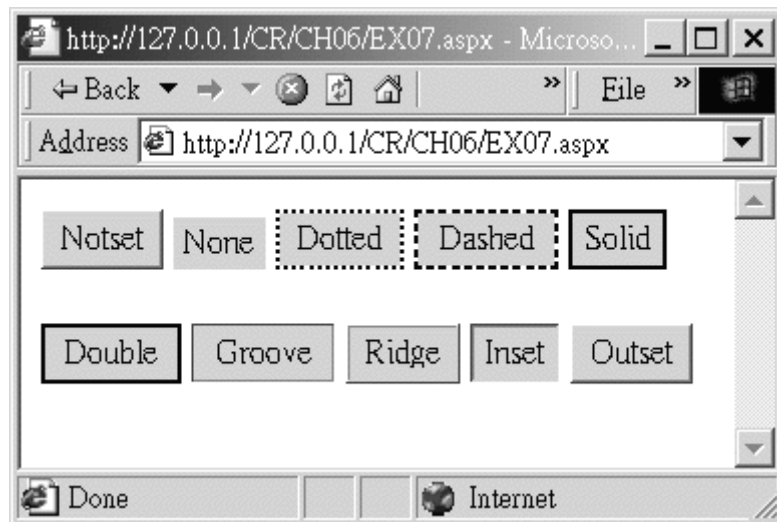
<ASP:Button Id="B7" Text="Groove" Borderstyle="Groove" Runat="Server"/>

<ASP:Button Id="B8" Text="Ridge" Borderstyle="Ridge" Runat="Server"/>

<ASP:Button Id="B9" Text="Inset" Borderstyle="Inset" Runat="Server"/>

<ASP:Button Id="B10" Text="Outset" Borderstyle="Outset"
Runat="Server"/><p>

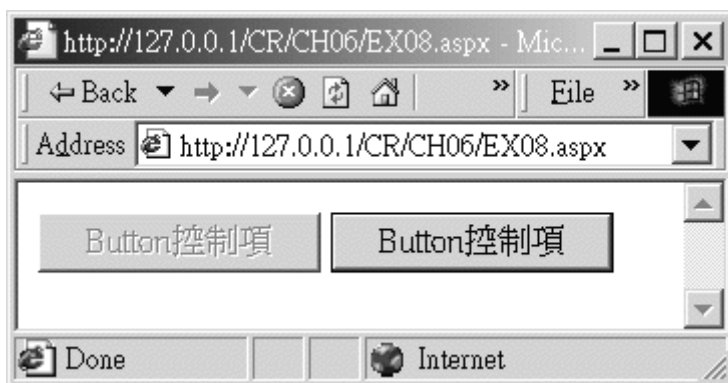
```



Enabled 属性

本属性称为致能，用来决定控件是否正常工作。本属性的默认值是 **True**，如要让控件失去作用，只要将控件的 **Enabled** 属性值设为 **False** 即可将它禁能。我们之前介绍的 **HTML** 控件也有一个类似的 **Disabled** 属性，称为禁能属性；和这里的 **Web** 控件刚好相反。

```
<ASP:Button Id="Button1" Text="Button 控件" Enabled="False"
Runat="Server" /><p>
<ASP:Button Id="Button2" Text="Button 控件" Runat="Server" />
```

Font 属性

Web 基础属性提供了六种属性让我们可以用来设定字型的样式，其属性以及设定值如下表所示：

属性	描述
Font-Bold	设定为 True 则会变成粗体。
Font-Italic	设定为 True 则会变成斜体。
Font-Names	设定为何种字型
Font-Size	设定字体大小，共有九种大小可供选择：Smaller、Larger、XX-Small、X-Small、Small、Medium、Large、X-Large、XX-Large
Font-Strikeout	设定为 True 则会出现删除线。
Font-Underline	设定为 True 则会出现底线。

下列范例展示了字型属性的设定：

```
<ASP:Label Id="Label1" Runat="Server" Font-Bold="True" Text="粗體"/>

<ASP:Label Id="Label2" Runat="Server" Font-Italic="True" Text="斜體"/>

<ASP:Label Id="Label3" Runat="Server" Font-Names="标楷体" Text="标楷体"
"/>

<ASP:Label Id="Label4" Runat="Server" Font-Strikeout="True" Text="删除
线"/>

<ASP:Label Id="Label5" Runat="Server" Font-Underline="True" Text="底线"
"/>

<ASP:Label Id="Label6" Runat="Server" Font-Size="XX-Large" Text="大字
体"/>
```



Height 属性、Width 属性

这两个属性用来设定 Web 控件的高和宽，单位是 **pixel**（像素）。范例及程序如下：

```
<ASP:Button Id="B1" Text="预设大小" Runat="Server" /><p>  
<ASP:Button Id="B2" Text="改变大小" Height="55" Width="92"  
Runat="Server"/>
```

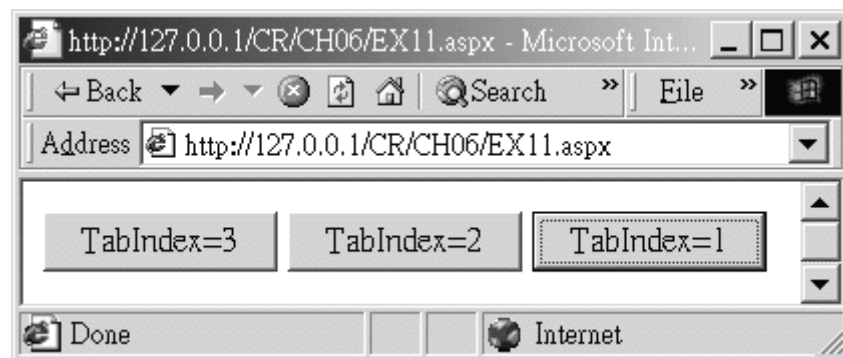


TabIndex 属性

用来设定当使用者按下「Tab」按钮时，Web 控件接收驻点的顺序，如果这个属性没有设定的话就是默认值零。如果 Web 控件的 TabIndex 属性值一样的话，则是以 Web 控件在 ASP.NET 网

页中被配置的顺序来决定。下列范例指定了 **Button Web** 控件的 **TabIndex** 属性，由于 **B3** 的 **TabIndex** 值最小，所以浏览网页的时候驻点是停留在 **B3** 上：

```
<ASP:Button Id="B1" Text="TabIndex=3" TabIndex="3" Runat="Server"/>
<ASP:Button Id="B2" Text="TabIndex=2" TabIndex="2" Runat="Server"/>
<ASP:Button Id="B3" Text="TabIndex=1" TabIndex="1" Runat="Server"/>
```



ToolTip 属性

ToolTip 就是小提示。有设定本属性时，当使用者停留在 **Web** 控件上时就会出现提示的文字：

```
<ASP:Button Id="B1" Text="我有小提示" ToolTip="这就是小提示"
Runat="Server"/>
```



Visible 属性

Visible 属性决定了控件的显示。设定本属性为 **False** 时，控件的使用者接口就会消失：

```
<ASP:Button Id="B1" Text="没隐藏的按钮" Runat="Server"/>
```

```
<ASP:Button Id="B2" Text="隐藏的按钮" Visible="False" Runat="Server"/>
```



內建控件（Intrinsic Control）

內建控件指的是一般如 **Button**、**HyperLink**、**TextBox** 等常会用在网页制作上的 **Web** 控件，使

用內建控件可以很快地为网页加入基本的操作接口。

Label Web 控件

Label Web 控件是最简单的控件，它的主要作用是用来显示文字。其使用语法为：

```
<ASP:Label
```

```
Id="被程序代码所控制的名称"
```

```
Runat="Server"
```

```
Text="所要显示的文字"
```

```
/>
```

或

```
<ASP:Label
```

```
Id="被程序代码所控制的名称"
```

```
Runat="Server"
```

```
>
```

```
所要显示的文字
```

```
</ASP:Label>
```

当我们要使用程序来改变其显示的文字时，只要改变它的 **Text** 属性即可。下列范例码配置了一个 **Label Web** 控件，并在 **Page_Load** 事件程序中将其 **Text** 属性设定为「这是一个 **Label** 控件」：

```
<Html>
```

```
<ASP:Label id="Label1" Runat="Server"/>
```

```
<Script Language="VB" Runat="Server">
```

```
Sub Page_Load(Sender As Object, e As EventArgs)
```

```
Label1.Text="这是一个 Label 控件"
```

```
End Sub
```

```
</Script>
```

```
</Html>
```



Image 控件

Image Web 控件是用来显示图片。其使用语法为：

```
<ASP:Image
```

```
Id="被程序代码所控制的名称"
```

```
Runat="Server"
```

```
ImageUrl="图片所在地址"
```

```
AlternateText="图形还没加载时所替代的文字"
```

```
ImageAlign="NotSet | AbsBottom | AbsMiddle | BaseLine | Bottom | Left |
```

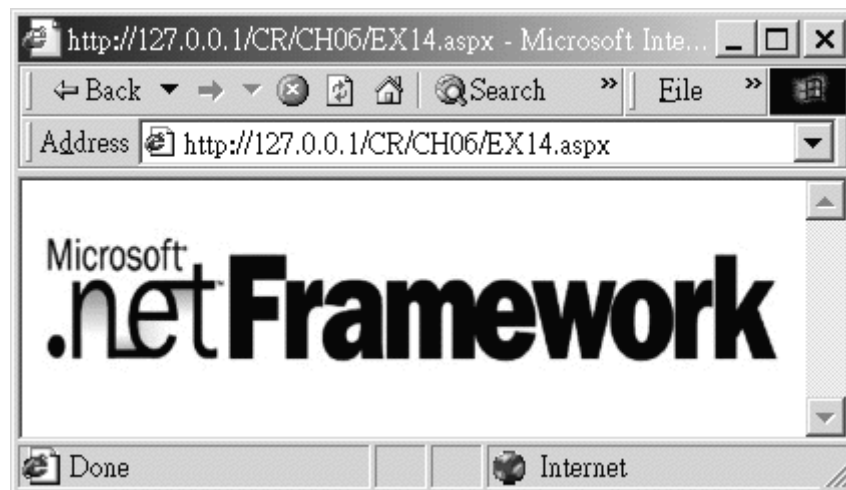
```
Middle |
```

```
Right | TextTop | Top"
```

```
/>
```


Image Web 控件最重要的属性是 `ImageUrl`，这个属性指明图形文件所在的目录或是网址；如档案和网页存放在同一个目录，则可以省略目录直接指定文件名即可。下列范例利用 Image Web 控件显示了 `stone.jpg` 这个图形：

```
<ASP:Image Id="Image1" ImageUrl="Framework.jpg" Runat="Server"/>
```



HyperLink 控件

HyperLink 控件可以用来设定超级链接，就是 HTML 元素的 `<A>` 标注。其使用语法为：

```
<ASP:Hyperlink  
Id="控件 Id"  
Runat="Server"
```

Text="超级链接文字或小提示文字"

ImageUrl="图片所在地址"

Target="超级链接所要显示的窗口"

/>

或

<ASP:Hyperlink

Id="被程序代码所控制的名称"

Runat="Server"

ImageUrl="图片所在地址"

Target="超级链接所要显示的窗口"

/>

超级链接文字

</ASP:Hyperlink>

我们只要设定 **NavigateUrl** 属性为欲浏览的地址，在使用者按下此连结时即可连至指定的地址。

而 **Target** 属性可以在有设框架（**Frame**）的网页上，决定此连结要开启在哪个框架或另外开启

新的窗口。设定 **ImageUrl** 属性则可以产生一个图形连结，在图形模式的 **HyperLink** 控件如果有

设定 **Text** 属性，则鼠标移到图形上时会出小提示。下列范例利用 **Hyperlink Web** 控件分别制作

了文字型态以及图形型态的超级链接：

```
<ASP:Hyperlink Id="hl1" Navigateurl="http://www.microsoft.com"  
Text="Microsoft"  
  
Target="_blank" Runat="server" /><p>  
  
<ASP:Hyperlink Id="hl2" Navigateurl="http://www.msn.com" Text="MSN"  
  
ImageUrl="vsdotnet.gif" Target="_blank" Runat="Server"/>
```



Button Web 控件

Button Web 控件是网页设计相当重要的 Web 控件。它主要作用在于接收使用者的 Click 事件，

并执行相对应的事件程序来完成程序的处理。其使用语法为：

```
<ASP:Button  
  
Id="被程序代码所控制的名称"  
  
Runat="Server"  
  
Text="按钮上的文字"  
  
Command="命令名称"  
  
CommandArgument="命令参数"  
  
OnClick="事件程序名"  
  
>
```

要使用 **Button Web** 控件的 **Click** 事件，除了要指定 **OnClick="事件名称"** 外，另外还必须将对象 **放在窗体标注中** 才会动作，不然将会没有作用。至于 **Command** 以及 **CommandArgument** 属性可以用来和 **DataList** 等控件配合使用，我们这里先不讨论。下列范例当我们按下 **Button** 控件后，便触发 **Click** 事件程序，并在程序中改变 **Label** 控件的 **Text** 属性：

```
<Html>  
  
<Form Id="Form1" Runat="Server">  
  
    <ASP:Button Id="B1" Text="请按我" OnClick="B1_Click"  
Runat="Server"/><p>  
  
    <ASP:Label Id="L1" Text="Label 控件" Runat="Server" />  
  
</Form>  
  
<Script Language="VB" Runat="Server">
```

```
Sub B1_Click(Sender As Object, e As EventArgs)

    L1.Text="改变后的 Label 控件"

End Sub

</Script>

</Html>
```



LinkButton Web 控件

LinkButton Web 控件的功能和 Button Web 控件一样，只不过它是类似超级链接的文字接口。其

使用语法为：

```
<ASP:LinkButton
```

Id="被程序代码所控制的名称"

Runat="Server"

Text="按钮上的文字"

Command="命令名称"

CommandArgument="命令参数"

OnClick="事件程序名"

/>

或

<ASP:LinkButton

Id="控件 Id"

Runat="Server"

Command="命令名称"

CommandArgument="命令参数"

OnClick="事件程序名"

/>

"按钮上的文字"

</ASP:LinkButton>

LinkButton 必须写在 <Form> 和 </Form> 之间，也要指定 OnClick 属性才会动作。下面的程

序代码将 Button 的范例换成用 LinkButton，执行结果还是一样：

```
<Html>

<Form Id="Form1" Runat="Server">

<ASP:LinkButton Id="B1" Text="請按我" OnClick="B1_Click"

Runat="Server"/><p>

<ASP:Label Id="L1" Text="Label 控件" Runat="Server" />

</Form>

<Script Language="VB" Runat="Server">

Sub B1_Click(Sender As Object, e As EventArgs)

L1.Text="改變後的 Label 控件"

End Sub

</Script>

</Html>
```



ImageButton Web 控件

ImageButton Web 控件的作用和上述两个控件一样，不过这个控件是用图片来当做按钮。其使用语法为：

```
<ASP:ImageButton  
  
Id="被程序代码所控制的名称"  
  
Runat="Server"  
  
Command="命令名称"  
  
CommandArgument="命令参数"  
  
OnClick="事件程序名"  
  
>
```

这里要特别注意事件程序的参数接收。ImageButton Web 控件在触发 Click 事件时，会传递使用者在图形的哪个位置上按下鼠标按钮；所以参数 e 的型态要更改为 ImageClickEventArgs，若还是维持原先的 EventArgs 将发生错误。下列范例码在使用者按下 ImageButton Web 控件时，显示鼠标在哪个位置上按下按钮：

```
<Html>  
  
<Form Id="Form1" Runat="Server">
```



```
<ASP:ImageButton Id="Button1" ImageUrl="vsdotnet.gif"
OnClick="Button1_Click"
Runat="Server" /><p>

<ASP:Label Id="Label1" Runat="Server" />

</Form>

<Script Language="VB" Runat="Server">

Sub Button1_Click(Sender As Object, e As ImageClickEventArgs)

Label1.Text="您位于影像的 " & e.x.ToString & ", " & e.y.ToString & _

                " 的位置按下鼠标"

End Sub

</Script>

</Html>
```



TextBox Web 控件

这个 Web 控件和 `<Input Type="Text">`、`<Input Type="Password">` 以及 `<TextArea>` 这三个 HTML 元素，都一样用来接收键盘键入的数据；不过 `TextBox` 可以用来取代上述三种 HTML 元素。其使用语法为：

```
<ASP:TextBox
```

```
Id="被程序代码所控制的名称"
```

```
Runat="Server"
```

```
AutoPostBack="True | False"
```

```
Columns="字符数目"
```

```
MaxLength="字符数目"
```

```
Rows="列数"
```

```
Text="字符串"

TextMode="SingleLine | Multiline | Password"

Wrap="True | False"

OnTextChanged="事件程序名称"

/>
```

TextBox Web 控件的属性说明，如下表所示：

属性	说明
AutoPostBack	设定当按「Enter」或是「Tab」键跳开文字输入盒时，是否要自动触发 OnTextChanged 事件。
Columns	设定 TextBox 的长度为可输入多少字符。
MaxLength	设定 TextBox 可以接受的最大字符数目。
Rows	设定 TextBox 的高度为多少列，本属性在 TextMode 属性设为 MultiLine 才生效。
Text	设定 TextBox 中所显示的内容，或是取得使用者的输入。
TextMode	<p>共有三种设定值：</p> <ol style="list-style-type: none">1. SingleLine: 只可以输入一行（即 <Input Type="Text">）2. PassWord: 输入的字符以*代替（即 <Input Type="Password">）3. MultiLine: 可做多行输入（即 <TextArea>）

Wrap	设定是否自动断行。本属性在 TextMode 属性设为 MultiLine 才生效
------	---

由上表可知，**TextBox** 的型态是由 **TextMode** 属性来决定的，若没有设定本属性则预设为

SingleLine。下列范例显示了三种型态的 **TextBox**：

```
<Html>
```

```
<Form Id="Form1" Runat="Server">
```

这是一般输入盒：

```
<ASP:TextBox Id="T1" TextMode="SingleLine" Runat="Server"/><br>
```

这是密码输入盒：

```
<ASP:TextBox Id="T2" TextMode="Password" Runat="Server"/><br>
```

这是多行输入盒：

```
<ASP:TextBox Id="T3" TextMode="Multiline" Rows="3" Runat="Server"/><br>
```

```
</Form>
```

```
</Html>
```



TextBox 有一个 OnTextChanged 事件, 这个事件是当 TextBox 内的文字传至 Server 端后, Server 发现文字的内容和上次的值不同时就会触发; 另外不管 Text 属性的内容是否有被改变, 一率先触发 Page_Load 事件。和 Button Web 控件一样, 使用本事件前必须先指定发生这个事件时所执行的事件程序。下列范例码中, 倘若使用者在文字输入盒中输入的内容和上次不一样时, 在按下「Tab」或「Enter」按钮后会显示文字的内容已经被改变的讯息:

```
<Html>

<Form Id="Form1" Runat="Server">

<ASP:Textbox Id="T1" AutoPostBack="True" OnTextChanged="T1_Changed"

        Runat="server" /><p>

<ASP:Label Id="Label1" Runat="Server" />

</Form>

<Script Language="VB" Runat="Server">
```

```
Sub Page_Load(Sender As Object, e As EventArgs)
```

```
    Label1.Text="文字的内容没有被改变"
```

```
End Sub
```

```
Sub T1_Changed(Sender As Object, e As EventArgs)
```

```
    Label1.Text="文字的内容已经被改变"
```

```
End Sub
```

```
</Script>
```

```
</Html>
```

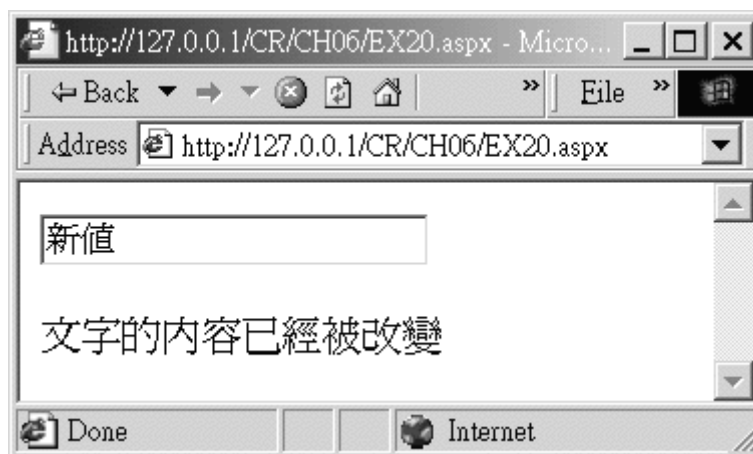


Table Web 控件、TableRow Web 控件及 TableCell Web 控件

Table Web 控件的用法和传统的 HTML 的 Table 元素差异很大,为了将网页设计对象导向,Table 内的列和字段也都跟着对象化了。前面我们已经提过 HTML 控件的 HtmlTable、HtmlTableRow、HtmlTableCell 这三个对象,基本上 Table Web 控件里的使用方式和 Table HTML 控件没有什么差别。其使用语法为:

```
<ASP:Table
  Id="被程序代码所控制的名称"
  Runat="Server"
  BackImageUrl="URL"
  CellSpacing="像素"
  CellPadding="像素"
  GridLines="Both | Horizontal | None | Vertical"
  HorizontalAlign="Center | Justify | Left | NotSet | Right"
/>
```

Table Web 控件的基本属性如下所示:

属性	说明
BackImageUrl	设定表格的背景图形。
CellPadding	设定储存格与表格边框的距离。
CellSpacing	设定储存格和储存格边框的距离。

GridLines	<p>设定表格内的水平线或垂直线是否出现，有四种值：</p> <ol style="list-style-type: none"> 1. None: 两者都不出现。 2. Horizontal: 只出现水平线。 3. Vertical: 只出现垂直线。 4. Both: 两者都出现。
HorizontalAlign	设定水平对齐方式。
Rows	TableRow 集合对象，用来设定或取得 Table 中有多少列。

我们知道 **TableCell** 对象是 **TableRow** 的子对象，而 **TableRow** 是 **Table** 的子物件。只要利用 **TableRow.Cells.Add** 及 **Table.Rows.Add** 方法就可以建立这些对象的关系。表格的制作方式有两种，一是使用类似 **HTML** 标注方法，另外一种是用程序动态新增。第一种方法如下范例码所示：

```
<Html>

<ASP:Table Id="Table1" BackImageUrl="vsback.gif" Border="1"
Runat="Server">

    <ASP:TableRow>

        <ASP:TableCell>第一列第一行</ASP:TableCell>

        <ASP:TableCell>第一列第二行</ASP:TableCell>

        <ASP:TableCell>第一列第三行</ASP:TableCell>
```



```
</ASP:TableRow>

<ASP:TableRow>

    <ASP:TableCell>第二列第一行</ASP:TableCell>

    <ASP:TableCell>第二列第二行</ASP:TableCell>

    <ASP:TableCell>第二列第三行</ASP:TableCell>

</ASP:TableRow>

<ASP:TableRow>

    <ASP:TableCell>第三列第一行</ASP:TableCell>

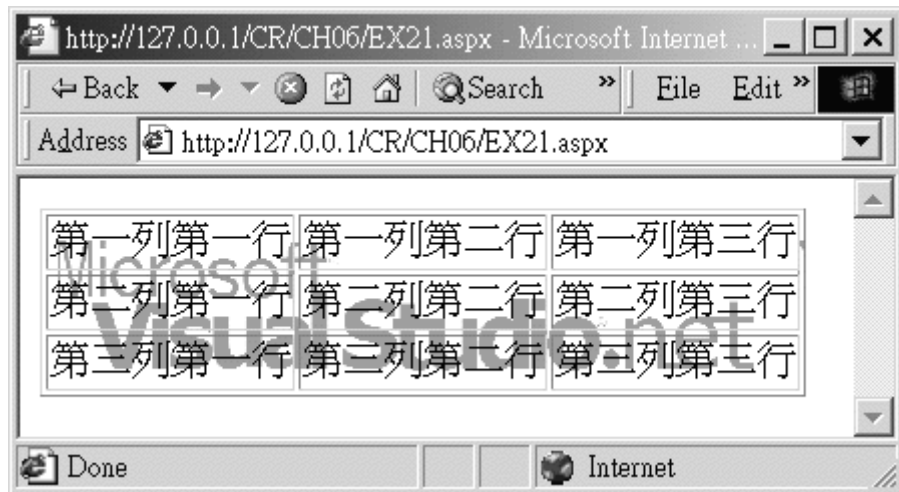
    <ASP:TableCell>第三列第二行</ASP:TableCell>

    <ASP:TableCell>第三列第三行</ASP:TableCell>

</ASP:TableRow>

</ASP:Table>

</Html>
```



上面这个程序看起来和 HTML 标注里的 Table 元素几乎一模一样,只不过是标注的名称改为 Web 控件的名称。另外字段内所要显示的的文字,除了使用上面程序的写法外上可写成下列的样式:

```
<ASP:TableCell Text="第一列第一行"/>
```

第二种用程序来动态新增的方法和 HtmlTable 控件一样,我们将 HTML 控件的九九表范例改成用 Web 控件来写,如下所示:

```
<Html>

<Form Id="Form1" Runat="Server">

    <ASP:Table Id="Table1" Border="1" Runat="Server" Font-Size=14/>

    <ASP:Button Id="Button1" Text="请按我" OnClick="Button1_Click"
Runat="Server"/>

</Form >

<Script Language="VB" Runat="Server">

Sub Button1_Click(Sender As Object, e As EventArgs)
```

```
Dim Cell As TableCell

Dim Row As TableRow

Dim X, Y As Short


For X=1 To 9 Step 1

    Cell=New TableCell


        For Y=1 To 9 Step 1

            Cell.Text+=CStr(X) & " * " & CStr(Y) & " = " & CStr(X * Y)

            If Y<>9 Then Cell.Text+="<br>"

        Next Y

        If X=1 Or X=4 Or X=7 Then Row=New TableRow

        Row.Cells.Add(Cell)

        If X=3 Or X=6 Or X=7 Then Table1.Rows.Add(Row)

    Next X


End Sub

</Script>

</Html>
```

上面这个程序的用法和 **HtmlTable** 控件一样，只不过将 **Html** 控件改成 **Web** 控件罢了。要在表

格中显示文字不是问题，若要在表格中放置控件也可以，只要使用 **TableCell** 对象中 **Controls**

集合的 **Add** 方法即可。下列范例码显示如何将对象放到表格中：

```
<Html>

<ASP:Table Id="Table1" BorderWidth="1px" GridLines="Both"

Cellspacing="0"

Cellpadding="1"

Runat="Server" />

<Script Language="VB" Runat="Server">

    Sub Page_Load(Sender As Object, e As EventArgs)

        Dim I, J As Short

        For I=0 To 4

            Dim Row As New TableRow

            For J=0 to 3

                Dim Cell As New TableCell

                Cell.Text=" Column=" & J

                If J=3 Then

                    Dim btnA As New Button

                    btnA.Text ="Column=3"
```

```

        Cell.Controls.Add(btnA)

    End If

    Row.Cells.Add(Cell)

Next

Table1.Rows.Add(Row)

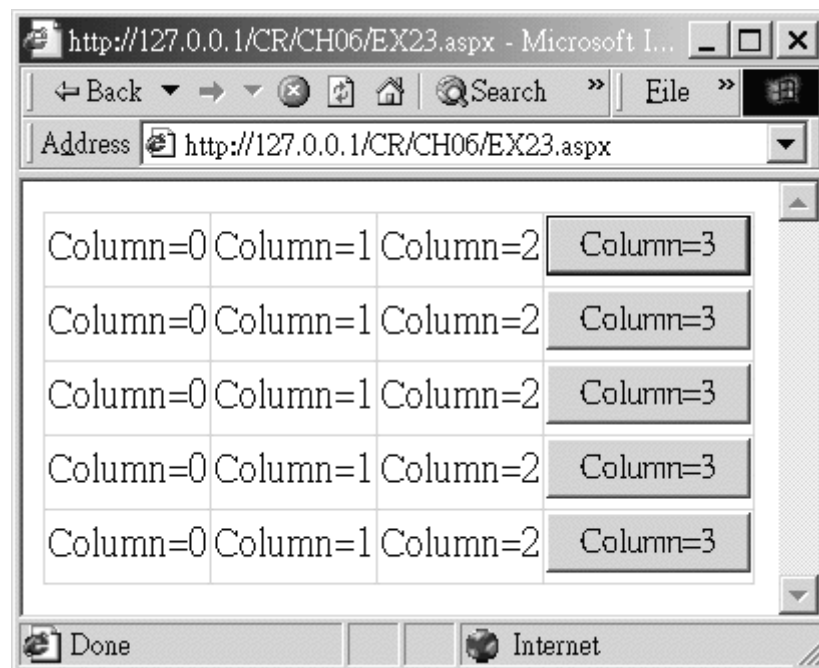
Next

End Sub

</Script>

</Html>

```



上述程序代码要产生每一列的第四栏时，我们就撰写程序动态的产生一个 **Button** 控件，然后将这个 **Button** 控件的 **Text** 属性设为 **Column=3** 后，利用 **Controls** 集合的 **Add** 方法将 **Button** 控件加入 **Cell** 对象的 **Controls** 集合对象中，最后产生一个第四字段为 **Button** 控件的 4 乘 5 表格。

Panel Web 控件

Panel Web 控件可以让我们群组控件，让我们决定在 **Panel** 中的控件是否要显示出来。其使用语法为：

```
<ASP:Panel  
  
Id="被程序代码所控制的名称"  
  
Runat="Server"  
  
BackImageUrl="URL"  
  
HorizontalAlign="Center | Justify | Left | NotSet | Right"  
  
Wrap="True | False"  
  
>  
  
  其它控件...  
  
</ASP:Panel>
```

Panel Web 控件的基本属性如下所示：

属性	说明
BackImageUrl	设定 Panel 的背景图形。
HorizontalAlign	设定水平对齐方式。
Wrap	设定是否自动折行，预设值为 True。
Visible	设定是否显示，预设值为 True。

下面范例为计算男女标准体重：

```
<Html>

<Form Id="Form1" Runat="Server">

<ASP:Panel Id="Main" Runat="Server"> <!--配置主 Panel-->

    标准体重计算程序<Hr>

    依您的性别进入不同的计算方法

    <ASP:Button Id="btnMan" Text="我是男生" OnClick="btnMan_Click"

Runat="Server"/>

    <ASP:Button Id="btnWoman" Text="我是女生" OnClick="btnWoman_Click"

        Runat="Server" />

</ASP:Panel>

<ASP:Panel Id="Man" Runat="Server"> <!--配置计算男生体重的 Panel-->

    输入您的身高： <ASP:TextBox Id="txtMan" Runat="Server" />
```

```
<ASP:Button Id="ShowResult1" Text="看结果" OnClick="ShowMan"
Runat="Server" />

</ASP:Panel>

<ASP:Panel Id="Woman" Runat="Server"> <!--配置计算女生体重的 Panel-->

    输入您的身高: <ASP:TextBox Id="txtWoman" Runat="Server" />

    <ASP:Button Id="ShowResult2" Text="看结果" OnClick="ShowWoman"
Runat="Server" />

</ASP:Panel>

<ASP:Panel Id="Result" Runat="Server"> <!--配置显示标准体重的 Panel-->

    您的标准体重为: <ASP:Label Id="lblBody" Runat="Server" />公斤

</ASP:Panel>

</Form>

<Script Language="VB" Runat="Server">

Sub Page_Load(Sender As Object, e As EventArgs)

If Not Page.IsPostBack Then '第一次执行时只显示主 Panel

    Man.Visible=False

    Woman.Visible=False

    Result.Visible=False

End If
```



```
End Sub
```

```
Sub btnMan_Click(Sender As Object, e As EventArgs)
```

```
    Main.Visible=False ' 将主 Panel 隐藏, 计算男生体重的 Panel 显示
```

```
    Man.Visible=True
```

```
End Sub
```

```
Sub btnWoman_Click(Sender As Object, e As EventArgs)
```

```
    Main.Visible=False ' 将主 Panel 隐藏, 计算女生体重的 Panel 显示
```

```
    Woman.Visible=True
```

```
End Sub
```

```
Sub ShowMan(Sender As Object, e As EventArgs)
```

```
    Man.Visible=False ' 将计算男生体重的 Panel 隐藏, 显示计算结果
```

```
    Result.Visible=True
```

```
    lblBody.Text=Cint((Cint(txtMan.Text)-80)*0.7)
```

```
End Sub
```

```
Sub ShowWoman(Sender As Object, e As EventArgs)
```

```
    Woman.Visible=False ' 将计算女生体重的 Panel 隐藏, 显示计算结果
```

```
    Result.Visible=True
```

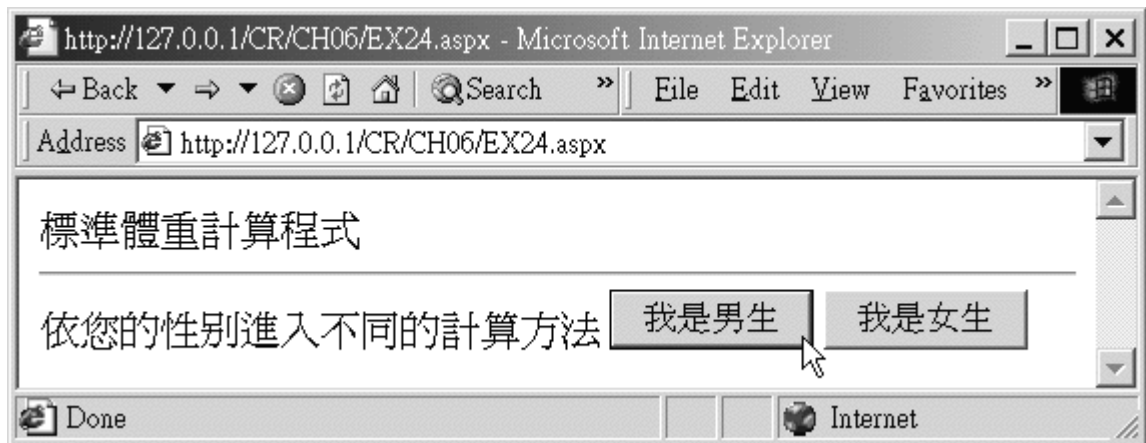
```
    lblBody.Text=Cint((Cint(txtWoman.Text)-70)*0.6)
```

```
End Sub
```

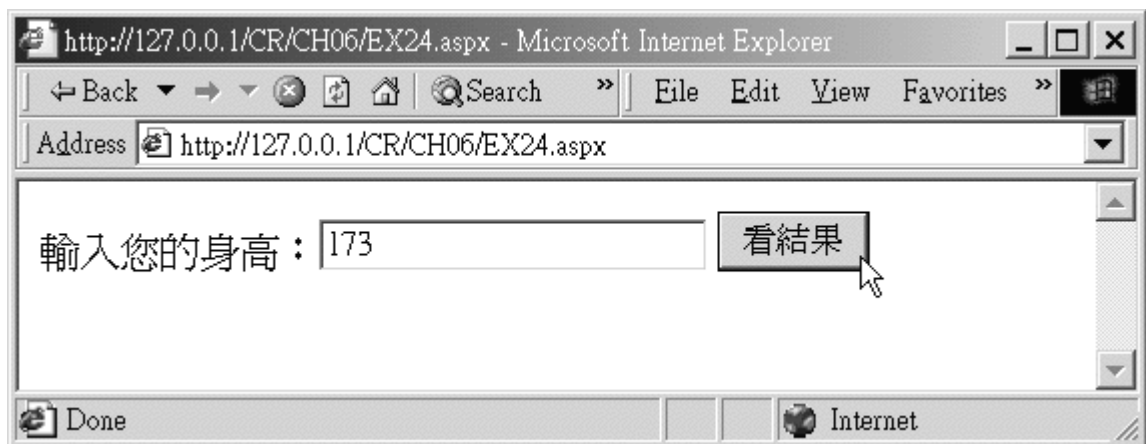
</Script>

</Html>

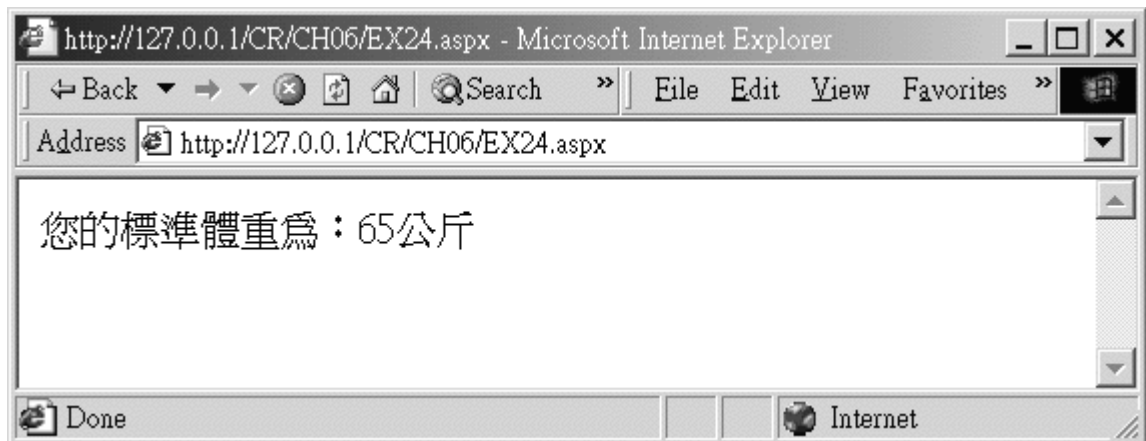
1. 程序开始执行时只会出现 Id 为 Main 的 Panel Web 控件，如下图所示：



2. 我们选择性别后，接着输入自己的身高：



3. 最后出现标准体重：



善用 Panel Web 控件，不但可以让我们群组对象外，还可以在同一个 ASPX 网页中执行许多程序，而不用分散成许多 ASPX 档案。

列举控件（ListControl）

当我们在使用标准窗口程序时，会遇到一些例如清单盒（ListBox）等控件；这些控件被限制只能选择控件内所提供项目的数据，这种类型的控件就是我们这里要讨论的列举控件。ASP.NET Web 列举控件包括了：

- CheckBox
- CheckBoxList
- DropDownList
- ListBox
- RadioButton
- RadioButtonList

AutoPostBack 属性及 Page.IsPostBack 属性

在介绍列举控件之前我们先来复习 AutoPostBack 以及 Page.IsPostBack 这两个属性。

Page.IsPostBack 属性

Page.IsPostBack 是用来检查目前网页是否为第一次加载，当使用者第一次浏览这个网页时

Page.IsPostBack 会传回 **False**，不是第一次浏览这个网页时就传回 **True**；所以当我们在

Page_Load 事件中就可以使用这个属性来避免做一些重复的动作。下列范例利用

Page.IsPostBack 属性来判断网页是不是第一次载入：

```
<Html>

<ASP:Label Id="lblA" Runat="Server"/>

<Form Id="Form1" Runat="Server">

    <ASP:Button Id="btnA" Runat="Server" Text="请按我"

OnClick="btnA_Click"/>

</Form>

<Script Language="VB" Runat="Server">

Sub Page_Load(Sender As Object, e As EventArgs)

If Page.IsPostBack Then

    lblA.Text="网页不是第一次载入"

Else

    lblA.Text="网页是第一次载入"

End If

End Sub
```

```
Sub btnA_Click(Sender As Object,e As EventArgs)

    Response.Write("您按了按钮")

End Sub

</Script>

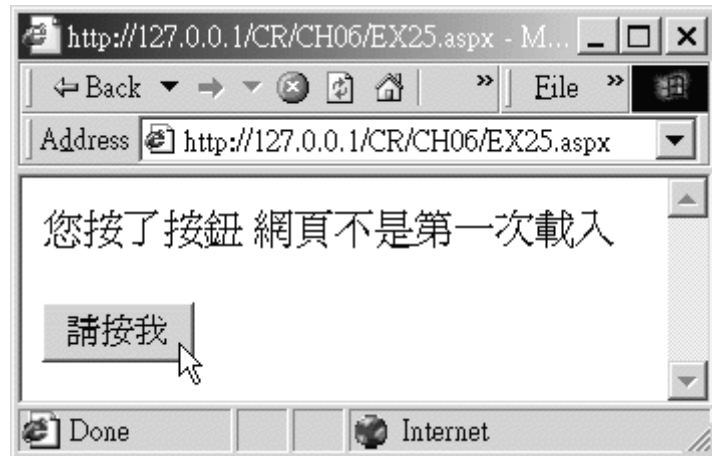
</Html>
```

所以第一次执行程序的时候，显示网页是第一次加载：



待按下按钮触发 **Page_Load** 事件时，由于网页不是第一次载入，**Page.IsPostBack** 属性此时就

传回 **True**，所以显示网页不是第一次加载：



AutoPostBack 属性

以 **TextBox Web** 控件为例，若我们把 **AutoPostBack** 属性在设定为 **True**，并且指定

OnTextChanged 的事件程序为何时，当使用者按下「Enter」或是「Tab」让光标离开此控件而

且控件的内容有所改变时，将自动传回控件现在的内容并触发 **Page_Load** 事件及

OnTextChanged 属性所设定的事件。支持 **AutoPostBack** 属性的 **Web** 控件以及事件如下表所示：

控件名称	指定触发事件的属性
CheckBox	OnCheckChanged
CheckBoxList	OnSelectedIndexChanged
DropDownList	OnSelectedIndexChanged
ListBox	OnSelectedIndexChanged
RadioButton	OnCheckChanged
RadioButtonList	OnSelectedIndexChanged

TextBox	OnTextChanged
---------	---------------

RadioButton Web 控件

RadioButton Web 控件的基本功能即是 HTML 控件的 `<Input Type="Radio">`。它的外观及功能

我们已经在 HTML 控件中展示过了，不过它比 HTML 控件的功能更强，其使用语法如下：

```
<ASP:RadioButton  
  
    Id="被程序代码所控制的名称"  
  
    Runat="Server"  
  
    AutoPostBack="True | False"  
  
    Checked="True | False"  
  
    GroupName="群组名称"  
  
    Text="标示控件的文字"  
  
    TextAlign="设定文字在控件的左边或右边"  
  
    OnCheckedChanged="事件程序名称"  
  
>
```

RadioButton Web 控件常用的属性如下表所示：

属性	说明
----	----

AutoPostBack	设定当使用者选择不同的项目时，是否自动触发 OnCheckedChanged 事件。
Checked	传回或设定是否该项目被选取。
GroupName	传回或设定按钮所属群组。
TextAlign	传回或设定项目所显示的文字是在按钮的左方或右方，预设是 Right。
Text	传回或设定 RadioButton 中所显示的内容。

下列范例码放置了两个 **RadioButton Web** 控件，并在 **Page_Load** 事件中设定控件的初始值。

```
<Html>

<Form Id="Form1" Runat="Server">

    <ASP:RadioButton Id="Radio1" Text="第一个 Radio" Runat="Server"/><br>

    <ASP:RadioButton Id="Radio2" Text="第二个 Radio" Runat="Server"/>

</Form>

<Script Language="VB" Runat="Server">

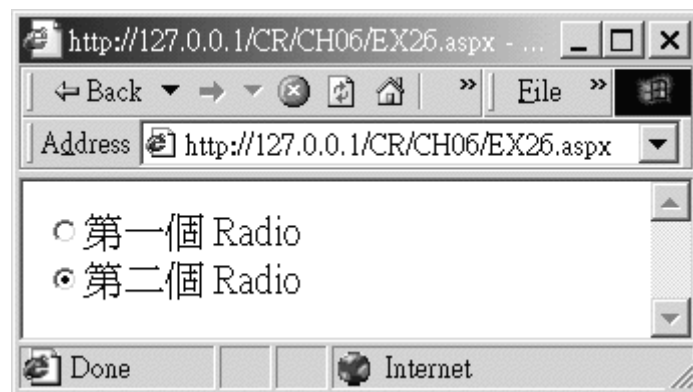
Sub Page_Load(Sender As Object,e As EventArgs)

    Radio2.Checked="True"    ' 让第二个 Radio 变成选取

End Sub

</Script>

</Html>
```



这个范例中 **RadioButton** 是两个独立的 **Web** 控件，若我们希望在一群 **RadioButton Web** 控制中只能选择一个时，只要将它们 **GroupName** 设为同一个即可。此时我们只能在这些 **RadioButton Web** 控件中选择其中一个。下列程序代码范例中，我们限制了只能在三个 **RadioButton Web** 控件中选择一个项目：

```
<Html>

<Form Id="Form1" Runat="Server">

<ASP:RadioButton Id="Radio1" Text="RadioButton1" GroupName="Group1"
Checked="True"

        Runat="Server"/><Br>

<ASP:RadioButton Id="Radio2" Text="RadioButton2" GroupName="Group1"
Runat="Server"/><Br>
```

```
<ASP:RadioButton Id="Radio3" Text="RadioButton3" GroupName="Group1"
Runat="Server"/><P>
```

```
<ASP:Button Id="Button1" Text="Check" OnClick="Button1_Click"
Runat="Server"/><P>
```

```
<ASP:Label Id="Label1" Runat="Server"/>
```

```
</Form>
```

```
<Script Language="VB" Runat="Server">
```

```
Sub Button1_Click(Sender As Object, e As EventArgs)
```

```
If Radio1.Checked Then Label1.Text="你选择了 RadioButton1"
```

```
If Radio2.Checked Then Label1.Text="你选择了 RadioButton2"
```

```
If Radio3.Checked Then Label1.Text="你选择了 RadioButton3"
```

```
End Sub
```

```
</Script>
```

```
</Html>
```



AutoPostBack 属性以及 CheckedChanged 事件

RadioButton Web 控件有 CheckedChanged 事件，这个事件是在当 RadioButton 控件的选择状态发生改变时触发；要触发这个事件，必须把 AutoPostBack 属性设为 True 才生效。下列程序代码范例将上述程序改成不需要按下按钮，只要使用者选择的项目不一样就会触发

CheckedChanged 事件：

```
<Html>

<Form Id="Form1" Runat="Server">

    <ASP:RadioButton Id="Radiol" Text="RadioButton1" GroupName="Group1"

        AutoPostBack="True"
```

```
OnCheckedChanged="Check_Clicked"
```

```
Runat="Server"/><Br>
```

```
<ASP:RadioButton Id="Radio2" Text="RadioButton2" GroupName="Group1"
```

```
AutoPostBack="True"
```

```
OnCheckedChanged="Check_Clicked"
```

```
Runat="Server"/><p>
```

```
<ASP:Label Id="Label1" Runat="Server"/>
```

```
</Form>
```

```
<Script Language="VB" Runat="Server">
```

```
Sub Check_Clicked(Sender As Object, e As EventArgs)
```

```
    If Radio1.Checked Then Label1.Text="Radio1"
```

```
    If Radio2.Checked Then Label1.Text="Radio2"
```

```
End Sub
```

```
</Script>
```

```
</Html>
```



RadioButtonList Web 控件

由于每一个 **RadioButton Web 控件** 是独立的控件，若要判断同一个群组内的 **RadioButton** 是否被选择，则必须判断所有的 **RadioButton Web 控件** 的 **Checked** 属性，这样判断实在是没效率。所以微软便制作了 **RadioButtonList Web 控件**，这个 **RadioButtonList** 控件可以管理许多选项，其使用语法如下：

```
<ASP:RadioButtonList  
  
    Id="被程序代码所控制的名称"  
  
    Runat="Server"  
  
    AutoPostBack="True | False"  
  
    CellPadding="像素"  
  
    *DataSource="<%数据系结叙述%>"  
  
    *DataTextField="数据源的字段"
```

```

    *DataValueField="数据源的字段"

    RepeatColumns="字段数量"

    RepeatDirection="Vertical | Horizontal"

    RepeatLayout="Flow | Table"

    TextAlign="Right | Left"

    OnSelectedIndexChanged="事件程序名称"

>

<ASP:ListItem/>

</ASP:RadioButtonList>

```

*关于和数据源的数据系结部分，我们在后面的章节再介绍。

RadioButtonList Web 控件的属性和 **RadioButton Web 控件**并不太相同,下表为 **RadioButtonList**

Web 控件的常用属性:

属性	说明
AutoPostBack	设定是否要致能 OnSelectedIndexChanged 事件。
CellPadding	设定 RadioButtonList Web 控件中各项目之间的距离，单位是像素。
*DataSource	设定数据系结所要使用的数据源。
*DataTextField	设定资料系结所要显示的字段。
*DataValueFiled	设定选项的相关数据要使用的字段。

Items	传回 RadioButtonList Web 控件中 ListItem 的参考。
RepeatColumns	设定 RadioButtonList Web 控件项目的横向字段数。
RepeatDirection	设定 RadioButtonList Web 控件的排列方式是以水平排列（Horizontal）还是垂直（Vertical）排列。
RepeatLayout	设定 RadioButtonList Web 控件的 ListItem 排列方式为要使用 Table 来排列还是直接排列，预设是 Table。
SelectedIndex	传回被选取到 ListItem 的 Index 值。
SelectedItem	传回被选取到 ListItem 参考，也就是 ListItem 本身。
TextAlign	设定 RadioButtonList Web 控件中各项目所显示的文字是在按钮的左方或右方，预设是 Right。

RadioButtonList Web 控件的使用方法和 HTML 控件的 Select 控件很像，只要先安置好

RadioButtonList Web 控件，接着设定它的子对象 ListItem Web 控件即可产生一组群组好的

RadioButton Web 控件。我们先来介绍 ListItem Web 控件：

ListItem Web 控件

ListItem Web 控件并不是一个独立存在的控件，它必须依附在下列几种 Web 控件下；例如

RadioButtonList Web 控件、DropDownList Web 控件以及 CheckBoxList 控件。一个 ListItem Web

控件代表的是一个 **ListControl Web** 控件的选项内容，也因为如此所以可以不需要指定 **Id** 属性。

其使用语法如下：

```
<ASP:ItemList  
  
    Id="被程序代码所控制的名称"  
  
    Runat="Server"  
  
    Selected="True | False"  
  
    Text="标示项目的文字"  
  
    Value="相关资料"  
  
>
```

或

```
<ASP:ItemList  
  
    Id="被程序代码所控制的名称"  
  
    Runat="Server"  
  
    Selected="True | False"  
  
    Value="相关资料"  
  
>
```

标示项目的文字

```
</ASP:ItemList>
```

下表列出 **ListItem Web** 控件常用的属性说明：

属性	说明
Selected	传回或设定此项目是否被选取
Text	标示项目的文字
Value	传回或设定和这个 Item 相关的数据

其使用的方式如下所示：

```
<ASP:ListItem>Item1</ASP:ListItem >
```

或

```
<ASP:ListItem Text="Item1" />
```

当我们使用程序来产生一个 **ListItem Web** 控件的实体时，其建构方式有三种：

1. Dim liA As New ListItem
2. Dim liA As New ListItem("Item1")
3. Dim liA As New ListItem("Item1","Item Value")

第二种方式在建构实体时，一并设定其 **Text** 属性；第三种方式则是设定其 **Text** 属性及 **Value** 属性。**Value** 属性和 **Text** 属性的型态一样都是字符串，但是 **Text** 属性的内容会显示出来而 **Value** 不会。当我们在页面上显示的内容和实际要做运算的数据不同时，就可以利用这个属性。下列范例

利用 **RadioButtonList Web** 控件以及 **Listitem Web** 控件让使用者选择性别，并且指定其相关的

Value 属性为 M 及 F，如下范例所示：

```
<Html>

<Form Id="Form1" Runat="Server">

    <ASP:RadioButtonList Id="rblA" Runat="Server">

        <ASP:ListItem Text="男" Selected="True" Value="M"/>

        <ASP:ListItem Text="女" Value="F"/>

    </ASP:RadioButtonList>

    <ASP:Button Id="B1" Runat="Server" Text=' 确定' OnClick="B1_Click"/>

</Form>

<ASP:Label Id="Label1" Runat="Server"/>

<Script Language="VB" Runat="Server">

Sub B1_Click(Sender As Object,e As EventArgs)

    Label1.Text="您选择了 " & rblA.SelectedItem.Text & "，它的相关值为 "

    & _

        rblA.SelectedItem.Value

End Sub

</Script>
```

```
</Html>
```



上列程序代码范例中，只要直接参考 **RadioButtonList Web** 控件的 **SelectedItem** 属性，就可以取得被选取到的 **Listltem** 对象。**RadioButtonList Web** 控件内的项目也可以用程序来动态的新增，我们只要先产生一个 **Listltem** 型态的对象变量，再用 **RadioButtonList Web** 控件 **Items** 集合的 **Add** 方法将这个对象加到 **Items** 集合内即可。下列程序代码范例动态的增加六个 **Listltem**，并排列成两栏：

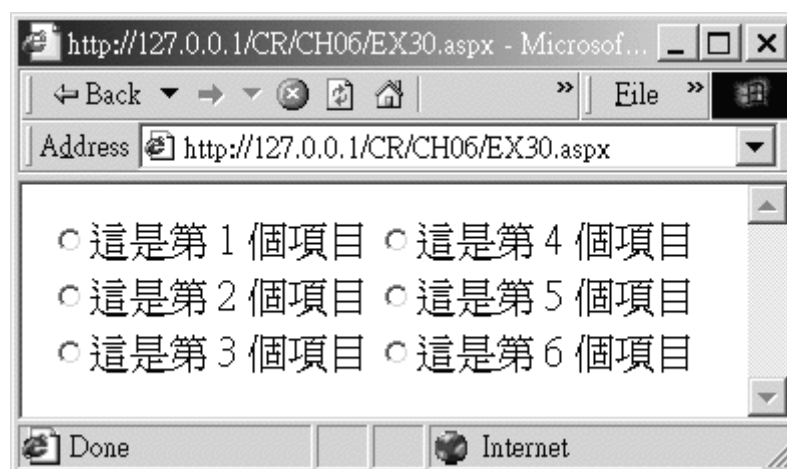
```
<Html>

<Form Id="Form1" Runat="Server">

    <ASP:RadioButtonList Id="rblA" RepeatColumns="2" Runat="Server" />

</Form>
```

```
<Script Language="VB" Runat="Server">  
  
Sub Page_Load(Sender As Object, e As EventArgs)  
  
    Dim shtI As Short  
  
    For shtI=1 To 6  
  
        Dim liA As New ListItem  
  
        liA.Text="这是第 " & shtI.ToString & " 个项目"  
  
        rblA.Items.Add(liA)  
  
    Next  
  
End Sub  
  
</Script>  
  
</Html>
```



CheckBox Web 控件

CheckBox Web 控件也是提供给使用者从选项中作选择的对象，相对于 HTML 控件为 `<Input Type="CheckBox">`。CheckBox Web 控件和 RadioButton Web 控件不同的地方是它可以重复选取，其使用语法如下所示：

```
<ASP:CheckBox  
  
    Id="被程序代码所控制的名称"  
  
    Runat="Server"  
  
    AutoPostBack="True | False"  
  
    Text="控件的文字"  
  
    TextAlign="控件文字出现在左方或右方"  
  
    Checked="True | False"  
  
    OnCheckedChanged="事件程序名称"  
  
>
```

CheckBox Web 控件常用的属性如下表所示：

属性	说明
AutoPostBack	设定当使用者选择不同的项目时，是否自动触发 OnCheckedChanged 事件。
Checked	传回或设定是否该项目被选取。

GroupName	传回或设定按钮所属群组。
TextAlign	传回或设定项目所显示的文字是在选取盒的左方或右方，预设是右方（Right）。
Text	传回或设定 CheckBox 中所显示的内容。

CheckBox Web 控件支持 **CheckedChanged** 事件，使用的方式和 **RadioButton Web** 控件一样。

下列程序代码范例中，我们在下面的程序中布置了一个 **CheckBox Web** 控件，并配合

CheckedChanged 事件和 **AutoPostBack** 属性。每当使用者按下画面中的 **CheckBox Web** 控件

时，由于 **Checked** 属性改变，所以触发 **CheckedChanged** 事件：

```
<Html>

<Form Id="Form1" Runat="Server">

    <ASP:CheckBox Id="CheckBox1" Text="Item Checked"

OnCheckedChanged="Check_Clicked"

                AutoPostBack="True" Runat="server"/><P>

    <ASP:Label Id="Label1" Runat="Server"/>

</Form>

<Script Language="VB" Runat="Server">

Sub Check_Clicked(Sender As Object, e As EventArgs)

    If CheckBox1.Checked Then

        Label1.Text="Checked"
```

```
Else

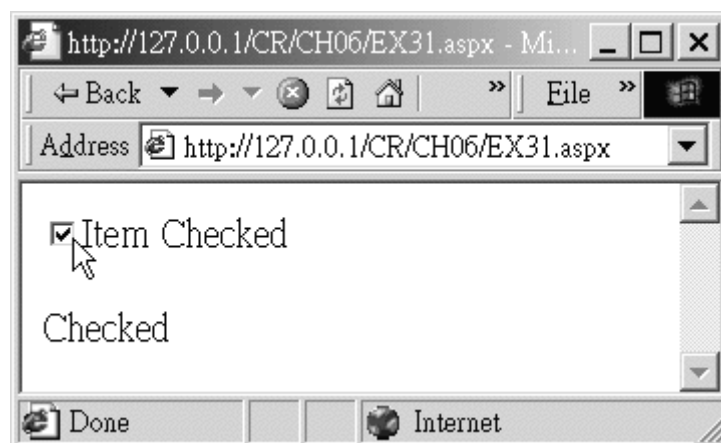
    Label1.Text="Not Checked"

End If

End Sub

</Script>

</Html>
```



CheckBoxList 控件

如果当我们要使用一群的 **CheckBox Web 控件** 时,在程序的判断上非常麻烦,因此 **CheckBoxList**

Web 控件和 **RadioButtonList Web 控件**一样是让我们方便的取得使用者选取的项目。其使用语

法如下:


```
<ASP:CheckBoxList

    Id="被程序代码所控制的名称"

    Runat="Server"

    AutoPostBack="True | False"

    CellPadding="像素"

    *DataSource="<%数据系结叙述%>"

    *DataTextField="数据源的字段"

    *DataValueField="数据源的字段"

    RepeatColumns="字段数量"

    RepeatDirection="Vertical | Horizontal"

    RepeatLayout="Flow | Table"

    TextAlign="Right | Left"

    OnSelectedIndexChanged="事件程序名称"

>

    <ASP:ListItem/>

</ASP:CheckBoxList>
```

*关于和数据源的数据系结部分，我们在后面的章节再介绍。

CheckBoxList Web 控件的属性和 **CheckBox Web** 控件的属性并不太相同，下表为 **CheckBoxList**

Web 控件的常用属性：

属性	说明
AutoPostBack	设定是否要致能 OnSelectedIndexChanged 事件。
CellPadding	设定 CheckBoxList Web 控件中各项目之间的距离，单位是像素。
*DataSource	设定数据系统所要使用的数据源。
*DataTextField	设定资料系统所要显示的字段。
*DataValueField	设定选项的相关数据要使用的字段。
Items	传回 CheckBoxList Web 控件中 ListItem 的参考。
RepeatColumns	设定 CheckBoxList Web 控件项目的横向字段数。
RepeatDirection	设定 CheckBoxList Web 控件的排列方式是以水平排列（Horizontal）还是垂直（Vertical）排列。
RepeatLayout	设定 CheckBoxList Web 控件的 ListItem 排列方式为要使用 Table 来排列还是直接排列，预设是 Table。
SelectedIndex	传回被选取到 ListItem 的 Index 值。
SelectedItem	传回被选取到 ListItem 参考，也就是 ListItem 本身。
SelectedItems	由于 CheckBoxList Web 控件可以复选，被选取的项目会被加入 ListItems 集合中；本属性可以传回 ListItems 集合，只读。
TextAlign	设定 CheckBoxList Web 控件中各项目所显示的文字是在按钮的左方或右方，预设是 Right。

下列程序代码范例显示一个简单的 **CheckBoxList Web** 控件，可让使用者选择：

```
<Html>

<Form Id="Form1" Runat="Server">

请输入您的兴趣:<br>

    <ASP:CheckBoxList Id="cblA" Runat="server">

        <ASP:ListItem>打球</ASP:ListItem>

        <ASP:ListItem>看书</ASP:ListItem>

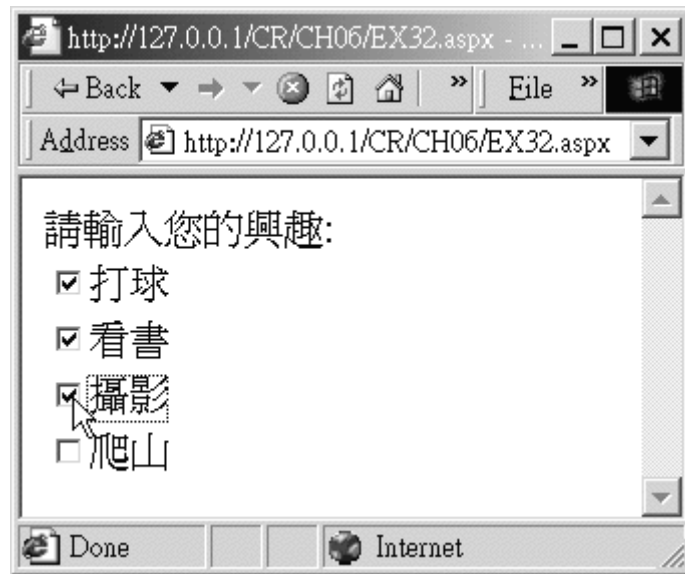
        <ASP:ListItem>摄影</ASP:ListItem>

        <ASP:ListItem>爬山</ASP:ListItem>

    </ASP:CheckBoxList>

</Form>

</Html>
```



CheckBoxList Web 控件的用法和 RadioButtonList Web 类似，不过 CheckBoxListd Web 控件的项目可以复选。选择完毕后的结果可以利用 **Items** 集合作检查，只要判断 **Items** 集合对象中哪一个项目的 **Selected** 属性为 **True**，即表示项目有被选择；如下范例所示：

```
<Html>

<Form Id="Form1" Runat="Server">

请输入您的兴趣:<br>

    <ASP:CheckBoxList Id="cblA" Runat="Server">

        <ASP:ListItem>打球</ASP:ListItem>

        <ASP:ListItem>看书</ASP:ListItem>

        <ASP:ListItem>摄影</ASP:ListItem>

        <ASP:ListItem>爬山</ASP:ListItem>

    </ASP:CheckBoxList>

    <ASP:Button Id="btnA" Text="确定" OnClick="btnA_Click" Runat="Server"/>

</Form>

</Html>
```

```
</Form>
```

```
<ASP:Label Id="lblA" Runat="Server"/>
```

```
<Script Language="VB" Runat="Server">
```

```
Sub btnA_Click(Sender As Object,e As EventArgs)
```

```
Dim shtI As Short
```

```
    lblA.Text=""
```

```
    For shtI=0 To cblA.Items.Count-1
```

```
        If cblA.Items(shtI).Selected=True Then
```

```
            lblA.Text & = "第 " & (shtI+1).ToString & " 个项目被选择<br>"
```

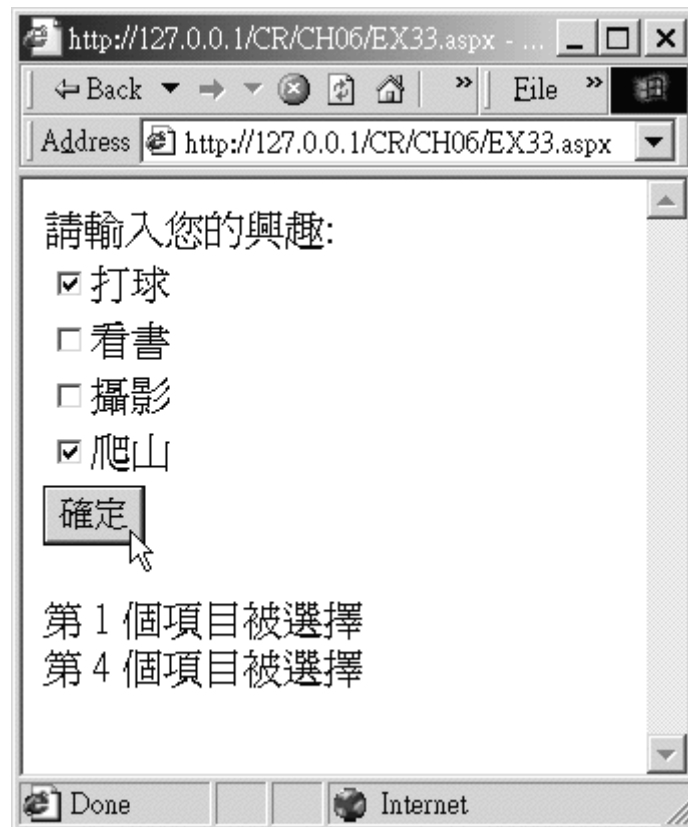
```
        End If
```

```
    Next
```

```
End Sub
```

```
</Script>
```

```
</Html>
```



DropDownList Web 控件

DropDownList Web 控件是一个下拉式的选单，功能和 RadioButtonList Web 控件很类似，提供使用者在一群选项中选择单一的值；不过 RadioButtonList Web 控件适合使用在较少量的选项群组项目，而 DropDownList Web 控件则适合用来管理大量的选项群组项目。其使用语法如下：

```
<ASP:DropDownList
```

```
    Id="被程序代码所控制的名称"
```

```
    Runat="Server"
```

```
    AutoPostBack="True | False"
```

```

*DataSource="<%数据系统叙述%>"

*DataTextField="数据源的字段"

*DataValueField="数据源的字段"

OnSelectedIndexChanged="事件程序名称"

>

<ASP:ListItem/>

</ASP:DropDownList>

```

*关于和数据源的数据系统结部分，我们在后面的章节再介绍。

下表为 **DropDownList Web** 控件的常用属性：

属性	说明
AutoPostBack	设定是否要致能 OnSelectedIndexChanged 事件。
*DataSource	设定数据系统所要使用的数据源。
*DataTextField	设定资料系统所要显示的字段。
*DataValueFiled	设定选项的相关数据要使用的字段。
Items	传回 DropDownList Web 控件中 ListItem 的参考。
SelectedIndex	传回被选取到 ListItem 的 Index 值。
SelectedItem	传回被选取到 ListItem 参考，也就是 ListItem 本身。

下列范例将 **DropDwonList Web** 控件填入十二个月：

```
<Html>

<ASP:DropDownList Id="ddlA" Runat="Server">

    <ASP:ListItem>1 月</ASP:ListItem>

    <ASP:ListItem>2 月</ASP:ListItem>

    <ASP:ListItem>3 月</ASP:ListItem>

    <ASP:ListItem>4 月</ASP:ListItem>

    <ASP:ListItem>5 月</ASP:ListItem>

    <ASP:ListItem>6 月</ASP:ListItem>

    <ASP:ListItem>7 月</ASP:ListItem>

    <ASP:ListItem>8 月</ASP:ListItem>

    <ASP:ListItem>9 月</ASP:ListItem>

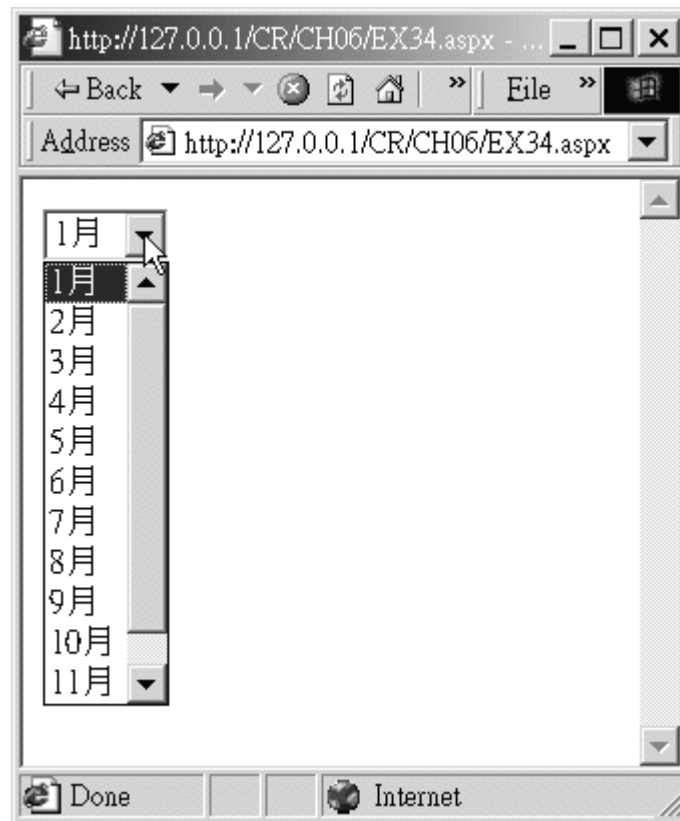
    <ASP:ListItem>10 月</ASP:ListItem>

    <ASP:ListItem>11 月</ASP:ListItem>

    <ASP:ListItem>12 月</ASP:ListItem>

</ASP:DropDownList>

</Html>
```

不过上列的方法太麻烦，我们可以利用 **Page_Load** 事件用程序动态的加入项目，如下范例所示：

```
<Html>

<ASP:DropDownList Id="ddlA" Runat="Server"/>

<Script Language="VB" Runat="Server">

Sub Page_Load(Sender As Object, e As EventArgs)

    Dim shtI As Short

    Dim liA As ListItem

    For shtI=1 To 12

        liA=New ListItem(shtI.ToString & "月")

        ddlA.Items.Add(liA)

    End For

End Sub

</Script>

</Html>
```

```
Next  
  
End Sub  
  
</Script>  
  
</Html>
```

这样程序代码就简洁多了。若要取得 **DropDownList Web** 控件被选取到的项目，则可以利用和 **RadioButtonList Web** 控件一样的方法，参考 **DropDownList Web** 控件的 **SelectedItem** 属性即可。下列范例当我们选择 **DropDownList Web** 控件完毕后，按下按钮会将选取到的选项显示出来：

```
<Html>  
  
<Form Id="From1" Runat="Server">  
  
    <ASP:DropDownList Id="ddlA" Runat="Server"/>  
  
    <ASP:Button Id="btnA" Text="请按我" OnClick="btnA_Click"  
Runat="Server"/>  
  
</Form>  
  
<ASP:Label Id="lblA" Runat="Server"/>  
  
<Script Language="VB" Runat="Server">  
  
Sub Page_Load(Sender As Object, e As EventArgs)  
  
    Dim shtI As Short  
  
    Dim liA As ListItem
```

```
For shtI=1 To 12

    liA=New ListItem(shtI.ToString & "月")

    ddlA.Items.Add(liA)

Next

End Sub

Sub btnA_Click(Sender As Object, e As EventArgs)

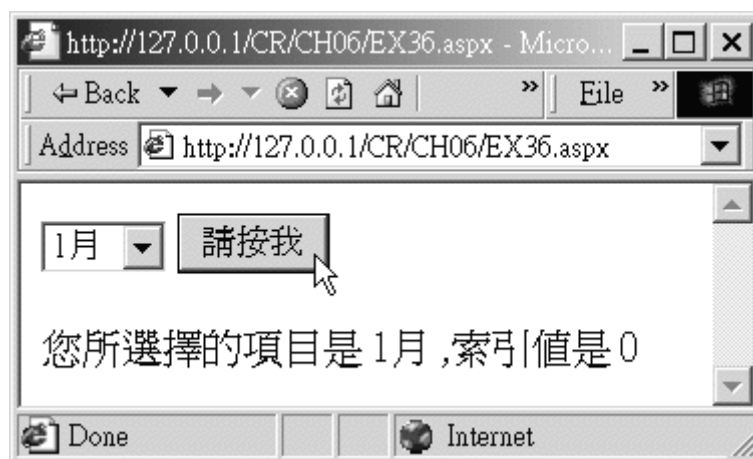
    lblA.Text="您所选择的项目是 " & ddlA.SelectedItem.Text & _

        " ,索引值是 " & ddlA.SelectedIndex.ToString

End Sub

</Script>

</Html>
```



DropDownList Web 控件所支持的事件是 **SelectedIndexChanged** 事件。若指定发生本事件要触发的程序，并将 **AutoPostBack** 属性设为 **True**，则当我们改变 DropDownList Web 控件里的选项时，便会触发这个事件。下列范例将前一个范例的 **Button Web** 控件拿掉，然后指定发生 **OnSelectedIndexChanged** 所要执行的事件程序，并将 **AutoPostBack** 设为 **True**：

```
<Html>

<Form Id="Form1" Runat="Server">

  <ASP:DropDownList Id="ddlA" AutoPostBack="True"

    OnSelectedIndexChanged="ddlA_Changed" Runat="Server"/>

</Form>

<ASP:Label Id="lblA" Runat="Server"/>

<Script Language="VB" Runat="Server">

Sub Page_Load(Sender As Object, e As EventArgs)

  Dim shtI As Short

  Dim liA As ListItem

  For shtI=1 To 12

    liA=New ListItem(shtI.ToString & "月")

    ddlA.Items.Add(liA)

  Next

End Sub
```

```
Sub ddlA_Changed(Sender As Object, e As EventArgs)

    lblA.Text="您所选择的项目是 " & ddlA.SelectedItem.Text & _

        " ,索引值是 " & ddlA.SelectedIndex.ToString

End Sub

</Script>

</Html>
```

选择正确的日期

这边我们来看一个比较实用的程序，它的作用是提供三个 **DropDownList Web** 控件让使用者选择日期。但是这里并不是只有提供三个 **DropDownList** 控件就了事，而是可以依据正确的年份以及月份提供正确的日期选择，以避免有如 2 月 31 日这种错误日期的发生。下列范例限制使用者只能选择包括今天以后五年内的日期：

```
<Html>

您认为两岸三通什么时候开放比较合适？

<Form Id="Form1" Runat="Server">

    <ASP:DropDownList Id="D1" AutoPostBack="True"

OnSelectedIndexChanged="DayChg"
```

```

        Runat="Server"/>年

        <ASP:DropDownList Id="D2" AutoPostBack="True"
OnSelectedIndexChanged="DayChg"

        Runat="Server"/>月

        <ASP:DropDownList Id="D3" AutoPostBack="True"
OnSelectedIndexChanged="DayChg"

        Runat="Server"/>日

    </Form>

    <ASP:Label Id="Label1" Runat="Server" />

    <Script Language="VB" Runat="Server">

Sub Page_Load(Sender As Object, e As EventArgs)

    If Page.IsPostBack=True Then Return ' 如果不是第一次加载就不执行

    Dim shtI As Short

    Dim liA As ListItem

    For shtI=Year(Now()) To Year(Now())+5 ' 加入未来五年内的时间

        liA=New ListItem(shtI)

        D1.Items.Add(shtI)

    Next

```

```

For shtI=1 To 12 '将月份填入

    liA=New ListItem(shtI)

    D2.Items.Add(shtI)

Next

D2.Items(Month(Now())-1).Selected=True '将选择到的项目显示为现在的
月份

For shtI=1 To Day(DateSerial(Year(Now()),Month(Now())+1,1-1)) '取得
现在的月份有几天

    liA=New ListItem(shtI)

    D3.Items.Add(shtI)

Next

D3.Items(Day(Now())-1).Selected=True '将选择到的项目显示为现在的日

End Sub

Sub DayChg(Sender As Object , e As EventArgs)

'下列判断如果使用者选择日期不是今天之后, 则将日及回指定到今天

If DateSerial(D1.SelectedItem.Text.ToInt16, _

    D2.SelectedItem.Text.ToInt16, _

    D2.SelectedItem.Text.ToInt16) < Now Then

```

```

D1.Items(0).Selected=True

D2.Items(Month(Now())-1).Selected=True

D3.Items(Day(Now())-1).Selected=True

Return

End If


Dim shtI As Short

Dim liA As ListItem

Dim shtD As Short

shtD=D3.SelectedItem.Text.ToInt16' 储存使用者选取的日期

D3.Items.Clear ' 移除所有的项目

For shtI=1 To Day(DateSerial(D1.SelectedItem.Text.ToInt16, _
                                D2.SelectedItem.Text.ToInt16+1, 1-1))

    liA=New ListItem(shtI)

    D3.Items.Add(shtI)

Next

' 下列判断如果使用者原先选的日子大于现在月份的日子,
' 就选择第一天; 否则将原日期填回去

If Day(DateSerial(D1.SelectedItem.Text.ToInt16, _
                    D2.SelectedItem.Text.ToInt16+1, 1-1)) < shtD Then

```



```
D3.Items(0).Selected=True

Else

    D3.Items(shtD-1).Selected=True

End If

Label1.Text="您所选择的日期为 " & D1.SelectedItem.Text & " 年 " & _

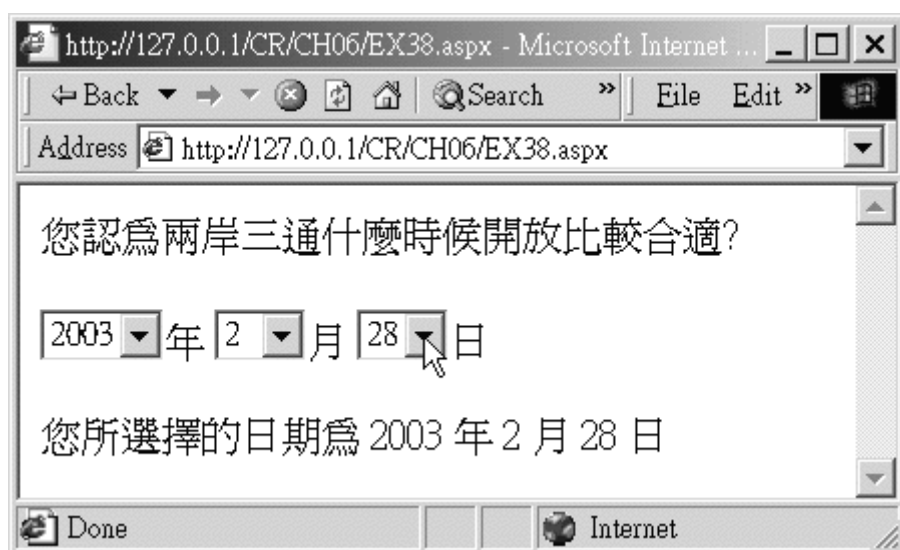
    D2.SelectedItem.Text & " 月 " & _

    D3.SelectedItem.Text & " 日"

End Sub

</Script>

</Html>
```



ListBox Web 控件

ListBox Web 控件和 DropDownList Web 控件的功能几乎是一样，只是 ListBox Web 控件是一次将所有的选项都显示出来。其使用语法如下：

```
<ASP:ListBox  
    Id="被程序代码所控制的名称"  
    Runat="Server"  
    AutoPostBack="True | False"  
    *DataSource="<%数据系统叙述%>"  
    *DataTextField="数据源的字段"  
    *DataValueField="数据源的字段"  
    Rows="一次要显示的列数"  
    SelectionMode="Single | Multiple"  
    OnSelectedIndexChanged="事件程序名称"  
>  
  
    <ASP:ListItem/>  
  
</ASP:ListBox>
```

*关于和数据源的数据系统部分，我们在后面的章节再介绍。

下表为 **ListBox Web** 控件的常用属性：

属性	说明
AutoPostBack	设定是否要致能 OnSelectedIndexChanged 事件。
*DataSource	设定数据系统所要使用的数据源。
*DataTextField	设定资料系统所要显示的字段。
*DataValueFiled	设定选项的相关数据要使用的字段。
Items	传回 ListBox Web 控件中 ListItem 的参考。
Rows	设定 ListBox Web 控件一次要显示的列数。
SelectedIndex	传回被选取到 ListItem 的 Index 值。
SelectedItem	传回被选取到 ListItem 参考，也就是 ListItem 本身。
SelectedItems	由于 ListBox Web 控件可以复选，被选取的项目会被加 入 ListItems 集合中； 本属性可以传回 ListItems 集合，只读。
SelectionMode	设定 ListBox Web 控件是否可以按住「Shift」或「Control」按钮进行复选， 默认值为「Single」。

下列范例是基本的 **ListBox** 控件用法，使用者只能单选 **ListBox Web** 控件中的项目：

```
<Html>
```

```
<Form Id="Form1" Runat="Server">
```

请选择您喜欢的月份(单选):


```
<ASP:ListBox Id="ListBox1" Runat="Server">
```

```
<ASP:ListItem>1 月</ASP:ListItem>
```

```
<ASP:ListItem>2 月</ASP:ListItem>
```

```
<ASP:ListItem>3 月</ASP:ListItem>
```

```
<ASP:ListItem>4 月</ASP:ListItem>
```

```
<ASP:ListItem>5 月</ASP:ListItem>
```

```
<ASP:ListItem>6 月</ASP:ListItem>
```

```
<ASP:ListItem>7 月</ASP:ListItem>
```

```
<ASP:ListItem>8 月</ASP:ListItem>
```

```
<ASP:ListItem>9 月</ASP:ListItem>
```

```
<ASP:ListItem>10 月</ASP:ListItem>
```

```
<ASP:ListItem>11 月</ASP:ListItem>
```

```
<ASP:ListItem>12 月</ASP:ListItem>
```

```
</ASP:ListBox>
```

```
</Form>
```

```
</Html>
```



ListBox Web 控件可以进行复选，只要将 **SelectionMode** 属性指定为 **Multiple**（复选）即可。下

列范例在使用者复选完毕按下确定按钮后，将使用者所选择的项目列出：

```
<Html>
```

```
<Form Id="Form1" Runat="Server">
```

请选择您喜欢的月份(复选):


```
<ASP:ListBox Id="ListBox1" SelectionMode="Multiple" Runat="Server">
```

```
<ASP:ListItem>1 月</ASP:ListItem>
```

```
<ASP:ListItem>2 月</ASP:ListItem>
```

```
<ASP:ListItem>3 月</ASP:ListItem>
```

```
<ASP:ListItem>4 月</ASP:ListItem>
```

```
<ASP:ListItem>5 月</ASP:ListItem>
```

```
<ASP:ListItem>6 月</ASP:ListItem>
```

```
<ASP:ListItem>7 月</ASP:ListItem>
```

```

    <ASP:ListItem>8 月</ASP:ListItem>

    <ASP:ListItem>9 月</ASP:ListItem>

    <ASP:ListItem>10 月</ASP:ListItem>

    <ASP:ListItem>11 月</ASP:ListItem>

    <ASP:ListItem>12 月</ASP:ListItem>

</ASP:ListBox>

<ASP:Button Id="btnA" Text="确定" OnClick="btnA_Click"
Runat="Server"/><br>

    <ASP:Label Id="Label1" Runat="Server"/>

</Form>

<Script Language="VB" Runat="Server">

Sub btnA_Click(Sender As Object, e As EventArgs)

    Dim shtI As Short

    Label1.Text="您喜欢的月份是: "

    For shtI=0 To ListBox1.Items.Count-1

        If ListBox1.Items(shtI).Selected=True Then

            Label1.Text & = ListBox1.Items(shtI).Text & " "

        End If

    Next

End Sub

```

</Script>

</Html>



ListBox Web 控件的事件和 DropDownList Web 控件一样, 只要将 **AutoPostBack** 属性设为 **True**,

再指定事件 **SelectedIndexChanged** 所要执行的事件程序即可。下列程序代码范例中, 只要使用

者选择不同的选项便触发 **SelectedIndexChanged** 事件:

请选择您喜欢的月份(单选):


```
<ASP:ListBox Id="ListBox1" AutoPostBack="True"
```

```
    OnSelectedIndexChanged ="ListBox1_Chg" Runat="Server">
```

```
<ASP:ListItem>1 月</ASP:ListItem>
```

```
<ASP:ListItem>2 月</ASP:ListItem>

<ASP:ListItem>3 月</ASP:ListItem>

<ASP:ListItem>4 月</ASP:ListItem>

<ASP:ListItem>5 月</ASP:ListItem>

<ASP:ListItem>6 月</ASP:ListItem>

<ASP:ListItem>7 月</ASP:ListItem>

<ASP:ListItem>8 月</ASP:ListItem>

<ASP:ListItem>9 月</ASP:ListItem>

<ASP:ListItem>10 月</ASP:ListItem>

<ASP:ListItem>11 月</ASP:ListItem>

<ASP:ListItem>12 月</ASP:ListItem>

</ASP:ListBox><br>

<ASP:Label Id="Label1" Runat="Server"/>

</Form>

<Script Language="VB" Runat="Server">

Sub ListBox1_Chg(Sender As Object, e As EventArgs)

    Label1.Text="您喜欢的月份是: " & ListBox1.SelectedItem.Text

End Sub

</Script>

</Html>
```


7. 进阶 Web 控件

数据验证 Web 控件

当我们我要求使用者输入数据的时候，一定要执行数据验证的工作。数据验证是一种限制使用者输入的限制（**Constraint**），可以确定使用者所输入的数据是正确的，或是强迫使用者一定要输入数据。先执行数据验证比输入错误的数据后，再让数据库响应一个错误讯息来的有效率；也可以确保使用者所输入的数据是一个有效值，而不会造成垃圾数据。数据验证 **Web** 控件可以帮助我们少写许多程序来验证使用者输入的数据，下表列出 **ASP.NET** 所提供的数据验证 **Web** 控件：

控件名称	说明
RequiredFieldValidator	验证使用者是否有输入数据。
CompareValidator	验证使用者输入的数据和某个值用比较运算符比较是否成立。
CustomValidator	自订的验证方式。
RangeValidator	验证使用者输入的数据是否在指定范围内。
RegularExpressionValidator	以特定规则验证使用者输入的数据。

RequireFieldValidator Web 控件

RequireFieldValidator Web 控件可以用来强迫使用必需输入数据，其使用语法为：

```
<ASP:RequireFieldValidator  
  
Id="被程序代码所控制的名称"  
  
Runat="Server"  
  
ControlToValidate="要验证的控件名称"  
  
ErrorMessage="所要显示的错误信息"  
  
Text="未输入数据时所显示的讯息"  
  
>
```

其常用属性说明如下表所示：

属性	说明
ControlToValidate	所要验证的控件名称。
ErrorMessage	所要显示的错误信息。
Text	未通过验证时所显示的讯息。

ControlToValidate 属性用来指明要检验的控件，而 **ErrorMessage** 属性用来提供给其它控件显示相关讯息，**Text** 属性在使用者的输入没有通过验证时立即显示。下列程序代码限制姓名字段一定要输入，否则无法触发按钮的事件程序：

```
<Html>
```

```
<Form Id="Form1" Runat="Server">

    姓名: <ASP:TextBox Id="txtName" Runat="Server"/>

    <ASP:RequiredFieldValidator Id="Validor1" Runat="Server"

        ControlToValidate="txtName"

        Text="必填项目"/><br>

    电话: <ASP:TextBox Id="txtTel" Runat="Server"/><br>

    住址: <ASP:TextBox Id="txtAdd" Runat="Server"/><br>

    <ASP:Button Id="btnOK" Text="确定" OnClick="btnOK_Click"

Runat="Server"/>

    <ASP:Label Id="lblMsg" Runat="Server"/>

</Form>


<Script Language="VB" Runat="Server">

Sub btnOK_Click(Sender As Object,e As EventArgs)

If Page.IsValid Then

    lblMsg.Text="验证成功!"

End If

End Sub


</Script>

</Html>
```

倘若使用没有输入姓名字段而按下确定按钮，不但不会触发任何事件程序外，还会显示提示讯息，

如下图所示：



倘若使用有输入姓名字段而按下确定按钮，则正常触发事件程序，如下图所示：



Page.IsValid 属性

要判断使用者输入的资料是否通过验证时，可以检查 **Page** 对象的 **IsValid** 属性。如果 **IsValid** 属性为 **True**，则表示所有的控件都通过验证；反之则代表有控件没有通过。

CompareValidator Web 控件

CompareValidator Web 控件可以验证使用者输入的数据，和某个值利用比较运算。其使用语法为：

```
<ASP:CompareValidator
```

```
Id="被程序代码所控制的名称"
```

```
Runat="Server"
```

```
ControlToValidate="要验证的控件名称"

Operator="DataTypeCheck | Equal | NotEqual | GreaterThan |
GreaterThanEqual | LessThan | LessThanEqual"

Type="资料型别"

ControlToCompare="要比较的控件名称" | ValueToCompare="要比较的值"

ErrorMessage="所要显示的错误信息"

Text="未通过验证时所显示的讯息"

/>
```

其常用属性说明如下表所示：

属性	说明
ControlToValidate	所要验证的控件名称
ErrorMessage	所要显示的错误信息。
Operator	所要执行的比较种类，有：DataTypeCheck（只比较数据型态）、Equal（等于）、NotEqual（不等于）、GreaterThan（大于）、GreaterThanEqual（大于等于）、LessThan（小于）、LessThenEqual（小于等于）。其中如果为DataTypeCheck 时，只需要填入要验证的数据型态，不需要设定ControlToCompare 或是 ValueToCompare。
Type	所要比较或验证的数据型别，可以设定为：Currency、Date、Double、Integer、

	String
ControlToCompare	要比较的控件名称。
ValueToCompare	要比较的值。
Text	未通过验证时所显示的讯息。

下列范例限制使用者所输入的年龄必需大于 **18** 岁：

```
<Html>

<Form Id="Form1" Runat="Server">

    姓名： <ASP:TextBox Id="txtName" Runat="Server"/><br>

    年龄： <ASP:TextBox Id="txtAge" Runat="Server"/>

    <ASP:CompareValidator Id="Validor1" Runat="Server"

        ControlToValidate="txtAge"

        ValueToCompare="18"

        Operator="GreaterThanOrEqual"

        Type="Integer"

        Text="您必须大于十八岁才可以浏览本站"/><br>

    住址： <ASP:TextBox Id="txtAdd" Runat="Server"/><br>

    <ASP:Button Id="btnOK" Text="确定" OnClick="btnOK_Click"

Runat="Server"/>
```

```
<ASP:Label Id="lblMsg" Runat="Server"/>
```

```
</Form>
```

```
<Script Language="VB" Runat="Server">
```

```
Sub btnOK_Click(Sender As Object, e As EventArgs)
```

```
    If Page.IsValid Then
```

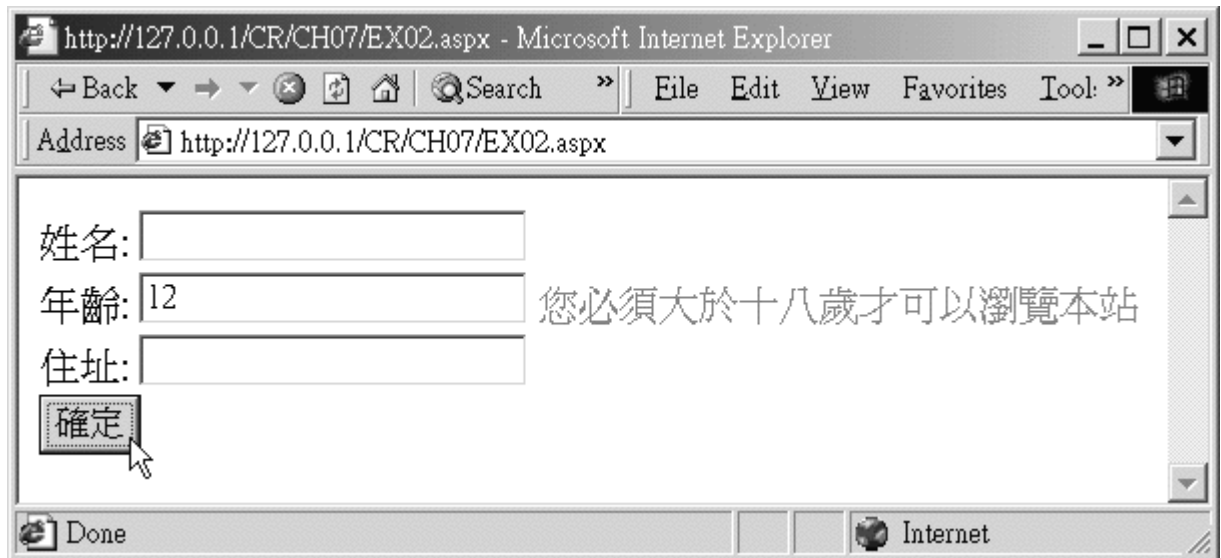
```
        lblMsg.Text="验证成功!"
```

```
    End If
```

```
End Sub
```

```
</Script>
```

```
</Html>
```



下列范例码限制使用者输入必须是整数型态的数据:

```
<Html>

<Form Id="Form1" Runat="Server">

    姓名: <ASP:TextBox Id="txtName" Runat="Server"/><br>

    年齡: <ASP:TextBox Id="txtAge" Runat="Server"/>

    <ASP:CompareValidator Id="Validator1" Runat="Server"

        ControlToValidate="txtAge"

        Operator="DataTypeCheck"

        Type="Integer"

        Text="您必須輸入數值"/><br>

    住址: <ASP:TextBox Id="txtAdd" Runat="Server"/><br>
```

```
<ASP:Button Id="btnOK" Text="确定" OnClick="btnOK_Click"
```

```
Runat="Server"/>
```

```
<ASP:Label Id="lblMsg" Runat="Server"/>
```

```
</Form>
```

```
<Script Language="VB" Runat="Server">
```

```
Sub btnOK_Click(Sender As Object, e As EventArgs)
```

```
    If Page.IsValid Then
```

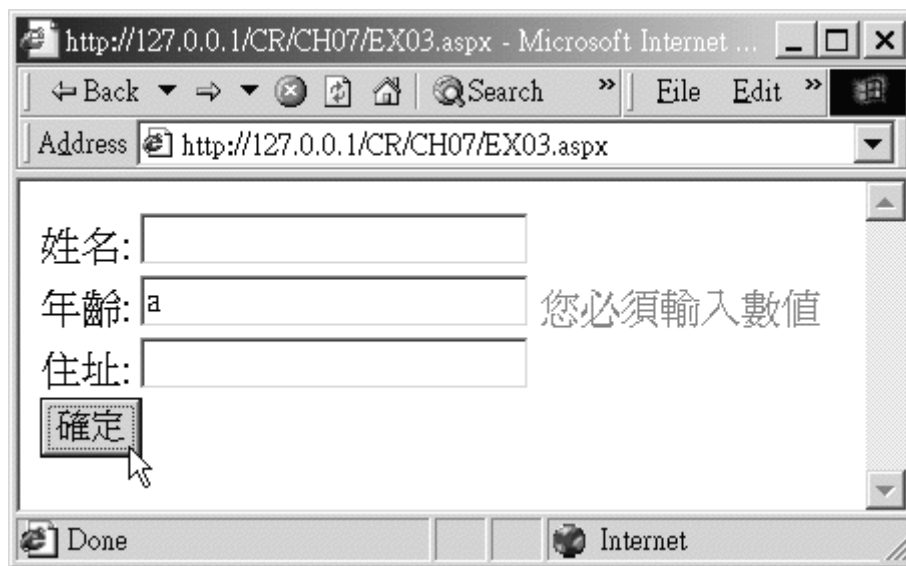
```
        lblMsg.Text="验证成功!"
```

```
    End If
```

```
End Sub
```

```
</Script>
```

```
</Html>
```



上述程序并没有限制使用者一定要输入年龄的数据，若要限制只用户一定要填入数据，可以搭配

RequireFieldValidator 来作验证；如下范例所示：

```
<Html>

<Form Id="Form1" Runat="Server">

    姓名: <ASP:TextBox Id="txtName" Runat="Server"/><br>

    年龄: <ASP:TextBox Id="txtAge" Runat="Server"/>

    <ASP:CompareValidator Id="Validator1" Runat="Server"

        ControlToValidate="txtAge"

        Operator="DataTypeCheck"

        Type="Integer"

        Text="您必须输入数值"/>

    <ASP:RequiredFieldValidator Id="Validator2" Runat="Server"

        ControlToValidate="txtAge"
```

```
Text="必填项目"/><br>

住址: <ASP:TextBox Id="txtAdd" Runat="Server"/><br>

<ASP:Button Id="btnOK" Text="确定" OnClick="btnOK_Click"

Runat="Server"/>

<ASP:Label Id="lblMsg" Runat="Server"/>

</Form>

<Script Language="VB" Runat="Server">

Sub btnOK_Click(Sender As Object,e As EventArgs)

If Page.IsValid Then

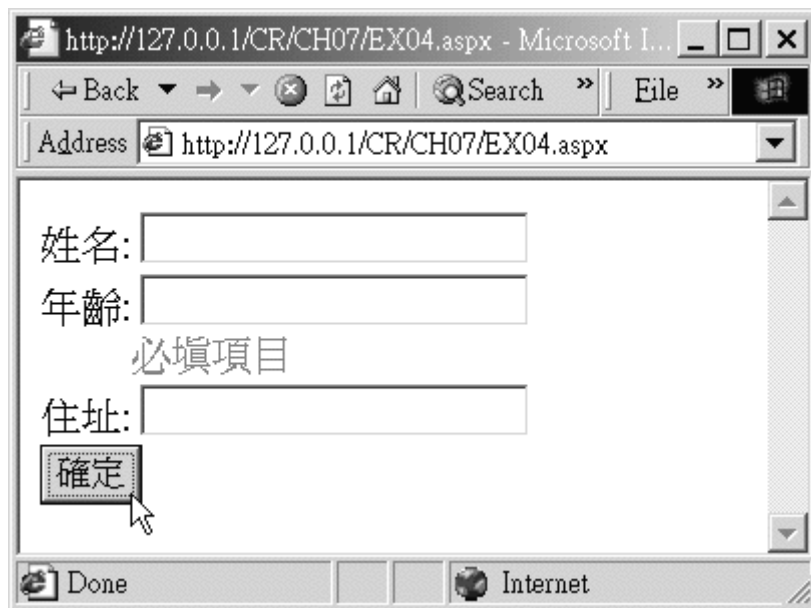
    lblMsg.Text="验证成功!"

End If

End Sub

</Script>

</Html>
```



RangeValidator Web 控件

RangeValidator Web 控件可以限制使用者所输入的数据在指定的范围之内，其使用语法为：

<ASP:RangeValidator

Id="被程序代码所控制的名称"

Runat="Server"

ControlToValidate="要验证的控件名称"

MinimumValue="最小值"

MaximumValue="最大值"

MinimumControl="限制最小值的控件名称"

MaximumControl="限制最大值的控件名称"

Type="资料型别"

ErrorMessage="所要显示的错误信息"

Text="未通过验证时所显示的讯息"

/>

其常用属性说明如下表所示：

属性	说明
ControlToValidate	所要验证的控件名称。
ErrorMessage	所要显示的错误信息。
MinimumValue	限制可以接受的最小值。
MaximumValue	限制可以接受的最大值。
MinimumControl	限制可以接受最小值所要参考的控件。
MaximumControl	限制可以接受最大值所要参考的控件。
Type	所要比较或验证的数据型别,可以设定为:Currency、Date、Double、Integer、String
Text	未通过验证时所显示的讯息。

<Html>

<Form Id="Form1" Runat="Server">

姓名: <ASP:TextBox Id="txtName" Runat="Server"/>

年龄: <ASP:TextBox Id="txtAge" Runat="Server"/>


```
<ASP:RangeValidator Id="Validor1" Runat="Server"

    ControlToValidate="txtAge"

    MaximumValue="40"

    MinimumValue="18"

    Type="Integer"

    Text="只接受 18-40"/><br>

住址: <ASP:TextBox Id="txtAdd" Runat="Server"/><br>

<ASP:Button Id="btnOK" Text="确定" OnClick="btnOK_Click"

Runat="Server"/>

<ASP:Label Id="lblMsg" Runat="Server"/>

</Form>

<Script Language="VB" Runat="Server">

Sub btnOK_Click(Sender As Object,e As EventArgs)

    If Page.IsValid Then

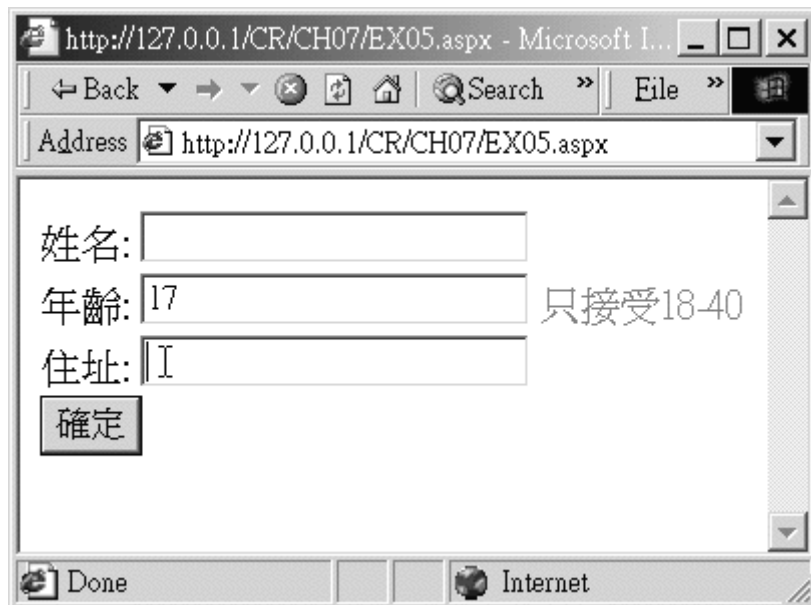
        lblMsg.Text="验证成功!"

    End If

End Sub

</Script>

</Html>
```



RegularExpressionValidator Web 控件

RegularExpressionValidator Web 控件可以用来执行更详细的验证，也就是说可以做更细微的限制。其使用语法为：

```
<ASP:RegularExpressionValidator
```

```
Id="被程序代码所控制的名称"
```

```
Runat="Server"
```

```
ControlToValidate="要验证的控件名称"
```

```
ValidationExpression="验证规则"
```

ErrorMessage="所要显示的错误信息"

Text="未通过验证时所显示的讯息"

/>

其常用属性说明如下表所示：

属性	说明
ControlToValidate	所要验证的控件名称。
ErrorMessage	所要显示的错误信息。
ValidationExpression	验证规则。
Textq	未通过验证时所显示的讯息。

ValidationExpression 验证规则

其中 ValidationExpression 验证规则属性为限制数据所输入的叙述，其常用符号如下表所示：

符号	说明
[]	用来定义单一字符的内容。
{}	用来定义需输入的字符个数。
.	表示任意字符。

*	表示最少可以不输入，最多到无限多个字符。
+	表示最少输入 1 个字符，最多到无限多个字符。
[^...]	表示不包含的字符。

[] 符号

[] 符号可以用来定义接受的单一字符，例如：

[a-zA-Z] 只接受 a-z 或是 A-Z 的英文字符。

[x-zX-Z] 只接收小写的 x-z 或大写的 X-Z。

[win] 只接收 w、i、n 的英文字母。

[^linux] 除了 l、i、n、u、x 之外的英文字母都接收。

{ } 符号

{ } 符号可以用来表示接收多少字符，例如：

[a-zA-Z]{4} 表示接受只接收四个字符。

[a-z]{4} 表示只接收共四个 a-z 小写字符。

`[a-zA-Z]{4,6}` 表示最少接受四个字符，最多接受六个字符。

`[a-zA-Z]{4,}` 表示最少接受四个字符，最多不限制。

.符号

`[.]` 符号可以用来表示接收除了空白外的任意字符，例如：

`{4}` 表示接收四个除了空白外的任意字符。

*符号

`[*]` 符号表示最少 0 个符合，最多到无限多个字符。例如：

`[a-zA-Z]*` 表示不限制数目，接受 `a-z` 或 `A-Z` 的字符，也可以不输入。

+符号

`[+]` 符号表示最少 1 个符合，最多到无限多个字符。例如：

`[a-zA-Z]+` 表示不限制数目，接受 `a-z` 或 `A-Z` 的字符，但是至少输入一个字符。

下列范例限制使用者输入的账号，必需要以英文字母为开头，而且最少要输入四个字符，最多可输入八个字符：

```
<Html>

<Form Id="Form1" Runat="Server">

    账号: <ASP:TextBox Id="txtId" Runat="Server"/>

    <ASP:RegularExpressionValidator Id="Validator1" Runat="Server"

        ControlToValidate="txtId"

        ValidationExpression="[a-zA-Z]{4,8}"

        Text="错误!"/><br>

    <ASP:Button Id="btnOK" Text="确定" OnClick="btnOK_Click"

Runat="Server"/>

    <ASP:Label Id="lblMsg" Runat="Server"/>

</Form>

<Script Language="VB" Runat="Server">

Sub btnOK_Click(Sender As Object, e As EventArgs)

    If Page.IsValid Then

        lblMsg.Text="验证成功!"

    End If

End Sub
```

End Sub

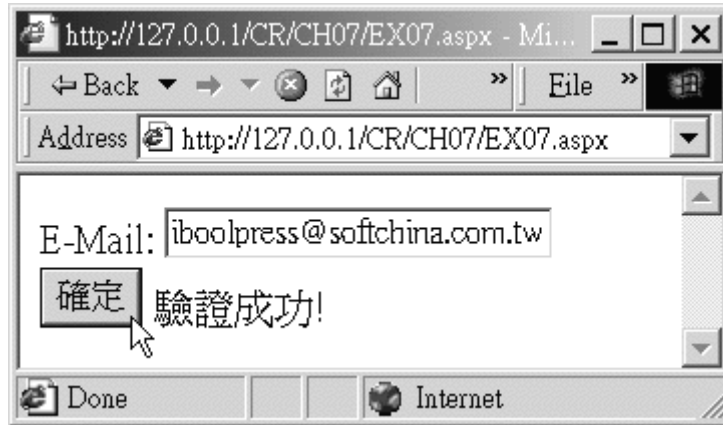
</Script>

</Html>



下列程序代码片段限制使用者输入的电子邮件信箱，必须是包含「@」：

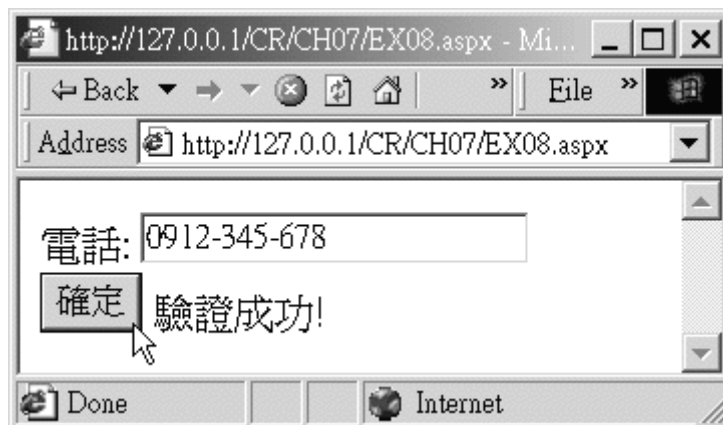
```
<ASP:RegularExpressionValidator Id="Validor1" Runat="Server"
    ControlToValidate="txtEmail"
    ValidationExpression=".+@.+"
    Text="错误!"/>
```



下列程序代码片段限制使用者输入的电话号码，必须要依使用习惯输入分隔线：

```
<ASP:RegularExpressionValidator Id="Validor1" Runat="Server"
    ControlToValidate="txtTel"
    ValidationExpression="[0-9]{2,4}-[0-9]{3,4}-[0-9]{3,4}"
    Text="错误!"/>
```

使用者输入 0800-006-089 或 0912-345-678 或 02-2311-8765 都可以接受。



|符号

如果我们想要限制使用者的输入，只要符合两个规则的其中一个即接受，可以使用「|」符号。

「|」符号表示或的意思，例如使用者只要输入全部四个数值或是全部四个字母都接受的话，验证规则可以写成[\[a-zA-Z\]{4} | \[0-9\]{4}](#)；表示四个所输入的数据如果不是全部都数值或是全部都英文字符则不正确。如果把验证规则设定成[\[a-zA-Z0-9\]{4}](#)的结果和不一样，这样表示任何四个混杂四个大小写字符或数值的数据都接受。

\符号

由于「()」、「[]」、「{}」或是「|」这些符号在验证叙述中有特定的意义，所以如果所要执行验证的字符如果包含这些符号，必需在这些符号前面加上符号「\」。例如验证叙述为

[「\\(\[0-9\]{2,3}\\)」](#)，表示所输入的数据必需为「(02)」或是「(035)」才符合验证规则。

ValidationSummary Web 控件

ValidationSummary Web 控件可以用来显示尚未通过验证的字段，其使用语法为：

```
<ASP:ValidatorSummary
```

```
Id="被程序代码所控制的名称"
```

```
Runat="Server"

DisplayMode=" BulletList | List | SingParagraph"

HeaderText="控件标题文字"

/>
```

其常用属性说明如下表所示：

属性	说明
DisplayMode	显示讯息的模式。可以设定为： BulletList ：以项目的方式显示。 List ：显示在不同列。 SingleParagraph ：显示在同一列。
HeaderText	控件的标题文字。

ValidatorSummary Web 控件最主要的功能是用来显示没有通过验证 **Web** 控件的 **ErrorMessage**

属性，所以要使用 **ValidatorSummary Web** 控件之前，必须先设定其它验证 **Web** 控件的

ErrorMessage。下列范例可以显示使用者尚未通过验证的数据域位：

```
<Html>

<Form Id="Form1" Runat="Server">

    姓名：<ASP:TextBox Id="txtName" Runat="Server"/>

    <ASP:RegularExpressionValidator Id="Validor1" Runat="Server"
```

ControlToValidate="txtName"

ErrorMessage="姓名字段"

ValidationExpression=". {6, 8}"

Text="错误!"/>

<ASP:RequiredFieldValidator Id="Validator2" Runat="Server"

ControlToValidate="txtName"

Text="必填项目"/>

电话: <ASP:TextBox Id="txtTel" Runat="Server"/>

<ASP:RegularExpressionValidator Id="Validator3" Runat="Server"

ControlToValidate="txtTel"

ErrorMessage="电话字段"

ValidationExpression="[0-9]{2, 4}-[0-9]{3, 4}-[0-9]{3, 4}"

Text="错误!"/>

<ASP:RequiredFieldValidator Id="Validator4" Runat="Server"

ControlToValidate="txtTel"

Text="必填项目"/>

<ASP:Button Id="btnOK" Text="确定" OnClick="btnOK_Click"

Runat="Server"/>

<ASP:Label Id="lblMsg" Runat="Server"/><p>

<ASP:ValidationSummary Id="vsA" Runat="Server"

```
        HeaderText="您尚未通过的字段: "

        DisplayMode="Bulletlist"/>

</Form>

<Script Language="VB" Runat="Server">

Sub btnOK_Click(Sender As Object,e As EventArgs)

    If Page.IsValid Then

        lblMsg.Text="验证成功!"

    Else

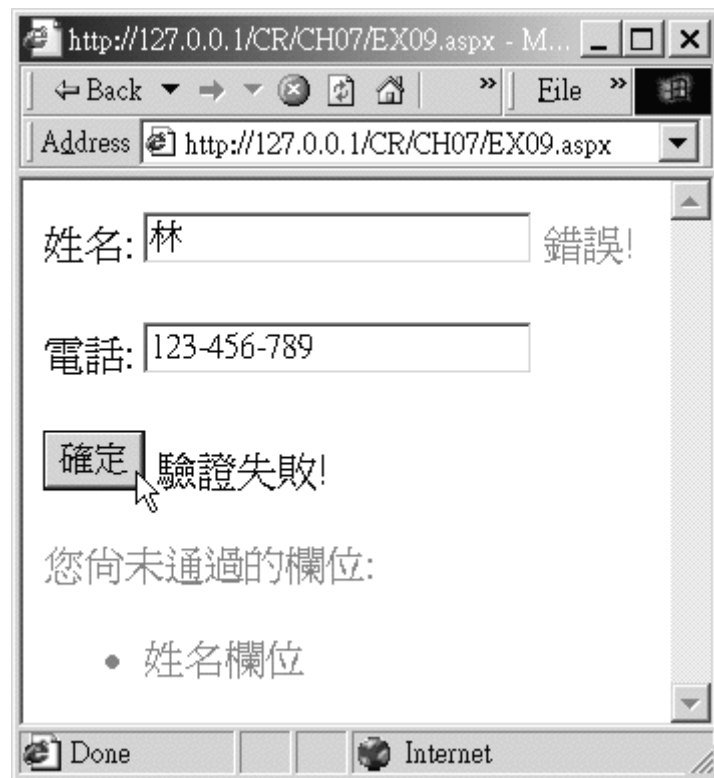
        lblMsg.Text="验证失败!"

    End If

End Sub

</Script>

</Html>
```



CustomValidator Web 控件

如果我们所要处理的数据有上列数据验证 **Web** 控件无法执行的特殊表达式，可以利用

CustomValidator Web 控件可以让我们自定数据的检验方式。其使用语法为：

```
<ASP:CustomValidator
```

```
Id="被程序代码所控制的名称"
```

```
Runat="Server"
```

```
ControlToValidate="要验证的控件名称"
```

```
OnServerValidate="自订的验证程序"
```

```
ErrorMessage="所要显示的错误信息"
```

```
Text="未通过验证时所显示的讯息"
```

```
/>
```

其常用属性说明如下表所示：

属性	说明
ControlToValidate	所要验证的控件名称。
OnServerValidate	执行验证的程序名称。
ErrorMessage	所要显示的错误信息。
Text	未通过验证时所显示的讯息。

CustomValidator Web 控件在执行自订的验证时，是呼叫 **OnServerValidate** 属性所指定的程序

来执行验证；当被呼叫的程序传回 **True** 时则表示验证成功，传回 **False** 则表示验证失败。其方

法的宣告语法如下所示：

```
Function 程序名称(Sender As Object, Value As String)As Boolean  
    ' 验证成功  
    Return True  
    ' 验证失败  
    Return False  
End Function
```

由于这个事件程序要明确的宣告传回值的数据型态,所以这里我们要在这个事件程序以 **Function** 来宣告,并且要指明传回值的型态为 **Boolean**。另外程序内所接收的第二个参数 **Value** 表示使用者输入的资料,我们可以将这个参数作验证后,再传回代表验证成功的 **True** 或验证失败的 **False**。

下列范例限制使用者必须输入十的倍数才可以通过验证:

```
<Html>

<Form Id="Form1" Runat="Server">

    请输入 10 的倍数: <ASP:TextBox Id="txtValue" Runat="Server"/>

    <ASP:Button Id="btnOK" Text="确定" OnClick="btnOK_Click"
Runat="Server"/><br>

    <ASP:CustomValidator Id="Validor1" Runat="Server"

        ControlToValidate="txtValue"

        OnServerValidate="CustVali"

        Text="必需输入十的倍数!"/><br>

<ASP:Label Id="lblMsg" Runat="Server"/>

</Form>

<Script Language="VB" Runat="Server">

Function CustVali(Sender As Object, Value As String)As Boolean

    If Value.ToInt16 Mod 10=0 Then
```

```
        Return True

    Else

        Return False

    End If

End Function

Sub btnOK_Click(Sender As Object, e As EventArgs)

    If Page.IsValid Then

        lblMsg.Text="验证成功!"

    Else

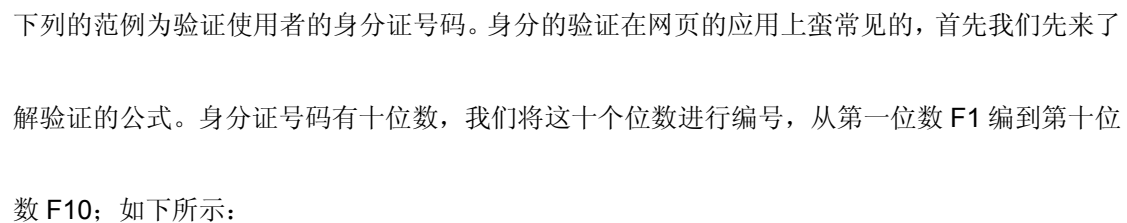
        lblMsg.Text="验证失败!"

    End If

End Sub

</Script>

</Html>
```

其中英文字母有特定的编号，如下表所示：

[illegible]

身分证验证的公式为：

总和=(F10 的个位数+(F10 的十位数
 $\times 9$)+(F9 $\times 8$)+(F8 $\times 7$)+(F7 $\times 6$)+(F6 $\times 5$)+(F5 $\times 4$)+(F4 $\times 3$)+(F3 $\times 2$)+(F2
+F1))

如果总和除以 10 可以被整除，那么表示使用者所输入的身分证号码是正确的。其验证的程序代码如下所示：

```
<Html>

<Form Id="Form1" Runat="Server">

    请输入身分证号码: <ASP:TextBox Id="txtValue" Runat="Server"/>

    <ASP:Button Id="btnOK" Text="确定" OnClick="btnOK_Click"
Runat="Server"/><br>

    <ASP:CustomValidator Id="Validor1" Runat="Server"

        ControlToValidate="txtValue"

        OnServerValidate="CustVali"/><br>

<ASP:Label Id="lblMsg" Runat="Server"/>

</Form>

<Script Language="VB" Runat="Server">
```

```

Function CustVali(Sender As Object, Value As String)As Boolean

    If Len(Value)<>10 Then      ' 如果位数小于 10

        Validor1.Text="请输入十位数!"

        Return False

    Else  If Char.IsLetter(Left(Value,1))=False Then  ' 检查第一个字符是
不是英文字母

        Validor1.Text="必须以英文字母开头!"

        Return False

    End if

    Dim shtI, shtJ, shtK As Short

    For shtI=2 To 10      ' 检查第二位至第十位是否为数值

        If IsNumeric(Mid(Value, shtI, 1))=False Then

            Validor1.Text="第二位至第九位必须是数值!"

            Return False

        End If

    Next

    ' 以下依序宣告 26 个英文字母所代表的编号

    Dim arF1() As Short = {10, 11, 12, 13, 14, 15, 16, 17, 34, 18, 19, 20, 21, _
        22, 35, 23, 24, 25, 26, 27, 28, 29, 30, 41, 42, 33}

```

```

    shtI=arF1(Asc(UCase(Left(Value,1)))-65)      '取得使用者输入英文字母
的编号

    For shtJ=2 To 8      '计算加总

        shtK+=Mid(Value, shtJ, 1).ToInt16 * (10-shtJ)

    Next

    shtK+=Mid(Value, 9, 1).ToInt16 + Mid(Value, 10, 1).ToInt16 + _

        Left(shtI.ToString, 1).ToInt16 + (Left(shtI.ToString,
2).ToInt16 * 9)

    If shtK Mod 10 <> 0 Then      '如果算式加总不能被 10 整除

        Validator1.Text="您所输入的身分证号码不正确!"

        Return False

    Else

        Return True      '验证成功, 传回 True

    End If

End Function

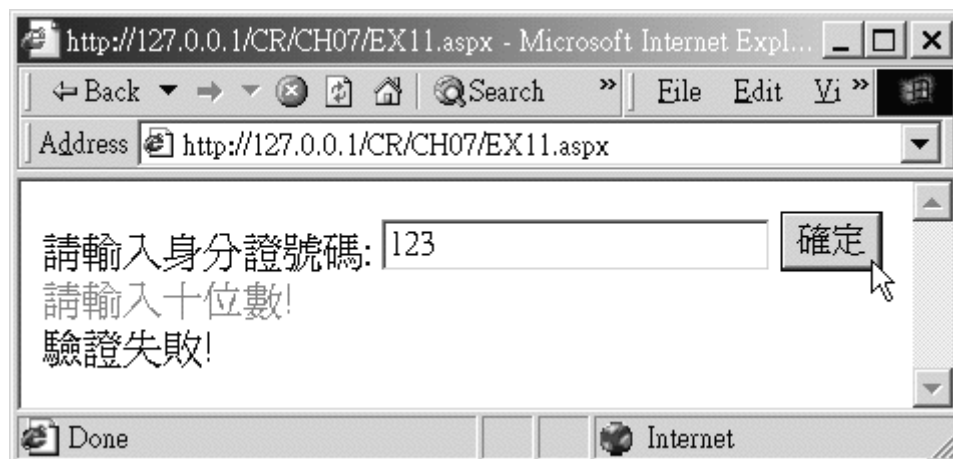
Sub btnOK_Click(Sender As Object,e As EventArgs)

    If Page.IsValid Then

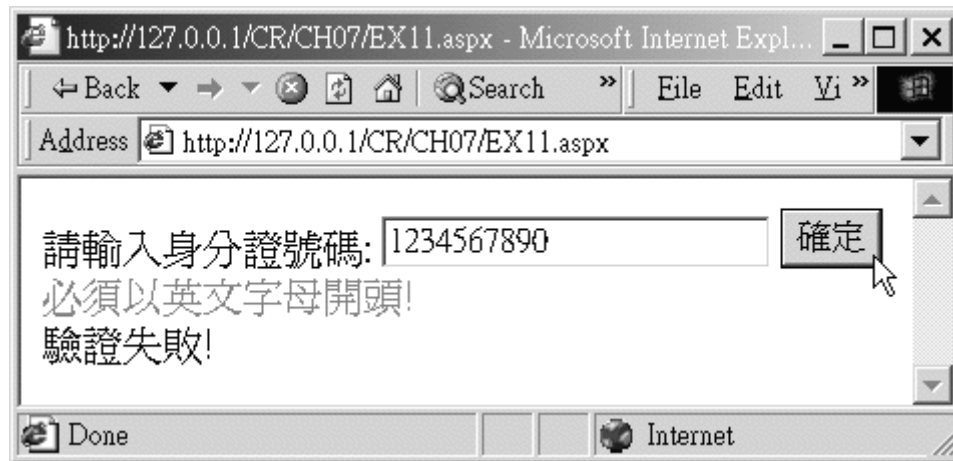
        lblMsg.Text="验证成功!"

```

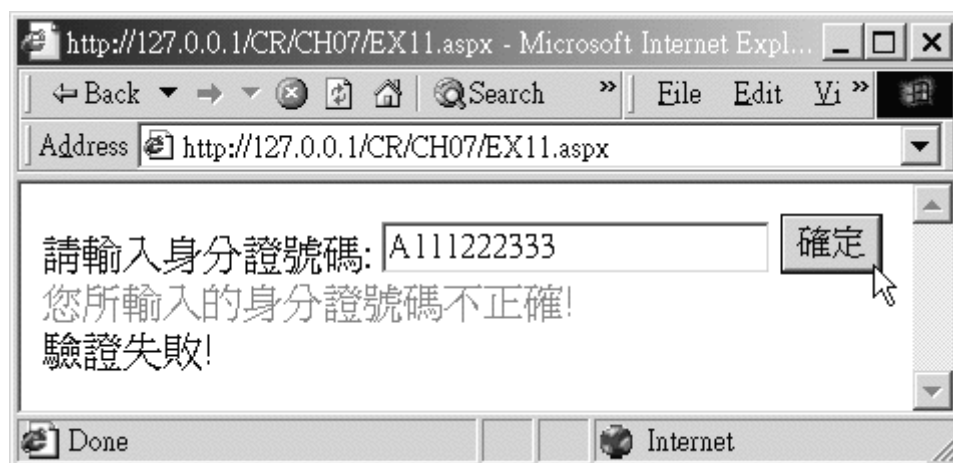
```
Else  
  
    lblMsg.Text="验证失败!"  
  
End If  
  
End Sub  
  
</Script>  
  
</Html>
```



如果只輸入三位数。



如果只輸入十位数值。



如果随便输入身份证号码。



如果输入 A123456789 正确的身份证号码。

AdRotator Web 控件

AdRotator Web 控件称为广告旋转版，我们常常在许多网页上用它来显示一些广告的内容，因为它可以用来控制一些图片要出现在网页的机率，以及点选后所重新导向的网址；所以每次使用者浏览网页时所会出现的广告都不尽相同。其使用语法为：

```
<ASP:AdRotator
```

```
Id="被程序代码所控制的名称"
```

```
Runat="SERVER"
```

```
AdvertisementFile="设定档名称"
```

```
KeywordFilter="要显示的分类广告"
```

```
Target="_TOP | _NEW | _CHILD | _SELF | _PARENT | _BLANK"
```

```
OnAdCreated="事件程序"
```

```
/>
```

其常用属性说明如下表所示：

属性	说明
AdvertisementFile	广告旋转版的设定文件名称。
KeywordFilter	要显示的分类广告。
Target	广告看板所要开启的窗口。
OnAdCreated	每次要产生新的看板内容时便触发此程序。

如果我们有指定 **OnAdCreated** 属性时，**AdRotator Web** 控件会在产生广告看板时触发我们所指定的事件，另外还会传送一些信息；所以我们宣告 **OnAdCreated** 事件程序的语法如下所示：

```
Sub 事件名称(Sender As Object, e As AdCreatedEventArgs)

    ...

End Sub
```

其中所传递的参数 **e** 有一些我们可以利用的属性，分别如下表所示：

属性	说明
AdProperties	可以取得目前广告旋转版的属性。
AlternateText	广告看板的提示文字。

ImageUrl	广告看板的使用图片的地址。
NavigateUrl	使用者选取时所要开启的连结。

产生 **AdRotator Web** 控件的设定文件

要使用 **AdRotator Web** 控件之前要先设定所要显示的广告看板的属性，其设定要以 XML 的格式来撰写。其使用语法如下所示：

```
<Advertisements>

  <Ad>

    <ImageUrl>要显示的图形文件地址</ImageUrl>

    <NavigateUrl>使用者选取时所要开启的连结</NavigateUrl>

    <AlternateText>提示文字</AlternateText>

    <Keyword>广告分类</Keyword>

    <Impressions>权值</Impressions>

  </Ad>

  其它广告设定...

</Advertisements>
```

其中权值表示广告看板所出现的机率。例如某个 **AdRotator Web** 控件所管理的广告有五则，假设每一则的权值都为 1，那么每一则广告出现的机率则为：

$$1 \div (1+1+1+1+1) \times 100\% = 20\%$$

所以每一则广告在被浏览的次数接近无限多次的时候，所会出现的机率则趋近于 20%。接下来

假设我们有下列五则广告：

	<TargetUrl> (圖檔名稱) : Banner1.gif <TargetUrl> (目標 URL) : B1.htm <AlternateText> (提示文字) : Small Business Server <Keyword> (分類) : Produce <Impressions> (權值) : 1
	<TargetUrl> (圖檔名稱) : Banner2.gif <TargetUrl> (目標 URL) : B2.htm <AlternateText> (提示文字) : Visual StudioNET <Keyword> (分類) : Produce <Impressions> (權值) : 1
	<TargetUrl> (圖檔名稱) : Banner3.gif <TargetUrl> (目標 URL) : B3.htm <AlternateText> (提示文字) : Windows XP <Keyword> (分類) : Produce <Impressions> (權值) : 1
	<TargetUrl> (圖檔名稱) : Banner4.gif <TargetUrl> (目標 URL) : B4.htm <AlternateText> (提示文字) : Driver And Hardware <Keyword> (分類) : Support <Impressions> (權值) : 1
	<TargetUrl> (圖檔名稱) : Banner5.gif <TargetUrl> (目標 URL) : B5.htm <AlternateText> (提示文字) : msdn online Library <Keyword> (分類) : Support <Impressions> (權值) : 1

依照我们的设定，产生下列广告旋转版的设定档：

```
<Advertisements>
```

```
<Ad>
```

```
<ImageUrl>Banner1.gif</ImageUrl>
```

```

    <NavigateUrl>B1.htm</NavigateUrl>

    <AlternateText>Small Business Server</AlternateText>

    <Keyword>Product</Keyword>

    <Impressions>1</Impressions>

</Ad>

<Ad>

    <ImageUrl>Banner2.gif</ImageUrl>

    <NavigateUrl>B2.htm</NavigateUrl>

    <AlternateText>Visual Studio.NET</AlternateText>

    <Keyword>Product</Keyword>

    <Impressions>1</Impressions>

</Ad>

...

</Advertisements>

```

其它的广告设定依此类推，我们将广告旋转版设定好后存放在 **Ads.xml** 文件里，接下来就可以使

用 **AdRotator Web** 控件来产生广告看板了。下列范例产生了一个广告看板，限制只有「**Product**」

类别的广告会被显示，并于广告看板被产生时将其属性显示出来：

```

<Html>

<ASP:AdRotator Id="arA" Runat="Server"

```

```
        AdvertisementFile="Ads.xml"

        BorderWidth="1"

        KeywordFilter="Product"

        OnAdCreated="arA_Create"/><p>

<ASP:Label Id="Label1" Runat="Server"/>

<Script Language="VB" Runat="Server">

Sub arA_Create(Sender As Object, e As AdCreatedEventArgs)

    Label1.Text = "广告看板的图形地址: " & e.ImageUrl & "<br>"

    Label1.Text + = "广告看板的目标连结: " & e.NavigateUrl & "<br>"

    Label1.Text + = "广告看板的提示文字: " & e.AlternateText & "<br>"

End Sub

</Script>

</Html>
```



按下后即开启新的连结:



样式对象（Style Object）

ASP.NET 支持样式对象（Style Object），样式对象可以让使用者设定一些如颜色与字型的外观

显示，让某些控件的外观显示更多样化。其使用语法为：

```
<ASP:控件类别
```

```
Id="被程序代码所控制的名称"
```

```
Runat="SERVER"
```

```
    样式对象-属性="设定值"
```

```
/>
```

下列范例简单的设定了日历 **Web** 控件的 **SelectedDayStyle** 对象, 这个对象决定了在日历控件上面的日期被点选时, 所会显示的样式:

```
<Html>

<Form runat="Server">

<asp:Calendar id="Calendar1" runat="server"

    SelectedDayStyle-BackColor="#DBDBDB"

    SelectedDayStyle-ForeColor="Red"

    SelectedDayStyle-Font-Bold="True"

    SelectedDayStyle-Font-Name="Arial"/>

</Form>

</Html>
```



样式对象总共分为三类，分别为基础样式、TableItem 样式以及 DataGridPager 样式；这些样式使用在不同的地方。

基础样式

TableItem 样式对象以及 DataGridPager 样式对象都支持下列基础样式，这些基础样式如下表所示：

属性	说明
StyleObject-BackColor	设定背景色。
StyleObject-BorderColor	设定边框颜色。

StyleObject-BorderStyle	设定边框样式。(Dashed、Dotted、Double、Groove、Inset、None、NotSet、Outset、Ridge、Solid)。
StyleObject-BorderWidth	边框宽度，*单位类别。
StyleObject-CssClass	CssClass。
StyleObject-Font-Bold	粗体字，True/False。
StyleObject-Font-Italic	斜体字，True/False。
StyleObject-Font-Name	字型名称。
StyleObject-Font-Names	字型名称。
StyleObject-Font-Overline	顶线。
StyleObject-Font-Size	字号。
StyleObject-Font-Strikeout	穿越线。
StyleObject-Font-Underline	底线。
StyleObject-ForeColor	前景色。
StyleObject-Height	对象高度，*单位类别。
StyleObject-Width	对象宽度，*单位类别。
StyleObject-IsEmpty	对象是否不存在，只能用程序取回。

TableItem 样式

TableItem 样式最主要用来设定 TableItem 的样式，这些样式如下表所示：

属性	说明
StyleObject-HorizontalAlign	设定水平的对齐方式。（Center、Justify、Left、NotSet、Right）
StyleObject-VerticalAlign	设定垂直的对齐方式。（Bottom、Middle、NoSet、Top）
StyleObject-Warp	设定边是否自动断行，True、False。

DataGridPager 样式

DataGridPager 样式最主要用来设定 DataGrid Web 控件的分页样式，这些样式如下表所示：

属性	说明
PagerStyle-Mode	设定分页方式，NextPrev、NumericPages。
PagerStyle-NextPageText	设定下一页的文字。
PagerStyle-PageButtonCount	设定共有分页按钮的文字风格。
PagerStyle-Position	设定分页的地址。（Bottom、Top、TopAndBottom）
PagerStyle-PrevPageText	设定上一页的文字。

PagerStyle-Visible	设定是否显示，True/False。
--------------------	--------------------

单位类别

ASP.NET 提供了 Unit 类别，支持许多种单位，如下表所示：

单位	缩写	说明
Pixel	Px	像素。
Point	Pt	点。
Pica	Pc	皮卡，接近 1/6 英吋。
Inch	in	英吋。
Ems	mm	公厘。
Centimeters	cm	公分。
Percentage	%	百分比。

Unit 类别可以利用程序产生并指定，或是直接在宣告控件的时候设定。下列范例为设定 TextBox

Web 控件的宽度：

```
<Html>  
  
<ASP:TextBox Id="A" Runat="Server"/>100px<br>
```

```
<ASP:TextBox Id="B" Runat="Server"/>200px<br>
<ASP:TextBox Id="C" Runat="Server"/>300px<br>
<ASP:TextBox Id="D" Runat="Server"/>2cm<br>
<ASP:TextBox Id="E" Runat="Server" Width="4in"/>4in<br>
<ASP:TextBox Id="F" Runat="Server" Width="30mm"/>30mm<br>
<ASP:TextBox Id="G" Runat="Server" Width="10%"/>10%<br>
<ASP:TextBox Id="H" Runat="Server" Width="10pt"/>10pt

<Script Language="VB" Runat="Server">

Sub Page_Load(Sender As Object, e As EventArgs)

    A.Width=New Unit(100) ' 预设为像素

    B.Width=New Unit(200, UnitType.Pixel)

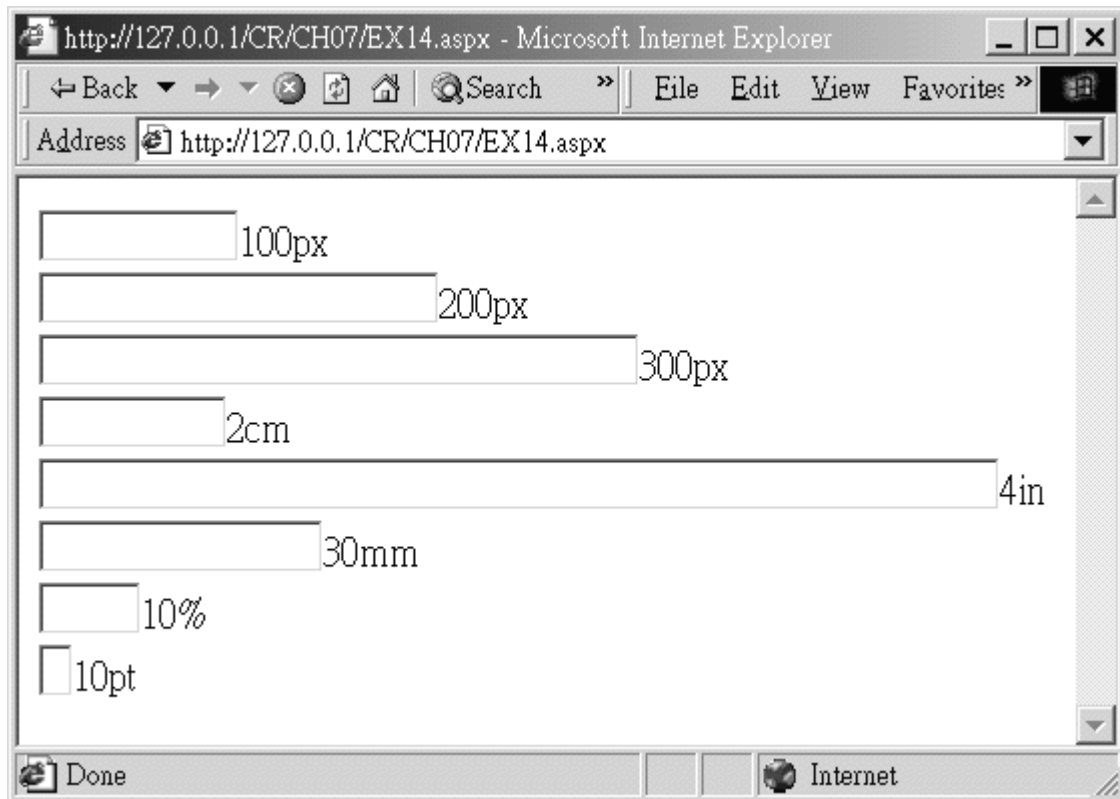
    C.Width=New Unit("300px")

    D.Width=New Unit("2cm")

End Sub

</Script>

</Html>
```



Calendar Web 控件

Calendar Web 控件可以让我们在网页上显示月历，也可以取得使用者在月历上点选的日子。其

使用语法为：

```
<ASP:Calendar
```

```
Id="被程序代码所控制的名称"
```

```
Runat="SERVER"
```

```
CellPadding="像素"
```

CellSpacing="像素"

DayNameFormat="FirstLetter | FirstTwoLetters | Full | Short"

FirstDayOfWeek="Default | Monday | Tuesday | Wednesday |

Thursday | Friday | Saturday | Sunday"

NextMonthText="HTML text"

NextPrevFormat="ShortMonth | FullMonth | CustomText"

PrevMonthText="HTML text"

SelectedDate="date"

SelectionMode="None | Day | DayWeek | DayWeekMonth"

SelectMonthText="HTML text"

SelectWeekText="HTML text"

ShowDayHeader="True | False"

ShowGridLines="True | False"

ShowNextPrevMonth="True | False"

ShowTitle="True | False"

TitleFormat="Month | MonthYear"

TodaysDate="date"

VisibleDate="date"

TodayDayStyle-property="value"

DayHeaderStyle-property="value"

```
DayStyle-property="value"

NextPrevStyle-property="value"

OtherMonthDayStyle-property="value"

SelectedDayStyle-property="value"

SelectorStyle-property="value"

StyeDayHeaderStyle-property="value"

TitleStyle-property="value"

TodayDayStyle-property="value"

WeekendDayStyle-property="value"

/>
```

Calendar Web 控件的常用属性

Calendar Web 控件的常用属性说明如下表所示：

属性	说明
CellPadding	储存格与表格边框的距离。
CellSpacing	储存格和储存格边框的距离。
DayNameFormat	显示星期几的格式。
FirstDayOfWeek	所要显示一周的第一天，可设定为：Default、Monday、Tuesday、Wednesday、

	Thursday、Friday、Saturday、Sunday。
NextMonthText	显示下个月的文字，以 HTML 设定。ShowNextPrevMonth 以及属性必须要设为 True，并且 NextPrevFormat 属性设定为 CustomText 才生效。
NextPrevFormat	要显示在标题列左右两侧下个月以及上个月的连结样式，可设定的属性为 ShortMonth、FullMonth 以及 CustomText（默认值）。本属性设为 CustomText 时，NextMonthText 属性以及 PrevMonthText 属性才生效。
PrevMonthText	显示上个月的文字，以 HTML 设定。ShowNextPrevMonth 以及属性必须要设为 True，并且 NextPrevFormat 属性设定为 CustomText 才生效。
SelectedDate	要突显的日期，预设是程序执行的日期。
SelectedDates	使用者所选取的多个日期，只读。
SelectionMode	设定使用者可以点选的日期，其可设定属性为 Day（默认值）、DayWeek、DayWeekMonth 或 None（无，单纯显示日期）。
SelectMonthText	要选取整月的文字，以 HTML 设定。本属性要将 SelectionMode 属性要设定成 DayWeekMonth 才生效。
SelectWeekText	要选取整周的文字，以 HTML 设定。本属性要将 SelectionMode 属性要设定成 DayWeek 或 DayWeekMonth 才生效。
ShowDayHeader	设定是否要显示星期的名称，True/False。
ShowGridLines	设定是否要显示网格线，True/False。

ShowNextPrevMonth	设定是否要显示上个月或下个月，True/False。
ShowTitle	设定是否要显示月历控件的标题列，True/False。如果设定为 False 时，会隐藏月份的名称及选择上下月的超级链接。
TitleFormat	设定标题列所要显示的日期格式，可设定为 MonthYear(默认值)或是 Month。
TodaysDate	用来当作今天的日期。设定 TodayDayStyle 属性时，在月历上才会显示今天的日期。
VisibleDate	决定用来显示哪个月份的日期，以这个日期的月份作决定。

下列范例显示一个简单的 **Calendar Web** 控件：

```
<Html>

<Form runat="Server">

<asp:Calendar id="Calendar1" runat="server"

    SelectedDate="2001/04/28"

    SelectionMode="DayWeekMonth"

    SelectMonthText="全选"

    ShowGridLines="True"

    ShowNextPrevMonth="True"

    NextMonthText="[<b>Next</b>]"

    TitleFormat="MonthYear"
```

```

SelectedDayStyle-BackColor="#DBDBDB"

SelectedDayStyle-ForeColor="Red"

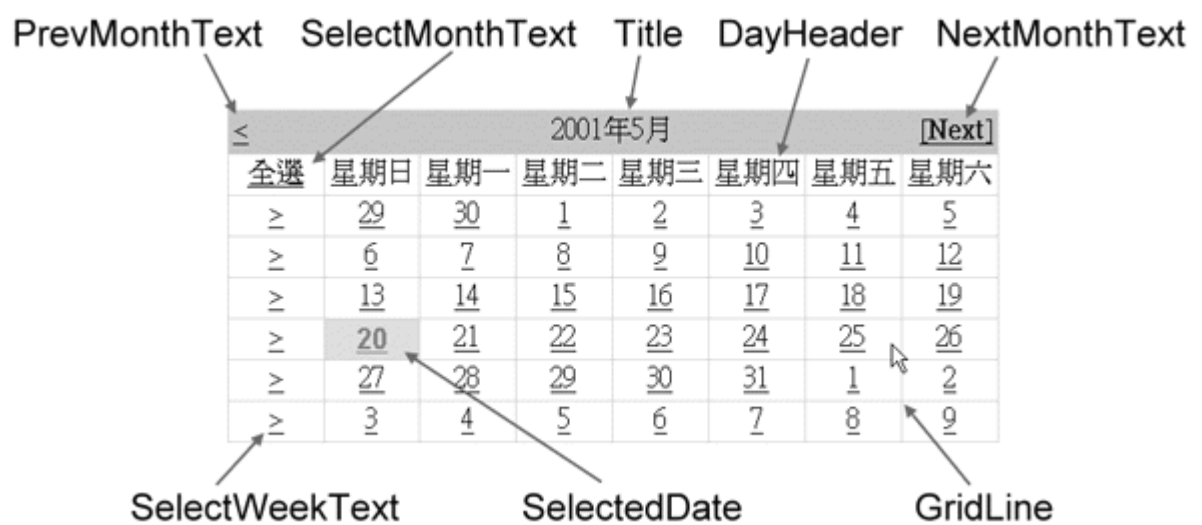
SelectedDayStyle-Font-Bold="True"

SelectedDayStyle-Font-Name="Arial"/>

</Form>

</Html>

```



Calendar Web 控件的样式对象

Calendar Web 控件支持许多样式对象，可以让我们可以更细微的设定其显示外观，如下表所示：

样式对象	样式类别	说明
DayHeaderStyle	TableItem	设定显示星期名称的样式。
DayStyle	TableItem	设定显示几日的样式。
NextPrevStyle	TableItem	设定显示上下月超级链接的样式。
OtherMonthDayStyle	TableItem	设定显示在月历上其它月份日期的样式。
SelectedDayStyle	TableItem	设定显示被选择日期的样式。
SelectorStyle	TableItem	设定选取整月或整周的超级链接样式。
TitleStyle	基础	设定标题列的样式。如果有设定 NextPrevStyle 的样式，则显示上下月的超级链接不受影响。
TodayDayStyle	TableItem	设定显示今天日期的样式。
WeekendDayStyle	TableItem	设定显示周末的样式。

下列范例自订了 **Calendar Web** 控件的外观：

```
<Html>

<Form Runat="Server">

<ASP:Calendar Id="Calendar1" Runat="Server"

    SelectedDate="2001/05/15" SelectionMode="DayWeekMonth"

    SelectMonthText="全选" ShowGridLines="True"
```

```
ShowNextPrevMonth="True" NextMonthText="[Next]"

TitleFormat="MonthYear" BorderColor="Black"

DayStyle-BorderColor="Black" DayStyle-BorderStyle="Dotted"

NextPrevStyle-ForeColor="White"

SelectorStyle-BackColor="#DBDBDB"

SelectorStyle-BorderColor="White"

SelectedDayStyle-BackColor="#DBDBDB"

SelectedDayStyle-ForeColor="Red"

SelectedDayStyle-Font-Bold="True"

SelectedDayStyle-Font-Name="Arial"

TitleStyle-BackColor="Black" TitleStyle-ForeColor="White"

TitleStyle-Font-Bold="True"

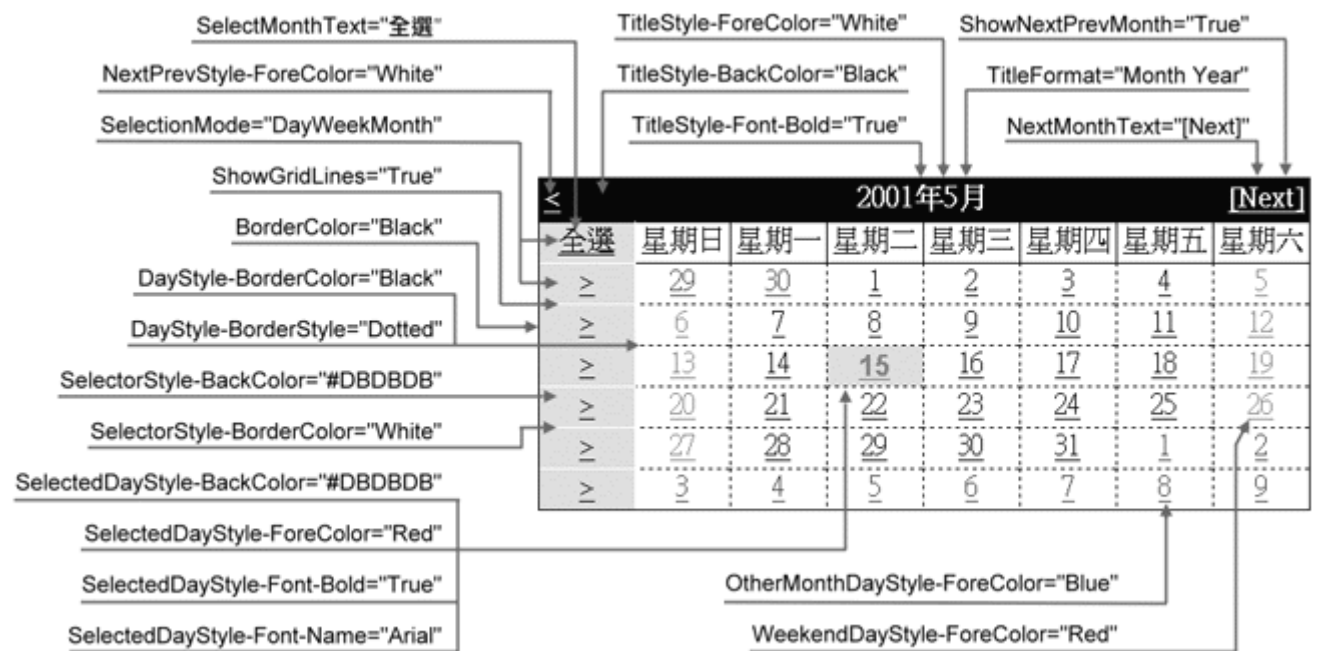
OtherMonthDayStyle-ForeColor="Blue"

WeekendDayStyle-ForeColor="Red"

/>

</Form>

</Html>
```



Calendar Web 控件的事件

Calendar Web 控件所支持的事件分别为 OnDayRender、OnVisibleMonthChanged 以及

OnSelectionChanged 这三个事件，接下来我们来了解如何使用。

OnSelectionChanged 事件

OnSelectionChanged 事件当使用者点选月历控件上的不同日期，或选了整月或整周时触发。其

宣告语法为：

```
Sub OnSelectionChanged(Sender As Object, e As EventArgs)
```

```
...
```

```
End Sub
```

下列范例显示使用者所点选的日子：

```
<Html>

<Form runat="Server">

<ASP:Calendar Id="calA" Runat="Server"

    SelectionMode="DayWeekMonth" ShowGridLines="True"

    ShowNextPrevMonth="True" NextMonthText="[Next]"

    SelectedDayStyle-BackColor="#DBDBDB"

    OnSelectionChanged="calA_SC"/>

</Form>

<ASP:Label Id="Label1" Runat="Server"/>

<Script Language="VB" Runat="Server">

Sub calA_SC(Sender As Object, e As EventArgs)

    Label1.Text="您所点选的日子为： " & calA.SelectedDate

End Sub

</Script>

</Html>
```



上述范例当使用者选取整周或整个月时，只能显示所选取的第一天；若要显示被选取的范围，利

用 **SelectedDates** 集合来取得使用者点选的范围。下列范例可以显示使用者选取的整周或整月：

```
<Html>

<Form runat="Server">

<ASP:Calendar Id="calA" Runat="Server"

    SelectionMode="DayWeekMonth" ShowGridLines="True"
```

```
ShowNextPrevMonth="True" NextMonthText="[Next]"

SelectedDayStyle-BackColor="#DBDBDB"

OnSelectionChanged="calA_SC"/>

</Form>

<ASP:Label Id="Label1" Runat="Server"/>

<Script Language="VB" Runat="Server">

Sub calA_SC(Sender As Object, e As EventArgs)

    Label1.Text="您所点选的为日期: " & calA.SelectedDate

    If calA.SelectedDates.Count > 1 Then

        Label1.Text += " 至 " &

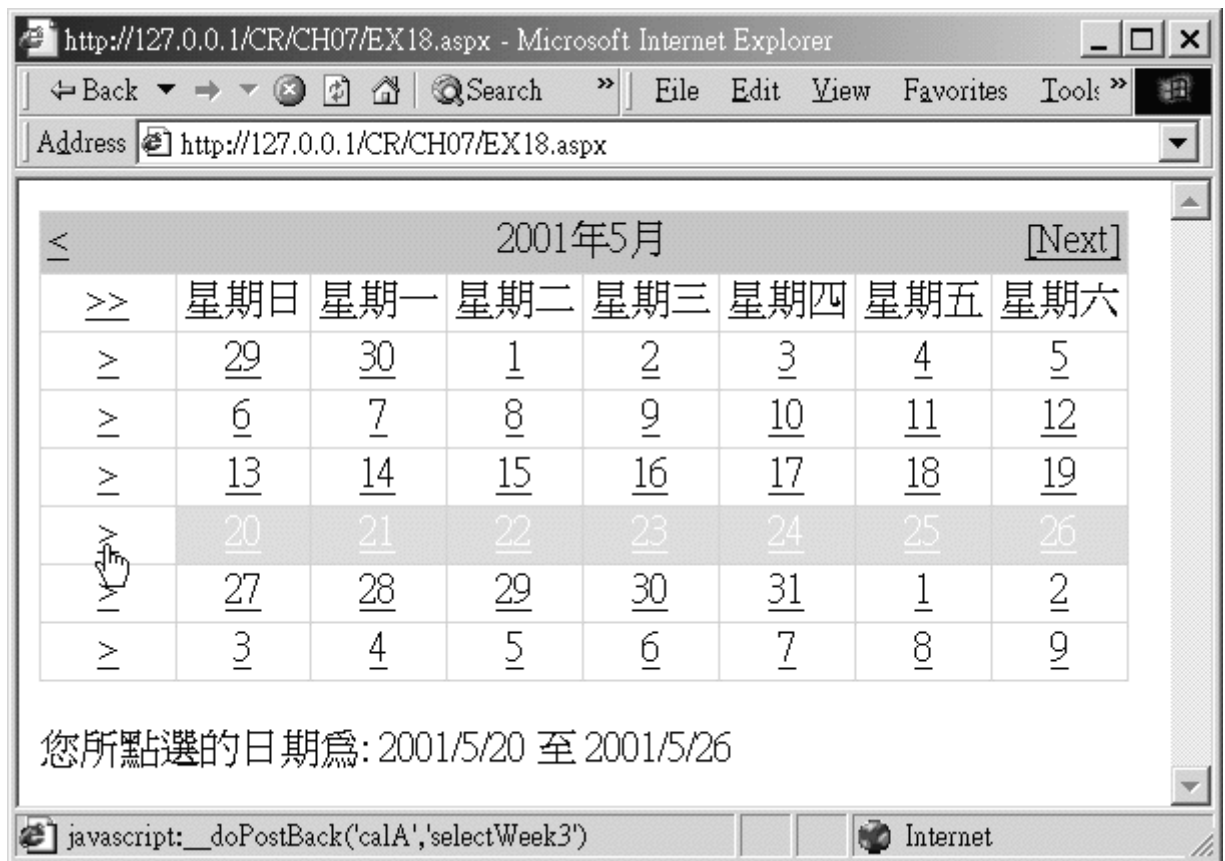
calA.SelectedDates.Item(calA.SelectedDates.Count-1)

    End If

End Sub

</Script>

</Html>
```

由于 **SelectedDates** 是集合对象，我们当然可以加入一些特定的日期到集合中，只要利用

SelectedDates.Add 方法即可。另外 **SelectedDates** 集合中有 **SelectRange** 属性可以用来指定某

个范围的日期。下列范例让使用者输入指定范围的日期，然后利用 **Calendar Web** 控件显示：

```
<Html>
```

```
<Form runat="Server">
```

请输入日期范围（格式 YYYY/MM/DD）:

从: <ASP:TextBox Id="Text1" Runat="Server"/>

至: <ASP:TextBox Id="Text2" Runat="Server"/>

```
<ASP:Button Id="btnOK" Text="确定" OnClick="btnOK_Click"
```

```
Runat="Server"/><br>
```

```
<ASP:Calendar Id="calA" Runat="Server"
```

```
    SelectionMode="DayWeekMonth" ShowGridLines="True"
```

```
    ShowNextPrevMonth="True" NextMonthText="[Next]"
```

```
    SelectedDayStyle-BackColor="#DBDBDB"/>
```

```
</Form>
```

```
<Script Language="VB" Runat="Server">
```

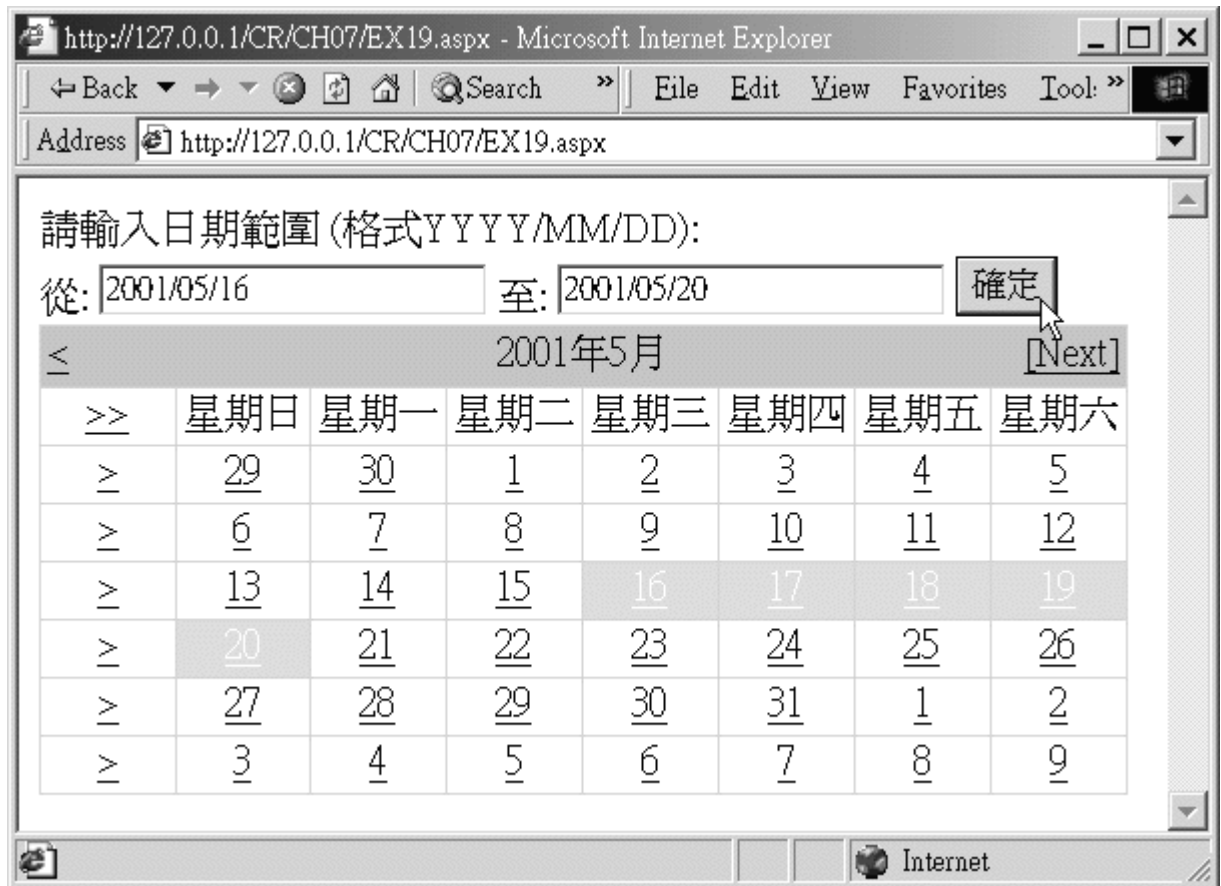
```
Sub btnOK_Click(Sender As Object, e As EventArgs)
```

```
    calA.SelectedDates.SelectRange(Text1.Text, Text2.Text)
```

```
End Sub
```

```
</Script>
```

```
</Html>
```



OnVisibleMonthChanged 事件

OnVisibleMonthChanged 事件当使用者点选月历控件标题列上的上个月或下个月按钮时触发。

其宣告语法为：

```
Sub OnVisibleMonthChanged(Sender As Object, e As MonthChangedEventArgs)
    ...
End Sub
```

其中参数 **e** 有两个属性，如下表所示：

参数	说明
e.NewDate	使用者所选的新日期。
e.PreviousDate	原先的日期。

OnDayRender 事件

OnDayRender 事件当月历控件在产生每一天的表格时触发。其宣告语法为：

```
Sub OnDayRender(Sender As Object, e As DayRenderEventArgs)

    ...

End Sub
```

其中参数 **e** 有 14 个属性，如下表所示：

参数	说明
e.Cell	TableCell 对象的参考。
e.Cell.RowSpan	传回或设定跨列数。
e.Cell.ColumnSpan	传回或设定跨栏数。
e.HorizontalAlign	传回或设定水平对齐方式。

e.VerticalAlign	传回或设定垂直对齐方式。
e.Cell.Warp	传回或设定字否自动断行，True/Flase。
e.Day	传回或设定要被产生的日。
e.Dat.Date	传回或设定要被产生的日期。
e.Day.DayNumberText	传回或设定日期的字符串型态的数值，例如 "20"。
e.Day.IsOtherMonth	传回所产生的日是不是属于其它月份，True/Flase。
e.Day.IsSelectable	传回或设定日期是否可以被选取，True/Flase。
e.Day.IsSelected	传回或设定日期是否被选取，True/Flase。
e.Day.IsToday	传回日期是否为今天，True/Flase。
e.Day.IsWeekend	传回日期是否是周末，True/Flase。

下列范例单纯的显示月历，并将本月的双数日期反像显示：

```

<%@Import Namespace="System.Drawing"%>

<Html>

<Form runat="Server">

<ASP:Calendar Id="calA" Runat="Server"

    SelectionMode="None" ShowGridLines="True"

    BorderColor="Gray" TitleStyle-BackColor="White"

```

```

        OnDayRender="calA_DayRender"/>

</Form>

<Script Language="VB" Runat="Server">

Sub calA_DayRender(Sender As Object, e As DayRenderEventArgs)

    If e.Day.DayNumberText.ToInt16 Mod 2 = 0 And Not e.Day.IsOtherMonth
Then

        e.Cell.BackColor=Color.Gray ' 以颜色名称设定颜色

        e.Cell.ForeColor=Color.FromARGB(255, 255, 255, 255) ' 设定颜色

    End If

End Sub

</Script>

</Html>

```

≤	2001年5月						≥
星期日	星期一	星期二	星期三	星期四	星期五	星期六	
29	30	1	2	3	4	5	
6	7	8	9	10	11	12	
13	14	15	16	17	18	19	
20	21	22	23	24	25	26	
27	28	29	30	31	1	2	
3	4	5	6	7	8	9	

要在程序中指定对象的颜色，要使用 **Color** 对象；要使用 **Color** 对象之前要先宣告 **Color** 对象的名称位置，其名称地址为 **System.Drawing**。**Color** 对象支持多种设定颜色的方式，其中最简单的方式是直接利用「**Color**. 颜色名称」的方式来宣告；或是以 **FromARGB** 方法指定颜色的 **Alpha**、**R**、**G**、**B** 值也可以。

8. Web 控件与数据源的系结 (Data Binding)

我们在第五章 ADO.NET 中已经介绍过如何和如 Access 以及 MS SQL Server 数据源进行互动，接下来我们要介绍如何将这些数据源的数据透过控件来展示。要将数据透过控件显示，可撰写一些程序进行手动的数据系结 (Data Binding)；或是透过控件本身的系结能力，让控件自动呈现数据。ASP.NET 可以当作数据源的对象很多，从最基本的变数，到 Array、ArrayList、Collection、DataSetView、DataView、DataSet、DataTable；或是一个对象的属性、一个叙述式、程序的传回值等都可以当作数据源。

数据系结叙述

要将控件和数据源进行系结的工作，最简单的方式就是直接把数据指定给控件的某个属性，或者是使用数据系结叙述。数据系结叙述可以让控件取得数据源的数据，只要在控件中需要数据源提供数据的地方插入 '<%#数据源%>' 这个叙述即可。

变数的系结

变量也可以当成数据源来提供数据。下列范例将 Label Web 控件利用数据系结叙述，取得变量的数据：

```
<Html>

<ASP:Label Id="Label1" Text='<#strMsg%>' Runat="Server"/>

<Script Language="VB" Runat="Server">

Dim strMsg As String    '宣告网页阶层的变量

Sub Page_Load(Sender As Object, e As EventArgs)

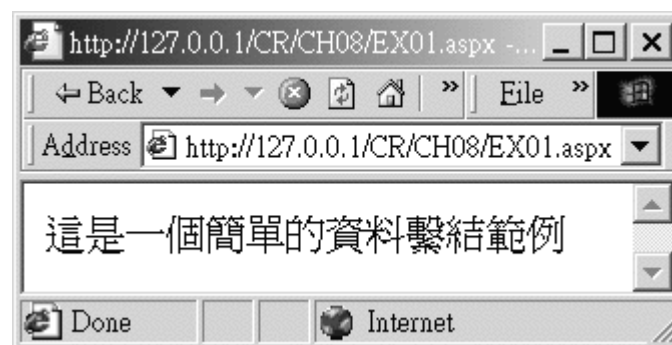
    strMsg = "这是一个简单的数据系结范例"

    Label1.DataBind()

End Sub

</Script>

</Html>
```



要将 Label1 的 Text 属性和数据源进行系统，要利用数据系统叙述 '<#strMsg%>' 表示要和 strMsg 这个变量进行数据系统。由于控件不会自动和数据源进行系统结的动作，我们必须使用控

件的 DataBind 方法来和数据源进行系统；所以我们在 Page_Load 事件程序中利用 Label Web 控件的 DataBind 方法和数据源进行数据系统，Label Web 控件才能取得数据源的数据。我们可以呼叫 Page 对象的 DataBind 方法，Page 对象的 DataBind 方法使用起来非常方便，在呼叫 Page 对象的 DataBind 方法时，Page 对象会自动呼叫所有控件的 DataBind 方法进行数据系统的工作，而不需要逐一呼叫每个控件的 DataBind 方法。另外特别注意，[由于数据系统叙述只可以和「网页阶层」变量进行数据系统的工作，所以变量必需在网页阶层的宣告区宣告才可以。](#)

变数的阶层

程序阶层变量

所谓程序阶层变量，就是在程序内宣告的变量。以前我们在 Sub 或是 Function 中宣告的变量，都是程序阶层的变量；也就是区域变量。[程序阶层变量在程序执行时被宣告产生，而程序执行完后也同时被毁灭](#)；也就是说其它的事件程序无法使用这个变量，因为它已经被毁灭了。下列范例宣告了一个程序阶层变量，并尝试在另外一个程序中将其它程序阶层的变量显示出来：

```
<Html>

<Script Language="VB" Runat="Server">

Sub Page_Init(Sender As Object, e As EventArgs)

Dim strTest As String '宣告程序阶层的变量
```

```
    strTest = "这是程序阶层的变量"

End Sub

Sub Page_Load(Sender As Object, e As EventArgs)

    Response.Write(strTest) ' 这里没有宣告 strTest 变量, 就是 Object 型态

End Sub

</Script>

</Html>
```

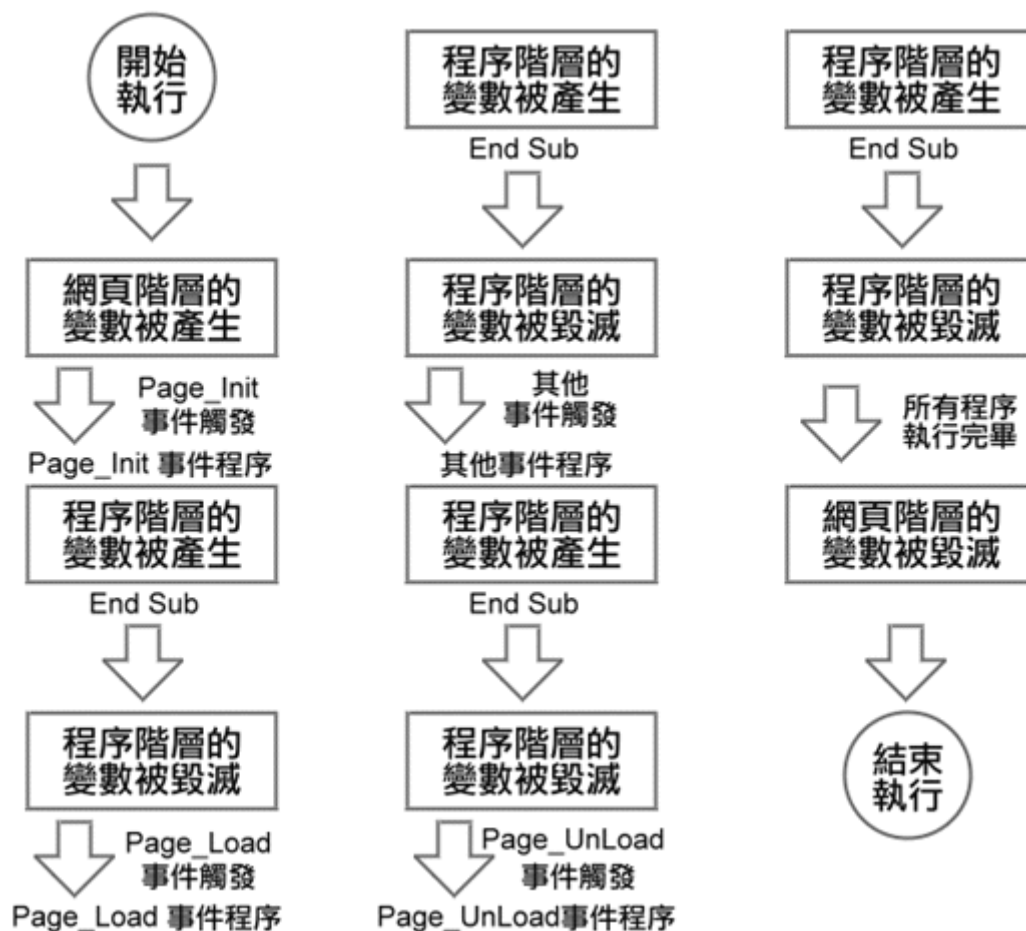
上述范例变量 **strTest** 在 **Page_Init** 事件中被宣告, 并指定其内容。不过由于该变量是属于程序阶层的变量, 所以在 **Page_Init** 事件程序执行完毕后即被消灭。因为在 **Page_Load** 事件程序中要显示变量 **strTest** 时, 由于 **strTest** 早已被消灭, 所以这里的 **strTest** 变量也是 **Page_Load** 程序阶层的变量, 对于没有宣告型态的变量就是 **Object** 型态, 无法显示任何的讯息。

网页阶层变数

而网页阶层的变量则是网页在加载执行时, 会先读取网页阶层宣告区中的宣告, 所有在网页阶层宣告区中宣告的变量都视为网页阶层变量: **网页阶层变量在网页执行时先被宣告产生, 在网页执行完毕后才会被毁灭**。所谓网页阶层的宣告区, 就是在程序外面, **<Script> </Script>** 之内的地方。为了程序的可读性, 我们习惯在 **<Script> </Script>** 标注中前面区域进行网页阶层的变量宣告。下列范例宣告了一个网页阶层变量 **strMSG**:

```
<Script Language="VB" Runat="Server">  
  
Dim strMsg As String      ' 网页阶层的宣告区  
  
Sub Page_Load(Sender As Object, e As EventArgs)  
  
    ...  
  
End Sub  
  
</Script>
```

网页阶层变量以及程序阶层变量的生命周期，如下插图所示：



我们知道网页开始执行时会先触发 **Page_Init** 事件，接着再触发 **Page_Load** 事件，所以我们利用这两个事件来作一个简单的试验。首先我们在网页层次的宣告区中宣告了一个变量 **strMsg**，另外在 **Page_Init** 事件程序中宣告了一个 **strTest** 变量；所以 **strMsg** 为网页阶层变量，而 **strTest** 为则为程序阶层变量，如下程序所示：

```

<Html>

<ASP:Label Id="Label1" Runat="Server"/>

<ASP:Label Id="Label2" Runat="Server"/>

```

```
<Script Language="VB" Runat="Server">

Dim strMsg As String '宣告网页阶层的变量


Sub Page_Init(Sender As Object, e As EventArgs)

Dim strTest As String '宣告程序阶层的变量

    strMsg = "这是窗体阶层的变量"

    strTest = "这是程序阶层的变量"

End Sub


Sub Page_Load(Sender As Object, e As EventArgs)

    Label1.Text=strMsg

    Label2.Text=strTest '这里没有宣告 strTest 变量, 就是 Object 型态

End Sub

</Script>

</Html>
```



上述范例的执行结果，只显示网页阶层变量的内容。这是因为在执行 **Page_Load** 事件程序的时候，由于 **strTest** 是在 **Page_Init** 事件程序中宣告的，所以 **Page_Init** 程序在执行完毕时立即将程序阶层变量 **strTest** 毁灭，执行到 **Page_Load** 事件程序时当然读取不到 **strTest** 这个程序阶层变量。而 **strMsg** 是在网页阶层宣告区中宣告的，在网页的所有程序执行完毕后才会被毁灭，所以在 **Page_Load** 事件程序中还是可以读到 **strMsg** 的内容。

程序的系结

由于 **Function** 程序有传回值，所以也可以当成数据源来提供数据。下列范例将 **Label Web** 控件利用数据系结叙述，取得程序传回值的数据：

```
<Html>

<ASP:Label Id="Label1" Text='<%#2 的平方为 " + Squ(2)%>'

Runat="Server"/>

<Script Language="VB" Runat="Server">

Sub Page_Load(Sender As Object, e As EventArgs)
```



```
Page.DataBind()

End Sub

Public Function Squ(intA)

    Return (intA*intA).ToString

End Function

</Script>

</Html>
```



上述范例除了接收 **Function** 所传回的结果外，并且在数据系结叙述中将两个字符串以字符串结合运算符「+」作结合。

基础 **Web** 控件与数据源的系结

CheckBoxList、DropDownList、ListBox 以及 RadioButtonList 这些基础 Web 控件有数据系统的
能力，因为他们有 DataSource、DataTextField 以及 DataValueField 这三个属性。DataSource
表示所要使用的数据源为何，DataTextField 表示控件所要显示的是资料源中的哪个字段，
DataValueField 表示 Web 控件使用某个数据源字段的值代表某个项目被选取的值。

系结至数组

数组也可以当作数据源来使用，不过如果 [数组要拿来当数据源使用，必须要是一维数组才可以](#)。

下列范例利用 ListBox Web 控件系结到一个数组中：

```
<Html>

请输入您的血型：

<ASP:ListBox Id="ListBox1" DataSource='<%#arA%>' Rows="4"

Runat="Server"/>

<Script Language="VB" Runat="Server">

Dim arA() As String = {"A", "B", "O", "AB"} '宣告网页阶层的数组

Sub Page_Load(Sender As Object, e As EventArgs)

    Page.DataBind()

End Sub

</Script>
```

```
</Html>
```



系结至 **ArrayList** 物件

ArrayList 对象和数组很类似，也可以被我们当作资料来源来使用。下列范例利用 **CheckBoxList**

Web 控件系结到一个 **ArrayList** 中：

```
<Html>
```

请选择您的兴趣：

```
<ASP:CheckBoxList Id="CheckBoxList1" DataSource='<%#alA%>'
```

```
Runat="Server"/>
```

```
<Script Language="VB" Runat="Server">
```

```
Dim alA As ArrayList = New ArrayList

Sub Page_Load(Sender As Object, e As EventArgs)

    alA.Add("爬山")

    alA.Add("打球")

    alA.Add("计算机")

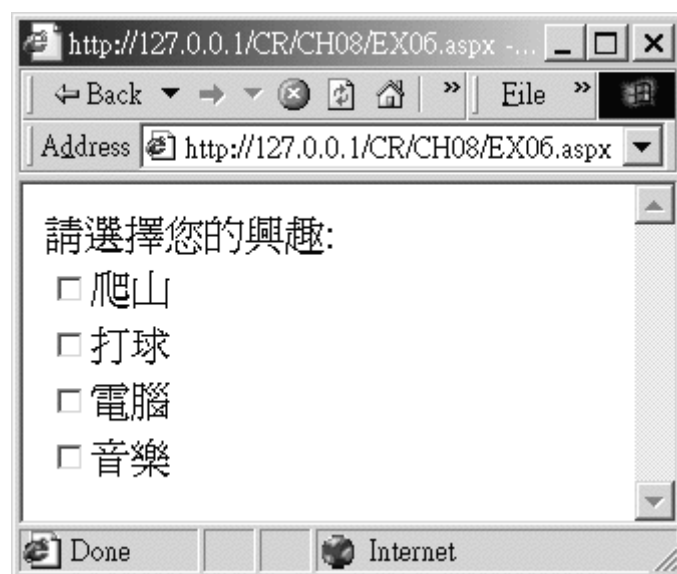
    alA.Add("音乐")

    Page.DataBind()

End Sub

</Script>

</Html>
```



数据系结属性

基本数据系结

我们知道数据系结叙述只能和网页阶层的变量系结，而利用程序代码设定数据系结属性就不需要宣告网页阶层的变量。下列范例在程序代码中设定 **DropDownList** 的数据系结属性，和程序阶层的 **ArrayList** 进行数据系结的工作：

```
<Html>

请选择居住城市：

<ASP:DropDownList Id="ddlA" Runat="Server"/>

<Script Language="VB" Runat="Server">

Sub Page_Load(Sender As Object, e As EventArgs)

Dim alA As ArrayList = New ArrayList '宣告程序阶层的 ArrayList

    alA.Add("台北市")

    alA.Add("新竹市")

    alA.Add("台中市")

    alA.Add("台南市")
```

```

alA.Add("高雄市")

ddlA.DataSource=alA '以程序的方式设定数据系结属性

Page.DataBind()

End Sub

</Script>

</Html>

```

系结至集合对象

集合对象也可以当成数据源使用。例如 **DataTable** 中的 **Rows** 集合，也可以当作数据源让控件系结。下列范例利用 **DropDownList Web** 控件将我们在第五章所建立的 **Members** 数据表中的 **Columns** 集合当成数据源使用：

```

<%@Import Namespace=System.Data.ADO%>

<%@Import Namespace=System.Data%>

<Html>

请选择一个字段：

<ASP:DropDownList Id="ddlA" Runat="Server"/>

<Script Language="VB" Runat="Server">

Sub Page_Load(Sender As Object, e As EventArgs)

```

```

Dim strConStr As String = "Provider=Microsoft.Jet.OLEDB.4.0;" & _
    "Data Source=C:\InetPub\wwwroot\CR\CH05\MyWeb.mdb"

Dim strComStr As String = "Select * From Members"

Dim dscA As ADODatasetCommand = New
ADODatasetCommand(strComStr, strConStr)

Dim dsDataSet As DataSet = New DataSet()

dscA.FillDataSet(dsDataSet, "Members")

ddlA.DataSource=dsDataSet.Tables("Members").Columns ' 将 Columns 集
合当成数据源

Page.DataBind()

End Sub

</SCRIPT>

</Html>

```

系结至 **DataView**

DataView 对象也可以当成数据源使用。下列范例利用 **RadioButtonList Web** 控件将 **Members**

数据表的 **DefaultView** 对象当成数据源使用：

```

<%@Import Namespace=System.Data.ADO%>

```

```

<%@Import Namespace=System.Data%>

<Html>

<Form Id=Form1 Runat="Server">

请选择一个使用者:

<ASP:RadioButtonList Id="rblA" AutoPostBack="True"

        OnSelectedIndexChanged="rblA_Chg" Runat="Server"/>

</Form>

<ASP:Label Id="Label1" Runat="Server"/>


<Script Language="VB" Runat="Server">

Sub Page_Load(Sender As Object, e As EventArgs)

If Page.IsPostBack=False Then ' 连接数据库读取数据及系结的工作只要作一
次

    Dim strConStr As String = "Provider=Microsoft.Jet.OLEDB.4.0;" & _

        "Data Source=C:\InetPub\wwwroot\CR\CH05\MyWeb.mdb"

    Dim strComStr As String = "Select * From Members"

    Dim dscA As ADODatasetCommand = New

ADODatasetCommand(strComStr, strConStr)

    Dim dsDataSet As DataSet = New DataSet()

    dscA.FillDataSet(dsDataSet, "Members")

```



```

        rblA.DataSource=dsDataSet.Tables("Members").DefaultView      ' 将
DataTable 当成数据源

        rblA.DataTextField="UserName"      ' 指定要显示的字段

        rblA.DataValueField="UserId"      ' 指定这个字段代表的值

        Page.DataBind()

End If

End Sub

Sub rblA_Chg(Sender As Object, e As EventArgs)

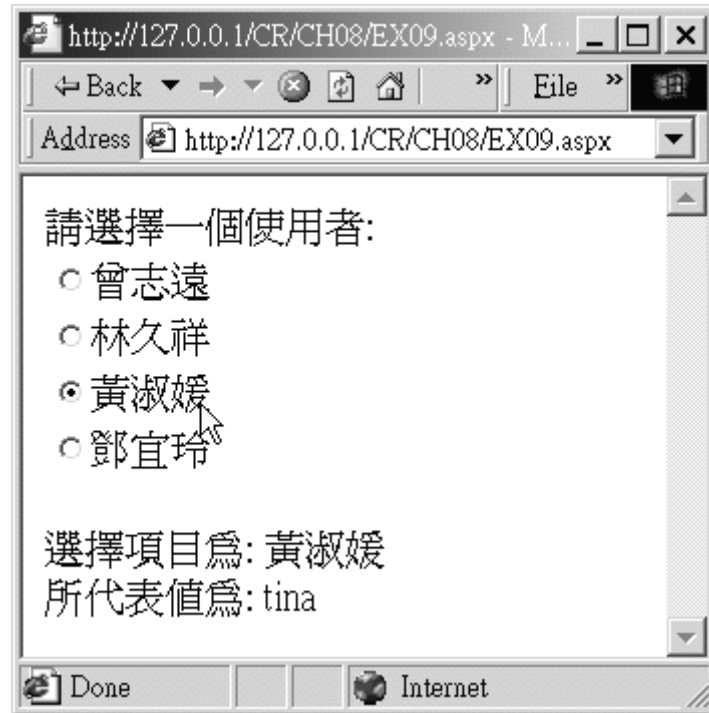
    Label1.Text="选择项目为: " & rblA.SelectedItem.Text & _
"<br>所代表值为: " & rblA.SelectedItem.Value

End Sub

</SCRIPT>

</Html>

```



包括 CheckBoxList、DropDownList、ListBox，以及 RadioButtonList Web 控件在內，一次只能显示一个字段的数据；所以我们在指定 DataSource 属性后还必须指定 DataTextField 属性，表示要用来显示的字段为何。另外 DataValueField 属性可以用来表明使用者选择了一个项目后，代表该项目的值为何。上述的例子中我们将 RadioButtonList Web 控件的 DataValueField 属性设定为 UserId，表示当使用者选择了使用者的姓名后，我们可以利用 ListItem 的 Value 属性取得这个项目的值；所以我们选择使用者「黄淑媛」后就可以取得其相关值「tina」，在许多应用上非常的方便。

含入檔的使用

我们可以将经常用到的程序另外储存于扩展名为 `.inc` 的档案中，要使用的时候再利用将之含入（`include`）即可。

含入檔的建立

例如我们有一个从 Access 的 `mdb` 檔中取回指定 `DataTable`，并将 `DataTable` 对象传回的

`Function` 程序；要将这个 `Function` 程序制作成含入文件，只要将 `Function` 程序写入 `<Script>` 标注中，并存于 `GetTable.inc` 檔中即可；如下程序代码所示：

```
<Script Language="VB" Runat="Server">

Function GetTable(ByVal strMDB As String, ByVal strTable As String)

    Dim strConStr As String = "Provider=Microsoft.Jet.OLEDB.4.0;" & _
        "Data Source=C:\InetPub\wwwroot\CR\" & strMDB

    Dim strComStr As String = "Select * From " & strTable

    Dim dscA As ADODatasetCommand = New
ADODatasetCommand(strComStr, strConStr)

    Dim dsDataSet As DataSet = New DataSet()

    dscA.FillDataSet(dsDataSet, strTable)

    Return dsDataSet.Tables(strTable)

End Function
```

```
</SCRIPT>
```

这个 Function 程序在使用的时候只要传入 Access 数据库档案的地址，以及所要取得的 Table 名称即可。

含入档的参考

而要用含入档时，只要加入 `<!--#Include File="文件名称.inc"-->` 即可。指定 File 属性可以直接指定相同层次的目录或是子目录的含入文件，但是 **不可以指定上一层目录或是其它目录的含入文件**。如果含入档不是存放于同一个目录或是同一个目录下的子目录，则可以指定 Virtual 属性。Virtual 属性是以 WWW 服务的根目录起算（一般是 "c:\inetpub\wwwroot"），例如我们的含入档放于 WWW 服务的根目录里，我们只要指定 `<!--#Include Virtual="文件名称.inc"-->` 即可。下列范例取回 MyWeb.mdb 数据库中的 Members 数据表，并利用 DataGrid Web 控件显示：

```
<%@Import Namespace=System.Data.ADO%>

<%@Import Namespace=System.Data%>

<!--#Include File="GetTable.inc"-->

<Html>

<ASP:DataGrid Id="dgDataGrid" Runat="Server"/>

<Script Language="VB" Runat="Server">
```

```

Sub Page_Load(Sender As Object, e As EventArgs)

    Dim dtDataTable As DataTable= GetTable("CH05\MyWeb.mdb", "Members")

    dgDataGrid.DataSource=dtDataTable.DefaultView    ' 将 DataView 当成数
据源

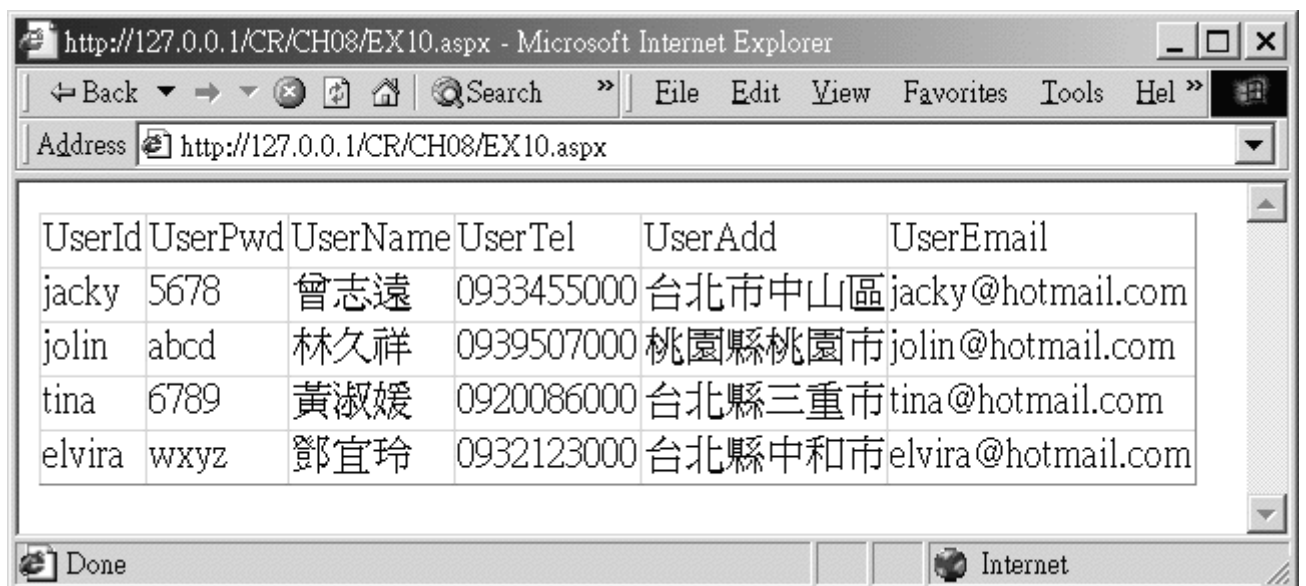
    Page.DataBind()

End Sub

</SCRIPT>

</Html>

```



后面我们会详细介绍 **DataGrid Web** 控件的使用。

Repeater Web 控件

Repeater Web 控件最主要的用途，是可以将数据依照我们所制定的格式逐一显示出来。只要将想要显示的格式先定义好，**Repeater Web** 就会依照我们所定义的格式来显示；这个预先定义好的格式我们称为「样版」（**Template**）。使用样版可以让我们的资料可以更容易、更美观的呈现给使用者；支持样版的 **Web** 控件有 **Repeater**、**DataList** 以及 **DataGrid**。接下来我们先来了解 **Repeater Web** 控件的使用语法：

```
<ASP:Repeater  
  
    Id="被程序代码所控制的名称"  
  
    Runat="Server"  
  
    DataSource='<%# 数据系结叙述 %>'  
  
>  
  
    <Template Name="样版名称">  
  
        以 HTML 所定义的样版  
  
    </Template >  
  
    其它样版定义...  
  
</ASP:Repeater>
```

Repeater Web 控件所支持的样版如下表所示：

样板名称	说明
HeaderTemplate	数据表头的样式。
ItemTemplate	呈现数据的样式。本样版为必要样版，不可省略。
AlternatingItemTemplate	如有定义本样版，则显示时会与 Item 样版交互出现。
SeparatorTemplate	分隔两笔数据的样式。
FooterTemplate	数据表尾的样式。

其中 **Item** 样版必须要定义才能顺利显示资料。下列范例利用 **Repeater Web** 控件显示使用者姓名及电子邮件信箱：

```
<%@Import Namespace=System.Data.ADO%>

<%@Import Namespace=System.Data%>

<!--#Include File="GetTable.inc"-->

<Html>

<ASP:Repeater Id="rpA" Runat="Server">

    <Template Name="ItemTemplate">

        <ASP:Image Id="I1" ImageUrl="ico1.gif" Runat="Server"/>

        姓名:<%#Container.DataItem("UserName")%><br>

        <ASP:Image Id="I2" ImageUrl="ico2.gif" Runat="Server"/>

        电邮:<%#Container.DataItem("UserEmail")%><br>
    </Template>
</ASP:Repeater>
</Html>
```

```
</Template>

</ASP:Repeater>

<Script Language="VB" Runat="Server">

Sub Page_Load(Sender As Object, e As EventArgs)

    Dim dtDataTable As DataTable=GetTable("CH05\MyWeb.mdb", "Members")

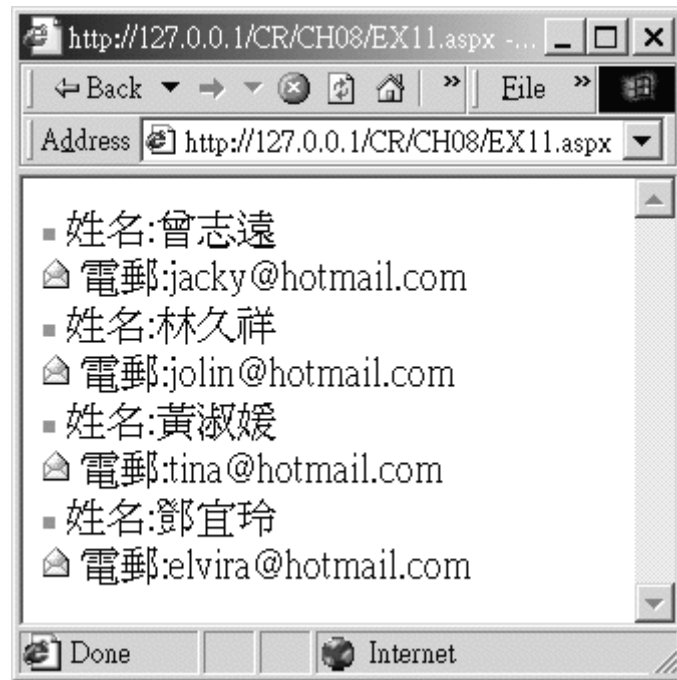
    rpA.DataSource=dtDataTable.DefaultView

    Page.DataBind()

End Sub

</SCRIPT>

</Html>
```

上述范例中定义了一个 **ItemTemplate** 样版，在这个样版中的数据系统叙述使用了

`<%#Container.DataItem("UserName")%>` 这个资料系统叙述，其中 **Container** 英文的意思是容器，这里是表示样版放置在哪个对象中；我们在 **Repeater Web** 控件中使用这个样版，所以这个样版的容器就是 **Repeater Web** 控件。当程序在执行时，使用 **Container** 关键词会传回 **Repeater Web** 控件的参考，这时候就可以利用 **Repeater Web** 控件的 **DataItem** 属性取得目前要显示的记录字段。下列范例展现了所有的样版：

```
<%@Import Namespace=System.Data.ADO%>

<%@Import Namespace=System.Data%>

<!--#Include File="GetTable.inc"-->

<Html>

<ASP:Repeater Id="rpA" Runat="Server">
```

```
<Template Name="HeaderTemplate">

    <ASP:Image ImageUrl="ico3.gif" Runat="Server"/> <br>

</Template>

<Template Name="ItemTemplate">

    <ASP:Image ImageUrl="ico1.gif" Runat="Server"/>

    姓名: <%#Container.DataItem("UserName")%><br>

    <ASP:Image ImageUrl="ico2.gif" Runat="Server"/>

    电邮: <%#Container.DataItem("UserEmail")%><br>

</Template>

<Template Name="AlternatingItemTemplate">

    <ASP:Image ImageUrl="ico1.gif" Runat="Server"/>

    姓名: <%#Container.DataItem("UserName")%>

    <ASP:Image ImageUrl="ico2.gif" Runat="Server"/>

    电邮: <%#Container.DataItem("UserEmail")%><br>

</Template>

<Template Name="SeparatorTemplate">

    <ASP:Image ImageUrl="ico5.gif" Runat="Server"/><br>

</Template>

<Template Name="FooterTemplate">

    <ASP:Image ImageUrl="ico4.gif" Runat="Server"/><br>
```

```
</Template>

</ASP:Repeater>

总共有 <ASP:Label Id="Label1" Runat="Server"/> 笔记录

<Script Language="VB" Runat="Server">

Sub Page_Load(Sender As Object, e As EventArgs)

    Dim dtDataTable As DataTable=GetTable("CH05\MyWeb.mdb", "Members")

    rpA.DataSource=dtDataTable.DefaultView

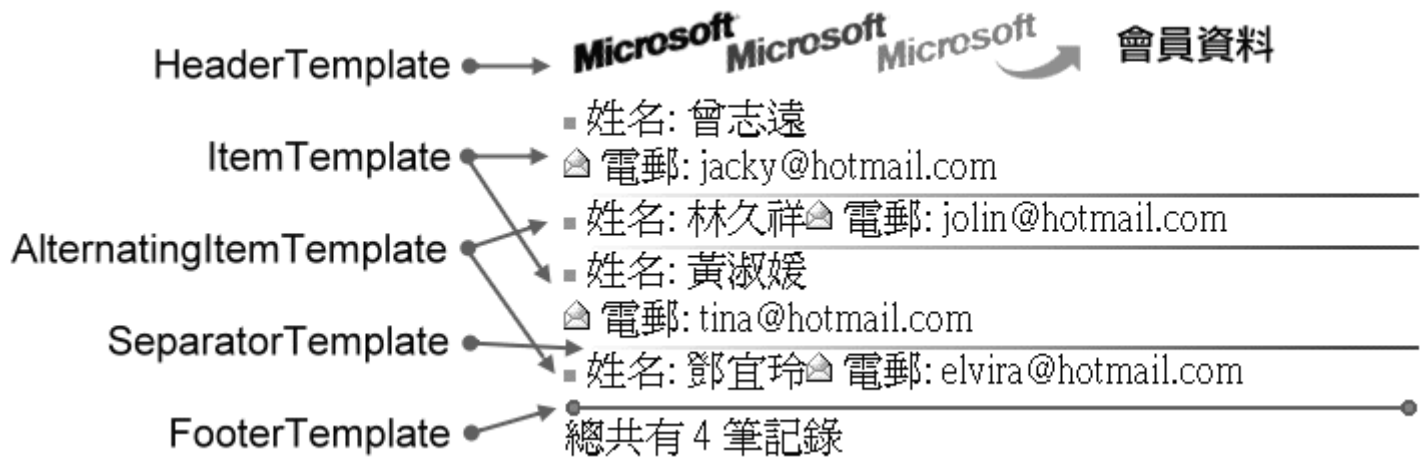
    Page.DataBind()

    Label1.Text=rpA.Items.Count

End Sub

</SCRIPT>

</Html>
```



DataList Web 控件

DataList Web 控件和 Repeater Web 控件有点类似，不过 DataList Web 控件除了可以将数据依照我们所制定的样版显示出来外，还可以进行 Repeater Web 控件无法作到的数据编辑。接下来我们先来了解 DataList Web 控件的使用语法：

```
<ASP:DataList
```

Id="被程序代码所控制的名称"

Runat="Server"

CellPadding="像素"

CellSpacing="像素"

DataKeyField="数据源的主键字段"

DataSource='<#数据系结叙述%>'

GridLines="None | Horizontal | Vertical | Both"

RepeatColumns="ColumnCount"

RepeatDirection="Vertical | Horizontal"

RepeatLayout="Flow | Table"

ShowFooter="True | False"

ShowHeader="True | False"

OnSCancelCommand="事件程序"

OnDeleteCommand="事件程序"

OnEditCommand="事件程序"

OnItemCommand="事件程序"

OnItemCreated="事件程序"

OnUpdateCommand="事件程序"

>

<Template Name="样版名称">

以 HTML 所定义的样版

</Template >

其它样版定义...

AlternatingItemStyle-Property="value"

EditItemStyle-Property="value"

```
FooterStyle-Property="value"

HeaderStyle-Property="value"

ItemStyle-Property="value"

SelectedItemStyle-Property="value"

SeparatorStyle-Property="value"

</ASP:DataList>
```

DataList Web 控件常用的属性如下表所示：

属性	说明
CellPadding	储存格与表格边框的距离。
CellSpacing	储存格和储存格边框的距离。
DataKeyField	设定在数据源中为主键的字段。
DataSource	设定数据系统所要使用的数据源。
EditItemIndex	设定要被编辑的字段名称。本属性设为 -1 可放弃编辑。
GridLines	设定是否要显示网格线。本属性在 RepeatLayout 属性设为 Table 时才有效。
Items	DataListItem 的集合对象。本对象只包含和数据源系结的 Item，也就是说不包含 Header、Footer 及 Separator 样版。
RepeatColumns	设定一次所要显示的字段数，默认值是 1。

RepeatDirection	<p>设定资料所要显示的方向。假设 RepeatColumns 属性为 2，如果设定为 Vertical，则资料的显示为上下依序显示，显示完毕后再显示第二栏；如果设定为 Horizontal，则资料的显示为左右依序显示，显示完毕后再显示第二列。</p> <p>RepeatColumns 设定为 1 时，本属性为 Vertical。</p>
RepeatLayout	<p>设定所要显示资料的方式是逐行显示（Flow）还是利用表格显示（Table）。</p> <p>默认值是 Table。</p>
SelectedIndex	设定哪一列被点选到。设定此属性时，该列会以 Selected 样版的样式来显示。
SelectedItem	传回被点选到的 Item。
ShowFooter	设定是否要显示脚注（Footer），True/False。
ShowHeader	设定是否要显示表头（Header），True/False。

DataList Web 控件所支持的样版如下表所示：

样板名称	说明
HeaderTemplate	数据表头的样式。
ItemTemplate	呈现数据的样式。本样版一定要宣告，不可省略。
AlternatingItemTemplate	如有定义本样版，则显示时会与 Item 样版交互出现。
EditItem	编辑数据的样版。
SelectedItem	选择项目时的样版。

SeparatorTemplate	分隔两笔数据的样式。
FooterTemplate	数据表尾的样式。

其中 **Item** 样版也是必须要定义才能顺利显示资料。另外 **DataList Web** 控件也支持许多样式对象，

可以让我们可以灵活的自订其显示外观，如下表所示：

样式对象	样式类别	说明
AlternatingItemStyle	TableItem	每一个交替项目所要显示的样式。
EditItemStyle	TableItem	项目在被编辑时所要显示的样式。
FooterStyle	TableItem	脚注所要显示的样式。
HeaderStyle	TableItem	标头所要显示的样式。
ItemStyle	基础	每一个项目所要显示的样式。
SelectedItemStyle	TableItem	项目在被选择时所要显示的样式。
SeparatorStyle	TableItem	所要显示的分隔样式。

下列范例利用 **DataList Web** 控件显示 **Members** 数据表，并指定要以表格的方式显示，每次显

示两行：

```
<%@Import Namespace=System. Data. ADO%>

<%@Import Namespace=System. Data%>

<!--#Include File="GetTable.inc"-->
```



```
<Html>

<Form runat="Server">

<ASP:DataList Id="dlA" RepeatColumns="2" GridLines="both"
Runat="Server">

    <Template Name="ItemTemplate">

        <ASP:Image ImageUrl="icol.gif" Runat="Server"/>

        姓名: <%#Container.DataItem("UserName")%>

    </Template>

</ASP:DataList>

</Form>

<Script Language="VB" Runat="Server">

Sub Page_Load(Sender As Object, e As EventArgs)

Dim dtDataTable As DataTable=GetTable("CH05\MyWeb.mdb", "Members")

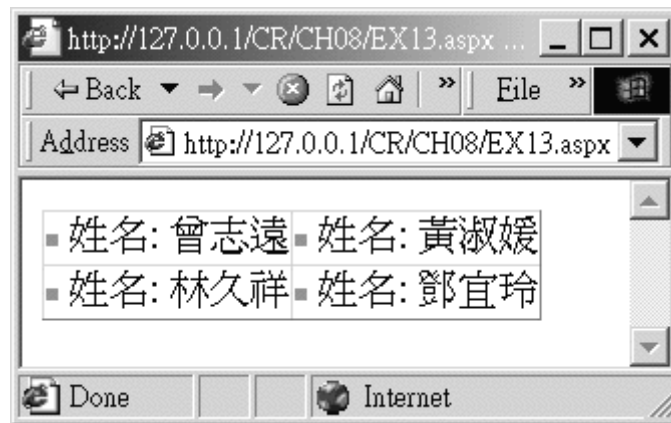
    dlA.DataSource=dtDataTable.DefaultView

    Page.DataBind()

End Sub

</SCRIPT>

</Html>
```



其中 RepeatDirection 属性决定数据排列的方式，如下图所示：

RepeatDirection="Vertical"



Rows(0)	Rows(1)
■ 姓名: 曾志遠	■ 姓名: 黃淑媛
■ 姓名: 林久祥	■ 姓名: 鄧宜玲
Rows(2)	Rows(3)

RepeatDirection="Horizontal"



Rows(0)	Rows(2)
■ 姓名: 曾志遠	■ 姓名: 林久祥
■ 姓名: 黃淑媛	■ 姓名: 鄧宜玲
Rows(1)	Rows(3)

DataList Web 控件支持六个事件，如下表所示：

事件名称	说明
OnCancelCommand	当在 EditItemTemplate 中所宣告的 Button 或 LinkButton 控件触发事件时，如果控件的 CommandName 属性为 Cancel 时，则触发本事件。
OnDeleteCommand	当在 ItemTemplate 中所宣告的 Button 或 LinkButton 控件触发事件时，如果控件的 CommandName 属性为 Delete 时，则触发本事件。
OnEditCommand	当在 ItemTemplate 中所宣告的 Button 或 LinkButton 控件触发事件时，如果控件的 CommandName 属性为 Edit 时，则触发本事件。
OnItemCommand	当在 ItemTemplate 中所宣告的 Button 或 LinkButton 控件触发事件时，如果 CommandName 属性的内容不是 Edit、Cancel、Delete 或 Update 时即触发本事件。
OnItemCreated	当 DataList 中的每一个项目被产生时触发。
OnUpdateCommand	当在 EditTemplate 中所宣告的 Button 或 LinkButton 控件触发事件时，如果控件的 CommandName 属性为 Update 时，则触发本事件。

OnItemCommand 事件

OnItemCommand 当在 ItemTemplate 中所宣告的 Button 或 LinkButton 控件触发事件时，如果该控件的 CommandName 属性内容不是 Edit、Cancel、Delete 或 Update 时，便触发本事件。

其宣告语法为：

```
Sub OnItemCommand(Sender As Object, e As DataListCommandEventArgs)

    ...

End Sub
```

其中参数 **e** 有 2 个属性，如下表所示：

参数	说明
e.CommandSource	触发事件之对象的参考。
e.Item	DataListItem 对象的参考。

要使用本事件前，首先必须在 **ItemTemplate** 中宣告 **LinkButton** 或 **Button Web** 控件。下列范例在 **ItemTemplate** 中宣告一个 **LinkButton Web** 控件，当使用者按下 **LinkButton Web** 控件后，会显示该项目的详细数据：

```
<%@Import Namespace=System.Data.ADO%>

<%@Import Namespace=System.Data%>

<!--#Include File="GetTable.inc"-->

<Html>

<Form runat="Server">

<ASP:DataList Id="d1A" OnItemCommand="d1A_ICMD" GridLines="both"

Runat="Server">

    <Template Name="ItemTemplate">
```

```

        <ASP:Image ImageUrl="ico1.gif" Runat="Server"/>

        姓名: <%#Container.DataItem("UserName")%>

        <ASP:LinkButton Id="lbShow" Text=">" Runat="Server"/>

    </Template>

    <Template Name="SelectedItemTemplate">

        <ASP:Image ImageUrl="ico1.gif" Runat="Server"/>

        姓名: <%#Container.DataItem("UserName")%>

        <ASP:LinkButton Id="lbClose" Text="<" Runat="Server"/><br>

        电话: <%#Container.DataItem("UserTel")%><br>

        住址: <%#Container.DataItem("UserAdd")%><br>

        电邮: <%#Container.DataItem("UserEmail")%><br>

    </Template>

</ASP:DataList>

</Form>

<Script Language="VB" Runat="Server">

Sub Page_Load(Sender As Object, e As EventArgs)

Dim dtDataTable As DataTable=GetTable("CH05\MyWeb.mdb", "Members")

    dlA.DataSource=dtDataTable.DefaultView

    Page.DataBind()

End Sub

```

```

Sub dlA_ICMD(Sender As Object, e As DataListCommandEventArgs)

    If e.CommandSource.Id="lbShow" Then

        dlA.SelectedIndex=e.Item.ItemIndex

    ElseIf e.CommandSource.Id="lbClose"

        dlA.SelectedIndex=-1

    End If

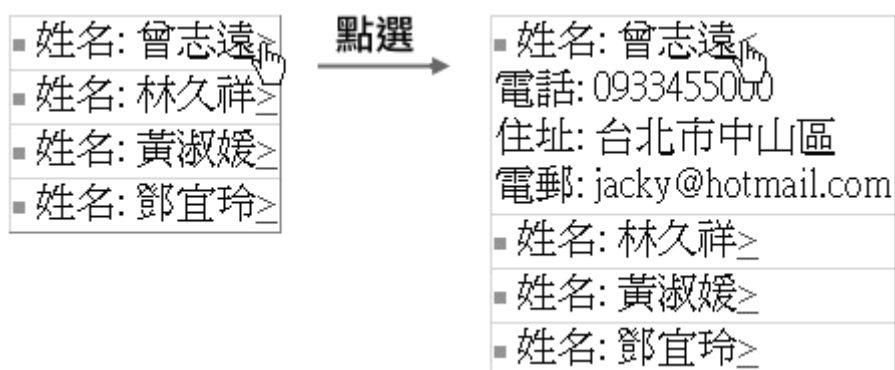
    dlA.DataBind()

End Sub

</SCRIPT>

</Html>

```



上述范例我们定义了 **ItemTemplate** 以及 **SelectedItemTemplate** 这两个样版，并指定 **DataList**

Web 控件的 **OnItemCommand** 属性为 **dlA_ICMD**，表示当 **Template** 中的控件引发事件时，只

要引发事件的控件其 **CommandName** 属性值不是 **Edit**、**Cancel**、**Delete** 或 **Update** 时，才会触发本事件。不过我们在 **Template** 中的 **LinkButton** 控件并没有指定其 **CommandName** 属性，符合触发 **OnItemCommand** 事件的条件；所以这个范例在触发事件时会执行 **dIA_ICMD** 事件程序。

由于 **DataList** 会引发事件，所以 **DataList** 控件必需被放置于 **<Form>** 标注之中。当事件触发时我们就可以透过 **e.CommandSource** 来取得引发事件的控件之参考，并判断其 **Id** 属性。若其 **Id** 属性为 **lbShow**，则将 **DataList Web** 控件的 **SelectedIndex** 属性指定为 **e.Item.ItemIndex**，**e.Item.ItemIndex** 表示被选到项目的 **Index** 值，所以该笔记录就会以 **SelectedItemTemplate** 来显示详细资料；若 **Id** 属性为 **lbClose**，则将 **DataList Web** 控件的 **SelectedIndex** 设为 **-1**，表示没有任何选项被选择。另外 **DataList** 设定 **SelectedIndex** 属性完毕后必须要再呼叫 **DataBind** 方法，让 **DataList Web** 控件重新再以新的设定来显示数据源中的数据。

数据的编辑

下列两个范例都是将 **DataList Web** 控件加入编辑数据的功能，第一个范例只用 **OnItemCommand** 事件，第二个范例综合 **OnEditCommand**、**OnCancelCommand**，以及 **OnUpdateCommand** 事件。使用者在选择一个项目展开 **SelectedItemTemplate** 样版显示详细资料后，若选择「编辑」选项时会显示 **EditItemTemplate** 样版，**EditItemTemplate** 样版是以 **TextBox** 来显示使用者资料，并且可以接受使用者的修改。修改完成后可以按「确定」执行将数据更新回数据源的动作，「放弃」可以回到显示使用者详细数据的选项：

范例一 只使用 **OnItemCommand** 事件

```
<%@Import Namespace=System.Data.ADO%>

<%@Import Namespace=System.Data%>

<Html>

<Form runat="Server">

<ASP:DataList Id="dlA" OnItemCommand="dlA_ICMD"

        GridLines="both" Runat="Server">

    <Template Name="ItemTemplate">

        <ASP:Image ImageUrl="ico1.gif" Runat="Server"/>

        姓名: <%#Container.DataItem("UserName")%>

        <ASP:LinkButton Id="lbShow" Text=">" Runat="Server"/>

    </Template>

    <Template Name="EditItemTemplate">

        <ASP:Image ImageUrl="ico1.gif" Runat="Server"/>

        姓名: <%#Container.DataItem("UserName")%><br>

        电话:<ASP:TextBox Id="T1"

Text=' <%#Container.DataItem("UserTel")%>'

        Runat="Server"/><br>
```



```
        住址:<ASP:TextBox Id="T2"
Text=' <%#Container.DataItem("UserAdd")%>'
Runat="Server"/><br>

        电邮:<ASP:TextBox Id="T3"
Text=' <%#Container.DataItem("UserEmail")%>'
Runat="Server"/><br>

        <ASP:LinkButton Id="lbCancel" Text="[放弃]" Runat="Server"/>

        <ASP:LinkButton Id="lbSubmit" Text="[确定]" Runat="Server"/><br>

    </Template>

    <Template Name="SelectedItemTemplate">

        <ASP:Image ImageUrl="icol.gif" Runat="Server"/>

        姓名: <%#Container.DataItem("UserName")%>

        <ASP:LinkButton Id="lbClose" Text="<" Runat="Server"/><br>

        电话: <%#Container.DataItem("UserTel")%><br>

        住址: <%#Container.DataItem("UserAdd")%><br>

        电邮: <%#Container.DataItem("UserEmail")%><br>

        <ASP:LinkButton Id="lbEdit" Text="[编辑]" Runat="Server"/><br>

    </Template>

</ASP:DataList>

</Form>
```

```

<Script Language="VB" Runat="Server">

Dim dscA As ADODatasetCommand=New ADODatasetCommand("Select * From
Members", _

    "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\InetPub\wwwroot\CR\Ch05\MyWeb.Mdb")

Dim dsDataSet As DataSet=New DataSet

Dim dtDataTable As DataTable

Sub Page_Load(Sender As Object, e As EventArgs)

    dscA.FillDataSet(dsDataSet, "Members")

    dtDataTable=dsDataSet.Tables("Members")

    dlA.DataSource=dtDataTable.DefaultView

    If Not Page.IsPostBack Then Page.DataBind()

End Sub

Sub dlA_ICMD(Sender As Object, e As DataListCommandEventArgs)

    If e.CommandSource.Id="lbShow" Then

        dlA.SelectedIndex=e.Item.ItemIndex

    ElseIf e.CommandSource.Id="lbClose" Then

        dlA.SelectedIndex=-1

```

```
ElseIf e.CommandSource.Id="lbEdit" Then
```

```
    dlA.EditItemIndex=e.Item.ItemIndex
```

```
ElseIf e.CommandSource.Id="lbCancel" Then
```

```
    dlA.EditItemIndex=-1
```

```
ElseIf e.CommandSource.Id="lbSubmit" Then
```

```
    Dim txtTemp As TextBox
```

```
    txtTemp=e.Item.FindControl("T1") '取回ListItem中名为T1的控件参
```

考

```
    dtDataTable.Rows(dlA.EditItemIndex)("UserTel")=txtTemp.Text
```

```
    txtTemp=e.Item.FindControl("T2") '取回ListItem中名为T2的控件参
```

考

```
    dtDataTable.Rows(dlA.EditItemIndex)("UserAdd")=txtTemp.Text
```

```
    txtTemp=e.Item.FindControl("T3") '取回ListItem中名为T3的控件参
```

考

```
    dtDataTable.Rows(dlA.EditItemIndex)("UserEmail")=txtTemp.Text
```

```
    dscA.Update(dsDataSet,"Members")
```

```
    dlA.EditItemIndex=-1
```

```
End If
```

```
dlA.DataBind()
```

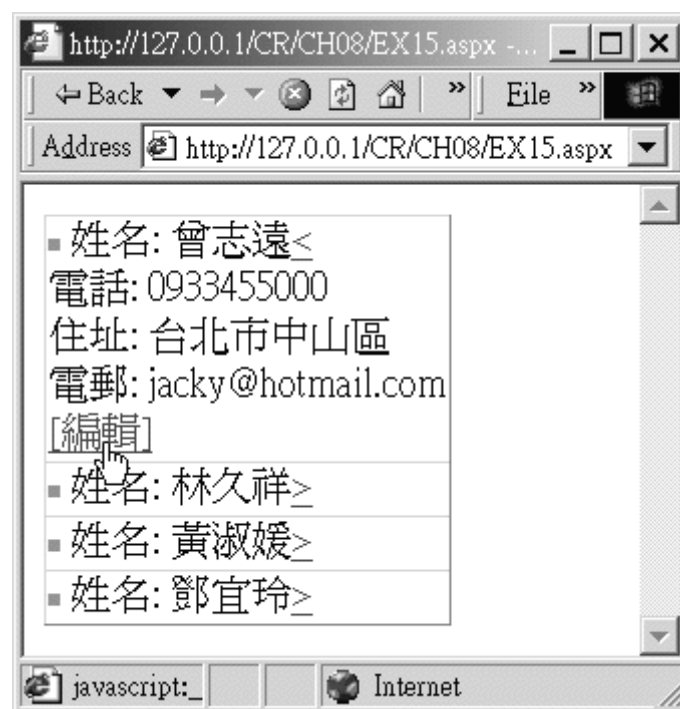
```
End Sub
```

```
</SCRIPT>
```

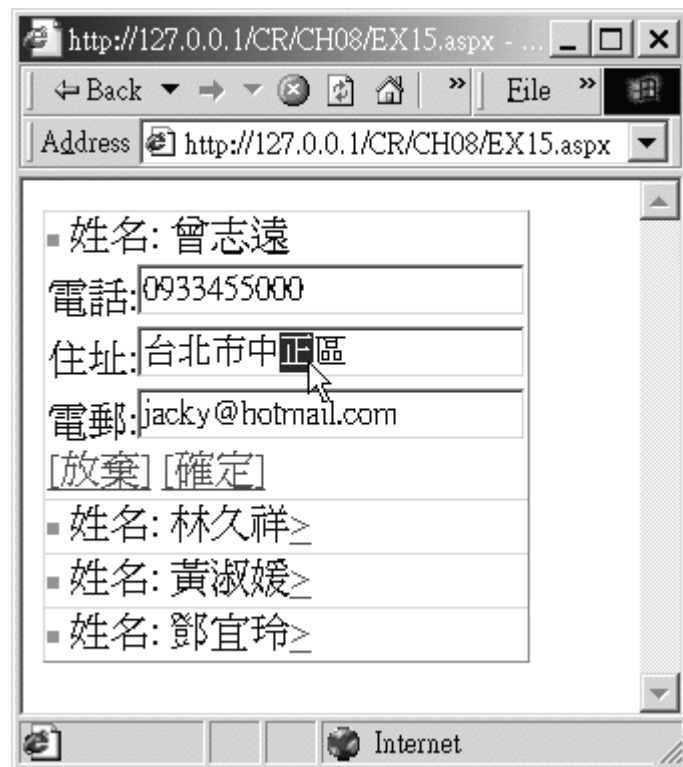
```
</Html>
```

由于我们要在许多程序中使用 **DataTable**、**DataSet** 以及 **DataSetCommand** 对象，所以我们将这些对象变量宣告在网页阶层的宣告区。程序开始执行时，先以 **ItemTemplate** 样版来显示资料。

待任意项目被选择后，便以 **SelectedItemTemplate** 样版来显示该项目，如下图所示：



选择「编辑」选项后，便以 **EditItemTemplate** 样版来显示所要编辑的记录；使用者可以在 **TextBox** 中编修数据，如下图所示：



待使用者将数据编辑完毕点选「确定」按钮时，我们就在 `dIA_ICMD` 事件程序中将使用者所作的修改更新回数据源；如下程序代码片段所示：

```
ElseIf e.CommandSource.Id="lbSubmit" Then
```

```
    Dim txtTemp As TextBox
```

```
    txtTemp=e.Item.FindControl("T1") '取回 ListItem 中名为 T1 的控件参
```

考

```
    dtDataTable.Rows(dIA.EditItemIndex)("UserTel")=txtTemp.Text
```

```

txtTemp=e.Item.FindControl("T2") '取回ListItem中名为 T2 的控件参
考

dtDataTable.Rows(dlA.EditItemIndex)("UserAdd")=txtTemp.Text

txtTemp=e.Item.FindControl("T3") '取回ListItem中名为 T3 的控件参
考

Then

dtDataTable.Rows(dlA.EditItemIndex)("UserEmail")=txtTemp.Text

dscA.Update(dsDataSet,"Members")

dlA.EditItemIndex=-1

End If

```

上述程序代码片段宣告一个 **TextBox** 型态的对象变量，用来存放 **ListItem** 中 **TextBox** 控件的参考；此时我们就可以用 **ListItem** 对象的 **FindControl()** 方法来取得指定的控件参考，**FindControl** 只要传入想要传回对象参考的 **Id** 属性即可。最后只要呼叫 **DataSetCommand** 对象的 **Update** 方法，就可以将使用者所作的修改更新回原来的数据源。所以选择「确定」回到 **SelectedItemTemplate** 样版的模式来显示数据时，就可以看到资料已经被更新了；如下图所示：



范例二 综合相关事件

```
<%@Import Namespace=System.Data.ADO%>

<%@Import Namespace=System.Data%>

<Html>

<Form runat="Server">

<ASP:DataList Id="d1A" OnItemCommand="d1A_ICmd"

OnEditCommand="d1A_ECmd"

OnCancelCommand="d1A_CCmd" GridLines="both"

Runat="Server">

<Template Name="ItemTemplate">
```

```

        <ASP:Image ImageUrl="ico1.gif" Runat="Server"/>

        姓名: <%#Container.DataItem("UserName")%>

        <ASP:LinkButton Id="lbShow" Text=">" CommandName="Show"
Runat="Server"/>

    </Template>

    <Template Name="EditItemTemplate">

        <ASP:Image ImageUrl="ico1.gif" Runat="Server"/>

        姓名: <%#Container.DataItem("UserName")%><br>

        电话:<ASP:TextBox Id="T1"
Text=' <%#Container.DataItem("UserTel")%>'

                Runat="Server"/><br>

        住址:<ASP:TextBox Id="T2"
Text=' <%#Container.DataItem("UserAdd")%>'

                Runat="Server"/><br>

        电邮:<ASP:TextBox Id="T3"
Text=' <%#Container.DataItem("UserEMail")%>'

                Runat="Server"/><br>

        <ASP:LinkButton Id="lbCancel" Text="[放弃]" CommandName="Cancel"

                Runat="Server"/>

        <ASP:LinkButton Id="lbSubmit" Text="[确定]" CommandName="Submit"

```



```

        Runat="Server"/><br>

</Template>

<Template Name="SelectedItemTemplate">

    <ASP:Image ImageUrl="icol.gif" Runat="Server"/>

    姓名: <%#Container.DataItem("UserName")%>

    <ASP:LinkButton Id="lbClose" Text="<" CommandName="Close"
Runat="Server"/><br>

    电话: <%#Container.DataItem("UserTel")%><br>

    住址: <%#Container.DataItem("UserAdd")%><br>

    电邮: <%#Container.DataItem("UserEmail")%><br>

    <ASP:LinkButton Id="lbEdit" Text="[编辑]" CommandName="Edit"
Runat="Server"/><br>

</Template>

</ASP:DataList>

</Form>

<Script Language="VB" Runat="Server">

Dim dscA As ADODatasetCommand=New ADODatasetCommand("Select * From
Members", _

```

```

        "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\InetPub\wwwroot\CR\Ch05\MyWeb.Mdb")

Dim dsDataSet As DataSet=New DataSet

Dim dtDataTable As DataTable

Sub Page_Load(Sender As Object, e As EventArgs)

    dscA.FillDataSet(dsDataSet, "Members")

    dtDataTable=dsDataSet.Tables("Members")

    dlA.DataSource=dtDataTable.DefaultView

    If Not Page.IsPostBack Then Page.DataBind()

End Sub

Sub dlA_ICmd(Sender As Object, e As DataListCommandEventArgs)

    If e.CommandSource.CommandName="Show" Then

        dlA.SelectedIndex=e.Item.ItemIndex

    ElseIf e.CommandSource.CommandName="Close" Then

        dlA.SelectedIndex=-1

    ElseIf e.CommandSource.CommandName="Submit" Then

        Dim txtTemp As TextBox

        txtTemp=e.Item.FindControl("T1") ' 取回 ListItem 中名为 T1 的控件参
考

```

```
dtDataTable.Rows(dlA.EditItemIndex)("UserTel")=txtTemp.Text  
txtTemp=e.Item.FindControl("T2") '取回ListItem中名为T2的控件参  
考
```

```
dtDataTable.Rows(dlA.EditItemIndex)("UserAdd")=txtTemp.Text  
txtTemp=e.Item.FindControl("T3") '取回ListItem中名为T3的控件参  
考
```

```
dtDataTable.Rows(dlA.EditItemIndex)("UserEmail")=txtTemp.Text  
dscA.Update(dsDataSet,"Members")  
  
dlA.EditItemIndex=-1  
  
End If  
  
dlA.DataBind()  
  
End Sub
```

```
Sub dlA_ECmd(Sender As Object, e As DataListCommandEventArgs)  
  
dlA.EditItemIndex=e.Item.ItemIndex  
  
dlA.DataBind()  
  
End Sub
```

```
Sub dlA_CCmd(Sender As Object, e As DataListCommandEventArgs)  
  
dlA.EditItemIndex=-1  
  
dlA.DataBind()  
  
End Sub
```

```
</SCRIPT>
```

```
</Html>
```

上述范例二的执行结果和范例一完全一样。

DataGrid Web 控件

DataGrid Web 控件是三个 **Web** 数据显示控件之中功能最强的。我们在撰写动态网页的时候，常需要将数据以不同的风格呈现出来；**DataGrid Web** 控件和上述介绍的 **Repeater Web** 控件及 **DataList Web** 控件都可以办到。但如果所要呈现的数据量非常庞大，而需要将这些数据分页展示的话，那就要靠 **DataGrid Web** 控件了；**DataGrid Web** 控件除了支持分页的功能外，也可以让使用者编辑数据。其使用语法为：

```
<ASP:DataGrid
```

```
    Id="被程序代码所控制的名称"
```

```
    Runat="Server"
```

```
    DataSource='<##数据系结叙述%>'
```

```
    AllowPaging="True | False"
```

```
    AllowSorting="True | False"
```

```
    AutoGenerateColumns="True | False"
```

```
    BackImageUrl="url"
```

CellPadding="像素"

CellSpacing="像素"

DataKeyField="主键字段"

GridLines="None | Horizontal | Vertical | Both"

HorizontalAlign="Center | Justify | Left | NotSet | Right"

PagedDataSource

PageSize="ItemCount"

ShowFooter="True | False"

ShowHeader="True | False"

VirtualItemCount="ItemCount"

AlternatingItemStyle-Property="value"

EditItemStyle-Property="value"

FooterStyle-Property="value"

HeaderStyle-Property="value"

ItemStyle-Property="value"

PagerStyle-Property="value"

SelectedItemStyle-Property="value"

OnCancelCommand="事件程序"

```
OnDeleteCommand="事件程序"

OnEditCommand="事件程序"

OnItemCommand="事件程序"

OnItemCreated="事件程序"

OnPageIndexChanged="事件程序"

OnSortCommand="事件程序"

OnUpdateCommand="事件程序"

/>
```

或

```
<ASP:DataGrid

    Id="被程序代码所控制的名称"

    Runat="Server"

    AutoGenerateColumns="False"

    DataSource='<%# DataBindingExpression %>'

    其它属性设定...

>

    <Property Name="Columns">

        <ASP:BoundColumn/>

        <ASP>EditCommandColumn/>
```

```

        <ASP:HyperlinkColumn/>

        <ASP:TemplateColumn>

            样版设定...

        </ASP:TemplateColumn>

    </Property>

</ASP:DataGrid>

```

DataGrid Web 控件常用的属性如下表所示：

属性	说明
AllowCustomPaging	设定是否允许自订分页。
AllowPaging	设定是否允许分页。
AllowSorting	设定是否允许排序数据。
AutoGenerateColumns	设定是否要自动产生数据源中每个字段的数据，预设值为 True。
BackImageUrl	设定表格背景所要显示的图形。
CellPadding	储存格与表格边框的距离。
CellSpacing	储存格和储存格边框的距离。
Columns	传回控件中所显示的字段数，只读。
CurrentPageIndex	设定控制项目目前所在的数据页数，只能用程序设定。

DataKeyField	设定在数据源中为主键的字段。
DataSource	设定数据系结所要使用的数据源。
EditItemIndex	设定要被编辑的字段名称。本属性设为 -1 可放弃编辑。
GridLines	设定是否要显示网格线。本属性在 RepeatLayout 属性设为 Table 时才有效。
HorizontalAlign	设定水平对齐的方式。
Items	DataListItem 的集合对象。本对象只包含和数据源系结的 Item，也就是说不包含 Header、Footer 及 Separator 样版。
PageCount	传回总共有几页，只读。
PageSize	设定每页所要显示的纪录笔数。
SelectedIndex	设定哪一列被点选到。设定此属性时，该列会以 Selected 样版的样式来显示。
SelectedItem	传回被点选到的 Item。
ShowFooter	设定是否要显示脚注（Footer），True/False。
ShowHeader	设定是否要显示表头（Header），True/False。
VirtualItemCount	设定所要显示的纪录笔数。如果 AllowCustomPaging 属性为 True，本属性则用来设定总共所要显示的页数；如果为 False，本属性则用来传回总共的页数。

DataGrid Web 控件所支持的样版如下表所示：

模板名称	说明
HeaderTemplate	数据表头的样式。
ItemTemplate	呈现数据的样式。本样版为必要样版，不可省略。
EditItem	编辑数据的样版。
FooterTemplate	数据表尾的样式。
Pager	数据分页的样式。

DataGrid Web 控件也支持许多样式对象，可以让我们可以灵活的自订其显示外观，如下表所示：

样式对象	样式类别	说明
AlternatingItemStyle	TableItem	每一个交替项目所要显示的样式。
EditItemStyle	TableItem	项目在被编辑时所要显示的样式。
FooterStyle	TableItem	脚注所要显示的样式。
HeaderStyle	TableItem	标头所要显示的样式。
ItemStyle	基础	每一个项目所要显示的样式。
PagerStyle	DataGridPager	分页的样式。
SelectedItemStyle	TableItem	项目在被选择时所要显示的样式。

DataGridPager 样式

DataGridPager 样式最主要用来设定 DataGrid Web 控件的分页样式，这些样式如下表所示：

属性	说明
PagerStyle-Mode	设定分页方式，NextPrev、NumericPages。
PagerStyle-NextPageText	设定下一页的文字。
PagerStyle-PageButtonCount	设定分页按钮的文字风格。
PagerStyle-Position	设定分页的地址。（Bottom、Top、TopAndBottom）
PagerStyle-PrevPageText	设定上一页的文字。
PagerStyle-Visible	设定是否显示，True/False。

DataGrid Web 控件支持八个事件，如下表所示：

事件名称	说明
OnCancelCommand	当在字段中的 Button 或 LinkButton 控件触发事件时，如果控件的 CommandName 属性为 Cancel 时，则触发本事件。
OnDeleteCommand	当在字段中的 Button 或 LinkButton 控件触发事件时，如果控件的 CommandName 属性为 Delete 时，则触发本事件。

OnEditCommand	当在字段中的 Button 或 LinkButton 控件触发事件时，如果控件的 CommandName 属性为 Edit 时，则触发本事件。
OnItemCommand	当在字段中的 Button 或 LinkButton 控件触发事件时，如果 CommandName 属性的内容不是 Edit、Cancel、Delete 或 Update 时即触发本事件。
OnItemCreated	当 DataList 中的每一个项目被产生时触发。
OnPageIndexChanged	当不同的数据页被选取时便触发。
OnSortCommand	当使用者选择要排序的字段时，即触发本事件。本事件必须将 DataGrid 的 AllowSorting 属性设为 True 才会触发。
OnUpdateCommand	当在字段中的 Button 或 LinkButton 控件触发事件时，如果控件的 CommandName 属性为 Update 时，则触发本事件。

Cancel、Delete、Edit、Item、Sort、Update 事件的宣告

OnCancelCommand、OnDeleteCommand、OnEditCommand、OnItemCommand、

OnSortCommand 以及 OnUpdateCommand 这六个事件程序的宣告语法，如下所示：

```
Sub 事件程序名称(Sender As Object, e As DataGridCommandEventArgs)

    ...

End Sub
```

其中参数 **e** 的属性如下表所示：

属性	说明
e.CommandSource	触发事件之对象的参考。
e.Item	DataItem 对象的参考。

OnItemCreated 事件的宣告

OnItemCreated 这个事件程序的宣告语法，如下所示：

```
Sub OnItemCreated(Sender As Object, e As DataGridItemCreatedEventArgs)

    ...

End Sub
```

其中参数 **e** 的属性如下表所示：

属性	说明
e.Item	DataItem 对象的参考。

OnPageIndexChanged 事件的宣告

OnPageIndexChanged 这个事件程序的宣告语法，如下所示：

```
Sub OnPageIndexChanged(Sender As Object, e As  
DataGridPageChangedEventArgs)  
  
    ...  
  
End Sub
```

其中参数 **e** 的属性如下表所示：

属性	说明
e.CommandSource	触发事件之对象的参考。
e.NewPageIndex	新的 PageIndex 值。
e.OldPageIndex	原来的 PageIndex 值。

DataGrid Web 控件的基础应用

下列范例利用 **DataGrid Web** 控件显示 **CH08** 目录中的 **Members** 数据表，总共有 **37** 笔记录：

```
<%@Import Namespace=System.Data.ADO%>  
  
<%@Import Namespace=System.Data%>  
  
<!--#Include File="GetTable.inc"-->  
  
<Html>  
  
<ASP:DataGrid Id="dgA" Runat="Server"/>
```

```
<Script Language="VB" Runat="Server">  
  
Sub Page_Load(Sender As Object, e As EventArgs)  
  
Dim dtDataTable As DataTable=GetTable("CH08\MyWeb.mdb", "Members")  
  
    dgA.DataSource=dtDataTable.DefaultView  
  
    Page.DataBind()  
  
End Sub  
  
</SCRIPT>  
  
</Html>
```

The screenshot shows a Microsoft Internet Explorer window with the address bar displaying `http://127.0.0.1/CR/CH08/EX17.aspx`. The main content area displays a table with 6 columns: `UserId`, `UserPwd`, `UserName`, `UserTel`, `UserAdd`, and `UserEmail`. The table contains 9 rows of user data. The status bar at the bottom shows 'Done' and 'Internet'.

UserId	UserPwd	UserName	UserTel	UserAdd	UserEmail
jacky	5678	曾志遠	0933455000	台北市中正區	jacky@hotmail.com
jolin	abcd	林久翔	0939507000	桃園縣桃園市	jolin@hotmail.com
tina	6789	黃淑媛	0920086000	台北縣三重市	tina@hotmail.com
elvira	wxyz	鄧宜玲	0932123000	台北縣中和市	elvira@hotmail.com
vivi	1252	邱苑玲	0225810000	台北市中正區	vivi@hotmail.com
mary	fh43	饒惠玲	0939694000	台北縣板橋市	mart@hotmail.com
jonny	dkgf	李維華	0921951000	台北縣林口鄉	johhy@hotmail.com
kevin	23dg	鄭博文	0227908000	台北市內湖區	kevin@hotmail.com
steven	1985	廖富熒	0222013000	台北縣新莊市	steven@hotmail.com

分页功能

由于将所有的 37 笔数据显示出来显的太杂乱，所以我们将 **DataGrid Web** 控件的分页功能打开。

只要在 **DataGrid Web** 控件的宣告中加入 `AllowPaging="True"`，表示要将资料分页展示；而宣告

`PageSize="5"` 表示每页要显示 5 笔记录。另外在分页的时候，**DataGrid Web** 控件的

`CurrentPageIndex` 属性会被改变，所以在点选分页按钮的时候会引发 `OnPageIndexChanged`

事件。如果要正确的支持分页的功能，我们就必须利用这个事件程序将 **DataGrid Web** 控件以新

的 `CurrentPageIndex` 属性再和数据源系结一次。如下所示：

```

<%@Import Namespace=System.Data.ADO%>

<%@Import Namespace=System.Data%>

<!--#Include File="GetTable.inc"-->

<Html>

<Form Runat="Server">

<ASP:DataGrid Id="dgA" Runat="Server"

AllowPaging="True" PageSize="5" OnPageIndexChanged="dgA_PageChg"/>

</Form>

<Script Language="VB" Runat="Server">

Sub Page_Load(Sender As Object, e As EventArgs)

    If Page.IsPostBack=False Then

        Dim dtDataTable As DataTable=GetTable("CH08\MyWeb.mdb", "Members")

        dgA.DataSource=dtDataTable.DefaultView

        Page.DataBind()

    End If

End Sub

Sub dgA_PageChg(Sender As Object, e As DataGridPageChangedEventArgs)

    Dim dtDataTable As DataTable=GetTable("CH08\MyWeb.mdb", "Members")

    dgA.DataSource=dtDataTable.DefaultView

```



```
Page.DataBind()
```

```
End Sub
```

```
</SCRIPT>
```

```
</Html>
```

UserId	UserPwd	UserName	UserTel	UserAdd	UserEmail
jacky	5678	曾志遠	0933455000	台北市中正區	jacky@hotmail.com
jolin	abcd	林久翔	0939507000	桃園縣桃園市	jolin@hotmail.com
tina	6789	黃淑媛	0920086000	台北縣三重市	tina@hotmail.com
elvira	wxyz	鄧宜玲	0932123000	台北縣中和市	elvira@hotmail.com
vivi	1252	邱苑玲	0225810000	台北市中正區	vivi@hotmail.com



↓ OnPageIndexChanged

UserId	UserPwd	UserName	UserTel	UserAdd	UserEmail
mary	fh43	饒惠玲	0939694000	台北縣板橋市	mart@hotmail.com
jonny	dkgf	李維華	0921951000	台北縣林口鄉	johhy@hotmail.com
kevin	23dg	鄭博文	0227908000	台北市內湖區	kevin@hotmail.com
steven	1985	廖富熒	0222013000	台北縣新莊市	steven@hotmail.com
gigi	kdji	鄭怡菁	0229354000	台北市福興路	gigi@hotmail.com



样式的设定

DataGrid Web 控件也支持一些样式对象，这些样式对象的设定方式我们在第七章已经讨论过，

所以我们不再深入讨论。下列程序代码片段设定了 DataGrid Web 控件的标头样式以及分页的样

式：

```
<ASP:DataGrid Id="dgA" AllowPaging="True" PageSize="5"
    OnPageIndexChanged="dgA_PageChg" Runat="Server"
    PagerStyle-Mode="NumericPages" BorderColor="#808080"
    HeaderStyle-Font-Names="Courier New"
HeaderStyle-Font-Bold="True"
    HeaderStyle-BackColor="#D1DCEB"
HeaderStyle-ForeColor="White"/>
```

UserId	UserPwd	UserName	UserTel	UserAdd	UserEmail		
jacky	5678	曾志遠	0933455000	台北市中正區	jacky@hotmail.com		
jolin	abcd	林久翔	0939507000	桃園縣桃園市	jolin@hotmail.com		
tina	6789	黃淑媛	0920086000	台北縣三重市	tina@hotmail.com		
elvira	wxyz	鄧宜玲	0932123000	台北縣中和市	elvira@hotmail.com		
vivi	1252	邱苑玲	0225810000	台北市中正區	vivi@hotmail.com		
1	2	3	4	5	6	7	8

自订字段内容

因为 **DataGrid Web** 控件的 **AutoGenerateColumn** 属性预设设为 **True**，表示会自动产生数据源中所有的字段。如果我们想自订 **DataGrid Web** 控制所要显示的字段，只要将 **AutoGenerateColumn** 属性设为 **False**，并设定 **Columns** 属性即可。其设定语法如下所示：

```
<Property Name="Columns">

    <ASP:BoundColumn/>

    <ASP:ButtonColumn/>

    <ASP:EditCommandColumn/>

    <ASP:HyperlinkColumn/>

    <ASP:TemplateColumn>

        样版设定...

    </ASP:TemplateColumn>

</Property>
```

DataGrid Web 控件允许我们定义的字段，如下表所示：

字段型态	说明
BoundColumn	字段内容以 Label 的方式呈现。
ButtonColumn	字段内容以超级链接或是按钮的方式呈现。

EditCommandColumn	提供数据编修的命令，不呈现字段数据。
HyperLinkColumn	字段内容以超级链接的方式呈现。
TemplateColumn	字段内容以自订样版的方式呈现。

下列程序代码片段展示了这几种字段：

```
<ASP:DataGrid Id="dgA" AllowPaging="True" PageSize="5"
    OnPageIndexChanged="dgA_PageChg" Runat="Server"
    PagerStyle-Mode="NumericPages" BorderColor="#808080"
    HeaderStyle-Font-Names="Courier New"
    HeaderStyle-BackColor="#D1DCEB"
    AutoGenerateColumns="False">
    <Property Name="Columns">
        <ASP:BoundColumn
            HeaderText="姓名"
            DataField="UserName"/>
        <ASP:ButtonColumn
            HeaderText="电话"
            ButtonType="PushButton"
            DataTextField="UserTel"/>
```

```
<ASP:HyperlinkColumn
```

```
    HeaderText="电邮"
```

```
    DataNavigateUrlField="UserEmail"
```

```
    DataTextField="UserEmail"/>
```

```
<ASP:TemplateColumn
```

```
    HeaderText="住址">
```

```
    <Template Name="ItemTemplate">
```

```
        <ASP:Image ImageUrl="icol.gif" Runat="Server"/>
```

```
        <%#Container.DataItem("UserAdd")%>
```

```
    </Template>
```

```
</ASP:TemplateColumn>
```

```
<ASP:EditCommandColumn
```

```
    HeaderText="编辑"
```

```
    ButtonType="PushButton"
```

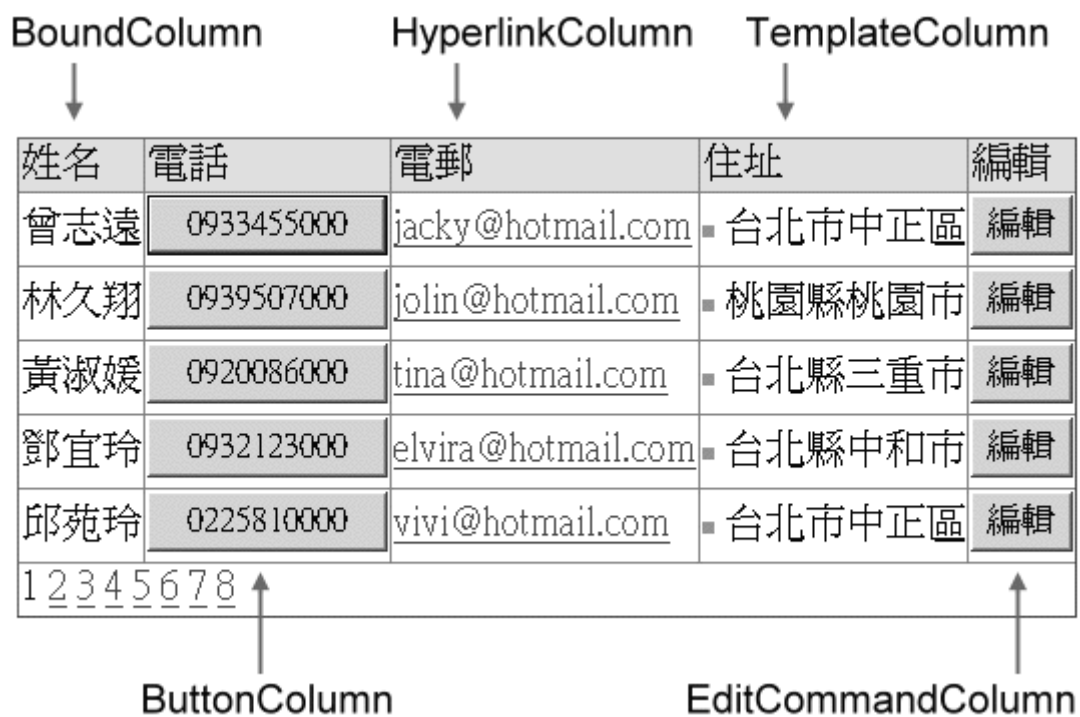
```
    CancelText="放弃"
```

```
    EditText="编辑"
```

```
    UpdateText="确定"/>
```

```
</Property>
```

```
</ASP:DataGrid>
```



其中这些字段的共同基础属性如下表所示：

属性	说明
FooterText	在字段底部显示的文字。
HeaderImageUrl	用来代替字段标题的影像文件地址。
HeaderText	字段标题所要显示的文字。
Owner	传回字段所属 DataGrid 的参考。
SortField	当使用者指明要以本字段来排序时，该字段在数据源的名称。

State	传回字段的状况。
Visible	设定是否要显示字段，True/False。

这些字段也支持许多样式对象，可以让我们可以灵活的自订其显示外观，如下表所示：

样式对象	样式类别	说明
FooterStyle	TableItem	脚注所要显示的样式。
HeaderStyle	TableItem	标头所要显示的样式。
ItemStyle	基础	每一个项目所要显示的样式。

BoundColumn

BoundColumn 最主要的功能是利用 **Label** 来显示数据源中的一个字段内容，其使用语法如下所

示：

```
<ASP:BoundColumn
```

```
    DataField="DataSourceField"
```

```
    DataFormatString="FormatString"
```

```
    FooterText="FooterText"
```

```
    HeaderImageUrl="url"
```

```
HeaderText="HeaderText"

ReadOnly="True | False"

SortField="DataSourceFieldToSortBy"

Visible="True | False"

FooterStyle-Property="value"

HeaderStyle-Property="value"

ItemStyle-Property="value"

/>
```

其中除了共同基础属性以及样式对象外，常用的属性如下表所示：

属性	说明
DataField	所要显示的资料源字段名称。
DataFormatString	所要显示的资料格式。
ReadOnly	设定字段是否为只读，True/False。

下列程序代码片段以 **BoundColumn** 来显示字段内容：

```
<ASP:DataGrid Id="dgA" AllowPaging="True" PageSize="5"

    OnPageIndexChanged="dgA_PageChg" Runat="Server"

    PagerStyle-Mode="NumericPages" BorderColor="#808080"
```



```
HeaderStyle-Font-Names="Courier New"

HeaderStyle-BackColor="#D1DCEB"

AutoGenerateColumns="False">

<Property Name="Columns">

    <ASP:BoundColumn

HeaderText="姓名"

HeaderStyle-Font-Bold="True"

HeaderStyle-HorizontalAlign="Center"

DataField="UserName"/>

    <ASP:BoundColumn

HeaderText="电话"

HeaderStyle-Font-Bold="True"

HeaderStyle-HorizontalAlign="Center"

DataField="UserTel"/>

    <ASP:BoundColumn

HeaderText="住址"

HeaderStyle-Font-Bold="True"

HeaderStyle-HorizontalAlign="Center"

DataField="UserAdd"/>
```

```
</Property>
```

```
</ASP:DataGrid>
```



ButtonColumn

BoundColumn 最主要的功能是利用 **LinkButton** 或 **PushButton** 来显示数据源中的一个字段内容，

并且可以触发 **DataGrid Web** 控件的事件。其使用语法如下所示：

```
<ASP:ButtonColumn
```

```
ButtonType="LinkButton | PushButton"
```

```
CommandName="命令名称"

DataTextField="DataSourceField"

DataTextFormatString="FormatString"

FooterText="FooterText"

HeaderImageUrl="url"

HeaderText="HeaderText"

ReadOnly="True | False"

SortField="DataSourceFieldToSortBy"

Text="ButtonCaption"

Visible="True | False"

/>
```

其中除了共同基础属性以及样式对象外，常用的属性如下表所示：

属性	说明
ButtonType	所要使用的按钮种类，预设 为 LinkButton。
CommandName	命令名称。
DataTextField	设定控件上所要显示的资料源字段名称。
DataTextFormatString	所要显示的资料格式。
Text	设定控件上所要显示的文字，若指定 DataTextField 属性，则本属性无

	效。
--	----

下列程序代码范例以 **BoundColumn** 来显示字段内容，并且显示使用者点选了哪一笔记录：

```
<%@Import Namespace=System.Data.ADO%>

<%@Import Namespace=System.Data%>

<!--#Include File="GetTable.inc"-->

<Html>

<Form Runat="Server">

<ASP:DataGrid Id="dgA" AllowPaging="True" PageSize="5"

    OnPageIndexChanged="dgA_PageChg" Runat="Server"

    PagerStyle-Mode="NumericPages" BorderColor="#808080"

    HeaderStyle-Font-Names="Courier New"

HeaderStyle-BackColor="#D1DCEB"

    HeaderStyle-Font-Bold="True" HeaderStyle-HorizontalAlign="Center"

    AutoGenerateColumns="False"

    OnItemCommand="dgA_ICmd">

<Property Name="Columns">

    <ASP: ButtonColumn

        HeaderText="姓名"
```

```

        DataTextField="UserName"/>

        <ASP:BoundColumn

        HeaderText="电话"

        DataField="UserTel"/>

        <ASP:BoundColumn

        HeaderText="住址"

        DataField="UserAdd"/>

    </Property>

</ASP:DataGrid>

</Form>

<ASP:Label Id="Label1" Runat="Server"/>

<Script Language="VB" Runat="Server">

Sub Page_Load(Sender As Object, e As EventArgs)

    If Page.IsPostBack=False Then

        Dim dtDataTable As DataTable=GetTable("CH08\MyWeb.mdb", "Members")

        dgA.DataSource=dtDataTable.DefaultView

        Page.DataBind()

    End If

    Label1.Text="您目前没有点选任何记录."

```

```
End Sub
```

```
Sub dgA_PageChg(Sender As Object, e As DataGridPageChangedEventArgs)
```

```
    Dim dtDataTable As DataTable=GetTable("CH08\MyWeb.mdb", "Members")
```

```
    dgA.DataSource=dtDataTable.DefaultView
```

```
    Page.DataBind()
```

```
End Sub
```

```
Sub dgA_ICmd(Sender As Object, e As DataGridCommandEventArgs)
```

```
    Dim shtRow As Short= e.Item.ItemIndex+1
```

```
    If shtRow<>0 Then
```

```
        Label1.Text="您点选了第 " & shtRow.ToString() & _
```

```
        " 个字段的数据, 这<br>笔数据在数据源中是第 "
```

```
        Label1.Text+=((dgA.CurrentPageIndex * dgA.PageSize) +
```

```
shtRow).ToString & _
```

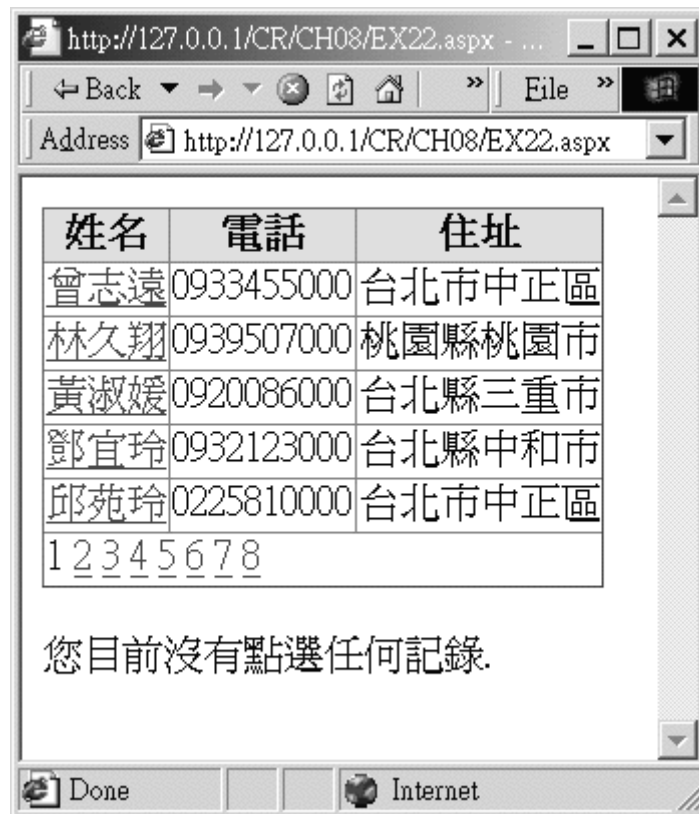
```
        "笔记录."
```

```
    End If
```

```
End Sub
```

```
</SCRIPT>
```

```
</Html>
```



上述范例我们指定 DataGrid Web 控件的 OnItemCommand 属性为 dgA_ICmd，表示按下显示使用者名称的 ButtonColumn 时会触发 dgA_CCmd 事件，并且执行 dgA_Icmd 事件程序；如下程序代码片段所示：

```
Sub dgA_ICmd(Sender As Object, e As DataGridCommandEventArgs)

    Dim shtRow As Short= e.Item.ItemIndex+1

    If shtRow<>0 Then

        Label1.Text="您点选了第 " & shtRow.ToString() & _
```

```
        " 个字段的数据，这<br>笔数据在数据源中是第 "
```

```
        Label11.Text+=((dgA.CurrentPageIndex * dgA.PageSize) +
```

```
shtRow).ToString & _
```

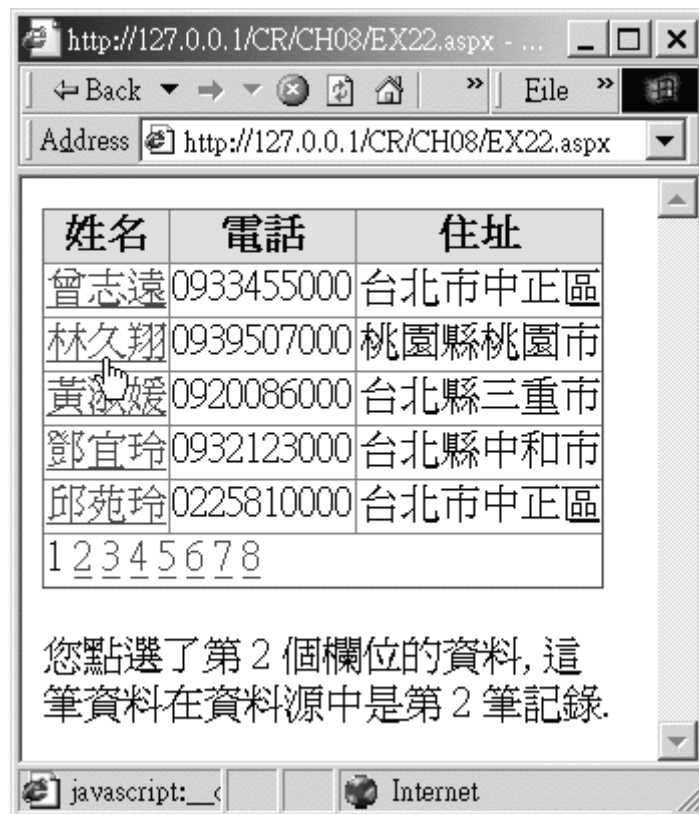
```
        "笔记录."
```

```
    End If
```

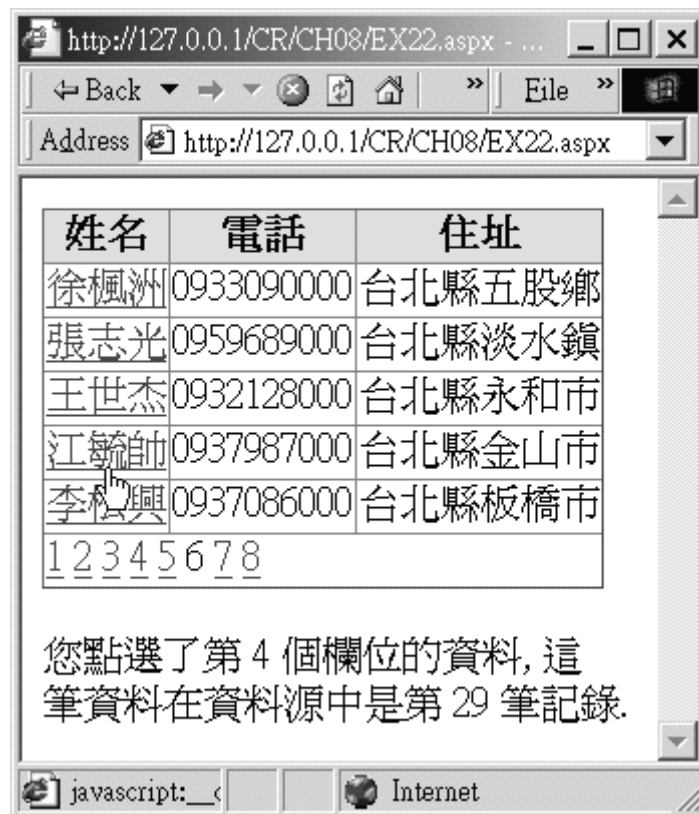
```
End Sub
```

在这个事件程序中，我们利用 **e.Item.ItemIndex** 属性取得使用者点选了第几个项目的 **Index** 值；

由于 **Index** 由 0 开始，所以我们加上 1 表示点选了第几个项目。如果没有任何项目被点选，则 **e.Item.ItemIndex** 属性传回 -1。由于变量 **shtRow** 为 **e.Item.ItemIndex** 加 1 的结果，所以我们藉由判断变量 **shtRow** 是否为 0 来得知使用者是否有选择选项。



因为 `e.Item.ItemIndex` 属性传回的是使用者在 `DataGrid Web` 控件上所点选的项目，并不是记录在数据源的地址；所以我们透过计算目前的页数索引乘以每页的纪录笔数后，再加上目前所在的字段即可得到该笔数据在数据源中的实际地址。所以使用者若点选了第六页的第四字段，表达式则为 $(5*5)+4$ ，结果 29 就是记录在数据源的实际顺序。



若要取得该笔记录在数据源中的索引值, 由于 **Index** 值是由 0 开始计数, 所以只要再减去 1 即可。

HyperlinkColumn

HyperlinkColumn 最主要的功能是以超级链接来显示资料源中的一个字段内容, 并可以指定使用

者点选该项目时所开启的地址。其使用语法如下所示:

```
<ASP:HyperlinkColumn
```

```
    DataNavigateUrlField="DataSourceField"
```

```
    DataNavigateUrlFormatString="FormatExpression"
```

```
    DataTextField="DataSourceField"
```

```
DataTextFormatString="FormatExpression"

FooterText="FooterText"

HeaderImageUrl="url"

HeaderText="HeaderText"

NavigateUrl="url"

ReadOnly="True | False"

SortField="DataSourceFieldToSortBy"

Target="window"

Text="HyperlinkText"

Visible="True | False"

/>
```

其中除了共同基础属性以及样式对象外，常用的属性如下表所示：

属性	说明
DataNavigateUrlField	设定要以哪个字段值来当成超级链接的目标地址。
DataNavigateUrlFormatString	设定要开启连结的格式字符串。
DataTextField	设定控件上所要显示的资料源字段名称。
DataTextFormatString	所要显示的资料格式。
NavigateUrl	设定要连结的 URL 地址。若指定了 DataNavigateUrlField 属性，

	则本属性无效。
Target	设定连结所要开启的目标。
Text	设定控件上所要显示的文字，若指定 DataTextField 属性，则本属性无效。

下列程序代码范例以 **Hyperlink Column** 来显示使用者电子邮件信箱字段，若使用者点选了一笔记录，则会开启邮件编辑软件：

```

<%@Import Namespace=System.Data.ADO%>

<%@Import Namespace=System.Data%>

<!--#Include File="GetTable.inc"-->

<Html>

<Form Runat="Server">

<ASP:DataGrid Id="dgA" AllowPaging="True" PageSize="5"

    OnPageIndexChanged="dgA_PageChg" Runat="Server"

    PagerStyle-Mode="NumericPages" BorderColor="#808080"

    HeaderStyle-Font-Names="Courier New"

HeaderStyle-BackColor="#D1DCEB"

    HeaderStyle-Font-Bold="True"

HeaderStyle-HorizontalAlign="Center"

```

```
        AutoGenerateColumns="False">

        <Property Name="Columns">

            <ASP:BoundColumn

                HeaderText="姓名"

                DataField="UserName"/>

            <ASP:BoundColumn

                HeaderText="电话"

                DataField="UserTel"/>

            <ASP:HyperlinkColumn

                HeaderText="电邮"

                DataNavigateUrlField="UserEmail"

                DataTextField="UserEmail"

                DataNavigateUrlFormatString="mailto:{0}" />

        </Property>

    </ASP:DataGrid>

</Form>

<Script Language="VB" Runat="Server">

Sub Page_Load(Sender As Object, e As EventArgs)

    If Page.IsPostBack=False Then
```

```

        Dim dtDataTable As DataTable=GetTable("CH08\MyWeb.mdb", "Members")

        dgA.DataSource=dtDataTable.DefaultView

        Page.DataBind()

    End If

End Sub

Sub dgA_PageChg(Sender As Object, e As DataGridPageChangedEventArgs)

    Dim dtDataTable As DataTable=GetTable("CH08\MyWeb.mdb", "Members")

    dgA.DataSource=dtDataTable.DefaultView

    Page.DataBind()

End Sub

</SCRIPT>

</Html>

```

上述范例我们以 **UserEmail** 字段做为显示以及欲连结的目标。由于我们希望使用者点选这个字段中的任何一个项目时，可以开启 **Outlook** 来进行邮件的编辑；所以我们必须在使用者的电子邮件信箱地址前将上「mailto:」。要达到这个目的，可以设定 **DataNavigateUrlFormatString** 属性；如下程序代码片段所示：

```
DataNavigateUrlFormatString="mailto: {0}"
```

其中「{0}」会被 **DataNavigateUrlField** 的内容所取代，所以执行结果如下：

姓名	電話	電郵
曾志遠	0933455000	jacky@hotmail.com
林久翔	0939507000	jolin@hotmail.com
黃淑媛	0920086000	tina@hotmail.com
鄧宜玲	0932123000	elvira@hotmail.com
邱苑玲	0225810000	vivi@hotmail.com
1	2	3
4	5	6
7	8	



若超级链接的内容是某个网址，只要将 `NavigateUrlFormatString` 属性的内容设定为「`http://{0}`」

即可。

EditCommandColumn

`EditCommandColumn` 最主要的功能是利用 `LinkButton` 或 `PushButton` 来下达编辑资料的命令，

并且可以触发 `DataGrid Web` 控件的事件。`EditCommandColumn` 进入编辑模式的时候会产生

`TextBox` 让使用者编辑，也可以和 `TemplateColumn` 一起配合使用，我们后面介绍

`TemplateColumn` 的时候会说明。其使用语法如下所示：

```
<ASP:EditCommandColumn

    ButtonType="LinkButton | PushButton"

    CancelText="CancelButtonCaption"

    EditText="EditButtonCaption"

    FooterText="FooterText"

    HeaderImageUrl="url"

    HeaderText="HeaderText"

    ReadOnly="True | False"

    SortField="DataSourceFieldToSortBy"

    UpdateText="UpdateButtonCaption"

    Visible="True | False"

/>
```

其中除了共同基础属性以及样式对象外，常用的属性如下表所示：

属性	说明
ButtonType	所要使用的按钮种类，预设 为 LinkButton。
EditText	编辑数据所要显示给使用者看的文字。
UpdateText	更新数据所要显示给使用者看的文字。
CancelText	放弃编辑所要显示给使用者看的文字。

当使用者点选了显示 **EditText** 的控件时, **DataGrid Web** 控件会自动触发 **OnEditCommand** 事件, 并执行 **OnEditCommand** 属性所指定的事件程序; 倘若点选了显示 **UpdateText** 以及 **CancelText** 内容的控制像, 也一样分别自动触发 **OnUpdateCommand** 以及 **OnCancelCommand** 事件。下列程序代码范例增加了一个 **EditCommandColumn** 字段, 并且显示使用者点选了哪一个按钮:

```
<%@Import Namespace=System.Data.ADO%>

<%@Import Namespace=System.Data%>

<!--#Include File="GetTable.inc"-->

<Html>

<Form Runat="Server">

<ASP:DataGrid Id="dgA" AllowPaging="True" PageSize="5"

    OnPageIndexChanged="dgA_PageChg" Runat="Server"

    PagerStyle-Mode="NumericPages" BorderColor="#808080"

    HeaderStyle-Font-Names="Courier New"

HeaderStyle-BackColor="#D1DCEB"

    HeaderStyle-Font-Bold="True" HeaderStyle-HorizontalAlign="Center"

    OnEditCommand="dgA_ECmd" OnUpdateCommand="dgA_UCmd"

    OnCancelCommand="dgA_CCmd" >

<Property Name="Columns">

    <ASP:BoundColumn
```

```

        HeaderText="姓名"

        DataField="UserName"/>

<ASP:BoundColumn

        HeaderText="电话"

        DataField="UserTel"/>

<ASP:EditCommandColumn

        HeaderText="编辑"

        ButtonType="PushButton"

        EditText="编辑"

        UpdateText="更新"

        CancelText="放弃"/>

</Property>

</ASP:DataGrid>

</Form>

<ASP:Label Id="Label1" Runat="Server"/>

<Script Language="VB" Runat="Server">

Dim dtDataTable As DataTable=GetTable("CH08\MyWeb.mdb", "Members")

Sub Page_Load(Sender As Object, e As EventArgs)

    If Page.IsPostBack=False Then

```

```
        dgA.DataSource=dtDataTable.DefaultView

        Page.DataBind()

    End If

End Sub

Sub dgA_PageChg(Sender As Object, e As DataGridPageChangedEventArgs)

    dgA.DataSource=dtDataTable.DefaultView

    Page.DataBind()

End Sub

Sub dgA_ECmd(Sender As Object, e As DataGridCommandEventArgs)

    Label1.Text="您点选了第 " & (e.Item.ItemIndex+1).ToString & " 个字  
段的编辑."

    dgA.EditItemIndex=e.Item.ItemIndex

    dgA.DataSource=dtDataTable.DefaultView

    Page.DataBind()

End Sub

Sub dgA_UCmd(Sender As Object, e As DataGridCommandEventArgs)

    Label1.Text="您点选了第 " & (e.Item.ItemIndex+1).ToString & " 个字  
段的更新."

    dgA.EditItemIndex=-1
```

```
        dgA.DataSource=dtDataTable.DefaultView

        Page.DataBind()

End Sub

Sub dgA_CCmd(Sender As Object, e As DataGridCommandEventArgs)

    Label1.Text="您点选了第 " & (e.Item.ItemIndex+1).ToString & " 个字  
段的放弃."

    dgA.EditItemIndex=-1

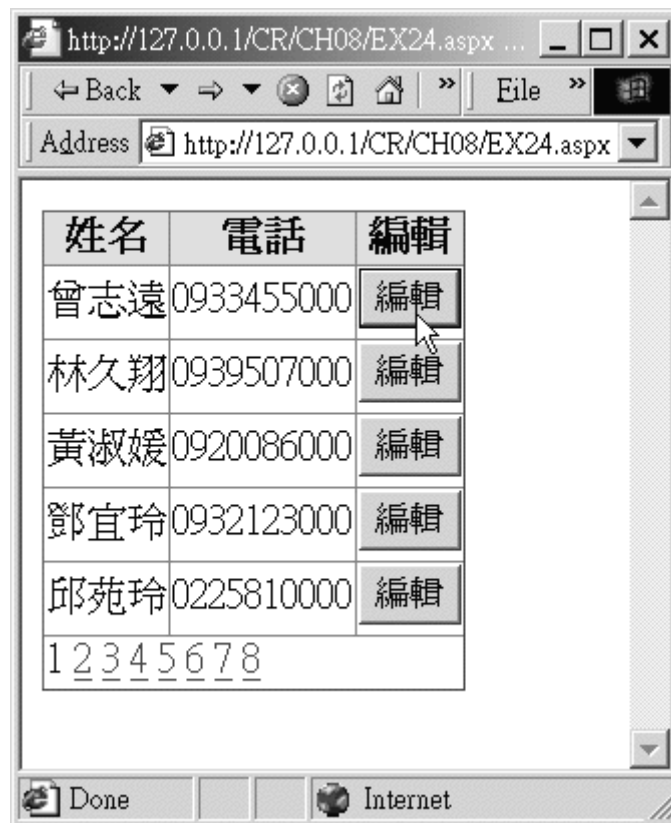
    dgA.DataSource=dtDataTable.DefaultView

    Page.DataBind()

End Sub

</SCRIPT>

</Html>
```



上述范例中，由于 **DataTable** 对象要让其它事件程序使用，所以我们在网页层次的宣告区宣告。

我们也指定了 **OnEditCommand** 事件、**OnUpdateCommand** 事件以及 **OnCancelCommand** 事件；当使用者按下「编辑」按钮时会自动触发我们所指定的 **dgA_ECcmd** 事件程序，如下程序代码范例所示：

```
Sub dgA_ECcmd(Sender As Object, e As DataGridCommandEventArgs)

    Label1.Text="您点选了第 " & (e.Item.ItemIndex+1).ToString & " 个字
段的编辑."

    dgA.EditItemIndex=e.Item.ItemIndex

    dgA.DataSource=dtDataTable.DefaultView

    Page.DataBind()
```

End Sub

上述程序代码片段将 **DataGrid** 的 **EditItemIndex** 属性设定为使用者所点选记录的索引值，所以该笔记录就会进入编辑模式；此时原来字段内的「编辑」按钮就会变成「更新」以及「放弃」。

使用者若编辑完毕后再点选「更新」或「放弃」按钮时，即触发所指定的 **OnUpdateCommand** 或是 **OnCancelCommand** 事件；这些事件最后将 **DataGrid** 对象的 **EditItemIndex** 属性设为 -1，表示没有任何项目被编辑，让数据回复到原来的显示模式。如下图所示：

姓名	電話	編輯
曾志遠	0933455000	更新 放棄
林久翔	0939507000	編輯
黃淑媛	0920086000	編輯
鄧宜玲	0932123000	編輯
邱苑玲	0225810000	編輯
1 2 3 4 5 6 7 8		

您點選了第 1 個欄位的編輯。

↓
點選更新

姓名	電話	編輯
曾志遠	0933455000	編輯
林久翔	0939507000	編輯
黃淑媛	0920086000	編輯
鄧宜玲	0932123000	編輯
邱苑玲	0225810000	編輯
1 2 3 4 5 6 7 8		

您點選了第 1 個欄位的更新。

↓
點選放棄

姓名	電話	編輯
曾志遠	0933455000	編輯
林久翔	0939507000	編輯
黃淑媛	0920086000	編輯
鄧宜玲	0932123000	編輯
邱苑玲	0225810000	編輯
1 2 3 4 5 6 7 8		

您點選了第 1 個欄位的放棄。

数据的更新

上述的程序代码范例还不能够将使用者所作的修改更新回数据源, 接下来我们将上述程序改成有数据更新的能力:

```
<%@Import Namespace=System.Data.ADO%>

<%@Import Namespace=System.Data%>

<Html>

<Form Runat="Server">

<ASP:DataGrid Id="dgA" AllowPaging="True" PageSize="5"

                OnPageIndexChanged="dgA_PageChg" Runat="Server"

                PagerStyle-Mode="NumericPages"

BorderColor="#808080"

                HeaderStyle-Font-Names="Courier New"

                HeaderStyle-BackColor="#D1DCEB"

                HeaderStyle-Font-Bold="True"

                HeaderStyle-HorizontalAlign="Center"

                AutoGenerateColumns="False"

                OnEditCommand="dgA_ECmd"
```

```
        OnUpdateCommand="dgA_UCmd"

        OnCancelCommand="dgA_CCmd" >

<Property Name="Columns">

    <ASP:BoundColumn

        HeaderText="姓名"

        DataField="UserName"/>

    <ASP:BoundColumn

        HeaderText="电话"

        DataField="UserTel"/>

    <ASP:EditCommandColumn

        HeaderText="编辑"

        ButtonType="PushButton"

        EditText="编辑"

        UpdateText="更新"

        CancelText="放弃" />

</Property>

</ASP:DataGrid>

</Form>

<Script Language="VB" Runat="Server">
```



```

Dim dscA As ADODatasetCommand=New ADODatasetCommand("Select * From
Members", _

    "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\InetPub\wwwroot\CR\Ch08\MyWeb.Mdb")

Dim dsDataSet As DataSet=New DataSet

Sub Page_Load(Sender As Object, e As EventArgs)

    If Page.IsPostBack=False Then

        BindGrid()

    End If

End Sub

' 将数据从数据源中取回，并和控件系结

Sub BindGrid()

    dscA.FillDataSet(dsDataSet, "Members")

    dgA.DataSource=dsDataSet.Tables("Members").DefaultView

    Page.DataBind()

End Sub

Sub dgA_PageChg(Sender As Object, e As DataGridPageChangedEventArgs)

    BindGrid()

End Sub

```

```

Sub dgA_ECmd(Sender As Object, e As DataGridCommandEventArgs)

    dgA.EditItemIndex=e.Item.ItemIndex

    BindGrid()

End Sub

Sub dgA_UCmd(Sender As Object, e As DataGridCommandEventArgs)

    BindGrid()

    Dim shtR As Short=(dgA.CurrentPageIndex * dgA.PageSize) +
dgA.EditItemIndex

    Dim txtTemp As TextBox

    txtTemp=e.Item.Cells(0).Controls(0) ' 取回第一个储存格中的
TextBox

    ' 将数据填回对应的 DataTable 中

    dsDataSet.Tables("Members").Rows(shtR)("UserName")=txtTemp.Te
xt

    txtTemp=e.Item.Cells(1).Controls(0) ' 取回第二个储存格中的
TextBox

    ' 将数据填回对应的 DataTable 中

    dsDataSet.Tables("Members").Rows(shtR)("UserTel")=txtTemp.Tex
t

```

```

' 将 DataTable 的数据更新回数据源

dscA.Update(dsDataSet, "Members")

dgA.EditItemIndex=-1

BindGrid()

End Sub

Sub dgA_CCmd(Sender As Object, e As DataGridCommandEventArgs)

    dgA.EditItemIndex=-1

    BindGrid()

End Sub

</SCRIPT>

</Html>

```

由于我们要在许多程序中和数据源系结，并使用 **DataSetCommand** 对象，所以我们将

DataSetCommand、**DataSet** 对象宣告在网页阶层的宣告区；并且撰写从数据源取回数据、呼

叫 **Page.DataBind()** 方法的程序 **BindGrid**：

```

Sub BindGrid()

    dscA.FillDataSet(dsDataSet, "Members")

    dgA.DataSource=dsDataSet.Tables("Members").DefaultView

```

```
Page.DataBind()
```

```
End Sub
```

另外我们也完成将使用者所作的改变更新回数据源的程序 **dgA_Ucmd**，如下程序代码片段所示：

```
Sub dgA_Ucmd(Sender As Object, e As DataGridCommandEventArgs)
```

```
    BindGrid()
```

```
    Dim shtR As Short=(dgA.CurrentPageIndex * dgA.PageSize) +  
dgA.EditItemIndex
```

```
    Dim txtTemp As TextBox
```

```
    txtTemp=e.Item.Cells(0).Controls(0) '取回第一个储存格中的  
TextBox
```

```
    '将数据填回对应的 DataTable 中
```

```
    dsDataSet.Tables("Members").Rows(shtR)("UserName")=txtTemp.Te  
xt
```

```
    txtTemp=e.Item.Cells(1).Controls(0) '取回第二个储存格中的  
TextBox
```

```
    '将数据填回对应的 DataTable 中
```

```
    dsDataSet.Tables("Members").Rows(shtR)("UserTel")=txtTemp.Tex  
t
```

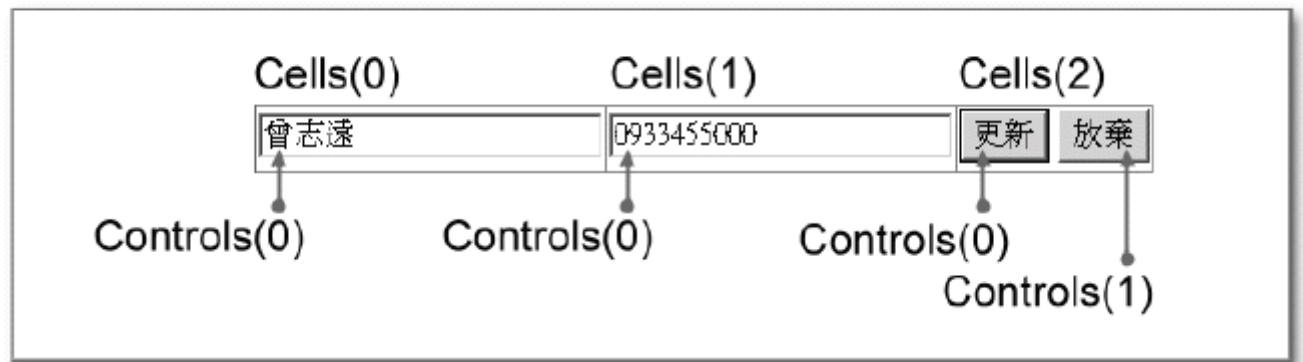
```
' 将 DataTable 的数据更新回数据源  
  
dscA.Update(dsDataSet, "Members")  
  
dgA.EditItemIndex=-1  
  
BindGrid()  
  
End Sub
```

由于我们将数据更新回数据源时，要指定数据在记录的绝对地址；所以我们将数据的绝对位置计算完毕后存入变量 **shtR** 中，待数据更新时使用。接下来的程序代码将使用者所编修的内容取回，并存回 **DataSet** 对象中，如下程序代码片段所示：

```
Dim txtTemp As TextBox  
  
txtTemp=e.Item.Cells(0).Controls(0)  
  
dsDataSet.Tables("Members").Rows(shtR)("UserName")=txtTemp.Text
```

首先我们宣告了一个指向 **TextBox** 对象的变量 **txtTemp**，并传回 **DataListItem** 中第一个字段的第一个控件之参考。因为 **DataListItem** 是以表格的方式来呈现字段所以每一个 **DataListItem** 都有一个 **Cells** 集合，用来管理每笔资料所要显示的所有字段，而呈现每个字段的控件被放置于 **Cell** 对象中；如下图所示：

DataListItem



所以要取得显示使用者名称的 `TextBox` 控件之内存地址，以下列叙述即可：

```
txtTemp=e.Item.Cells(0).Controls(0)
```

而假设要取回「放弃」`TextBox` 的内存地址，并将其 `Text` 属性显示出来；那么只要写成下列程

序代码片段即可：

```
txtTemp=e.Item.Cells(2).Controls(1)
```

```
Response.Write(txtTemp.Text)
```

我们取得 `TextBox` 的参考后，就可以将使用者所作的修改更新回 `DataTable` 中，如下程序代码

片所示：

```
dsDataSet.Tables("Members").Rows(shtR)("UserName")=txtTemp.Text
```

将所有的字段更新完毕后，最后只要利用 `DataSetCommand` 对象的 `Update` 方法将数据更新回数据源即可。

TemplateColumn

`TemplateColumn` 可以让我们自定义字段所要显示的样版。其使用语法如下所示：

```
<ASP:TemplateColumn>

    FooterText="表尾文字"

    HeaderImageUrl="url"

    HeaderText="标头文字"

    ReadOnly="True | False"

    SortField="DataSourceFieldToSortBy"

    Visible="True | False"

    <Template Name="HeaderTemplate">

        以 HTML 所定义的标头样版

    </Template>

    <Template Name="ItemTemplate">

        以 HTML 所定义的显示样版

    </Template>
```

```
<Template Name="EditItemTemplate">

    以 HTML 所定义的编辑样版

</Template>

<Template Name="FooterTemplate">

    以 HTML 所定义的表尾样版

</Template>

</ASP:TemplateColumn>
```

其可以定义的样版如下表所示：

样版	说明
EditItemTemplate	字段进入编辑模式时的样版。
FooterTemplate	字段的表尾样版。
HeaderTemplate	字段的标头样版。
ItemTemplate	字段的显示样版。

下列范例定义了三个字段的显示样版：

```
<%@Import Namespace=System.Data.ADO%>

<%@Import Namespace=System.Data%>
```



```
<!--#Include File="GetTable.inc"-->

<Html>

<Form Runat="Server">

<ASP:DataGrid Id="dgA" AllowPaging="True" PageSize="5"

    OnPageIndexChanged="dgA_PageChg" Runat="Server"

    PagerStyle-Mode="NumericPages" BorderColor="#808080"

    HeaderStyle-Font-Names="Courier New"

HeaderStyle-BackColor="#D1DCEB"

    HeaderStyle-HorizontalAlign="Center"

AutoGenerateColumns="False" >

    <Property Name="Columns">

        <ASP:TemplateColumn>

            <Template Name="HeaderTemplate">

                姓名

            </Template>

            <Template Name="ItemTemplate">

                <ASP:Image ImageUrl="ico7.gif" Runat="Server"/>

                <%#Container.DataItem("UserName")%>

            </Template>
```

```
</ASP:TemplateColumn>
```

```
<ASP:TemplateColumn>
```

```
    <Template Name="HeaderTemplate">
```

```
        电话
```

```
    </Template>
```

```
    <Template Name="ItemTemplate">
```

```
        <ASP:Image ImageUrl="ico8.gif" Runat="Server"/>
```

```
        <%#Container.DataItem("UserTel")%>
```

```
    </Template>
```

```
</ASP:TemplateColumn>
```

```
<ASP:TemplateColumn>
```

```
    <Template Name="HeaderTemplate">
```

```
        住址
```

```
    </Template>
```

```
    <Template Name="ItemTemplate">
```

```
        <ASP:Image ImageUrl="ico9.gif" Runat="Server"/>
```

```
        <%#Container.DataItem("UserAdd")%>
```

```
    </Template>
```

```
</ASP:TemplateColumn>
```

```
</Property>
```

```
</ASP:DataGrid>
```

```
</Form>
```

```
<ASP:Label Id="Label1" Runat="Server"/>
```

```
<Script Language="VB" Runat="Server">
```

```
Sub Page_Load(Sender As Object, e As EventArgs)
```

```
Dim dtDataTable As DataTable=GetTable("CH08\MyWeb.mdb", "Members")
```

```
    If Page.IsPostBack=False Then
```

```
        dgA.DataSource=dtDataTable.DefaultView
```

```
        Page.DataBind()
```

```
    End If
```

```
End Sub
```

```
Sub dgA_PageChg(Sender As Object, e As DataGridPageChangedEventArgs)
```

```
    Dim dtDataTable As DataTable=GetTable("CH08\MyWeb.mdb", "Members")
```

```
    dgA.DataSource=dtDataTable.DefaultView
```

```
    Page.DataBind()
```

```
End Sub
```

```
</SCRIPT>
```

```
</Html>
```

姓名	電話	住址
<input type="checkbox"/> 曾志遠	<input type="checkbox"/> 0933455000	<input type="checkbox"/> 台北市中正區
<input type="checkbox"/> 林久翔	<input type="checkbox"/> 0939507000	<input type="checkbox"/> 桃園縣桃園市
<input type="checkbox"/> 黃淑媛	<input type="checkbox"/> 0920086000	<input type="checkbox"/> 台北縣三重市
<input type="checkbox"/> 鄧宜玲	<input type="checkbox"/> 0932123000	<input type="checkbox"/> 台北縣中和市
<input type="checkbox"/> 邱苑玲	<input type="checkbox"/> 0225810000	<input type="checkbox"/> 台北市中正區
1	2	3
4	5	6
7	8	

DataGrid Web 控件的进阶应用

使用 Template 的数据更新

上述的程序代码范例中，对于数据编修的功能还不是很完整，无法将记录的修改状况更新回数据源。接下来我们利用 `EditCommandColumn`、`TemplateColumn` 以及一些事件程序将数据更新的功能完成。

```
<%@Import Namespace=System.Data.ADO%>

<%@Import Namespace=System.Data%>

<Html>

<Form Runat="Server">
```

```
<ASP:DataGrid Id="dgA" AllowPaging="True" PageSize="5"

    OnPageIndexChanged="dgA_PageChg" Runat="Server"

    PagerStyle-Mode="NumericPages" BorderColor="#808080"

    HeaderStyle-Font-Names="Courier New"

HeaderStyle-BackColor="#D1DCEB"

    HeaderStyle-HorizontalAlign="Center"

AutoGenerateColumns="False"

    OnEditCommand="dgA_ECmd" OnUpdateCommand="dgA_UCmd"

    OnCancelCommand="dgA_CCmd" >

<Property Name="Columns">

    <ASP:TemplateColumn>

        <Template Name="HeaderTemplate">

            姓名

        </Template>

        <Template Name="ItemTemplate">

            <ASP:Image ImageUrl="ico7.gif" Runat="Server"/>

            <%#Container.DataItem("UserName")%>

        </Template>

        <Template Name="EditItemTemplate">

            <ASP:Image ImageUrl="ico8.gif" Runat="Server"/>
```

```
<ASP:TextBox Id="txtUserName"
Text=' <%#Container.DataItem("UserName")%>'
Runat="Server"/>
</Template>
</ASP:TemplateColumn>
<ASP:TemplateColumn>
<Template Name="HeaderTemplate">
电话
</Template>
<Template Name="ItemTemplate">
<ASP:Image ImageUrl="ico7.gif" Runat="Server"/>
<%#Container.DataItem("UserTel")%>
</Template>
<Template Name="EditItemTemplate">
<ASP:Image ImageUrl="ico8.gif" Runat="Server"/>
<ASP:TextBox Id="txtUserTel"
Text=' <%#Container.DataItem("UserTel")%>'
Runat="Server"/>
</Template>
</ASP:TemplateColumn>
```

```

<ASP:TemplateColumn>

    <Template Name="HeaderTemplate">

        住址

    </Template>

    <Template Name="ItemTemplate">

        <ASP:Image ImageUrl="ico7.gif" Runat="Server"/>

        <%#Container.DataItem("UserAdd")%>

    </Template>

    <Template Name="EditItemTemplate">

        <ASP:Image ImageUrl="ico8.gif" Runat="Server"/>

        <ASP:TextBox Id="txtUserAdd"
Text=' <%#Container.DataItem("UserAdd")%>'
        Runat="Server"/>

    </Template>

</ASP:TemplateColumn>

<ASP:EditCommandColumn

    HeaderText="编辑"

    EditText="编辑"

    UpdateText="更新"

    CancelText="放弃"/>

```

```

        </Property>

</ASP:DataGrid>

</Form>

<Script Language="VB" Runat="Server">

Dim dscA As ADODatasetCommand=New ADODatasetCommand("Select * From
Members", _

        "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\InetPub\wwwroot\CR\Ch08\MyWeb.Mdb")

Dim dsDataSet As DataSet=New DataSet

Dim dtDataTable As DataTable

Sub Page_Load(Sender As Object, e As EventArgs)

    If Page.IsPostBack=False Then

        BindGrid()

    End If

End Sub

Sub BindGrid()      ' 数据系结程序

    dscA.FillDataSet(dsDataSet,"Members")

```



```

        dtDataTable=dsDataSet.Tables("Members")

        dgA.DataSource=dtDataTable.DefaultView

        Page.DataBind()

End Sub

Sub dgA_PageChg(Sender As Object, e As DataGridPageChangedEventArgs) '
换页程序

    BindGrid()

End Sub

Sub dgA_ECmd(Sender As Object, e As DataGridCommandEventArgs) ' 进入编辑模式
dgA.EditItemIndex=e.Item.ItemIndex BindGrid()

End Sub

Sub dgA_UCmd(Sender As Object, e As DataGridCommandEventArgs) ' 数据更新程序

Dim shtR As Short=(dgA.CurrentPageIndex * dgA.PageSize) +

e.Item.ItemIndex

Dim txtTemp As TextBox

        dscA.FillDataSet(dsDataSet,"Members") ' 先将 DataTable 从资料来源取回

        dtDataTable=dsDataSet.Tables("Members") ' 为了便利起见而将dtDataTable 指向目标数据表

```

```
txtTemp=e.Item.FindControl("txtUserName") ' 搜寻
```

DataListItem 内的控件

```
dtDataTable.Rows(shtR)("UserName")=txtTemp.Text ' 将找到
```

的控制之 Text 属性更新回

' 相对应的记录中

```
txtTemp=e.Item.FindControl("txtUserTel")
```

```
dtDataTable.Rows(shtR)("UserTel")=txtTemp.Text
```

```
txtTemp=e.Item.FindControl("txtUserAdd")
```

```
dtDataTable.Rows(shtR)("UserAdd")=txtTemp.Text
```

```
dscA.Update(dsDataSet,"Members") ' 将所作的改变更新回数据源
```

```
dgA.EditItemIndex=-1
```

```
BindGrid()
```

End Sub

```
Sub dgA_CCmd(Sender As Object, e As DataGridCommandEventArgs)
```

```
dgA.EditItemIndex=-1
```

```
BindGrid()
```

End Sub

</SCRIPT>

</Html>

姓名	電話	住址	編輯
<input type="checkbox"/> 曾志遠	<input type="checkbox"/> 0933455000	<input type="checkbox"/> 台北市中正區	編輯
<input type="checkbox"/> 林久翔	<input type="checkbox"/> 0939507000	<input type="checkbox"/> 桃園縣桃園市	編輯
<input type="checkbox"/> 黃淑媛	<input type="checkbox"/> 0920086000	<input type="checkbox"/> 台北縣三重市	編輯
<input type="checkbox"/> 鄧宜玲	<input type="checkbox"/> 0932123000	<input type="checkbox"/> 台北縣中和市	編輯
<input type="checkbox"/> 邱苑玲	<input type="checkbox"/> 0225810000	<input type="checkbox"/> 台北市中正區	編輯
1	2	3	4
5	6	7	8

上述范例码指定了 **DataGrid Web** 控件的 **OnEditCommand**、**OnUpdateCommand**、以及 **OnCancelCommand** 事件。所以使用者按下编辑选项并触发 **OnEditCommand** 事件时，会执行 **dgA_ECcmd** 事件程序让该笔记录进入编辑模式；如下程序代码片段所示：

```
Sub dgA_ECcmd(Sender As Object, e As DataGridCommandEventArgs)

    dgA.EditItemIndex=e.Item.ItemIndex

    BindGrid()

End Sub
```

由于我们在 **EditItemTemplate** 中指定进入编辑模式时，以 **TextBox** 来让使用者编修数据，所以当使用者点选「编辑」选项时会依照我们所设定的样版来显示编辑模式；如下程序代码片段所示：

<Template Name= "EditItemTemplate">

```

        <ASP:Image ImageUrl="ico8.gif" Runat="Server"/>

        <ASP:TextBox Id="txtUserAdd"
Text=' <%#Container.DataItem("UserAdd")%>'
Runat="Server"/>

    </Template>

```

姓名	電話	住址	編輯
<input type="text" value="曾志遠"/>	<input type="text" value="0933455000"/>	<input type="text" value="台北市中正區"/>	更新 放棄
<input type="text" value="林久翔"/>	<input type="text" value="0939507000"/>	<input type="text" value="桃園縣桃園市"/>	編輯
<input type="text" value="黃淑媛"/>	<input type="text" value="0920086000"/>	<input type="text" value="台北縣三重市"/>	編輯
<input type="text" value="鄧宜玲"/>	<input type="text" value="0932123000"/>	<input type="text" value="台北縣中和市"/>	編輯
<input type="text" value="邱苑玲"/>	<input type="text" value="0225810000"/>	<input type="text" value="台北市中正區"/>	編輯
1 2 3 4 5 6 7 8			

当使用者编辑完毕按下确定后，即执行 **dgA_UCmd** 这个事件程序，其程序代码片段如下所示：

```

Sub dgA_UCmd(Sender As Object, e As DataGridCommandEventArgs) '数据更新程序

Dim shtR As Short=(dgA.CurrentPageIndex * dgA.PageSize) +
e.Item.ItemIndex

Dim txtTemp As TextBox

```

```

        dscA.FillDataSet(dsDataSet, "Members")      ' 先将 DataTable 从资料源
取回

        dtDataTable=dsDataSet.Tables("Members")    ' 为了便利起见而将
dtDataTable 指向目标数据表

        txtTemp=e.Item.FindControl("txtUserName")  ' 搜寻 DataListItem 内的
控件

        dtDataTable.Rows(shtR)("UserName")=txtTemp.Text    ' 将找到的控制之
Text 属性更新回

        ' 相对应的记录中

        '... 程序代码略

        dscA.Update(dsDataSet, "Members")  ' 将所作的改变更新回数据源

dgA.EditItemIndex=-1 BindGrid()

End Sub

```

上述程序代码片段中我们先将数据源中的 **Members** 数据表取回，然后为了便利使用方便我们宣告了一个指向 **DataTable** 对象的变量 **dtDataTable**。这里有一点要特别注意，[在还没将 DataGrid Web 控件中使用者所修改的资料更新回 DataTable 前，不可以呼叫 DataBind 方法](#)；否则使用者所修改的数据会被原始的数据所覆盖，而造成更新回数据源的数据还是原始数据的错误，一定

要小心。接下来我们利用 **DataListItem** 的 **FindControl** 方法将 **TextBox** 控件的参考取回，如下程

序代码片段所示：

```
txtTemp=e.Item.FindControl("txtUserName") ' 搜寻 DataListItem 内的控件  
dtDataTable.Rows(shtR)("UserName")=txtTemp.Text ' 将找到的控制之  
Text 属性更新回  
' 相对应的记录中
```

因为我们使用样版来呈现我们想要的格式，在 **DataListItem** 中的每一个字段可能由许多复杂的控件组成；所以 **TextBox** 不见得会是 **TableCell** 对象中的第一个对象，故利用指定 **Cell** 中 **Controls** 集合的 **Index** 值来传回控件的参考，可能需要参考很多阶层。所以我们直接以 **ID** 属性来搜寻控件，并传回控件的参考比较方便。最后将找到的 **TextBox** 控件之 **Text** 属性逐一更新回对应的字段后，再呼叫 **DataSetCommand** 的 **Update** 方法即大功告成。

姓名	電話	住址	編輯
曾志遠	0933455000	台北市文山區	更新放棄
林久翔	0939507000	桃園縣桃園市	編輯
黃淑媛	0920086000	台北縣三重市	編輯
鄧宜玲	0932123000	台北縣中和市	編輯
邱苑玲	0225810000	台北市中正區	編輯
1 2 3 4 5 6 7 8			



姓名	電話	住址	編輯
曾志遠	0933455000	台北市文山區	編輯
林久翔	0939507000	桃園縣桃園市	編輯
黃淑媛	0920086000	台北縣三重市	編輯
鄧宜玲	0932123000	台北縣中和市	編輯
邱苑玲	0225810000	台北市中正區	編輯
1 2 3 4 5 6 7 8			

限制使用者对数据的更新

倘若我们不允许使用者修改某个字段的数据，在没有设定 **Template** 的状况下只要指定字段的

ReadOnly 属性为 **True** 即可；而若有指定 **Template** 的状况，只要不要设定该字段

EditItemTemplate 即可。下图为没有指定 **EditItemTemplate** 的情形：

姓名	電話	住址	編輯
<input type="checkbox"/> 曾志遠 <input type="checkbox"/>	<input type="text" value="0933455001"/>	<input type="checkbox"/> 台北市中正區	更新 放棄
<input type="checkbox"/> 林久翔 <input type="checkbox"/>	<input type="text" value="0939507000"/>	<input type="checkbox"/> 桃園縣桃園市	編輯
<input type="checkbox"/> 黃淑媛 <input type="checkbox"/>	<input type="text" value="0920086000"/>	<input type="checkbox"/> 台北縣三重市	編輯
<input type="checkbox"/> 鄧宜玲 <input type="checkbox"/>	<input type="text" value="0932123000"/>	<input type="checkbox"/> 台北縣中和市	編輯
<input type="checkbox"/> 邱苑玲 <input type="checkbox"/>	<input type="text" value="0225810000"/>	<input type="checkbox"/> 台北市中正區	編輯
1 2 3 4 5 6 7 8			

字段的排序

要支持排序功能，只要將 **DataGrid Web** 控件的 **AllowSorting** 屬性設為 **True**，並且指定要排序

時所要執行的程序為何即可。下列范例支持排序的功能，使用者可以點選欲排序的字段，該字段

數據即以升冪方式排列；若再點選該字段一次，則該字段改成以降序的方式排列：

```
<%@Import Namespace=System.Data.ADO%>

<%@Import Namespace=System.Data%>

<!--#Include File="GetTable.inc"-->

<Html>

<Form Runat="Server">

<ASP:DataGrid Id="dgA" AllowPaging="True" PageSize="10"

    OnPageIndexChanged="dgA_PageChg" Runat="Server"

    PagerStyle-Mode="NumericPages" BorderColor="#808080">
```



```
        HeaderStyle-Font-Names="Courier New"

HeaderStyle-BackColor="#D1DCEB"

        HeaderStyle-HorizontalAlign="Center"

        AllowSorting="True" OnSortCommand="dgA_Sort">

</ASP:DataGrid>

<ASP:TextBox Id="txtHidden" Runat="Server" Visible="False"

Enabled="False"/>

</Form>

<Script Language="VB" Runat="Server">

Dim dtDataTable As DataTable=GetTable("CH08\MyWeb.mdb", "Members")

Sub Page_Load(Sender As Object, e As EventArgs)

    If Page.IsPostBack=False Then

        txtHidden.Text="UserName"

        dgA.DataSource=dtDataTable.DefaultView

        Page.DataBind()

    End If

End Sub

Sub dgA_PageChg(Sender As Object, e As DataGridPageChangedEventArgs)

    If txtHidden.Enabled=True Then

        dtDataTable.DefaultView.Sort=txtHidden.Text
```

```
        Else

            dtDataTable.DefaultView.Sort=txtHidden.Text & " Desc"

        End If

        dgA.DataSource=dtDataTable.DefaultView

        Page.DataBind()

End Sub

Sub dgA_Sort(Sender As Object, e As DataGridSortCommandEventArgs)

    If txtHidden.Enabled=False Then

        dtDataTable.DefaultView.Sort=e.SortField

    Else

        dtDataTable.DefaultView.Sort=e.SortField & " Desc"

    End If

    txtHidden.Enabled=Not txtHidden.Enabled

    txtHidden.Text=e.SortField

    dgA.DataSource=dtDataTable.DefaultView

    Page.DataBind()

End Sub

</SCRIPT>

</Html>
```

由于 DataGrid Web 控件并不会自动排序数据，所以我们藉由 OnSortCommand 事件程序来修

改 DataTable.DefaultView 的 Sort 属性。由于换页的时候原来的排序状况无法保留，所以我们

透过一个 TextBox Web 控件，并将其 Visible 属性设为 False 来帮我们保持排序的状况。其中

TextBox 的 Text 属性用来记录所要排序的字段名称，而 Enabled 属性用来表示排序的方式是升

幂还是降序。其执行结果如图所示：

<u>UserId</u>	<u>UserPwd</u>	<u>UserName</u>	<u>UserTel</u>	<u>UserAdd</u>	<u>UserEmail</u>
bear	1252	鄭智雄	0223922000	台北市金山路	bear@hotmail.com
chee	iehh	黃世程	0927535000	台北縣中和市	chee@hotmail.com
chen	5952	陳正忠	0918452000	台北市大安區	chen@hotmail.com
chezn	kike	林啓仁	0289413000	台北縣中和市	chzen@hotmail.com
dar	ddeq	林志達	0336222000	桃園縣南豐街	dar@hotmail.com
der	dkke	廖得宏	0915266000	台北市芝玉路	der@hotmail.com
elvira	wxyz	鄧宜玲	0932123000	台北縣中和市	elvira@hotmail.com
fon	fie3	徐楓洲	0933090000	台北縣五股鄉	fon@hotmail.com
gigi	kdji	鄭怡菁	0229354000	台北市福興路	gigi@hotmail.com
hand	p00p	江毓帥	0937987000	台北縣金山市	hand@hotmail.com
1 2 3 4					

第一次点选时以升幂方式排序

User <u>I</u> d	UserP <u>W</u> d	UserN <u>A</u> me	UserT <u>E</u> l	UserA <u>D</u> d	UserE <u>M</u> ail
wum <u>i</u>	dkii	黃玉銘	0952591000	台北縣三重市	wumi@hotmail.com
wul <u>i</u> n	kloi	林永祥	0229431000	台北縣中和市	wulin@hotmail.com
we <u>i</u>	i93s	張偉信	0922383000	台北縣淡水鎮	wei@hotmail.com
wang	ikoe	王亦榮	0938012000	台北縣土城市	wang@hotmail.com
viv <u>i</u>	1252	邱苑玲	0225810000	台北市中正區	vivi@hotmail.com
tom	dkie	黃堂育	0933006000	台北縣中和市	tom@hotmail.com
tina	6789	黃淑媛	0920086000	台北縣三重市	tina@hotmail.com
tan	diie	梅思騰	0229748000	台北縣三重市	tan@hotmail.com
steven	1985	廖富熒	0222013000	台北縣新莊市	steven@hotmail.com
sing	idkk	李松興	0937086000	台北縣板橋市	sing@hotmail.com
1 2 3 4					

第二次点选时以降序方式排序

「主/明细」资料表

「主/明细」资料表常用来显示关联数据表间的数据。例如在 **MyWeb** 数据库中有两个有关联的数据表，分别为会员数据表以及订单数据表；会员数据表中记录了基本的会员数据，而订单数据表记录了会员所下的订单。我们可以透过一个 **DataGrid Web** 控件将这个会员数据表呈现出来后，点选我们所要检视的会员；此时该会员所下过的订单细目就会列在另一个 **DataGred Web** 控件中，如下图所示：

姓名	電話	住址	電郵
曾志遠	0933455000	台北市中正區	jacky@hotmail.com
林文翔	0939507000	桃園縣桃園市	jolin@hotmail.com
黃淑媛	0920086000	台北縣三重市	tina@hotmail.com
鄧宜玲	0932123000	台北縣中和市	elvira@hotmail.com
邱苑玲	0225810000	台北市中正區	vivi@hotmail.com
1 2 3 4 5 6 7 8			

日期	產品名稱	單價	數量	小計
2001-05-01T00:00:00	DVD Player	8000	1	8000
2001-05-10T00:00:00	DVD 測試片	1250	1	1250
2001-05-12T00:00:00	光纖線	1000	1	1000

總共有3筆記錄

要达到这个功能，只要利用 DataGrid Web 控件的 OnItemCommand 事件就可以办到。我们将

主数据表的使用者名称以 ButtonColumn 来呈现，这样使用者在这个字段上点选时，就会触发

OnItemCommand 事件；我们就可以透过这个事件取得使用者所点选的记录，并设定好条件将

子记录从数据源取回后呈现在明细数据表上：

```
<%@Import Namespace=System.Data.ADO%>
```

```
<%@Import Namespace=System.Data%>
```

```
<Html>
```

```
<Form Runat="Server">
```

```
<ASP:DataGrid Id="dgA" AllowPaging="True" PageSize="5"
    OnPageIndexChanged="dgA_PageChg"
Runat="Server"
    PagerStyle-Mode="NumericPages"
BorderColor="#808080"
    HeaderStyle-Font-Names="Courier New"
    HeaderStyle-BackColor="#D1DCEB"
    HeaderStyle-Font-Bold="True"
    HeaderStyle-HorizontalAlign="Center"
    AutoGenerateColumns="False"
    OnItemCommand="dgA_ICmd">
    <Property Name="Columns">
        <ASP:ButtonColumn
            HeaderText="姓名" DataTextField="UserName"/>
        <ASP:BoundColumn
            HeaderText="电话" DataField="UserTel"/>
        <ASP:BoundColumn
            HeaderText="住址" DataField="UserAdd"/>
        <ASP:BoundColumn
            HeaderText="电邮" DataField="UserEmail"/>
```

```
</Property>

</ASP:DataGrid><br>

<ASP:DataGrid Id="dgB" Runat="Server"

                PagerStyle-Mode="NumericPages"

BorderColor="#808080"

                HeaderStyle-Font-Names="Courier New"

                HeaderStyle-BackColor="#D1DCEB"

                HeaderStyle-Font-Bold="True"

                HeaderStyle-HorizontalAlign="Center"

                AutoGenerateColumns="False">

    <Property Name="Columns">

        <ASP:BoundColumn

                HeaderText="日期" DataField="OrderDate"/>

        <ASP:BoundColumn

                HeaderText="产品名称" DataField="ProductName"/>

        <ASP:BoundColumn

                HeaderText="单价" DataField="UnitPrice"/>

        <ASP:BoundColumn

                HeaderText="数量" DataField="Quantity"/>
```

```

        <ASP:BoundColumn
            HeaderText="小计" DataField="Total"/>

    </Property>

</ASP:DataGrid>

</Form>

<ASP:Label Id="Label1" Runat="Server"/>

<Script Language="VB" Runat="Server">

Dim dscA As ADODatasetCommand=New ADODatasetCommand("Select * From
Members", _

    "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\InetPub\wwwroot\CR\Ch08\MyWeb.Mdb")

Dim dsDataSet As DataSet=New DataSet

Sub Page_Load(Sender As Object, e As EventArgs)

    If Page.IsPostBack=False Then

        dscA.SelectCommand.CommandText="Select * From Members"

        dscA.FillDataSet(dsDataSet, "Members")

        dgA.DataSource=dsDataSet.Tables("Members").DefaultView

        dgA.DataBind()

```



```

End If

    Label11.Text="您目前没有点选任何记录."
End Sub

Sub dgA_PageChg(Sender As Object, e As DataGridPageChangedEventArgs)

    dscA.FillDataSet(dsDataSet, "Members")

    dgA.DataSource=dsDataSet.Tables("Members").DefaultView

    dgA.DataBind()

    dgB.Visible=False

End Sub

Sub dgA_ICmd(Sender As Object, e As DataGridCommandEventArgs)

    Dim shtR As Short=(dgA.CurrentPageIndex * dgA.PageSize) +
e.Item.ItemIndex

    If shtR>=0 Then

        dscA.FillDataSet(dsDataSet, "Members")

        dscA.SelectCommand.CommandText="Select * From Orders Where
UserId=' " & _

dsDataSet.Tables("Members").Rows(shtR)("UserId") & "' "

        dscA.FillDataSet(dsDataSet, "Orders")

        dgB.DataSource=dsDataSet.Tables("Orders").DefaultView

```

```
dgB.DataBind()

dgB.Visible=True

Label1.Text="总共有" & dsDataSet.Tables("Orders").Rows.Count & "
笔记录"

End If

End Sub

</SCRIPT>

</Html>
```

输出数据的润饰

我们在呈现数据的时候，要特别注意使用者接口的设计。一个好的使用者接口，除了美观外还可以让使用者容易使用以及阅读数据。

数据格式的润饰

我们在呈现数据的时候，不要将未经修饰过的数据呈现给使用者。例如金额一万元，如果我们直接显示「10000」，可能会导致使用者看成一千或十万，造成使用者阅读数据上的困扰。若我们将一万元润饰后输出为「NT\$10,000」，不但让使比较好阅读，也会让使用者减少犯错的机会。

下列画面为润饰过的结果：

姓名	電話	住址	電郵
曾志遠	0933455000	台北市中正區	jacky@hotmail.com
林久翔	0939507000	桃園縣桃園市	jolin@hotmail.com
黃淑媛	0920086000	台北縣三重市	tina@hotmail.com
鄧宜玲	0932123000	台北縣中和市	elvira@hotmail.com
邱苑玲	0225810000	台北市中正區	vivi@hotmail.com
1 2 3 4 ...			

日期	產品名稱	單價	數量	小計
2001/5/1	DVD Player	NT\$8,000.00	1	NT\$9,000.00
2001/5/10	DVD 測試片	NT\$1,250.00	1	NT\$1,250.00
2001/5/12	光纖線	NT\$1,000.00	1	NT\$1,000.00

上述数据除了将 **DataGrid Web** 控件以颜色来区隔记录外，最主要将日期、单价以及小计这三个计字段的数据修饰的更容易阅读。要修饰字段的输出，只要设定字段的 **DataFormatString** 属性即可；其使用语法如下：

```
DataFormatString="{0:格式字符串}"
```

我们知道在 **DataFormatString** 中的 {0} 表示数据本身，而在冒号后面的格式字符串代表我们所希望数据显示的格式；另外在指定的格式符号后可以指定小数所要显示的位数。例如原来的数据为「12.34」，若格式设定为 {0:N1}，则输出为「12.3」。其常用的数值格式如下表所示：

格式字符串	资料	结果
-------	----	----

"{0:C}"	12345.6789	\$12,345.68
"{0:C}"	-12345.6789	(\$12,345.68)
"{0:D}"	12345	12345
"{0:D8}"	12345	00012345
"{0:E}"	12345.6789	1234568E+004
"{0:E10}"	12345.6789	1.2345678900E+004
"{0:F}"	12345.6789	12345.68
"{0:F0}"	12345.6789	12346
"{0:G}"	12345.6789	12345.6789
"{0:G7}"	123456789	1.234568E8
"{0:N}"	12345.6789	12,345.68
"{0:N4}"	123456789	123,456,789.0000
"Total: {0:C}"	12345.6789	Total: \$12345.68

其常用的日期格式如下表所示：

格式	说明	输出格式
d	精简日期格式	MM/dd/yyyy
D	详细日期格式	dddd, MMMM dd, yyyy

f	完整格式 (long date + short time)	dddd, MMMM dd, yyyy HH:mm
F	完整日期时间格式 (long date + long time)	dddd, MMMM dd, yyyy HH:mm:ss
g	一般格式 (short date + short time)	MM/dd/yyyy HH:mm
G	一般格式 (short date + long time)	MM/dd/yyyy HH:mm:ss
m, M	月日格式	MMMM dd
s	适中日期时间格式	yyyy-MM-dd HH:mm:ss
t	精简时间格式	HH:mm
T	详细时间格式	HH:mm:ss

上述画面完整程序如下所示：

```

<%@Import Namespace=System.Data%>

<%@Import Namespace=System.Data.ADO%>

<html>

<Form Runat="Server">

<ASP:DataGrid Id="dgA" AllowPaging="True" PageSize="5"

OnPageIndexChanged="dgA_PageChg"

Runat="Server"

```

```

        PagerStyle-Mode="NumericPages"

BorderColor="#808080"

        HeaderStyle-Font-Names="Courier New"

        HeaderStyle-BackColor="#D1DCEB"

        HeaderStyle-Font-Bold="True"

        HeaderStyle-HorizontalAlign="Center"

        AutoGenerateColumns="False"

        OnItemCommand="dgA_ICmd"

Alternatingitemstyle-BackColor="#ffff99">

<Property Name="PagerStyle">

    <ASP:DataGridPagerStyle

        HorizontalAlign="Center" BackColor="Gainsboro"

        PageButtonCount="4" Mode="NumericPages"/></Property>

<Property Name="AlternatingItemStyle">

    <ASP:TableItemStyle ForeColor="Black"

BackColor="Gainsboro"/></Property>

<Property Name="ItemStyle">

    <ASP:TableItemStyle ForeColor="#000040"/></Property>

<Property Name="HeaderStyle">

    <ASP:TableItemStyle Font-Names="Courier New" Font-Bold="True"

```

```

        HorizontalAlign="Center" BackColor="#D1DCEB"/></Property>

<Property Name="Columns">

<ASP:ButtonColumn DataTextField="UserName" HeaderText="姓名"/>

<ASP:BoundColumn DataField="UserTel" HeaderText="电话"/>

<ASP:BoundColumn DataField="UserAdd" HeaderText="住址"/>

<ASP:BoundColumn DataField="UserEmail" HeaderText="电邮"/>

</Property>

</ASP:DataGrid><p>

<ASP:DataGrid Id="dgB" Runat="Server"

        PagerStyle-Mode="NumericPages"

        BorderColor="#808080"

        HeaderStyle-Font-Names="Courier New"

        HeaderStyle-BackColor="#D1DCEB"

        HeaderStyle-Font-Bold="True"

        HeaderStyle-HorizontalAlign="Center"

        AutoGenerateColumns="False"

        bgcolor="Transparent" >

<Property Name="PagerStyle">

```

```
<ASP:DataGridPagerStyle Mode="NumericPages"/></Property>

<Property Name="AlternatingItemStyle">

    <ASP:TableItemStyle ForeColor="Black"

BackColor="Gainsboro"/></Property>

<Property Name="ItemStyle">

    <ASP:TableItemStyle ForeColor="#000040"

BackColor="White"/></Property>

<Property Name="HeaderStyle">

    <ASP:TableItemStyle Font-Names="新细明体" Font-Bold="True"

HorizontalAlign="Center" BackColor="#D1DCEB"/></Property>

<Property Name="Columns">

    <ASP:BoundColumn DataField="OrderDate" HeaderText="日期"

DataFormatString="{0:d}"/>

    <ASP:BoundColumn DataField="ProductName" HeaderText="产品名称"/>

    <ASP:BoundColumn DataField="UnitPrice" HeaderText="单价"

DataFormatString="NT${0:N0}"/>

    <ASP:BoundColumn DataField="Quantity" HeaderText="数量"/>

    <ASP:BoundColumn DataField="Total" HeaderText="小计"

DataFormatString="NT${0:N0}"/>

</Property>
```



```

</ASP:DataGrid>

</Form>

<ASP:Label Id="Label1" Runat="Server"/>

<Script Language="VB" Runat="Server">

Dim dscA As ADODatasetCommand=New ADODatasetCommand("Select * From
Members", _

    "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\InetPub\wwwroot\CR\Ch08\MyWeb.Mdb")

Dim dsDataSet As DataSet=New DataSet

Sub Page_Load(Sender As Object, e As EventArgs)

    If Page.IsPostBack=False Then

        dscA.SelectCommand.CommandText="Select * From Members"

        dscA.FillDataSet(dsDataSet, "Members")

        dgA.DataSource=dsDataSet.Tables("Members").DefaultView

        dgA.DataBind()

    End If

    Label1.Text="您目前没有点选任何记录."

End Sub

Sub dgA_PageChg(Sender As Object, e As DataGridPageChangedEventArgs)

```

```

        dscA.FillDataSet(dsDataSet, "Members")

        dgA.DataSource=dsDataSet.Tables("Members").DefaultView

        dgA.DataBind()

        dgB.Visible=False

End Sub

Sub dgA_ICmd(Sender As Object, e As DataGridCommandEventArgs)

    Dim shtR As Short=(dgA.CurrentPageIndex * dgA.PageSize) +
e.Item.ItemIndex

    If shtR>=0 Then

        dscA.FillDataSet(dsDataSet, "Members")

        dscA.SelectCommand.CommandText="Select * From Orders Where
UserId=' " & _

dsDataSet.Tables("Members").Rows(shtR)("UserId") & " "

        dscA.FillDataSet(dsDataSet, "Orders")

        dgB.DataSource=dsDataSet.Tables("Orders").DefaultView

        dgB.DataBind()

        dgB.Visible=True

        Label1.Text="总共有" & dsDataSet.Tables("Orders").Rows.Count & "
笔记录"

```

```
End If  
  
End Sub  
  
</SCRIPT>  
  
</html>
```

使用者接口的可视化

上述范例利用点选使用者姓名的方式，来呈现该使用者所购买过的东西，这样的使用者接口还不够美观及直觉。接下来我们利用图示以来表示被点选到的字段为何，并将被点选到的字段以蓝底白字显示，如下图所示：

選擇	姓名	電話	住址	電郵
<input type="checkbox"/>	曾志遠	0933455000	台北市中正區	jacky@hotmail.com
<input checked="" type="checkbox"/>	林久翔	0939507000	桃園縣桃園市	jolin@hotmail.com
<input type="checkbox"/>	黃淑媛	0920086000	台北縣三重市	tina@hotmail.com
<input type="checkbox"/>	鄧宜玲	0932123000	台北縣中和市	elvira@hotmail.com
<input type="checkbox"/>	邱苑玲	0225810000	台北市中正區	vivi@hotmail.com
1 2 3 4 ...				

日期	產品名稱	單價	數量	小計
2001/4/5	卡通VCD	NT\$200	2	NT\$400
2001/5/6	CD	NT\$300	1	NT\$300
2001/5/10	耳機	NT\$800	1	NT\$800

總共有3筆記錄

```

<%@Import Namespace=System.Data%>

<%@Import Namespace=System.Data.ADO%>

<html>

<Form Runat="Server">

<ASP:DataGrid Id="dgA" AllowPaging="True" PageSize="5"

    OnPageIndexChanged="dgA_PageChg" Runat="Server"

    PagerStyle-Mode="NumericPages" BorderColor="#808080"

    HeaderStyle-Font-Names="Courier New"

    HeaderStyle-BackColor="#D1DCEB"

    HeaderStyle-Font-Bold="True"

    HeaderStyle-HorizontalAlign="Center"

```

```
AutoGenerateColumns="False"

OnItemCommand="dgA_ICmd"

alternatingitemstyle-backcolor="#ffff99">

<Property Name="PagerStyle">

    <ASP:DataGridPagerStyle

        HorizontalAlign="Center" BackColor="Gainsboro"

        PageButtonCount="4" Mode="NumericPages"/></Property>

<Property Name="AlternatingItemStyle">

    <ASP:TableItemStyle ForeColor="Black"

BackColor="Gainsboro"/></Property>

<Property Name="SelectedItemStyle">

<ASP:TableItemStyle ForeColor="White" BackColor="Blue"

Font-Bold="True"/></Property>

<Property Name="ItemStyle">

<ASP:TableItemStyle ForeColor="#000040"/></Property>

<Property Name="HeaderStyle">

<ASP:TableItemStyle Font-Names="Courier New" Font-Bold="True"

HorizontalAlign="Center"

        BackColor="#D1DCEB"/></Property>

<Property Name="Columns">
```

```
<ASP:ButtonColumn Text="<Img Border=0 Src=select.gif>" HeaderText="选择"
```

```
ItemStyle-HorizontalAlign="Center"/>
```

```
<ASP:BoundColumn DataField="UserName" HeaderText="姓名"/>
```

```
<ASP:BoundColumn DataField="UserTel" HeaderText="电话"/>
```

```
<ASP:BoundColumn DataField="UserAdd" HeaderText="住址"/>
```

```
<ASP:BoundColumn DataField="UserEmail" HeaderText="电邮"/>
```

```
</Property>
```

```
</ASP:DataGrid><p>
```

```
<ASP:DataGrid Id="dgB" Runat="Server"
```

```
PagerStyle-Mode="NumericPages" BorderColor="#808080"
```

```
HeaderStyle-Font-Names="Courier New"
```

```
HeaderStyle-BackColor="#D1DCEB"
```

```
HeaderStyle-Font-Bold="True"
```

```
HeaderStyle-HorizontalAlign="Center"
```

```
AutoGenerateColumns="False" backcolor="Transparent" >
```

```
<Property Name="PagerStyle">
```

```
<ASP:DataGridPagerStyle Mode="NumericPages"/>
```

```
</Property>
```

```
<Property Name="AlternatingItemStyle">

<ASP:TableItemStyle ForeColor="Black" BackColor="Gainsboro"/>

</Property>

<Property Name="ItemStyle">

<ASP:TableItemStyle ForeColor="#000040" BackColor="White"/>

</Property>

<Property Name="HeaderStyle">

<ASP:TableItemStyle Font-Names="新细明体" Font-Bold="True"

HorizontalAlign="Center" BackColor="#D1DCEB"/>

</Property>

<Property Name="Columns">

<ASP:BoundColumn DataField="OrderDate" HeaderText="日期"

DataFormatString="{0:d}"/>

<ASP:BoundColumn DataField="ProductName" HeaderText="产品名称"/>

<ASP:BoundColumn DataField="UnitPrice" HeaderText="单价"

DataFormatString="NT${0:N0}"/>

<ASP:BoundColumn DataField="Quantity" HeaderText="数量"/>

<ASP:BoundColumn DataField="Total" HeaderText="小计"

DataFormatString="NT${0:N0}"/>
```

```
</Property>
```

```
</ASP:DataGrid>
```

```
</Form>
```

```
<ASP:Label Id="Label1" Runat="Server"/>
```

```
<Script Language="VB" Runat="Server">
```

```
Dim dscA As ADODatasetCommand=New ADODatasetCommand("Select * From  
Members", _
```

```
    "Provider=Microsoft.Jet.OLEDB.4.0;Data
```

```
Source=C:\InetPub\wwwroot\CR\Ch08\MyWeb.Mdb")
```

```
Dim dsDataSet As DataSet=New DataSet
```

```
Sub Page_Load(Sender As Object, e As EventArgs)
```

```
    If Page.IsPostBack=False Then
```

```
        dscA.SelectCommand.CommandText="Select * From Members"
```

```
        dscA.FillDataSet(dsDataSet, "Members")
```

```
        dgA.DataSource=dsDataSet.Tables("Members").DefaultView
```

```
        dgA.DataBind()
```

```
    End If
```

```
    DgA.SelectedIndex=-1
```



```
Label1.Text="您目前没有点选任何记录."
```

```
End Sub
```

```
Sub dgA_PageChg(Sender As Object, e As DataGridPageChangedEventArgs)
```

```
    dscA.FillDataSet(dsDataSet, "Members")
```

```
    dgA.DataSource=dsDataSet.Tables("Members").DefaultView
```

```
    dgA.DataBind()
```

```
    dgB.Visible=False
```

```
End Sub
```

```
Sub dgA_ICmd(Sender As Object, e As DataGridCommandEventArgs)
```

```
    Dim shtR As Short=(dgA.CurrentPageIndex * dgA.PageSize) +
```

```
e.Item.ItemIndex
```

```
Sub dgA_ICmd(Sender As Object, e As DataGridCommandEventArgs)
```

```
    If e.Item.ItemIndex>-1 Then
```

```
        Dim shtR As Short=(dgA.CurrentPageIndex * dgA.PageSize) +
```

```
e.Item.ItemIndex
```

```
        dgA.SelectedIndex=e.Item.ItemIndex
```

```
        Dim dgiA As DataGridItem=dgA.SelectedItem
```

```
        Dim celSel As TableCell = dgiA.Controls(0)
```

```
        celSel.Text="<Img Border=0 Src=open.gif>"
```

```
        dscA.FillDataSet(dsDataSet, "Members")
```

```

        dscA.SelectCommand.CommandText="Select * From Orders Where
        UserId=' " & _

        dsDataSet.Tables("Members").Rows(shtR)("UserId") & "' "

        dscA.FillDataSet(dsDataSet, "Orders")

        dgB.DataSource=dsDataSet.Tables("Orders").DefaultView

        dgB.DataBind()

        dgB.Visible=True

        Label1.Text="总共有" & dsDataSet.Tables("Orders").Rows.Count & "
        笔记录"

    Else

        DgA.SelectedIndex=-1

    End If

End Sub

</SCRIPT>

</html>

```

上述程序代码中，我们定义了 **SelectedItemStyle** 属性，并设定其前景色为白色、背景色为蓝色，以及字型为粗体；如下程序代码范例所示：

```

<Property Name="SelectedItemStyle">

```

```
<ASP:TableItemStyle ForeColor="White" BackColor="Blue"  
Font-Bold="True"/></Property>
```

然后我们增加一个 **ButtonColumn** 字段，并设定其 **Text** 属性为「<img Border=0

Src=select.gif>」，这样的结果会让字段显示图钉的影像，如下程序代码片段所示：

```
<Property Name="Columns">  
  
<ASP:ButtonColumn Text="<img Border=0 Src=select.gif>" HeaderText="选  
择"  
  
ItemStyle-HorizontalAlign="Center"/>  
  
<ASP:BoundColumn DataField="UserName" HeaderText="姓名"/>  
  
<ASP:BoundColumn DataField="UserTel" HeaderText="电话"/>  
  
<ASP:BoundColumn DataField="UserAdd" HeaderText="住址"/>  
  
<ASP:BoundColumn DataField="UserEmail" HeaderText="电邮"/>  
  
</Property>
```

接下来就要处理使用者点选图形的动作了。我们设定 **OnItemCommand** 属性为 **dgA_ICmd**，所

以当使用者点选图形时便执行下列程序：

```
Sub dgA_ICmd(Sender As Object, e As DataGridCommandEventArgs)  
  
If e.Item.ItemIndex>-1 Then
```

```

        Dim shtR As Short=(dgA.CurrentPageIndex * dgA.PageSize) +
e.Item.ItemIndex

        dgA.SelectedIndex=e.Item.ItemIndex

        Dim dgiA As DataGridViewItem=dgA.SelectedItem

        Dim celSel As TableCell = dgiA.Controls(0)

        celSel.Text="<Img Border=0 Src=open.gif>"

        dscA.FillDataSet(dsDataSet, "Members")

        dscA.SelectCommand.CommandText="Select * From Orders Where
UserId=' " & _

dsDataSet.Tables("Members").Rows(shtR)("UserId") & " "

        dscA.FillDataSet(dsDataSet, "Orders")

        dgB.DataSource=dsDataSet.Tables("Orders").DefaultView

        dgB.DataBind()

        dgB.Visible=True

        Label1.Text="总共有" & dsDataSet.Tables("Orders").Rows.Count & "
笔记录"

End If

End Sub

```

上述程序代码先判断触发这个事件时，使用者是否有点选一个项目；倘若点选了任何一个项目，就将 dgA 这个 DataGrid Web 控件的 SelectedIndex 属性设为被点选的项目；接下来的三行程序将图钉改成钉住的影像：

```
Dim dgiA As DataGridItem=dgA.SelectedItem

    Dim celSel As TableCell = dgiA.Controls(0)

    celSel.Text="<Img Border=0 Src=open.gif>"
```



上述程序代码片段第一行宣告一个 DataGridItem 对象变量 dgiA，并指向被选到的 DataGridItem 参考；然后取回被点选项目中的第一个控件，也就是显示图钉影像的第一个字段，并将其 Text 属性改成被钉下去的图形「open.gif」。

强化 DataGrid Web 控件的编辑接口

接下来的议题比较复杂，不过详细研究了解后可以产生亲和力较佳的使用者接口。DataGrid Web 控件在进入编辑模式时，可以自订一些可视化的输入组件，代替使用者在 TextBox 中输入。这样一来可以让使用者快速轻松输入外，也可以避免一些输入的错误。首先我们来看看加强后的编辑画面，我们利用 DropDownList Web 控件以及 CheckBox Web 控件，协助使用者输入性别、血型、星座以及是否订阅电子报，如下画面所示：

編輯	姓名	性別	血型	星座	是否訂閱電子報
	<input type="checkbox"/> 林煌章	男	AB	天秤座	是
	<input type="checkbox"/> 陳毅帆	男	A	金牛座	否
	<input type="checkbox"/> 林久翔	男	O	處女座	是
	<input type="checkbox"/> 林進德	男	A	射手座	是
	<input type="checkbox"/> 林欣予	女	B	獅子座	是
1					

按下编辑选项后，提供可视化的工具供使用者使用：

編輯	姓名	性別	血型	星座	是否訂閱電子報
	<input type="checkbox"/> 林煌章	男	AB	天秤座	是
	<input type="checkbox"/> 陳毅帆	男	A	金牛座	否
 	<input type="checkbox"/> 林久翔	<input checked="" type="radio"/> 男 <input type="radio"/> 女	A	處女座	<input checked="" type="checkbox"/>
	<input type="checkbox"/> 林進德	男	A	射手座	是
	<input type="checkbox"/> 林欣予	女	B	獅子座	是
1					

編輯	姓名	性別	血型	星座	是否訂閱電子報
	<input type="checkbox"/> 林煌章	男	AB	天秤座	是
	<input type="checkbox"/> 陳毅帆	男	A	金牛座	否
 	<input type="checkbox"/> 林久翔	<input checked="" type="radio"/> 男 <input type="radio"/> 女	B	處女座	<input checked="" type="checkbox"/>
	<input type="checkbox"/> 林進德	男	A	射手座	是
	<input type="checkbox"/> 林欣予	女	B	獅子座	是
1					

使用者修改完毕后，数据已经更新：

編輯	姓名	性別	血型	星座	是否訂閱電子報
	+ 林煌章	男	AB	天秤座	是
	+ 陳毅帆	男	A	金牛座	否
	+ 林久翔	男	B	射手座	否
	+ 林進德	男	A	射手座	是
	+ 林欣予	女	B	獅子座	是
1					

上述范例画面的完整程序代码如下所示：

```
<%@Import Namespace=System.Data.ADO%>

<%@Import Namespace=System.Data%>

<Html>

<Form Runat="Server">

<ASP:DataGrid Id="dgA" AllowPaging="True" PageSize="5"

    OnPageIndexChanged="dgA_PageChg" Runat="Server"

    PagerStyle-Mode="NumericPages" BorderColor="#808080"

    HeaderStyle-BackColor="#0066CC" HeaderStyle-ForeColor="White"
```

```

        HeaderStyle-HorizontalAlign="Center"

AutoGenerateColumns="False"

        OnEditCommand="dgA_ECmd" OnUpdateCommand="dgA_UCmd"

        OnCancelCommand="dgA_CCmd" >

<Property Name="EditItemStyle">

        <ASP:TableItemStyle HorizontalAlign="Center" BackColor="#D1DCEB"/>

</Property>

<Property Name="ItemStyle">

        <ASP:TableItemStyle HorizontalAlign="Center"/>

</Property>

<Property Name="Columns">

        <ASP:EditCommandColumn

                HeaderText="编辑"

                EditText="<Img Border=0 Src=edit.gif>"

                UpdateText="<Img Border=0 Src=save.gif>"

                CancelText="<Img Border=0 Src=cancel.gif>"/>

        <ASP:TemplateColumn>

                <Template Name="HeaderTemplate">

                        姓名

```



```

</Template>

<Template Name="ItemTemplate">

    <ASP:Image ImageUrl="ico7.gif" Runat="Server"/>

    <%#Container.DataItem("UserName")%>

</Template>

<Template Name="EditItemTemplate">

    <ASP:Image ImageUrl="ico8.gif" Runat="Server"/>

    <ASP:TextBox Id="txtName"
Text=' <%#Container.DataItem("UserName")%>'
        Runat="Server"/>

</Template>

</ASP:TemplateColumn>

<ASP:TemplateColumn>

    <Template Name="HeaderTemplate">

        性别

    </Template>

    <Template Name="ItemTemplate">

        <%#Container.DataItem("UserSex")%>

    </Template>

    <Template Name="EditItemTemplate">

```

```

        <ASP:RadioButtonList Id="rblSex" RepeatDirection="Horizontal"
            SelectedIndex=' <%#GetProperty("rblSex")%>'
Runat="Server">

        <ASP:ListItem>男</ASP:ListItem>

        <ASP:ListItem>女</ASP:ListItem>

    </ASP:RadioButtonList>

</Template>

</ASP:TemplateColumn>

<ASP:TemplateColumn>

    <Template Name="HeaderTemplate">

        血型

    </Template>

    <Template Name="ItemTemplate">

        <%#Container.DataItem("UserBlood")%>

    </Template>

    <Template Name="EditItemTemplate">

        <ASP:DropDownList Id="ddlBlood" DataSource=' <%#arBlood%>'
            SelectedIndex=' <%#GetProperty("ddlBlood")%>' Runat="Server">

    </ASP:DropDownList>

    </Template>

```

```
</ASP:TemplateColumn>
```

```
<ASP:TemplateColumn>
```

```
    <Template Name="HeaderTemplate">
```

```
        星座
```

```
    </Template>
```

```
    <Template Name="ItemTemplate">
```

```
        <%#Container.DataItem("UserCons")%>
```

```
    </Template>
```

```
    <Template Name="EditItemTemplate">
```

```
        <ASP:DropDownList Id="ddlCons" DataSource=' <%#arCons%'>
```

```
        SelectedIndex=' <%#GetProperty("ddlCons")%' Runat="Server"/>
```

```
    </Template>
```

```
</ASP:TemplateColumn>
```

```
<ASP:TemplateColumn>
```

```
    <Template Name="HeaderTemplate">
```

```
        是否订阅电子报
```

```
    </Template>
```

```
    <Template Name="ItemTemplate">
```

```
        <%#Container.DataItem("OrderNews")%>
```

```
    </Template>
```

```

        <Template Name="EditItemTemplate">

            <ASP:CheckBox Id="cbOrderNews"

Checked=<%#GetProperty("cbOrderNews")%>

                Runat="Server"/>

        </Template>

    </ASP:TemplateColumn>

</Property>

</ASP:DataGrid>

</Form>

<Script Language="VB" Runat="Server">

Dim dscA As ADODataSetCommand=New ADODataSetCommand("Select * From

Users", _

    "Provider=Microsoft.Jet.OLEDB.4.0;Data

Source=C:\InetPub\wwwroot\CR\Ch08\MyWeb.Mdb")

Dim dsDataSet As DataSet=New DataSet

Dim dtDataTable As DataTable

Dim arCons() As String={"天秤座","天蝎座","水瓶座","巨蟹座","金牛座","

射手座", _

    "处女座","狮子座","双子座","双鱼座","魔羯座","牡羊座"}

```

```
Dim arBlood() As String={"A","B","O","AB"}

Sub Page_Load(Sender As Object, e As EventArgs)

If Page.IsPostBack=False Then

    BindGrid()

End If

End Sub


Sub BindGrid() ' 本程序用来执行控件和数据源间的系结

    dscA.FillDataSet(dsDataSet,"Users")

    dtDataTable=dsDataSet.Tables("Users")

    dgA.DataSource=dtDataTable.DefaultView

    Page.DataBind()

End Sub


Sub dgA_PageChg(Sender As Object, e As DataGridPageChangedEventArgs)

    BindGrid()

End Sub


Sub dgA_ECmd(Sender As Object, e As DataGridCommandEventArgs)

    dgA.EditItemIndex=e.Item.ItemIndex

    BindGrid()

End Sub
```

Sub dgA_UCmd(Sender As Object, e As DataGridCommandEventArgs) '更新数据源的程序

```
Dim shtR As Short=(dgA.CurrentPageIndex * dgA.PageSize) +  
e.Item.ItemIndex
```

```
dscA.FillDataSet(dsDataSet, "Users")
```

```
dtDataTable=dsDataSet.Tables("Users")
```

```
Dim objControl As Object
```

```
objControl=e.Item.FindControl("txtName")
```

```
dtDataTable.Rows(shtR)("UserName")=objControl.Text
```

```
objControl=e.Item.FindControl("rblSex") '传回 rblSex 的参考
```

```
dtDataTable.Rows(shtR)("UserSex")=objControl.SelectedItem.Text '将  
使用者的选择更新
```

```
                                '至 DataTable 中所  
对应的字段
```

```
objControl=e.Item.FindControl("ddlBlood")
```

```
dtDataTable.Rows(shtR)("UserBlood")=objControl.SelectedItem.Text
```

```
objControl=e.Item.FindControl("ddlCons")
```

```
dtDataTable.Rows(shtR)("UserCons")=objControl.SelectedItem.Text
```

```
objControl=e.Item.FindControl("cbOrderNews")
```

```

dtDataTable.Rows(shtR)("OrderNews")=IIF(objControl.Checked,"是","否
")

dscA.Update(dsDataSet,"Users")

dgA.EditItemIndex=-1

BindGrid()

End Sub

Sub dgA_CCmd(Sender As Object, e As DataGridCommandEventArgs)

    dgA.EditItemIndex=-1

    BindGrid()

End Sub

' 进入编辑模式时，让编辑数据的控件显示正确值的程序

Function GetProperty(ByVal strCtlName As String)

Dim shtR As Short=(dgA.CurrentPageIndex * dgA.PageSize) +

dgA.EditItemIndex

If shtR>-1 Then

    Dim shtI As Short

    Select Case strCtlName    ' 判断控件名称

        Case "rblSex"    ' 传回性别

```

```
Return IIf(dtDataTable.Rows(shtR)("UserSex")="男", 0, 1) ' 如果等于男  
则传回 0, 否则传回 1
```

```
Case "ddlBlood" ' 传回血型
```

```
For shtI=0 To 3
```

```
If dtDataTable.Rows(shtR)("UserBlood")=arBlood(shtI) Then
```

```
' 判断资料表中的资
```

```
Return shtI ' 料在数组的索引值
```

```
Exit Function
```

```
End IF
```

```
Next
```

```
Case "ddlCons" ' 传回星座
```

```
Dim strCons=dtDataTable.Rows(shtR)("UserCons")
```

```
For shtI=0 To 11
```

```
If arCons(shtI)=strCons Then
```

```
Return shtI
```

```
Exit Function
```

```
End If
```

```
Next
```

```
Case "cbOrderNews" ' 传回是否订阅电子报
```



```

        Return IIF(dtDataTable.Rows(shtR)("OrderNews")="是
", "True", "False")

    End Select

End If

End Function

</SCRIPT>

</Html>

```

上述程序代码我们利用定义每个字段的 **EditItemTemplate**，并且利用数据系结叙述取得该字段数据的正确值。例如下列宣告了性别字段的编辑样版，使用者在进入编辑模式时以

RadioButtonList Web 控件来让使用者选择：

```

<Template Name="EditItemTemplate">

    <ASP:RadioButtonList Id="rblSex" RepeatDirection="Horizontal"

        SelectedIndex='<#GetProperty("rblSex")%>'

Runat="Server">

        <ASP:ListItem>男</ASP:ListItem>

        <ASP:ListItem>女</ASP:ListItem>

    </ASP:RadioButtonList>

</Template>

```

而 **GetProperty** 程序可以检查每个控件所应该显示的状态值，只要在呼叫本程序时传入该控件的名称，这个程序就会到 **DataTable** 中将控件的适当状态设定传回，如下程序代码片段所示：

```
Function GetProperty(ByVal strCtlName As String)

Dim shtR As Short=(dgA.CurrentPageIndex * dgA.PageSize) +

dgA.EditItemIndex

If shtR>-1 Then

    Dim shtI As Short

    Select Case strCtlName    '判断控件名称

        Case "rblSex"    '传回性别

            Return IIf(dtDataTable.Rows(shtR)("UserSex")="男", 0, 1)    '如果等于
            男则传回 0,

            ' 否则传回 1

    End Select

End If

End Function
```

上述程序代码先取得记录在 **DataTable** 中的绝对地址，再判断使用者所传入的控件名称是否为 **rblSex**。若为 **rblSex** 所执行的数据系结动作，则传回 **IIF** 函式的判断结果。**IIF** 的使用语法如下所示：

```
变量=IIF(条件判断式, 成立所传回的资料, 不成立所传回的资料)
```

IIF 中的条件判断式的最结果若为 **True**，则传回成立所传回的资料；若不成立则传回不成立所传回的资料。所以若目前 **UserSex** 这个字段的值若为「男」则传回数值 **0**，若不成立则为传回 **1**；此时 **RadioButtonList Web** 控件收到后即可显示数据正确的选项，其它栏控件取得正确状态的方式也是如此。

9. HTTP 物件

ASP.NET 提供了许多基础对象让我们使用，前面我们所使用的 **Response** 等对象就是属于 HTTP 对象中的一员。这些对象可以提供我们相当多的功能，例如可以在两个网页之间传递变量、输出数据，以及纪录变量值等。

Request 物件

Request 对象主要是让伺服器取得客户端浏览器的一些数据。因为 **Request** 对象是 **Page** 对象的成员之一，所以在程序中不需要做任何宣告即可直接使用；**Request** 对象正确的对象类别名称是 **HttpRequest**。**Request** 对象的属性和方法相当多，下表列出常用的属性：

属性	说明	型态
ApplicationPath	传回目前正在值行程序之伺服端的虚拟目录。	String
Browser	传回有关客户端浏览器的功能信息。	HttpBrowserCapabilities
ClientCertificate	传回有关客户端端安全认证的信息。	HttpClientCertificate
ConnectionID	传回目前客户端所发出的网页浏览请求之联机 ID。	Long

ContentEncoding	<p>传回客户端所支持的字符设定。</p> <p>中文 InternetExplorer 预设是 ChineseTraditional (Big5)。</p>	Encoding
ContentType	传回目前需求的 MIME 内容类型。	String
Cookies	传回一个 HttpCookieCollection 对象集合。	HttpCookieCollection
FilePath	传回目前执行网页的相对地址。	String
Files	传回客户端上传的档案集合。	HttpFileCollection
Form	传回有关窗体变量的集合。	NameValueCollection
Headers	传回有关 HTTP 标头的集合。	NameValueCollection
HttpMethod	<p>传回目前客户端 HTTP 数据传输的方式是 Post 或 Get。</p>	String
IsAuthenticated	传回目前的 HTTP 联机是否有效。	Boolean
IsSecureConnection	传回目前 HTTP 联机是否是安全。	Boolean
Params	<p>传回 QueryString、Form、Server Variable 以及 Cookies 全部的集合。</p>	NameValueCollection
Pathq	传回目前请求网页的相对地址。	String
PhysicalApplicationPath	传回目前执行的 Server 端程序在 Server	String

	端的真实路径。	
PhysicalPath	传回目前请求网页在 Server 端的真实	String
QueryString	传回附在网址后面的参数内容。	NameValueCollection
RawUrl	传回目前请求页面的原始 URL。	String
RequestType	传回客户端 HTTP 数据的传输方式使用 Get 或 Post。	String
ServerVariables	传回网页 Server 变量的集合。	NameValueCollection
TotalBytes	传回目前的输入串流有多少 Bytes。	Integer
Url	传回有关目前请求的 URL 信息。	HttpUrl
UserAgent	传回客户端浏览器的版本信息。	String
UserHostAddress	传回远方客户端机器的主机 IP 地址。	String
UserHostName	传回远方客户端机器的 DNS 名称。	String
UserLanguages	传回一个储存客户端机器使用的语言。	String

下表列出常用的方法：

方法	说明	语法
MapPath	传回实际路径。	MapPath(ByVal 虚拟路径 As String) As String
SaveAs	将 HTTP 请求的信息储存到磁盘中。	MapPath(ByVal virtualPath As String ,

		ByVal baseVirtualDir As String, ByVal allowCrossAppMapping As Boolean) As String
--	--	--

读取物件或参数的值

Request 对象的基本用法就是读取对象或是参数的内容。下列程序代码范例中有两个程序，我们

先在 **EX01.aspx** 中放入一个 **HtmlAnchor** 对象，并将它的 **Href** 属性设为

「**http://127.0.0.1/CR/CH09/EX02.aspx?data1=100**」；网址后的「**?data1=100**」即是我们所要

传递的参数，**data1** 是参数名称而 **100** 是内容。接下来启动这个网页，并点选「将数据传到另一

个网页」这个超级链接。当网页跳至 **EX02.aspx** 时，**Request** 对象便将参数 **data1** 的内容读取

出来，并使用 **Response** 对象的 **Write** 方法将它印出来。

```
<Html>

<A Href="http://127.0.0.1/CR/CH09/EX02.aspx?data1=100">将资料传到另一个网页</A>

</Html>

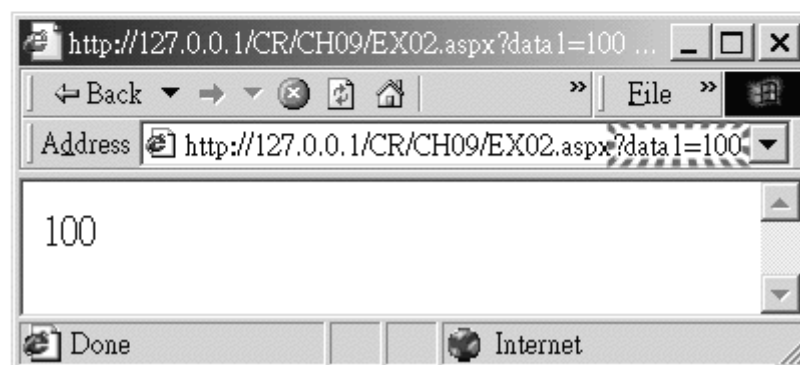
</程序>
```


<程序>

<Html>

<%Response.Write(Request("data1"))%>

</Html>



取得客户端浏览器的信息

欲取得目前和服务端联机之浏览器的信息，可以使用 **Browser** 属性。**Browser** 属性是一个集合对象，所以也可以使用一个 **HttpBrowserCapabilities** 型态的对象变量来接收 **Browser** 属性的传回值。下列范例我们使用 **HttpBrowserCapabilities** 型态的变量来取得了浏览器的部分信息：

```
<Html>

<Script Language="VB" Runat="Server">

Sub Page_Load(Sender As Object, e As EventArgs)

    Dim bc As HttpBrowserCapabilities = Request.Browser

    Response.Write("<p>浏览器信息:</p>")

    Response.Write("浏览器 = " & bc.Browser & "<br>")

    Response.Write("型态 = " & bc.Type & "<br>")

    Response.Write("名称 = " & bc.Browser & "<br>")

    Response.Write("版本 = " & bc.Version & "<br>")

    Response.Write("使用平台 = " & bc.Platform & "<br>")

    Response.Write("是否为测试版 = " & bc.Beta & "<br>")

    Response.Write("是否为 16 位的环境 = " & bc.Win16 & "<br>")

    Response.Write("是否为 32 位的环境 = " & bc.Win32 & "<br>")

    Response.Write("是否支持框架(Frame) = " & bc.Frames & "<br>")

    Response.Write("是否支持表格(Table) = " & bc.Tables & "<br>")

    Response.Write("是否支持 Cookie = " & bc.Cookies & "<br>")
```

```
Response.Write("是否支持 VB Script = " & bc.VBScript & "<br>")

Response.Write("是否支持 Java Script = " & bc.JavaScript & "<br>")

Response.Write("是否支持 Java Applets = " & bc.JavaApplets & "<br>")

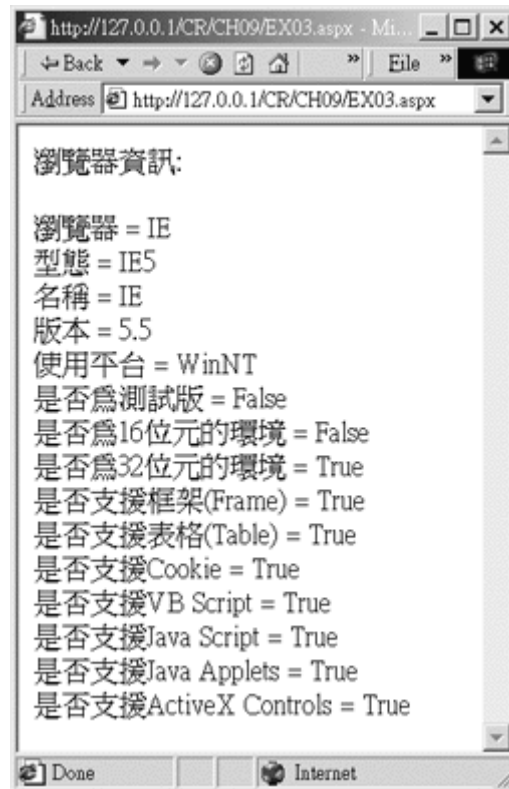
Response.Write("是否支持 ActiveX Controls = " & bc.ActiveXControls &
"<br>")

End Sub

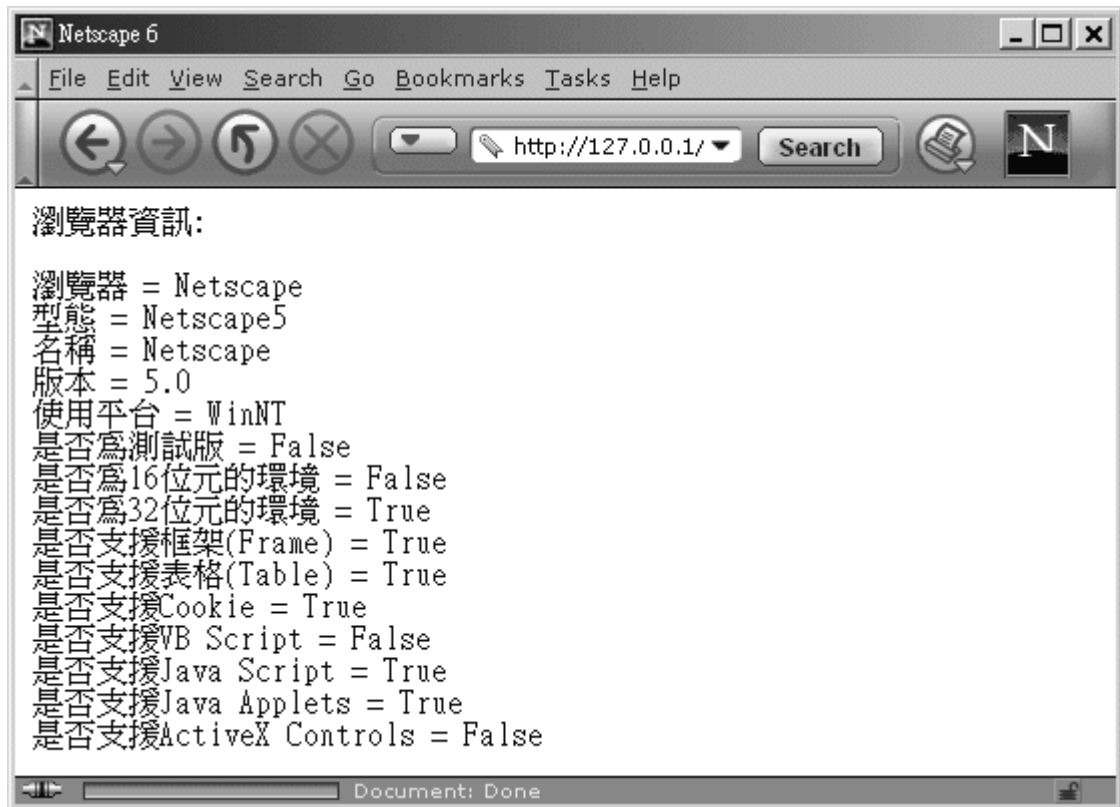
</Script>

</Html>
```

使用 InternetExplorer 浏览的结果:



使用 Netscape 浏览的结果:



MapPath 方法

使用 **MapPath** 方法接收一个字符串型态的参数，并且传回目前所在的实际路径以及传入字符串的结合。这个方法应用在需要使用实际路径的地方，例如在和数据源连结时必须指定完整的实际路径；就可以使用本方法取回。下列程序代码范例中利用 **Response** 对象的 **Write** 方法将目前的实际路径结合 **MyWeb.mdb** 字符串后印出：

```
<Html>
```

```
<Script Language="VB" Runat="Server">
```

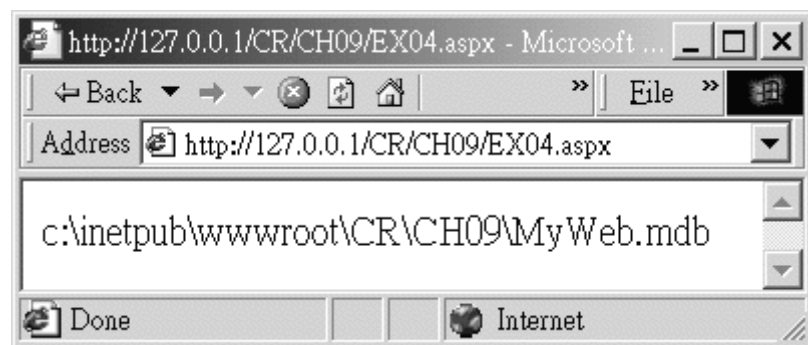
```
Sub Page_Load(Sender As Object, e As EventArgs)

    Response.Write(Request.MapPath("MyWeb.mdb"))

End Sub

</Script>

</Html>
```



取出使用者所上传的参数

前面我们已经了解如何取得使用者上传的参数值，但那是在已经知道参数名称的状况之下才可

以；而使用 **QueryString** 属性我们可以只利用索引来取得参数值，**QueryString** 属性的型别是

NameValueCollection。下面的程序中我们先定义一个 **NameValueCollection** 型态变量来接收

QueryString 的内容，然后使用一组巢状循环来取得参数名称及内容：

```
<Html>
```

```
<Script Language="VB" Runat="Server">

Sub Page_Load(Sender As Object, e As EventArgs)

    Dim shtLoop1, shtLoop2 As Short

    Dim arA(), arB() As String

    Dim colA As NameValueCollection

    colA=Request.QueryString

    arA = colA.AllKeys ' 取得全部的键值并存到一个数组中

    For shtLoop1 = 0 To UBound(arA)

        Response.Write("参数名: " & arA(shtLoop1))

        arB = colA.GetValues(shtLoop1) ' 利用外循环的索引来取得参数内容并
存到一个数组中

        For shtLoop2 = 0 To UBound(arB)

            Response.Write(" 内容: " & arB(shtLoop2) & "<br>")

        Next shtLoop2

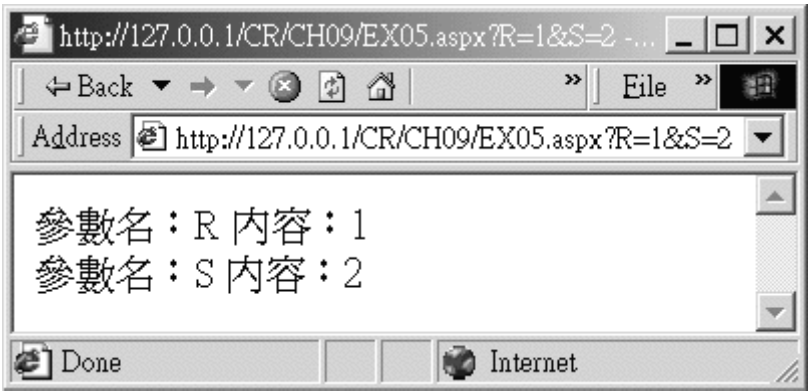
    Next shtLoop1

End Sub

</Script>

</Html>
```

由于 AllKeys 方法和 GetValues 方法的传回值都是数组，所以必须宣告两个数组来接收。我们在网址列输入「http://127.0.0.1/CR/CH09/EX05.aspx?R=1&S=2」后的执行结果，如下图所示：



Response 物件

Response 对象主要是输出数据到客户端，我们在前面章节已经应用过 Response 对象的 Write 方法。Response 对象正确的对象类别名称是 HttpResponse，和 Request 对象一样是属于 Page 对象的成员，所以也是不用宣告便可以直接使用。Response 对象提供了许多属性和方法，其常用的属性如下表所示：

属性	说明	型态
BufferOutput	设定 HTTP 输出是否要做缓冲处理，预设为 True。	Boolean
Cache	传回目前网页快取的设定。	HttpCachePolicy

Charset	设定或取得 HTTP 的输出字符编码。	String
Cookies	传回目前请求的 HttpCookieCollection 对象集合。	HttpCookieCollection
IsClientConnected	传回客户端是否仍然和 Server 连接。	Boolean
StatusCode	传回或设定输出至客户端浏览器的 HTTP 状态码，预设是 200。	Integer
StatusDescription	传回或设定输出至客户端浏览器的 HTTP 状态说明字符串，预设是 OK。	String
SuppressContent	设定是否将 HTTP 的内容送至客户端浏览器，若为 True 则网页将不会传至 Client 端。	Boolean

其常用的方法如下表所示：

方法	说明	语法
AppendToLog	将自订的纪录信息加到 IIS 的纪录文件中。	AppendToLog (ByVal param As String)
BinaryWrite	将一个二进制的字符串写入 HTTP 输出串流。	BinaryWrite (ByVal buffer As Byte)
Clear	将缓冲区的内容清除。	Clear ()
ClearHeaders	将缓冲区中所有的页面标头清除。	ClearHeaders ()

Close	关闭客户端的联机。	Close()
End	将目前缓冲区中所有的内容送到客户端然后关闭联机。	End()
Flush	将缓冲区中所有的数据送到客户端。	Flush()
Redirect	将网页重新导向另一个地址。	Redirect(ByVal url As String)
Write	将数据输出到客户端。	Write(ByVal String As String)
WriteFile	将一个档案直接输出至客户端。	WriteFile(ByVal filename As String)

使用缓冲区（Buffer）

由于 **Response** 对象的 **BufferOutput** 属性预设为 **True**，所以要输出到客户端的数据暂时都储存在缓冲区内，等到所有的事件程序以及所有的页面对象全部解译完毕后，才将所有在缓冲区中的数据送到客户端的浏览器。接下来我们做个实验来观察缓冲区如何运作：

```
<Html>

<%

Response.Write("清除之后的数据<Br>")

%>

<Script Language="VB" Runat="Server">
```

```
Sub Page_Load(Sender As Object, e As EventArgs)

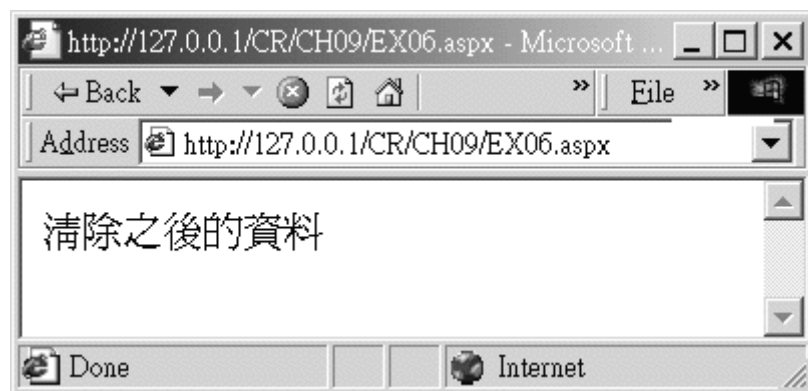
    Response.Write("清除缓冲区之前的数据" & "<br>")

    Response.Clear()

End Sub

</Script>

</Html>
```



上述程序代码范例首先在 **Page_Load** 事件中送出「清除缓冲区之前的数据」这一行，此时的数据存在缓冲区中。接着使用 **Response** 对象的 **Clear** 方法将缓冲区的数据清除，故刚刚送出的字符串已经被清除。然后 IIS 开始读取 HTML 组件的部分，最后将结果送至 **Client** 端的浏览器。由执行结果只出现「清除之后的数据」得知，使用 **Clear** 方法之前的数据并没有出现在浏览器上，由此可知程序在一开始是存在缓冲区内。接下来我们将相同的程序中加入

「 **Response.BufferOutput=False** 」叙述：

```
<Html>

<%

    Response.Write("清除之后的数据<Br>")

%>

<Script Language="VB" Runat="Server">

Sub Page_Load(Sender As Object, e As EventArgs)

    Response.BufferOutput=False

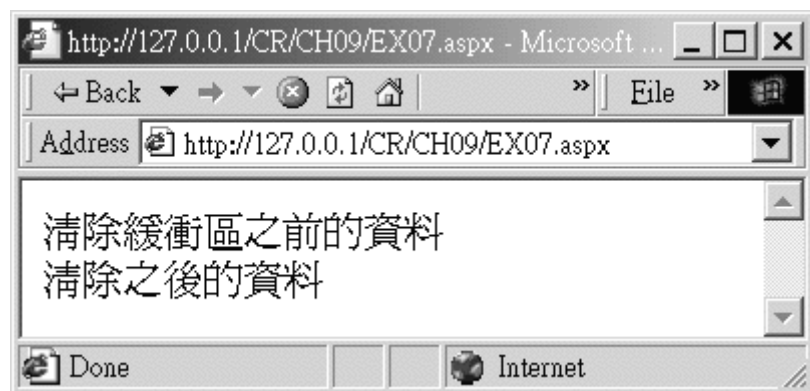
    Response.Write("清除缓冲区之前的数据" & "<Br>")

    Response.Clear()

End Sub

</Script>

</Html>
```



我们可以发现执行的结果并没有因为使用 **Clear** 方法而将缓冲区的数据清除，这表示数据是直接输出而没有存放在缓冲区内。

侦测使用者的联机状态

当网页在执行需要时间的复杂的运算或循环时，使用者的浏览器会一直处于等待的状态。此时若使用者停止浏览的动作，IIS 还是继续执行运算的话，相当浪费系统有限的资源。因此我们可以在执行这些需要等待的运算时，判断 **Response** 对象的 **IsClientConnected** 属性；若为 **False** 则代表使用者已经离线，此时只要使用 **Response** 对象的 **End** 方法来结束网页的执行即可。这样 **Server** 就不会执行无用的工作，可以空出更多的资源来让他人使用。下列的程序代码范例执行一个终值达 100000 的循环，我们在循环中加入判断 **IsClientConnection** 属性的叙述；只要 **Client** 端离线就终止执行，而 **Client** 端的浏览器上也不会出现任何信息：

```
<Html>

<Script Language="VB" Runat="Server">

Sub Page_Load(Sender As Object,e As Eventargs)

Dim i As Long

For i=0 To 100000

    If Response.IsClientConnected=False Then Response.End

Next
```

```
Response.Write("执行完毕")
```

```
End Sub
```

```
</Script>
```

```
</Html>
```

地址的重新导向

Response 对象的 **Redirect** 方法可以将连结重新导向到其它地址,使用时只要传入一个字符串型

态的 **URL** 即可,传入在网址后附加参数的 **URL** 字符串也可以。下列范例使用了 **TextBox Web**

控件让使用者输入一个 **URL**,使用者点选了 **Button1** 后即可将连结导向新的地址:

```
<Html>
```

```
<Form Runat="Server">
```

请输入一个 URL:


```
<Asp:TextBox Id="TextBox1" Runat="Server" />
```

```
<Asp:Button Id="Button1" Text="确定" OnClick="Button1_Click"
```

```
Runat="Server" />
```

```
</Form>
```

```
<Script Language="VB" Runat="Server">
```

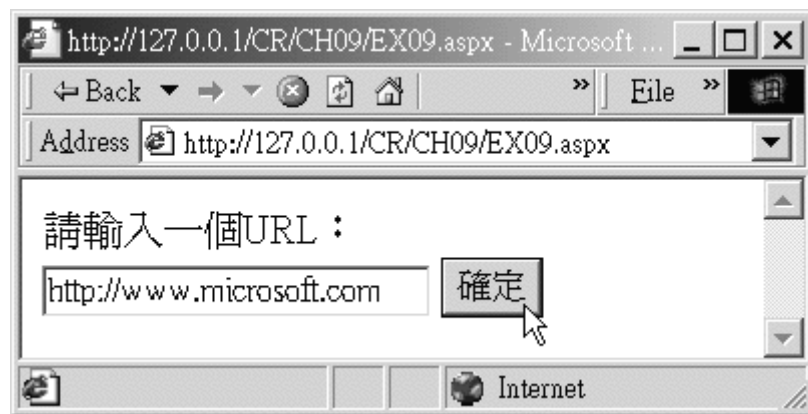
```
Sub Button1_Click(Sender As Object,e As EventArgs)
```

```
Response.Redirect(TextBox1.Text)

End Sub

</Script>

</Html>
```



直接输出文字文件

Response 对象提供了我们一个直接输出文字文件的 **WriteFile** 方法。若所要输出的档案和执行的网页在同一个目录，只要直接传入文件名称就可以了；若不是在同一个目录，则要指定详细的目录名称。下列程序代码范例中，我们直接输出一个内含 **HTML** 元素的文字文件到网页上；欲输出的档案内容如下所示：

```
<Input Type="Button" Value="Button1">
```

這是一個文字文件

```
<Input Type="Button" Value="Button1">
```

程序內容如下所示

```
<Html>

<Script Language="VB" Runat="Server">

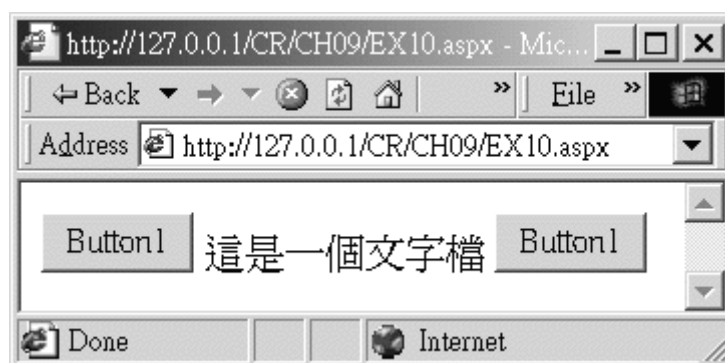
Sub Page_Load(Sender As Object, e As EventArgs)

Response.WriteFile("Output.txt")

End Sub

</Script>

</Html>
```



Server 物件

Server 对象也是 Page 对象的成员之一，主要提供一些处理网页请求时所需的功能；例如建立 COM 对象、将字符串的编译码等工作。Server 对象的对象类别名称是 HttpServerUtility，下表列出 Server 对象常用属性：

属性	说明	型态
MachineName	传回 Server 端机器名称。	String
ScriptTimeout	传回请求逾时的时间。	Integer

下表列出 Server 对象常用方法：

方法	说明	语法
CreateObject	使用 ProgId 来建立一个 COM 组件。	CreateObject (ByVal ProgId As String) As Object
CreateObjectFromClsid	使用 ClsId 来建立一个 COM 组件。	CreateObjectFromClsid (ByVal ClsId As String) As Object
CreateObjectStatic	使用 ProgId 来建立一个 COM 组件。	CreateObjectStatic (ByVal ProgId As String) As Object
HtmlDecode	将编码后的字符串译码回原来的 HTML 数据。	HtmlDecode (ByVal String As String) As String
HtmlEncode	将字符串编码为 HTML 可以	HtmlEncode (ByVal String As String) As

	辨识的信息。	String
MapPath	传回实际路径和传入字符串的结合字符串，和 Request 对象的	MapPath(ByVal path As String) As String MapPath 方法一样。
UrlDecode	将编码后的 URL 的字符串译码。	UrlDecode(ByVal String As String) As String
UrlEncode	将代表 URL 的字符串编码。	UrlEncode(ByVal String As String) As String

使用 COM 组件

COM（Component Object Model），一种对象的格式。凡是依照 COM 的规格所制作出来的组件，皆可以提供功能让其它程序或组件所使用。要使用 COM 组件，可以使用 Server 对象的 CreateObject、CreateObjectFromClsid、CreateObjectStatic 这三种方法。但是若要建立有使用者接口的 COM 组件（就是 ActiveX 控件，另一种规格的 COM 组件），则必须使用 <Object> 标注。CreateObject 和 CreateObjectStatic 使用方式一样，我们只要传入 ProgId（用来识别 COM 组件的唯一代码，每一个组件皆不同）即可；而 CreateObjectStatic 方法则是传入代表这个 COM 组件的 ProgId（COM 组件在操作系统里的名称）。其语法如下所示：

```
对象变量=Server.CreateObject("ProgId")
```

```
对象变量=Server.CreateObjectStatic("ProgId")
```

```
对象变量=Server.CreateObjectFromClsid("ClsId")
```

下列范例建立一个可以存取档案的 **FileSystemObject** 对象，并读取一个文字文件。以下为文字

文件内容：

使用档案系统对象来读取存在磁盘的文件

案记得将换行字符改成 HTML 的断行字符

程序内容如下所示：

```
<Html>

<Script Language="VB" Runat="Server">

Sub Page_Load(Sender As Object, e As EventArgs)

Dim Content As String

Dim objFile As Object

fsoA=Server.CreateObject("Scripting.FileSystemObject")

objFile=fsoA.OpenTextFile(Server.MapPath("TextFile.txt"),1,False)

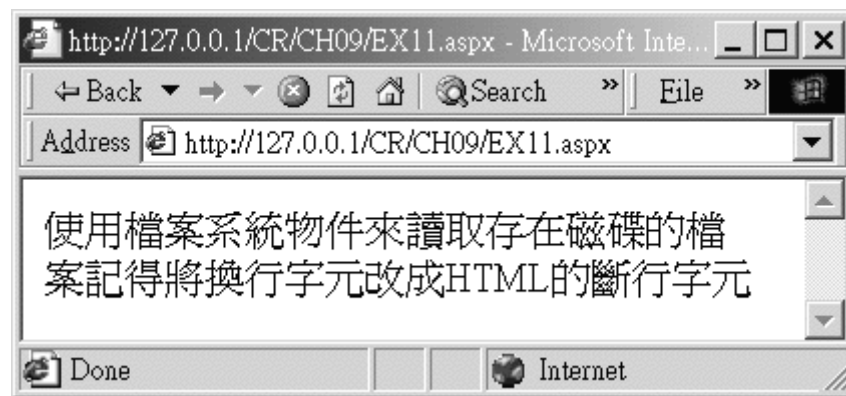
If objFile.AtEndOfStream=False Then

    Content=objFile.ReadAll

    Content=Replace(Content,Chr(13),"<Br>")

    Response.Write(Content)
```

```
End If  
  
End Sub  
  
</Script>  
  
</Html>
```



上述程序代码中我们建立一个名为 **fsoA** 的 **FileSystemObject** 对象，然后用 **fsoA** 对象的 **OpenTextFile** 方法来开启文字文件。因为 **OpenTextFile** 的传回值是 **TextStream** 对象，所以使用一个对象变量 **objFile** 来接收。接下来判断 **objFile** 对象的 **AtEndOfStream** 属性，若为 **True** 代表已经到达档案的结尾位置，则不做任何动作。接下来使用 **objFile** 对象的 **ReadAll** 方法将档案内容全部读出到字符串变量 **Content** 内，然后将 **Content** 内的断行字符 **VbCrLf** 转换为 HTML 断行标注 **
** 后输出 **Content** 的内容。上述程序将 **CreateObject** 改为 **CreateObjectFromClsId** 方法，并将传入档案对象的 **Clsid** 「0D43FE01-F093-11CF-8940-00A0C9054228」，最后的结果都是一样的。

```
<Html>

<Script Language="VB" Runat="Server">

Sub Page_Load(Sender As Object, e As EventArgs)

Dim Content As String

Dim objFile As Object

fsoA=Server.CreateObjectFromClsid("0D43FE01-F093-11CF-8940-00A0C90542
28")

objFile=fsoA.OpenTextFile(Server.MapPath("TextFile.txt"), 1, False)

    If objFile.AtEndOfStream=False Then

        Content=objFile.ReadAll

        Content=Replace(Content, Chr(13), "<Br>")

        Response.Write(Content)

    End If

End Sub

</Script>

</Html>
```

HtmlEncode 以及 HtmlDecode 方法

当我们想在网页上显示 HTML 标注时，若在网页中直接输出则会被浏览器解译为 HTML 的内容，所以要透过 **Server** 对象的 **HtmlEncode** 方法将它编码再输出；而若要将编码后的结果译码回原本的内容，则使用 **HtmlDecode** 方法。下列程序代码范例使用 **HtmlEncode** 方法将「HTML 内容」编码后输出至浏览器，再利用 **HtmlDecode** 方法将把编码后的结果译码还原：

```
<Html>

<Script Language="VB" Runat="Server">

Sub Page_Load(Sender As Object,e As EventArgs)

    Dim strHtmlContent As String

    strHtmlContent=Server.HtmlEncode("<B>HTML 内容</B>")

    Response.Write(strHtmlContent)

    Response.Write("<P>")

    strHtmlContent=Server.HtmlDecode(strHtmlContent)

    Response.Write(strHtmlContent)

End Sub

</Script>

</Html>
```



上述范例的输出结果可以发现到,编码后的HTML标注变成了 `HTML 内容`, 这是因为 `` 变成了 ``, `` 变成了 ``, 所以我们才能在页面中显示 HTML 标注。

UrlEncode 以及 UriDecode 方法

我们在传递网页参数时是将数据附在网址后面传递,但是遇到一些如「#」、「&」的特殊字符会读不到这些字符之后的参数。所以在需要传递特殊字符的场合,我们只要先将欲传递的内容先以 `UrlEncode` 加以编码,就可以保证所传递过去的值可以顺利被读到,而 `UriDecode` 方法则是将编码过的内容译码还原。下列范例我们使用两个 `HtmlAnchor` 控件来比较编码传递和未编码的结果,传递的参数内容是「a# @ #b」:

```
<Html>

<A Href="http://127.0.0.1/CR/CH09/EX02.aspx?data1=a# @ #b">未编码的参数内容</A><br>
```

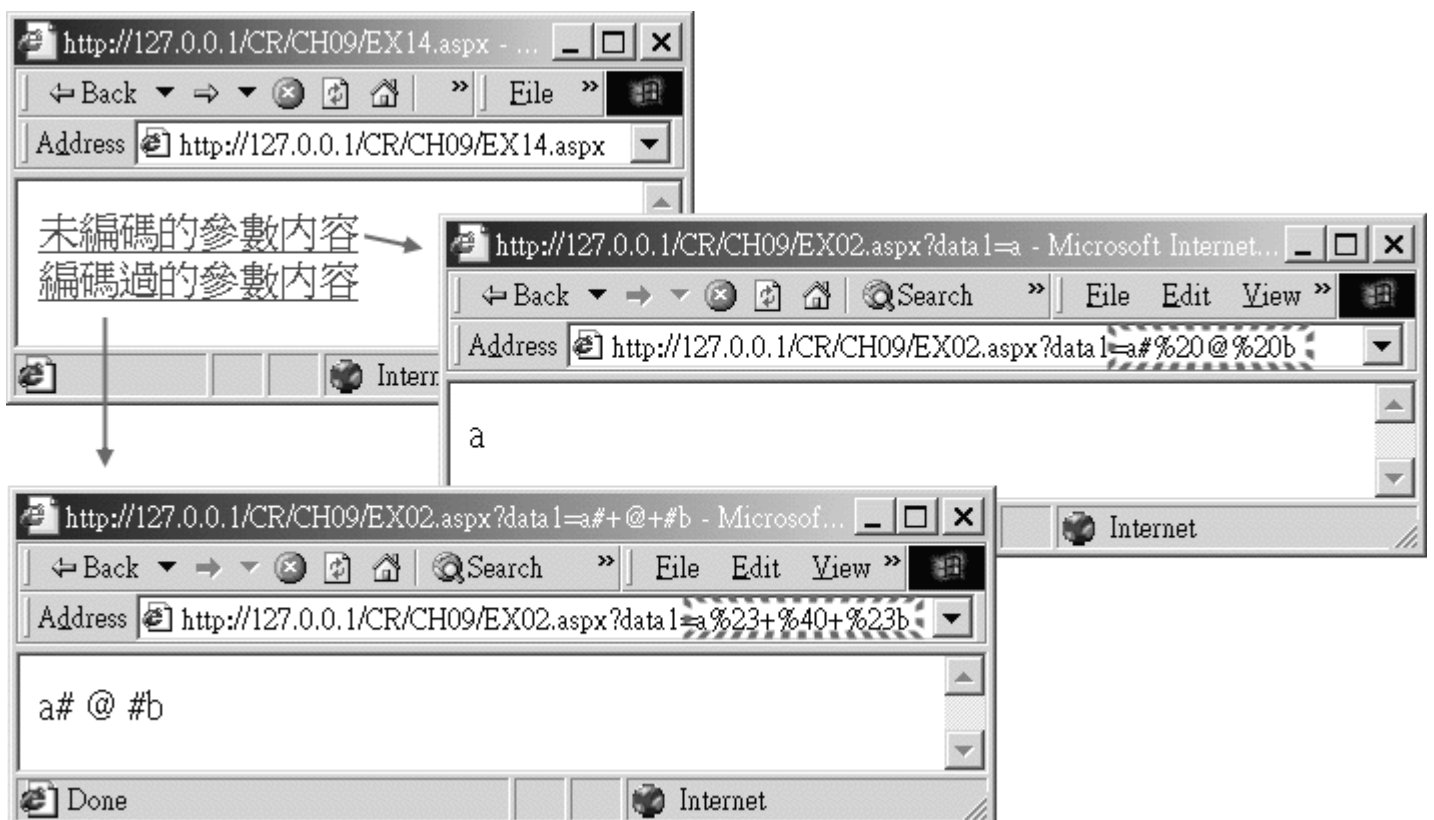
<A

Href="http://127.0.0.1/CR/CH09/EX02.aspx?data1=<%Response.Write(Serve
r.UrlEncode

("a# @ #b"))%>">

编码过的参数内容

</Html>



我们先按下「未编码的参数内容」可以发现到传递过去的参数内容只显示 **a**，这和我们当初所设定的结果并不相符。接着我们按下「编码过的参数内容」，结果显示出正确的参数内容。

Application 物件

Application 对象可以产生一个全部的 **Web** 应用程序都可以存取的变量，这个变量的可视范围涵盖全部的使用者；也就是只要正在使用这个网页程序的联机都可以存取这个变量。**Application** 对象正确的对象类别名称是 **HttpApplication**，每个 **Application** 对象变量都是 **Application** 对象集合中的对象之一，由 **Application** 对象统一管理。使用 **Application** 对象变量的语法如下所示：

```
Application("变量")="变量内容"
```

Application 对象变量的内容也可以是 **COM** 组件，产生的语法如下：

```
Application("对象名称")=Server.CreateObject(ProgId)
```

以上述语法所产生的对象变量可以完全使用该对象的属性与方法。**Application** 对象变量的生命周期止于关闭 **IIS** 或使用 **Clear** 方法清除，**Application** 对象是 **Page** 对象的成员，可以直接叫用。

下表为 **Application** 对象的常用属性：

属性	说明	型态
All	传回全部的 Application 对象变量到一个 Object 型态的数组。	Object ()

AllKeys	传回全部的 Application 对象变量名称到一个 String 型态的数组中。	String()
Count	取得 Application 对象变量的数量。	Integer
Item	让我们使用索引或是 Application 变量名称传回内容值。	Object

下表为 Application 对象的常用方法：

方法	说明	语法
Add	新增一个新的 Application 对象变量。	Add (ByVal name As String, ByVal value As Object)
Clear	清除全部的 Application 对象变量。	Clear ()
Get	使用索引值或变量名称传回变量值。	<div>1. Get (ByVal index As Integer) As Object</div> <div>2. Get (ByVal name As String) As Object</div>
GetKey	使用索引值来取得变量名称。	GetKey (ByVal index As Integer) As String
Lock	锁定全部的 Application 变数。	Lock ()
Remove	使用变量名称移除一个 Application	Remove (ByVal name As String)
RemoveAll	移除全部的 Application 对象变量。	RemoveAll ()

Set	使用变量名称更新一个 Application 对象变量的内容。	Set (ByVal name As String, ByVal value As Object)
UnLock	解除锁定 Application 变数。	UnLock()

前面我们已经介绍过如何定义一个 **Application** 对象变量，这边我们就来介绍一些其它的

Application 对象的使用方法。

取得 **Application** 对象变量内容的方法

取得 **Application** 对象变量的语法如下所示：

```
变数=Application("变量名称")
```

接下来我们来实际操作 **Application** 变量。下列范例宣告了三个 **Application** 对象变量，利用循环

显示后并清除：

```
<Html>

<Script Language="VB" Runat="Server">

Sub Page_Load(Sender As Object,e As EventArgs)

    Dim shtI As Short

    Application.Add("App1","Value1")

    Application.Add("App2","Value2")
```

```
Application.Add("App3", "Value3")

For shtI=0 To Application.Count-1

    Response.Write("變數名: " & Application.GetKey(shtI) )

    Response.Write(" , 變數值: " & Application.Item(shtI) & "<P>")

Next

Application.Clear()

End Sub

</Script>

</Html>
```



上述范例我们利用 **Application** 的 **Add** 方法产生 **Application** 变量并指定初始值，所有的

Application 变量都放在 **Application** 集合中由 **Application** 对象管理。所以我们要取出集合中的对

象可以使用 **For...Next** 循环或是 **For Each** 循环，循环中分别使用 **GetKey** 方法传回变量名称，**Item**

属性传回变量内容，程序最后将全部在集合中的变量用 **Clear** 方法清除。

锁定 **Application** 物件

因为 **Application** 对象变量每个联机都可以使用，所以可能造成两个以上的使用者同时存取同一

个变量的情形，可能会导致存入的数据不正确。要避免这种情况，我们只要利用 **Application** 对

象的 **Lock** 方法将变量暂时锁定禁止他人写入，等操作完毕后再利用 **Application** 对象的 **UnLock**

方法解除锁定。使用方法如下：

```
Application.Lock
```

```
Application("变量")="内容"
```

```
Application.Unlock
```

Session 物件

Session 对象的功能和 **Application** 对象一样，都是用来储存跨网页程序的变量或是对象，但

Session 对象和 **Application** 对象变量有些特性不太一样。**Session** 对象变量只针对单一网页使用

者，也就是说各个联机的机器有各自的 **Session** 对象变量，不同的联机无法互相存取。**Application**

对象变量中止于停止 IIS 服务，但是 Session 对象变量终止于联机机器离线时，也就是当网页使用者关掉浏览器或超过设定 Session 变量对象的有效时间时，Session 对象变量就会消失。

Session 对象正确的对象类别名称是 HttpSessionState，和 Application 对象一样是属于 Page 对象的成员；所以可以直接使用。Session 对象的使用方式和 Application 对象变量相当类似，其使用语法如下：

```
Session("变量名")="内容"
```

Session 也可以存放 COM 组件，其使用语法如下：

```
Session("名称")=Server.CreateObject (ProgId)
```

下表为 Session 对象常用的属性：

属性	说明	型态
All	传回全部的 Session 对象变量到一个数组。	Object ()
Count	传回 Session 对象变量的个数。	Integer
Item	以索引值或变量名称来传回或设定 Session	Item(String) As Object 对象变量的内容。
TimeOut	传回或设定 Session 对象变量的有效时间， 当联机使用者 超过有效时间没有动作， Session 对象便失效。默认值为 20 分钟。	Integer

下表为 Session 对象常用方法：

方法	说明	语法
Add	新增一个 Session 对象变量。	Add(ByVal name As String, ByVal value As Object)
Clear	清除所有的 Session 对象变量。	Clear()
Remove	以变量名称来移除变数。	Remove(ByVal name As String)
RemoveAll	清除所有的 Session 对象变量。	RemoveAll()

Session 对象变量最常应用在存放使用者的状态。例如在使用者登入的页面上，我们可以将代表使用者登入网页的成功与否状态储存到一个变量中，然后在其它网页加入判断使用者是否登入成功与否的程序代码。如果登入成功才可以浏览某些网页，如果登入失败则限制或拒绝使用者的浏览。下列范例使用者必须成功通过登入网页 EX16.aspx 的验证，才可以浏览 EX17.aspx 网页。

其验证的程序从会员数据表中判断使用者所填入的名称及密码是否正确，如下所示：

```
Sub btnSubmit_Click(Sender As Object, e As EventArgs)

    Dim strConStr As String = "Provider=Microsoft.Jet.OLEDB.4.0;" & _
        "Data Source=C:\InetPub\wwwroot\CR\CH08\MyWeb.mdb"

    Dim strComStr As String = "Select * From Members Where UserId = '" &
        _
        txtID.Text & "' And UserPwd = '" & txtPassword.Text & "'"
```

```
Dim dscA As ADODatasetCommand = New
ADODatasetCommand(strComStr, strConStr)

Dim dsDataSet As DataSet = New DataSet()

dscA.FillDataSet(dsDataSet, "Members")

If dsDataSet.Tables("Members").Rows.Count=1 Then

    Session("Id")=txtID.Text

    Session("IsPassed")="True"

    Page.Navigate("EX17.aspx") ' 将连结导向至 EX17.aspx

Else

    Label1.Text="验证失败！请重新输入"

End If

End Sub

Sub btnReset_Click(Sender As Object, e As EventArgs)

    txtID.Text=""

    txtPassword.Text=""

End Sub

</Script>

</Html>
```


上述范例我们将使用者所输入的使用者名称及密码转成 SQL 查询叙述,并存放于变量 **strComStr** 中; 如下所示:

```
Dim strComStr As String = "Select * From Members Where UserId = '" & _  
    txtID.Text & "' And UserPwd = '" & txtPassword.Text & "'"
```

倘若使用者输入的账号及密码分别为「rex」以及「sewq」, 则 **strComStr** 的内容为「Select * From Members Where UserId = 'rex' And UserPwd = 'sewq'」。由于 SQL 的语法规则字符串必需被单引号「'」包围, 所以请特别注意单引号的部分。我们将上述 SQL 陈述的执行结果填入 **DataSet** 对象中, 并且判断 **DataTable** 对象 **Rows** 集合的 **Count** 属性, 如果为 1 就表示有找到该使用者; 如下程序代码片段所示:

```
If dsDataSet.Tables("Members").Rows.Count=1 Then  
    Session("Id")=txtID.Text  
    Session("IsPassed")="True"  
    Page.Navigate("EX17.aspx") ' 将连结导向至 EX17.aspx  
Else  
    Label1.Text="验证失败! 请重新输入"  
End If
```

上述程序代码若使用者通过验证，则将 **Session** 对象的 **Id** 变量以及 **IsPassword** 变量的值分别填入使用者所数入的账号以及 **True**，然后将网页连结导向 **EX17.aspx**。我们就可以在 **EX17.aspx** 在 **Page_Load** 事件程序中取回 **Session** 对象中 **IsPassed** 变量，如下程序所示：

```
<Html>

您的输入已经通过验证.

<Script Language="VB" Runat="Server">

Sub Page_Load(Sender As Object, e As EventArgs)

    If Session("IsPassed")=Nothing Or Session("IsPassed")<>"True" Then

        Response.Redirect("EX16.aspx")

    End If

End Sub

</Script>

</Html>
```

如果我们通过验证，就可以成功的浏览 **EX17.aspx** 网页。倘若我们尝试直接利用浏览器直接浏览 **EX17.aspx** 网页时，因为 **Session("IsPassed")** 变量内并没有内容，所以网页会自动导向 **EX16.aspx** 登入网页。



设定 Session 对象变量的有效期限

因为每一个和 Server 端联机的客户端都是独立的 Session，所以 Server 端需要额外的资源来管理这些 Session。有时候使用者正在浏览网页时，突然去做其它的事情而没有把网页的联机关闭；如果 Server 端一直浪费资源在管理这些 Session 上，那么势必会让服务器的效率降低。所以当使用者超过一段时间没有动作时，我们就可以将 Session 释放。要更改 Session 对象的有效期限，只要设定 Timeout 属性即可；Timeout 属性的默认值是 20 分钟。下列范例将 Session 对象的 Timeout 属性设定为一分钟：

```
<Html>
```

```
<Form Runat="Server">
```

```
<Asp:Button Id="Button1" Text="显示" OnClick="Button1_Click"
Runat="Server" />
```

```
目前时间: <Asp:Label Id="Label1" Runat="Server" />
```

```
<P>
```

```
第一个 Session 的值: <Asp:Label Id="Label2" Runat="Server" /><Br>
```

```
第二个 Session 的值: <Asp:Label Id="Label3" Runat="Server" /><Br>
```

```
</Form>
```

```
<Script Language="VB" Runat="Server">
```

```
Sub Page_Load(Sender As Object, e As EventArgs)
```

```
If Not Page.IsPostBack Then
```

```
    Session("Session1")="Value1"
```

```
    Session("Session2")="Value2"
```

```
    Session.Timeout=1
```

```
    Label1.Text=Format(Now(), "HH:MM:SS")
```

```
    Label2.Text=Session("Session1")
```

```
    Label3.Text=Session("Session2")
```

```
End If
```

```
End Sub
```

```
Sub Button1_Click(Sender As Object, e As EventArgs)

    Label1.Text=Format(Now(), "hh:mm:ss")

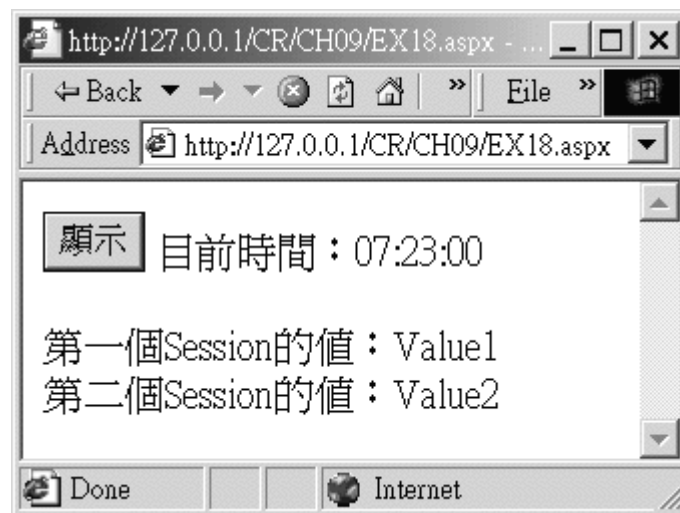
    Label2.Text=Session("Session1")

    Label3.Text=Session("Session2")

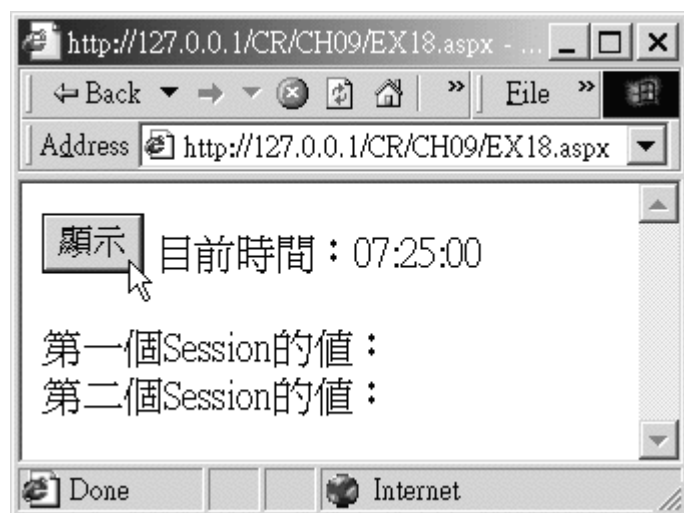
End Sub

</Script>

</Html>
```



第一次进入这个网页时，**Session** 对象变量的值会被显示出来；接着我们不要做任何动作静待一分钟，一分钟过后按下 **Button1** 时，**Session** 对象变量的内容便被释放：



Cookie 物件

Cookies、Session 和 Application 对象很类似，也是一种集合对象，都是用来在保存数据。但

Cookies 和其它对象最大的不同是 Cookies 将数据存放于客户端的磁盘上，而 Application 以及

Session 对象是将数据存放于 Server 端。Application、Session 以及 Cookies 对象的差异如下表

所示：

物件	数据存放位置	生命周期
Application	Server 端的内存上。	终止于 IIS 关闭时。
Session	存放在 Server 端的内存上。	终止于设定的时间或使用者离线。
Cookies	以档案的型式存放在客户端的磁盘上。	可一直存在或终止于所设定的时间为止。

Cookies 对象不隶属于 **Page** 对象，所以用法和 **Application** 及 **Session** 对象不同。**Cookies** 对象分别属于 **Request** 对象和 **Response** 对象，每一个 **Cookie** 变量都是被 **Cookies** 对象所管理，它的正确对象类别名称是 **HttpCookie Collection**。要储存一个 **Cookie** 变量，要透过 **Response** 对象的 **Cookies** 集合；其使用语法如下：

```
Response.Cookies(Name As String).Value="资料"
```

而要取回 **Cookie**，则是用 **Request** 对象的 **Cookies** 集合，并将指定的 **Cookie** 传回；其使用语法如下所示：

```
变数=Request.Cookies(Name As String).Value
```

Cookies 对象常用的属性如下表所示：

属性	说明	型态
All	传回全部的 Cookie 变量到一个数组中。	HttpCookie()
AllKeys	传回全部 Cookie 变量的名称到一个字符串型态的数组中。	String()
Count	传回 Cookie 变数的数量。	Integer
Item	以 Cookie 变量名称或索引值来传回 Cookie 变量的内容。	1. Item(String)As HttpCookie 2. Item(Index)As

		HttpCookie
--	--	------------

Cookies 对象常用的方法如下表所示：

方法	说明	语法
Add	新增一个 Cookie 变量到 Cookies 集合内。	Add(ByVal cookie As HttpCookie)
Clear	将 Cookies 集合内的变量全部清除。	Cookie Clear()
Get	以 Cookie 变量名称或索引值传回 Cookie 变量的值。	1. Get(ByVal index As Integer) As HttpCookie 2. Get(ByVal name As String) As HttpCookie
GetKey	以索引值来取回 Cookie 变量名称。	GetKey(ByVal index As Integer) As String
Remove	以 Cookie 变量名称移除 Cookie 变数。	Remove(ByVal name As String)
Set	更新一个 Cookie 变数的值。	Set(ByVal cookie As HttpCookie)

而 Cookie 变量常用的属性如下：

属性	说明	型态
Expires	设定 Cookie 变量的有效时间，默认值是 1000 分钟。若设为 0 则可以实时删除 Cookie 变量。	DateTime

Name	取得 Cookie 变量的名称。	String
Value	取得或设定 Cookie 变量的内容值。	String

下列范例新增两个 **Cookie** 变数，并利用 **For...Next** 循环分别利用 **Cookies** 集合的 **Item** 属性以及 **Get** 方法将 **Cookie** 变量传回：

```
<Html>

<Script Language="VB" Runat="Server">

Sub Page_Load(Sender As Object,e As EventArgs)

    Dim shtI As Short

    Response.Cookies("Cookie1").Value="Microsoft VisualStudio .Net"

    Response.Cookies("Cookie2").Value="ASP.Net"

    For shtI=0 To Request.Cookies.Count-1

        Response.Write("变量名称: " & Request.Cookies.Item(shtI).Name & _
            "<br>变量内容: " & Request.Cookies.Get(shtI).Value &
            "<br>")

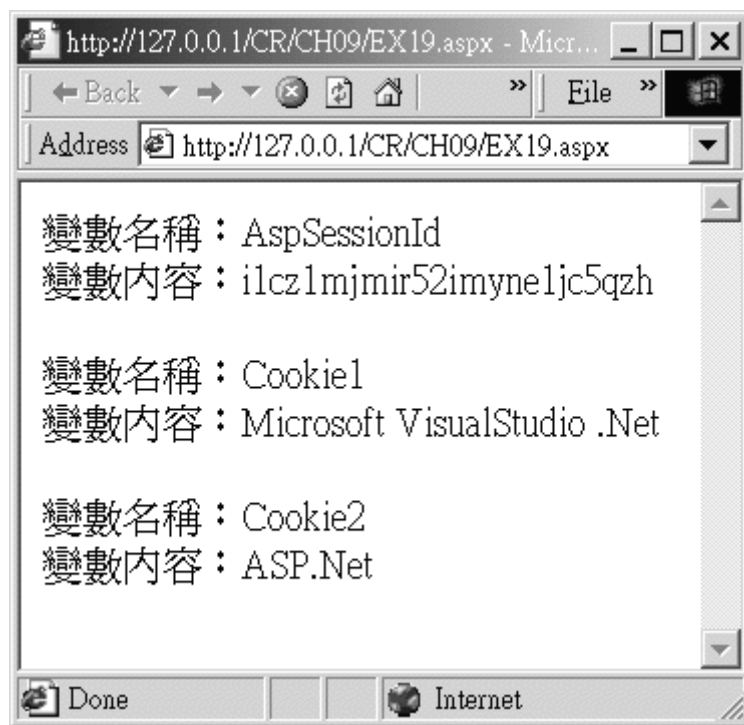
    Next

    Response.Cookies.Clear()

End Sub

</Script>

</Html>
```



除了我们所加入的 **Cookie** 变量之外，另外多了一个名为 **AspSessionId** 的 **Cookie** 变量。这个 **Cookie** 变量最主要是被 **ASP.NET** 用来识别每个连结，由 **ASP.NET** 在每个客户端建立连结时自动产生；每一次的连结其 **AspSessionId** 的内容都不同。

自订 **CookieCollection** 及 **Cookie** 物件

除了使用系统预设的 **Cookies** 对象外，我们也可以自行定义属于自己的 **Cookies** 对象。我们知道 **Cookie** 对象是属于 **CookieCollection** 集合对象中的成员，所以我们可以自行在程序中宣告并

使用这些对象；这样的好处可以避免和一些系统所自动产生的 **Cookie** 变量混杂。下列范例以上

述范例为基础，改为自定的 **Cookies** 对象来操作 **Cookie** 变量：

```
<Html>

<Script Language="VB" Runat="Server">

Sub Page_Load(Sender As Object, e As EventArgs)

    Dim CookieCollection As HttpCookieCollection=New HttpCookieCollection

    Dim Cookie1 As HttpCookie=New HttpCookie("Cookie1")

    Dim Cookie2 As HttpCookie=New HttpCookie("Cookie2")

    Cookie1.Value="Microsoft VisualStudio .NET"

    Cookie2.Value="ASP.NET"

    CookieCollection.Add(Cookie1)    ' 将 Cookie 对象加入 Cookies 集合中

    CookieCollection.Add(Cookie2)

    Dim shtI As Short

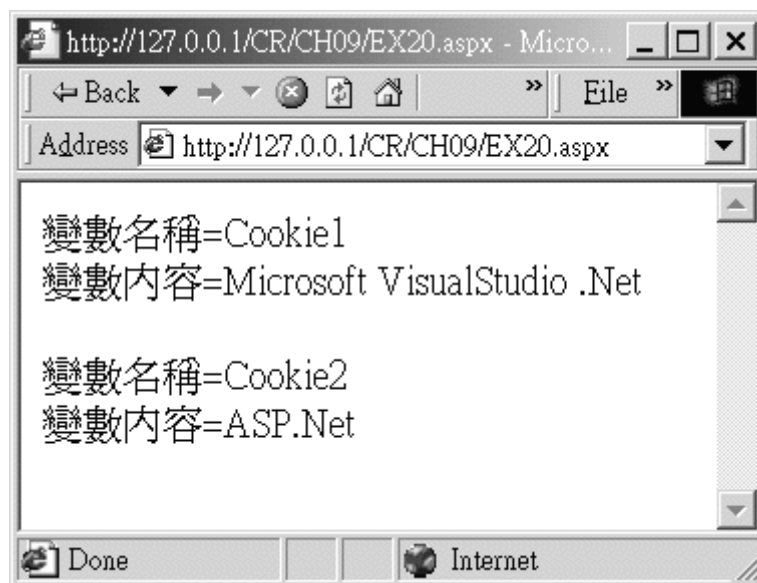
    For shtI=0 To CookieCollection.Count-1

        Response.Write("变量名称=" & CookieCollection.Item(shtI).Name &
" <br>")

        Response.Write("变量内容=" & CookieCollection.Get(shtI).Value &
" <p>")

    Next
```

```
CookieCollection.Clear()  
  
End Sub  
  
</Script>  
  
</Html>
```



上述范例中分别产生一个 `HttpCookieCollection` 对象以及两个 `HttpCookie` 对象。当我们在产生

`HttpCookie` 对象时至少要将 `Name` 属性设定好，或者以下列的语法产生：

```
变量=New HttpCookie("Cookie 名称","数据")
```

当设产生 **Cookie1** 及 **Cookie2** 这两个变量后，我们就可以使用 **CookieCollection** 对象的 **Add** 方法将 **Cookie** 对象加入集合中；接着在循环中分别读出他们的变量名称及内容。我们观察执行结果，发现系统 **Cookie** 变量 **AspSessionId** 就没有出现在我们所自订的 **Cookies** 集合对象中了。

设定 **Cookie** 变量的生命周期

Cookie 变量虽然存放在 **Client** 端机器上，却也不是永远不会消失的。系统预设给 **Cookie** 变量的有效时间是 1000 分钟，不过我们可以在程序中自行设定有效日期，只要指定 **Cookie** 变量的 **Expires** 属性即可。使用语法如下所示：

```
Response.Cookies(CookieName).Expires=#日期#
```

若我们没有指定 **Expires** 属性，则 **Cookie** 变量将不会被储存，会像 **Session** 一样浏览器关闭结束浏览便被毁灭。不过 **Cookie** 一旦设定有效期限后，除非我们将 **Expires** 属性设为「dbNull」，否则有日期期限的 **Cookie** 无法被移除。所谓「dbNull」值代表「空」值，「空」的意思是什么都没有；所以有设定有效日期的 **Cookie** 就可以被移除。下列范例在使用者登入后，在一个月浏览页都不需要再登入；并且每次登入时，程序自动将 **Cookie** 有效期限往登入日期后延长一个月：

```
<Html>

<ASP:Panel Id="Pan1" Runat="Server">

<Form Runat="Server">
```

```
<Table>
```

```
<Tr>
```

```
<Td>账号: </Td>
```

```
<Td><Asp:TextBox Id="txtID" Runat="Server" /></Td>
```

```
</Tr>
```

```
<Tr>
```

```
<Td>密码: </Td>
```

```
<Td><Asp:TextBox TextMode="Password" Id="txtPassword"
```

```
Runat="Server" /></Td>
```

```
</Tr>
```

```
</Table>
```

```
<ASP:Button Id="btnSubmit" Text="确定" OnClick="btnSubmit_Click"
```

```
Runat="Server"/>
```

```
<ASP:Button Id="btnReset" Text="清除" OnClick="btnReset_Click"
```

```
Runat="Server"/>
```

```
<ASP:Label Id="Label1" Text="请输入账号及密码" Runat="Server"/>
```

```
</Form>
```

```
</ASP:Panel>
```

```
<ASP:Panel Id="Pan2" Runat="Server">
```

```
Hi! <ASP:Label Id="lblMsg" Runat="Server"/> ,欢迎光临
```

```
</ASP:Panel>
```

```
<Script Language="VB" Runat="Server">
```

```
Sub Page_Load(Sender As Object, e As EventArgs)
```

```
    If Request.Cookies.Item("MyWeb_UserID")=DBNull Or _
```

```
        Request.Cookies.Item("MyWeb_UserID").Value="" Then
```

```
        Pan2.Visible=False
```

```
    Else
```

```
Response.Cookies.Item("MyWeb_UserID").Expires=Now.AddMonths(1)
```

```
    lblMsg.Text=Request.Cookies.Item("MyWeb_UserID").Value
```

```
    Pan1.Visible=False
```

```
    End If
```

```
End Sub
```

```
Sub btnSubmit_Click(Sender As Object, e As EventArgs)
```

```
    If txtID.Text="charles" and txtPassword.Text="1234" Then
```

```
        Response.Cookies("MyWeb_UserID").Value=txtID.Text
```

```
        Response.Cookies.Item("MyWeb_UserID").Expires=Now.AddMonths(1)
```

```
        Pan1.Visible="False"
```

```

        Pan2.Visible="True"

        lblMsg.Text=txtID.Text

    End If

End Sub

Sub btnReset_Click(Sender As Object, e As EventArgs)

    txtID.Text=""

    txtPassword.Text=""

End Sub

</Script>

</Html>

```

上述程序代码范例中我们使用两个 **Panel**，分别为 **Pan1** 以及 **Pan2**；**Pan1** 为要求使用者输入账号及密码，而 **Pan2** 则为欢迎语。程序执行时若使用者的 **Cookie** 不存在或没有数据，将欢迎语隐藏；如下列程序代码片段所示：

```

Sub Page_Load(Sender As Object, e As EventArgs)

    If Request.Cookies.Item("MyWeb_UserID")=DBNull Or _

        Request.Cookies.Item("MyWeb_UserID").Value="" Then

        Pan2.Visible=False

    Else

```



```
Response.Cookies.Item("MyWeb_UserID").Expires=Now.AddMonths(1)

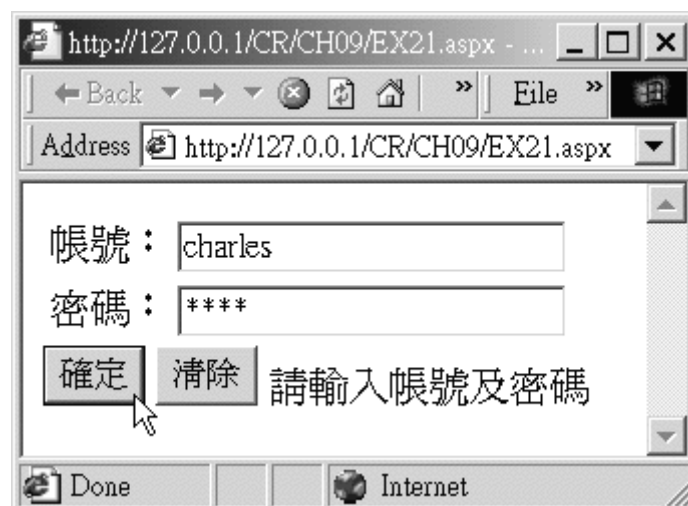
    lblMsg.Text=Request.Cookies.Item("MyWeb_UserID").Value

    Pan1.Visible=False

End If

End Sub
```

上列程序代码片段中我们先判断 **Cookie** 对象是否存在，以及确定 **Cookie** 中是有内容；如果 **Cookie** 不存在或是 **Cookie** 内没有资料，则显示 **Pan1** 要求使用者登入的画面；倘若 **Cookie** 存在，则显示 **Pan2** 出现欢迎语：



使用者若输入正确的使用者名称后，我们便将使用者账号写入 **Cookie** 中，并指定有效期限为一个
个月内；如下程序代码片段所示：

```
Sub btnSubmit_Click(Sender As Object, e As EventArgs)

    If txtID.Text="charles" and txtPassword.Text="1234" Then

        Response.Cookies("MyWeb_UserID").Value=txtID.Text

        Response.Cookies.Item("MyWeb_UserID").Expires=Now.AddMonths(1)

        Pan1.Visible="False"

        Pan2.Visible="True"

        lblMsg.Text=txtID.Text

    End If

End Sub
```

下次使用者再浏览网页的时候，只要在一个月内有登入过，就不需要再输入使用者账号及密码，
而直接出现下列画面：



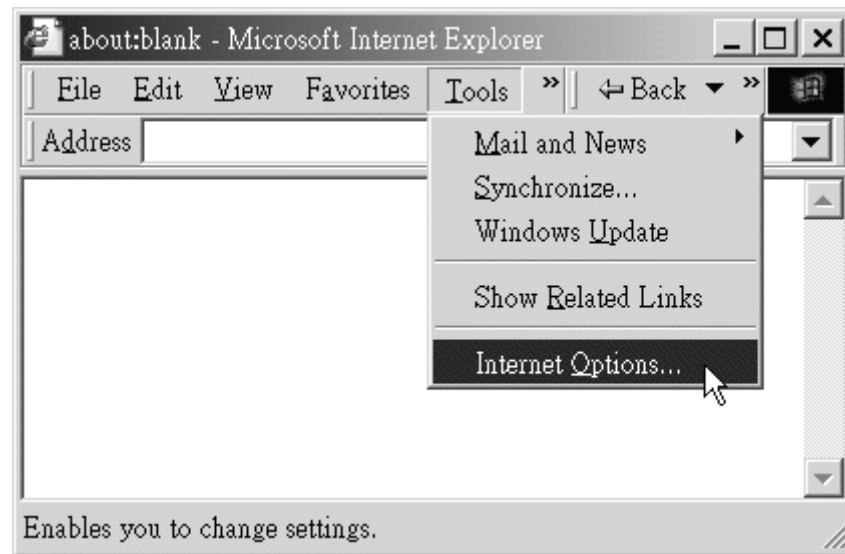
Cookie 验证的安全考量

在使用 **Cookie** 验证使用者时，必须要考虑到身分验证的问题。因为使用者可能在非私人的计算机上浏览，或是个人计算机的安全防护不完善；导致其它人可能使用同一个浏览器上站，这样一来任何人都可以顺利的通过 **Cookie** 的验证。对于有机密考量的数据或是有价交易的处理，势必造成漏洞；设计这方面的网站时，要小心仔细考量 **Cookie** 的应用。

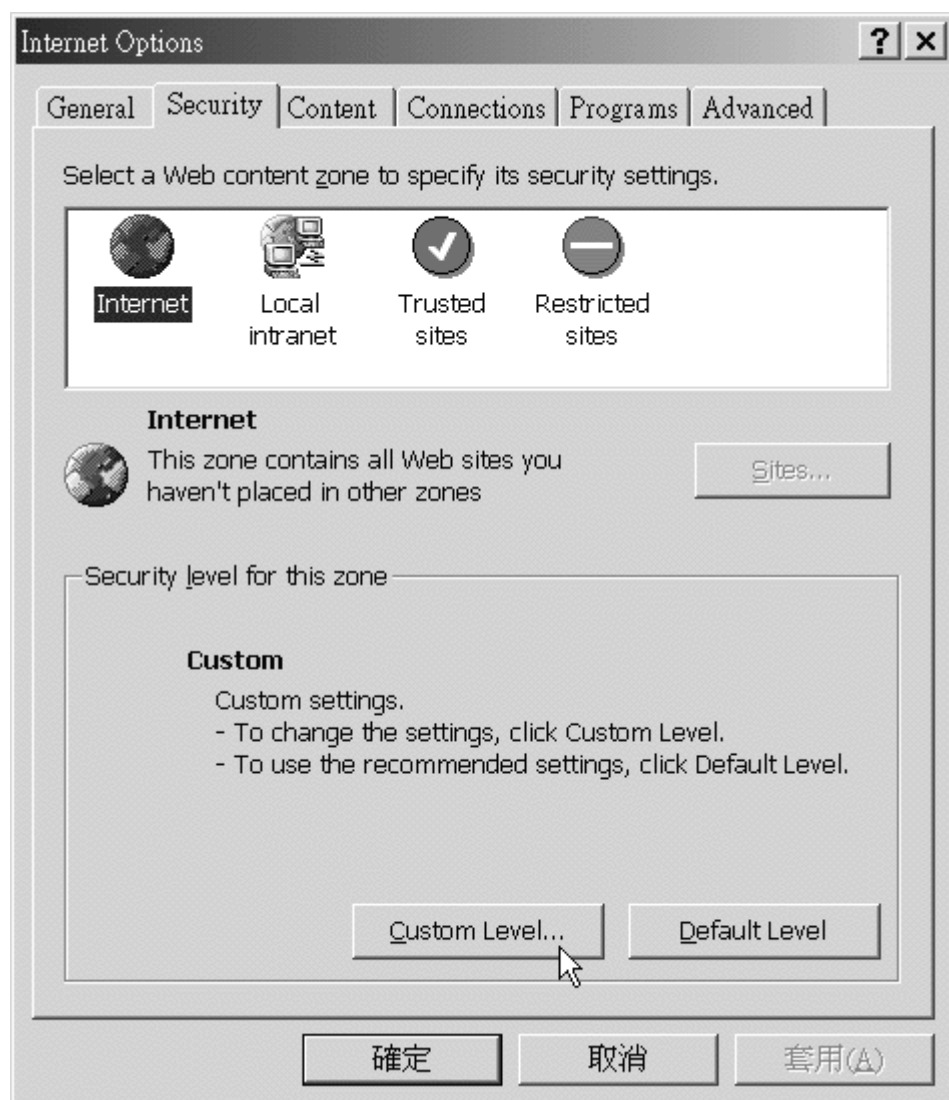
Session 和 Cookie 的关系

以前的 **Session** 对象使用时要打开浏览器的 **Cookie** 支持，IE 的默认值是开启。检查 **Cookie** 的功能是否被关闭的步骤如下：

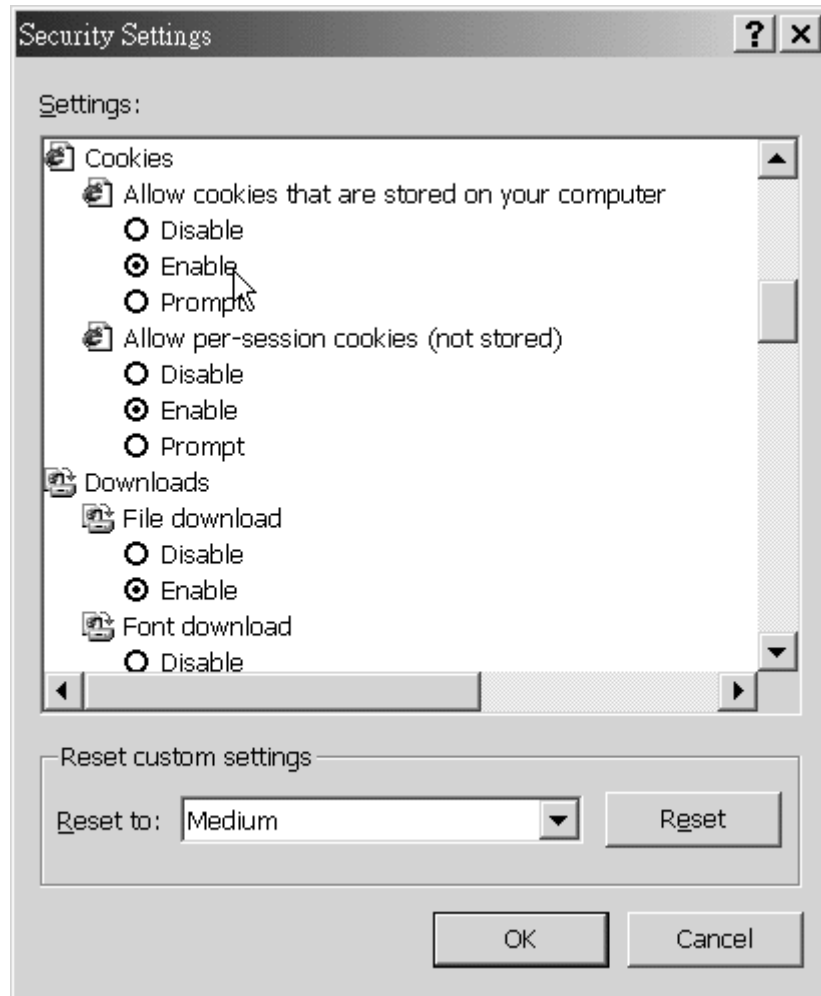
1. 选择 IE 的「Tools」选单，点选「Internet Options」选项：



2. 按下「Security」页签里的「Custom Level...」按钮:



3. 确认「Security Settings」窗口内「Cookies」群组里的「Allow cookies that are stored on your computer」和「Allow per-session cookies (not stored)」的 Enable 选项是否被核取，若无选取则在选取后按「OK」离开。



不过在 ASP.NET 中就算不开启浏览器的 Cookie 支持也可以让 Session 动作，只要在 Config.web

档案中加入 < **sessionstate cookieless="true"** / > 这个叙述就可以让 Session 不再依赖

Cookie；但是此时的 Config.web 文件一定要放在 www 的根目录里，也就是预设的

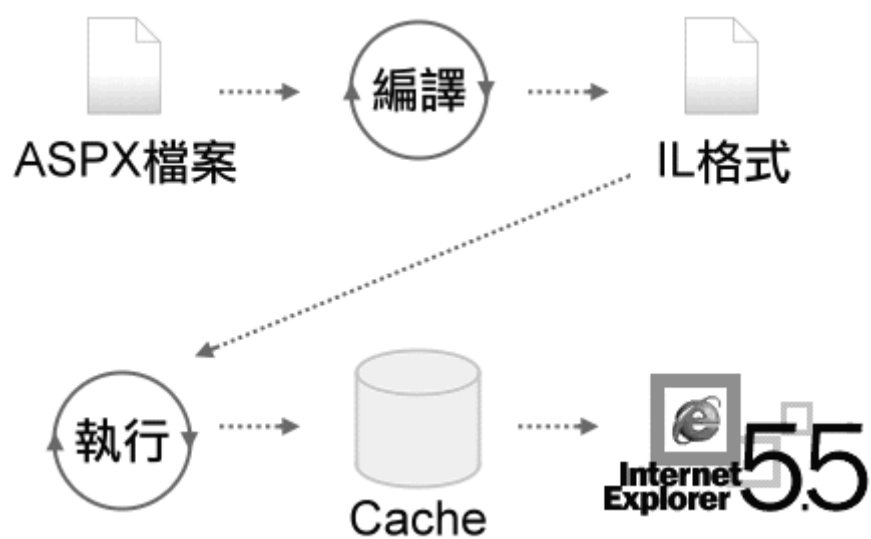
C:\inetpub\wwwroot 里。

Cache 物件

Cache（快取）是将输出的数据存在内存中，当要输出数据时直接由内存取得数据做输出；由于是从内存中抓取数据，所以效率也就提升了。ASP.NET 的快取方式有两种，分别是 Output Cache 以及 Data Cache。

Output Cache（输出快取）

我们在第一章中提到当我们第一次执行 aspx 网页时，整个程序会先被编译为 IL 格式后再执行；其实在执行 IL 和最后输出网页之间还可以开启 Cache 的功能。当 aspx 网页第一次被执行时，会先被编译成 IL 格式后再执行，然后将执行的结果储存在 Output Cache 中再下载至客户端的浏览器。



等到以后网页要输出时，只要原网页内容没有被修改及快取中有数据，就可以直接从 **Cache** 中直接将网页的内容下载给使用者；当然就会提升执行的效率。输出快取对象 **Cache** 是属于 **Response** 对象的成员。**Cache** 对象正确的对象类别名称为 **HttpCachePolicy**，其常用的属性及方法如下表所示：

属性或方法	说明
SetExpires 属性	设定 Cache 的有效时间。
SetCacheability 方法	设定 Cache 的有效范围。
SetNoServerCaching 方法	当使用这个方法后，Server 将会停止对目前的输出作 Cache 的动作。

要启动输出快取只要在网页的开头加入下列宣告：

```
<%@ OutputCache Duration="秒数"%>
```

其中 **Duration** 代表这个快取会在内存中存在的时间，在这段时间之内一律使用快取内的网页数据，直到超过所设定的时间才会执行更新的动作。**Output Cache** 也可以利用程序来控制。因为 **Cache** 对象属于 **Response** 对象，所以要使用 **Cache** 只要以「**Response.Cache.属性或方法**」的语法即可使用 **Output Cache**。所以如果我们要设定输出快取的有效时间，其使用语法如下所示：

```
Response.Cache.SetExpires(时间)
```

另外 **SetCacheability** 可以设定 **Cache** 的有效范围，其使用语法如下所示：


```
Response.Cache.SetCacheability(HttpCacheability.Public |  
HttpCacheability.Private)
```

倘若传入的参数如为 **HttpCacheability.Public**，表示所有的 **Client** 端都可以使用这个 **Cache**；

HttpCacheability.Private 则只有这个联机可以使用。下列范例我们设定 **Cache** 有效时间为三秒，

也就是说在这三秒内按下重新整理后，其输出的时间是一样的：

```
<Html>  
  
<Form Runat="Server">  
  
目前时间：<ASP:Label Id="lblTime" Runat="Server" />  
  
</Form>  
  
<Script Language="VB" Runat="Server">  
  
Sub Page_Load(Sender As Object, e As EventArgs)  
  
    Response.Cache.SetExpires(DateTime.Now.AddSeconds(3))  
  
    Response.Cache.SetCacheability(HttpCacheability.Public)  
  
    lblTime.Text=Format(Now(), "hh:mm:ss")  
  
End Sub  
  
</Script>  
  
</Html>
```

Data Cache（资料快取）

Data Cache 可以提升网页执行的效能。**Data Cache** 是针对单一的变量或对象，使用方式和 **Session** 对象类似。由于它也是 **Page** 对象的属性成员之一，所以不用宣告就可以直接使用，**Data Cache** 对象正确的对象类别名称是 **Cache** 对象。其常用的属性如下表所示：

属性	说明	型态
Count	传回 Cache 变数的数量。	Integer
Item	以变量名称来设定或传回 Cache 变量的值。	Item(ByVal key As String) As Object
MaxItems	传回或设定 Cache 变量的最大数量，如果新增的数量超过这个上限，则系统会将较旧的 Cache 变量移除，以挪出空间来存放新值。	Integer

其常用的方法如下表所示：

方法	说明	语法
Get	以变量名称来传回变数的值。	Cache Get(ByVal key As String) As Object
Insert	新增一个 Cache 变数。	1. Insert(ByVal key As String,ByVal value As Object)

		2. Insert(ByVal key As String,ByVal value As Object,ByVal dependencies As CacheDependency)
Remove	以变量名称来移除一个 Cache 变数。	Remove(ByVal key As String) As Object

数据快取的使用语法如下所示：

```
Cache("变量名称")="变量内容"
```

下列范例我们分别用几种不同的方法来新增及显示 **Cache** 变量：

```
<Html>

<Script Language="VB" Runat="Server">

Sub Page_Load(Sender As Object,e As EventArgs)

    Cache("Cache1")="Value1"

    Cache.Insert("Cache2","Value2")

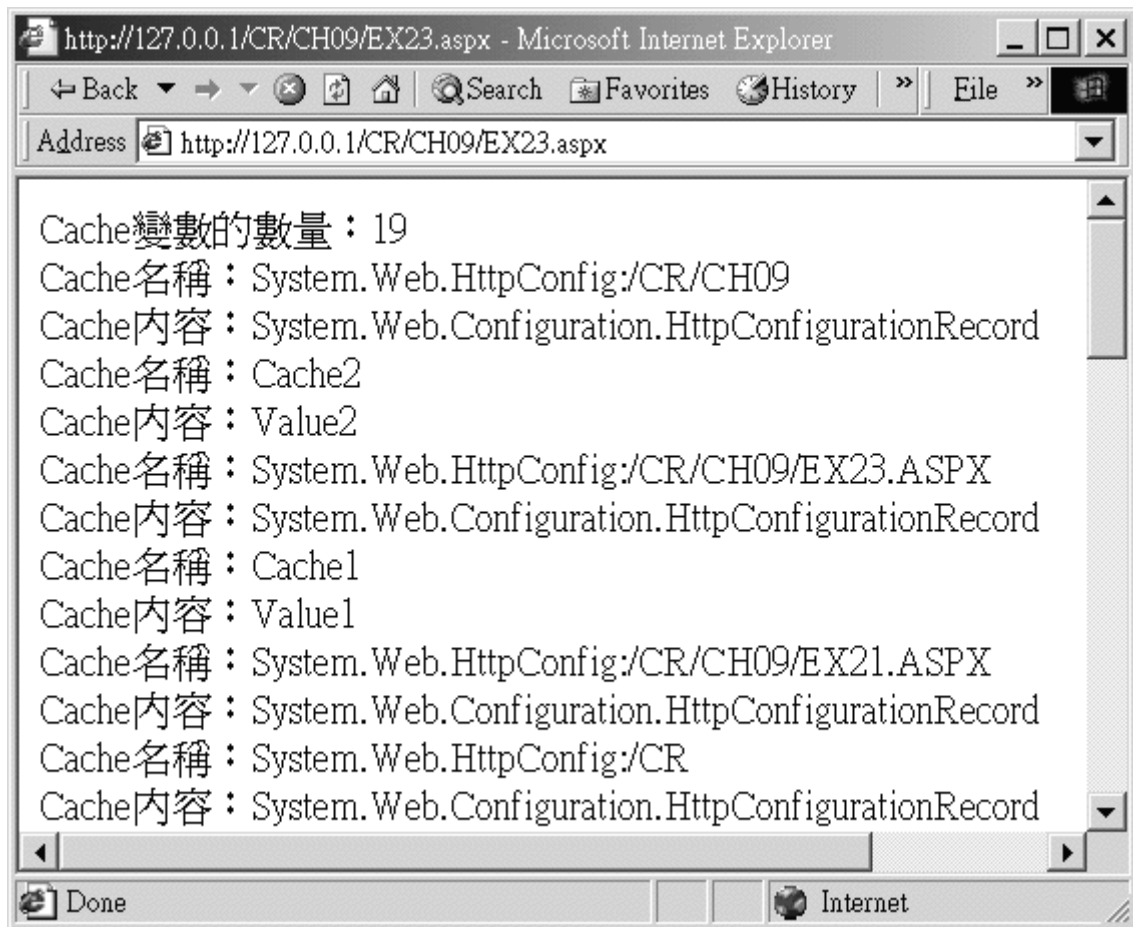
    Dim obj As Object

    Response.Write("Cache 变数的数量： " & Cache.Count & "<br>")

    For Each obj In Cache

        Response.Write("Cache 名称： " & obj.Key & "<br>" & _
```

```
Cache 内容: " & Cache(obj.Key).ToString &  
"  
<br>")  
Next  
End Sub  
</Script>  
</Html>
```



上述程序中我們利用 **For Each....Next** 來印出 **Cache** 中的變數。我們觀察上述的執行結果發現畫面出現許多不是我們所定義的 **Cache** 變量，這是因為 **Cache** 中有許多由系統所使用的變量（以 **System** 開頭的皆是）。

10. 进阶议题

除错（Debug）

错误的种类

我们在开发程序的时候难免会发生错误，这些错误有时是程序设计师在编写程序代码时不小心打错，有时是客户端使用者因操作不当而引发的；即使再小心的程序设计师也不可能完全避免程序出错。因此如何避免错误、找出错误便成了程序设计中不可缺少的一环。程序的错误大致可分为三种，分别为语法错误、执行时期错误，以及逻辑错误。

语法错误（Syntax Error）

程序在撰写的时候没有按照规定的语法就会出错。这种错误较为常发生在初学者，例如关键词拼错、有 **If** 却忘了加 **Then**、字符串没有用双引号围起来等，都会引发语法错误；但这一类错误会随着对程序语言的熟练度而渐渐减少。

执行时期错误（Runtime Error）

程序在执行时所发生错误即为执行时期错误。例如以 0 作为除数导致程序无法继续执行，如下列

程序代码片段所示：

```
X=1/0
```

这个程序的语法并没有错误。但是 0 不可为除数，因此执行到这一行时便会引发执行时期错误。

要解决执行时期错误必须另外加入错误处理程序，有关错误处理的方法我们在后面章节再讨论。

逻辑错误（Logic Error）

程序执行的结果不是我们所预期的，便称为逻辑错误。这可能是因为程序设计师的观念就本身就

不正确，所以这种错误并不好发现。因为程序的语法内容并没有错，要解决这类的问题必须配合

一些工具和方法，才能找出错误的地方。

以程序代码来除错

在程序代码中加入除错用的程序代码来除错是最直接的方法。以逻辑错误来说，因程序本身并没有任何语法错误存在，大多是变量的内容或程序执行流程上出了问题；因此将程序执行过程中的变量值显示出来可以帮助我们了解变量的变化，进而找出程序的问题出在哪里。下列范例以程序代码来显示变量的内容：

```
<Html>

<ASP:Label Id="Label1" Runat="Server" />

<Script Language="VB" Runat="Server">

Sub Page_Load(Sender As Object, e As EventArgs)

    Dim I As Short

    Dim J As Short

    J=10

    For I=0 To 10

        I=I+1

    Next

    Response.Write("I: " & I.ToString & "<p>")    ' 将变数 I 的值印出
```

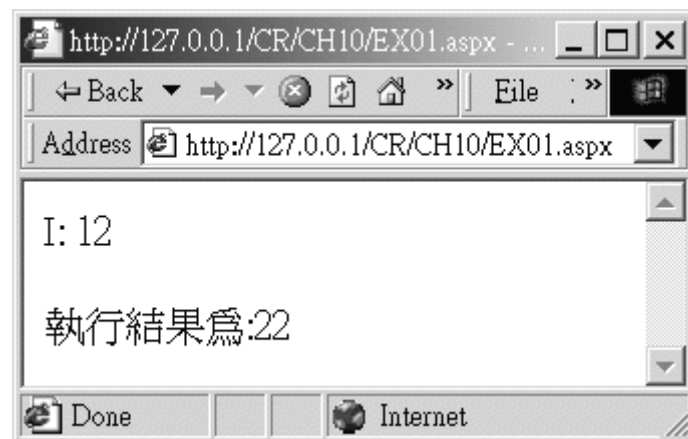
```
J=J+I

Label1.Text="执行结果为:" & J.ToString

End Sub

</Script>

</Html>
```



上述例子程序在执行时一点问题也没有，我们预期最后的显示值是 20，可是却出现 22 这个不符

合我们答案的数字；这时我们可以利用 **Response.Write** 方法显执行完循环后的变量 I 的内容。

这个范例很单纯的植入检查变量值的叙述，在某些单纯的网页上使用有其便利性；不过程序代码

一但复杂，所要检视的元素也会相对的成长，所以我们使用一些 **ASP.NET** 提供的追踪程序方法较方便。

观察错误讯息

我们故意撰写一个错误的程序，然后以浏览器浏览：

```
<Html>

<Form Runat="Server">

<ASP:Button Id="Button1" Text="Click" OnClick="Button1_Click"

Runat="Server" /><P>

<ASP:Label Id="Label1" Runat="Server" /><P>

</Form>

<Script Language="VB" Runat="Server">

Sub Button1_Click(Sender As Object, e As EventArgs)

    Label1.Text=2/0

End Sub

</Script>
```

</Html>



由上图我们可以清楚的了解程序的错误类型以及错误的位置。上面这个程序在编译时期就出错

误，原因是我们在运算时使用 0 当除数，所以发生除 0 错误。

Trace 功能

在对程序进行除错工作的同时，都希望能够有系统的整理这些除错讯息。可是以往的网页程序设计并不提供这样的功能，必须使用一堆如 `Response.Write()` 方法来显示所需要信息，既不清楚又没有效率。现在 **ASP.NET** 提供了一个新的对象来方便我们除错，那就是 **Trace** 对象。**Trace** 对象是 **Page** 对象的成员，因此不需另行宣告就可直接使用，它的正确类别名称为 **TraceContext**。**Trace** 对象会列出整个网页的每一个请求等讯息，并且将这些信息整理好。要使用 **Trace** 对象，首先要在网页的开头处加上如下的宣告：

```
<%@ Page Trace="True"%>
```

如下面这个程序：

```
<%@Page Trace="True"%>

<Html>

<Form Runat="Server">

    <ASP:TextBox Id="Text1" Runat="Server"/>×

    <ASP:TextBox Id="Text2" Runat="Server"/>

    <ASP:Button Id="Button1" Text="=" OnClick="Button1_Click"

Runat="Server"/>

    <ASP:Label Id="Label1" Runat="Server"/><p>
```

```
</Form>
```

```
<Script Language="VB" Runat="Server">
```

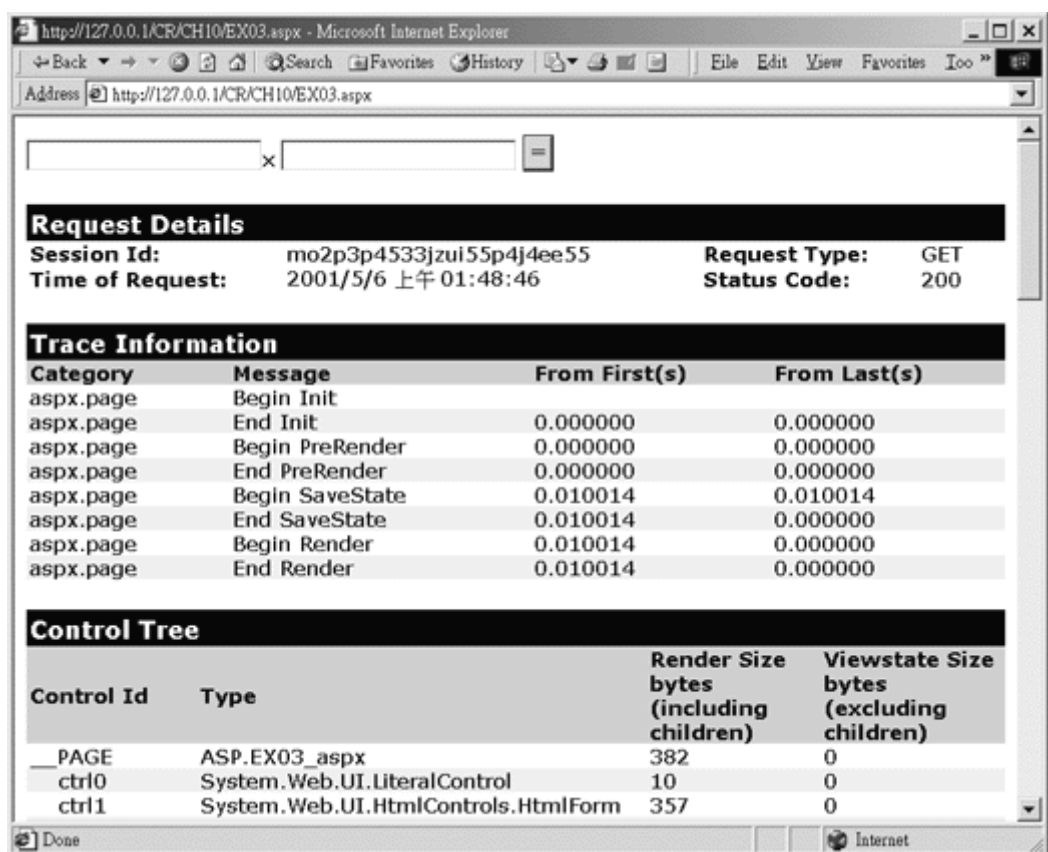
```
Sub Button1_Click(Sender As Object, e As EventArgs)
```

```
    Label1.Text=Cint(Text1.Text) * Cint(Text2.Text)
```

```
End Sub
```

```
</Script>
```

```
</Html>
```



我们会看到网页除了有我们所放的控件之外还多了一些信息，下表是 Trace 对象后各个标题的说明：

标题	说明
Request Details	显示本机和 Server 联机的识别码、此网页被处理的时间以及请求的型态。
Trace Information	当我们使用 Trace 对象时；欲显示字符串的区域。
Control Tree	列出全部的控件并显示他们的大小。
Cookies Collection	Cookie 集合的全部内容。

Headers Collection	显示 Client 端浏览器的信息。
Server Variables	一些有关 Server 端信息的变量。

Trace 对象的常用属性如下表所示：

属性	说明
IsEnabled	设定目前的网页中 Trace 的功能是否被致能，我们可以使用这个属性判断是否需要输出 Trace 的信息；当使用 <@ Page Trace="True"> 指令时这个属性便为 True。
TraceMode	<p>TraceMode 共有三种设定值：</p> <ol style="list-style-type: none"> 1. Default：默认值。相当于 SortByTime，以时间来排序。 2. SortByCategory：将 Trace 对象送出的讯息按种类及文字顺序排序。 3. SortByTime：Trace 对象的讯息按照命令被处理的先后显示。

Trace 对象的常用方法如下表所示：

方法	说明
Write	输出 Trace 讯息。其语法为 Write(Trace 种类字符串,Trace 讯息内容) 。

Warn	和 Write 一样，但是显示在页面的颜色为红色。其语法为 Warn(Trace 种类字符串,Trace 讯息内容) 。
------	---

下列范例利用 **Trace** 对象的 **Warn** 方法送出追踪讯息：

```
<%@Page Trace="True"%>

<Html>

<Form Runat="Server">

    <Asp:TextBox Id="Text1" Runat="Server"/>×

    <Asp:TextBox Id="Text2" Runat="Server"/>

    <Asp:Button Id="Button1" Text="=" OnClick="Button1_Click"

Runat="Server"/>

    <Asp:Label Id="Label1" Runat="Server"/><p>

</Form>

<Script Language="VB" Runat="Server">

Sub Button1_Click(Sender As Object, e As EventArgs)

    Trace.Warn("Trace 讯息", "Click 事件开始")

    Trace.Warn("Text1 内容", Text1.Text)
```

```
Trace.Warn("Text2 内容", Text2.Text)
```

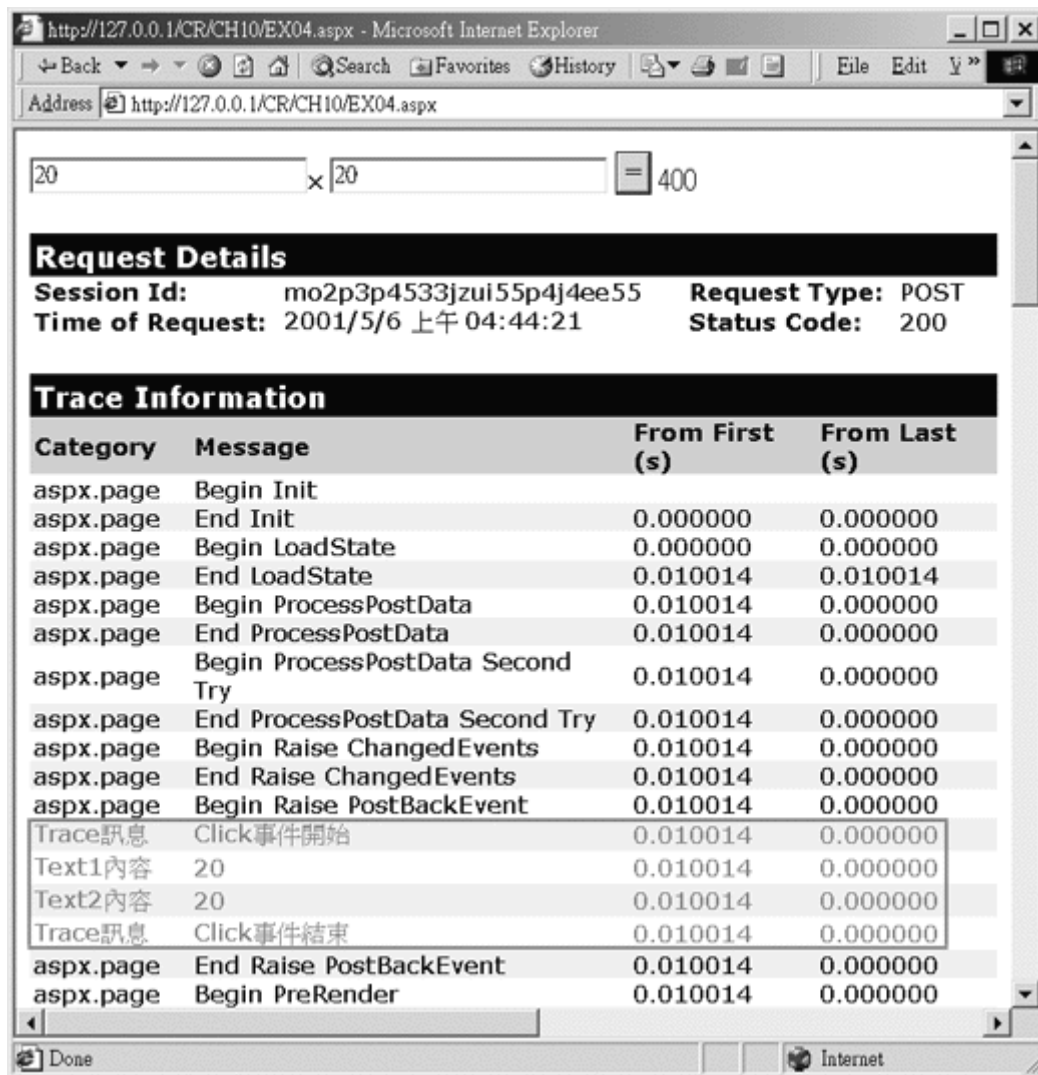
```
Label1.Text=Cint(Text1.Text)*Cint(Text2.Text)
```

```
Trace.Warn("Trace 讯息", "Click 事件结束")
```

```
End Sub
```

```
</Script>
```

```
</Html>
```



上述范例我们在 Button1 的 Click 事件开始处加入 `Trace.Warn("Trace 讯息 ", "Click 事件开始")`，然后在事件结束位置加上 `Trace.Warn("Trace 讯息 ", "Click 事件结束")`。接着我们按下 Button1 后，发现在 Trace Information 的区块中多了我们刚刚所自订的讯息。当我们不再需要 Trace 对象的讯息时，并不需要逐一的去删除；只要将 `<@Page Trace="True">` 从网页中移除，则 Trace 对象的信息将不会出现。而程序中的 Trace 叙述可以在以后使用，非常方便。

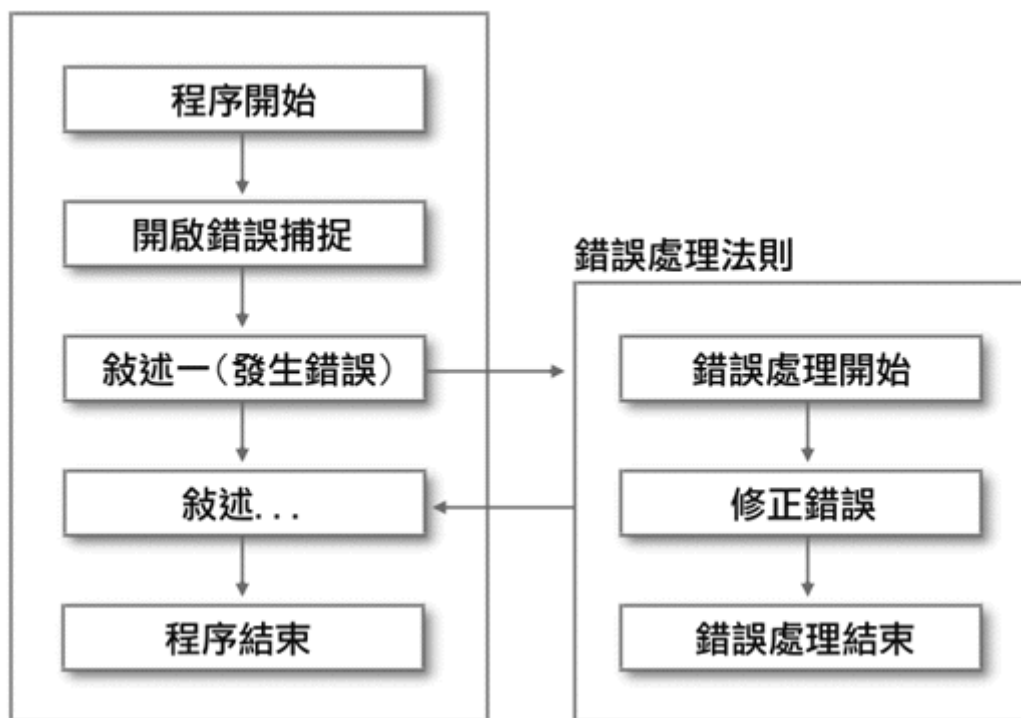
错误处理程序

前面所提的 **Trace** 的确是很方便，不过它只能针对程序开发时或维护时使用。而在执行程序时常因为使用者操作错误或程序写作不周详，而发生发生执行时期错误（**Runtime Error**）。此时如果让使用者的浏览器显示错误的讯息，那么对于网站的形象就会造成负面的影响。所以网页开发时我们必须假设所有使用者会发生的错误，并且撰写程序来处理这些错误；这个处理错误的程序称为「错误处理法则」。VB.NET 支持了两种错误处理法则，一个是从以前的 VB 就支持的 **On Error Goto** 叙述，另一个是全新的 **Try...Catch** 叙述。这两种比较起来以 **Try...Catch** 较结构化，但是 **On Error Goto** 的使用较为简单，两者各有利弊。

On Error Goto 叙述

OnError 错误处理法则大约分为三个步骤：

1. 开启错误捕捉码，指定错误发生时应该执行哪些程序代码。
2. 撰写修正错误的程序。
3. 结束错误处理继续执行正常的程序。



在介绍 On Error Goto 叙述之前我们先了解 Err 对象。当程序发生错误时，所有和错误相关的讯息都会被纪录在这个对象里；我们能利用 Err 对象来检视目前发生错误的错误讯息、错误代码及简述，或是用来引发一个自订的错误。Err 对象的属性如下所示：

属性	说明
Description	错误的内容说明。
Number	目前发生错误的代码。

Source	可以用来存放一些额外的信息，例如在对象中引发一个错误时可以注明错误的出处。
--------	---------------------------------------

Err 对象的方法如下所示：

方法	内容
Clear	将 Number 属性清除为 0。
Raise	引发一个错误。

致能错误捕捉

接着我们要介绍如何建立错误处理法则。首先我们在程序的开始处输入「On Error Goto 标注」

来致能错误捕捉的功能，并指定程序发生错误时程序代码要转移的错误处理法则。下列程序代码

片段将错误捕捉打开，并指定程序发生错误时会跳到标签「ErrorHandler」来执行错误处理法则：

```
Sub 程序名称(Sender As Object, e As EventArgs)

On Error Goto ErrorHandler

...

```

```
Exit Sub
```

```
ErrorHandler:
```

```
错误处理码...
```

```
...
```

```
跳出错误处理法则
```

```
End Sub
```

上述程序代码片段中，我们输入叙述 **OnError Goto...** 开启错误捕捉，并指定发生错误时跳至

ErrorHandler 标签处执行错误处理法则；卷标的宣告只要在卷标名后加上冒号「:」即可。我们

在程序的最后面的地址将错误处理法则加入，并且在错误处理法则标号前加 **Exit Sub** 叙述；这

是因为程序是逐行执行的，倘若没有发生任何错误还是会跳入错误处理法则执行错误处理的工

作。所以我们在错误处理法则标号前加入 **Exit Sub** 叙述，以避免没有发生错误时跳入错误处理

法则中执行错误处理码；如果我们执行的程序是 **Function** 程序时，在错误处理法则前要加入的

叙述为 **Exit Function**。

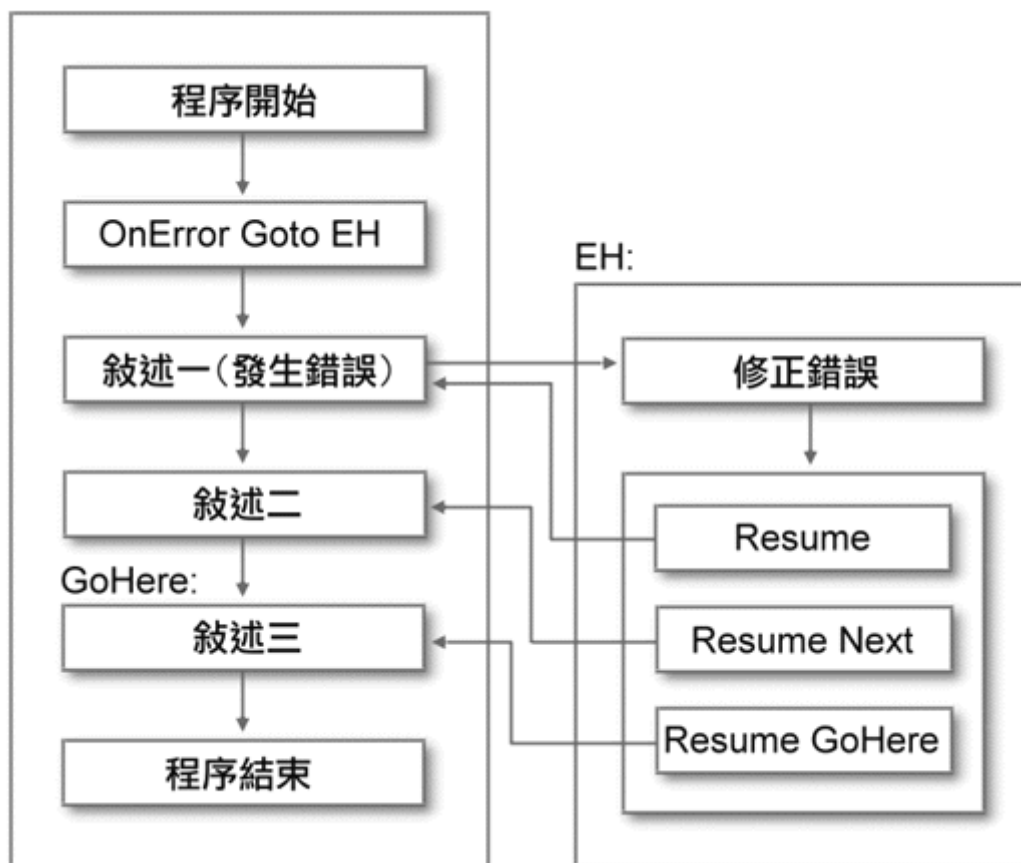
跳出错误处理法则

在错误处理法则中程序的执行遇到 **End Sub** 或 **End Function** 叙述时便结束程序的执行。若我们

要让程序回到程序中继续执行，则要使用 **Resume** 叙述。**Resume** 叙述的使用有三种方式，如

下表所示：

叙述	说明
Resume	回到发生错误的那一行继续执行。
Resume Next	回到发生错误的下一行继续执行。
Resume 行号/标签	跳至指定的行号或标签继续执行。



在 ASP.NET 中使用第一种 **Resume** 时要特别注意，这种 **Resume** 叙述最主要用在一般的

Windows 应用程序上，倘若在 **ASP.NET** 中使用 **Resume** 叙述会导致无穷循环。因为 ASP.NET

回到错误的那一行后会将原本发生错误的程序代码再执行一次，接着又进入错误处理程序，如此

反复进行下去成为无穷循环。下列范例我们建立一个具有完整错误处理法则的网页：

```

<Html>

<Form Runat="Server">

    <ASP:TextBox Id="Text1" Runat="Server"/>X

    <ASP:TextBox Id="Text2" Runat="Server"/>

```

```

    <ASP:Button Id="Button1" Text="" OnClick="Button1_Click"
Runat="Server"/>

    <ASP:Label Id="Label1" Runat="Server"/><p>

    <ASP:Label Id="Label2" Runat="Server"/>

</Form>

<Script Language="VB" Runat="Server">

Sub Page_Load(Sender As Object,e As EventArgs)

    Label1.Text=""

    Label2.Text=""

End Sub

Sub Button1_Click(Sender As Object,e As EventArgs)

    On Error Goto ErrHandler '将错误捕捉打开

    Label1.Text=Cint(Text1.Text)/Cint(Text2.Text)

Exit Sub

ErrHandler:

    If Err.Number=13 Then '如果发生错误码为 13

        Label2.Text=Err.Description & "<br>两边都必须输入数字!"

```

```
Resume Next ' 回到发生错误的下一行

End If

End Sub

</Script>

</Html>
```



当 **Text1** 及 **Text2** 中都输入数值则显示正常的结果，但如果其中任一个 **TextBox** 输入非数值型态的数据便引发一个错误；此时程序会跳到 **ErrorHandler** 卷标处执行错误处理法则。而非数字型态的值做型态转换会引发代码为 **13** 的错误，所以我们可以使用 **If** 叙述判断，当 **Err** 对象的 **Number** 属性为 **13** 时，**Label2** 将会显示「两边都必须输入数值」，接着回到发生错误的下一行继续执行。

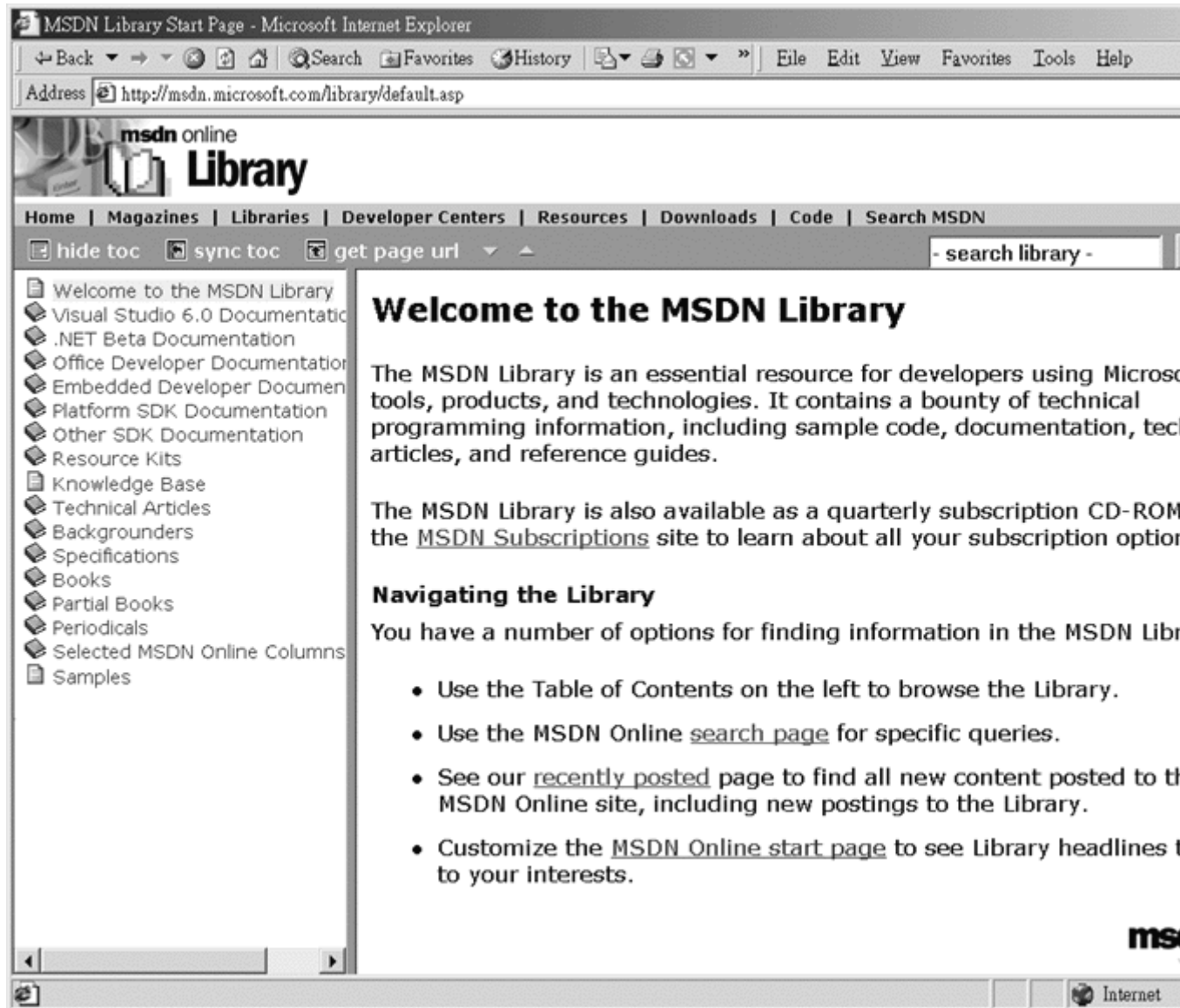
Tips

关于错误码以及其错误的讯息，如果你有 MSDN (Microsoft Developer Network) Library 光盘，

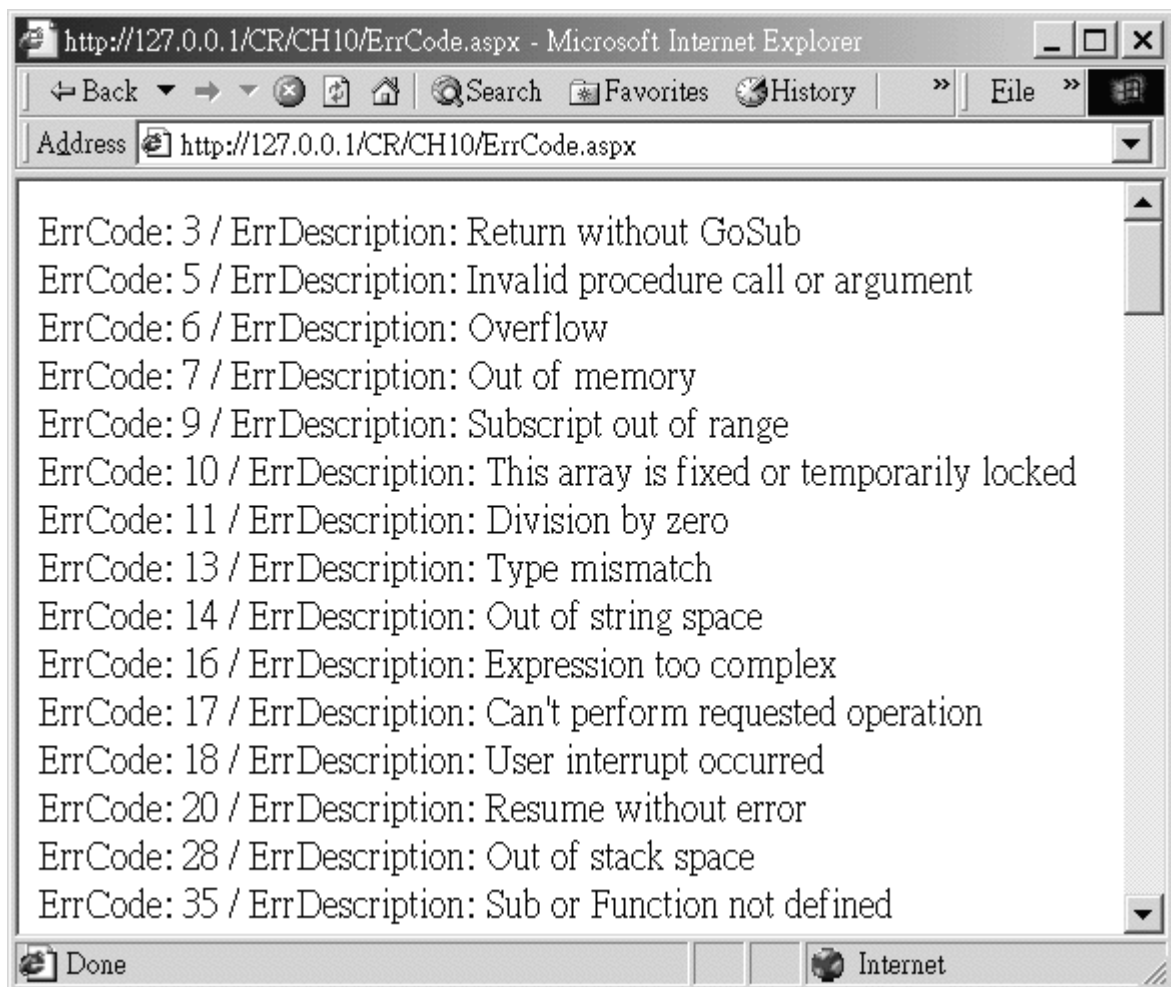
可以查询其中的「Trappable Errors」议题，或是浏览微软 MSDN 的官方网站

「[Http://msdn.microsoft.com/library/default.asp](http://msdn.microsoft.com/library/default.asp)」在线查询，或是执行 ErrCode.aspx 范例程序

可以列出错误码以及错误描述。



MSDN Online Library



错误码以及错误描述

Try...Catch...Finally 和 Throw 叙述

介绍完了 On Error 叙述后，相信大家对于错误处理的方式及流程应该有基础的认识了。接着我

们要来介绍 .NET 架构里最新的错误处理方式 Try...Catch。我们前面提到 On Error 虽然简单但

是不够结构化，因此 .NET 便加入了 Try...Catch 的除错方式，使我们能够更有效率的撰写错误处理法则。

例外 (Exception)

错误被称为例外(Exception)，而例外也是对象；它和 Err 对象不一样，不同的例外被做成不同的对象。例如将非数值型态的数据以型态转换函数处理时，会发生 **FormatException** 型态的例外；而提供这个例外信息的是 **FormatException** 对象。

Try...Catch

Try...Catch 的概念基本上和 On Error 叙述一样，它也是当发生错误时就跳到例外处理程序中，其结构如下所示：

Try

程序执行的区块

Catch 变量 As 例外对象

例外处理程序的区块

```
End Try
```

我们可以针对不同的例外来建构不同的例外处理程序，例如：

```
Try
```

```
    程序代码
```

```
Catch 变量 As 例外对象
```

```
    例外处理程序代码
```

```
Catch 变量 As 例外对象
```

```
    例外处理程序代码
```

```
[Finally]
```

```
    例外处理程序代码
```

```
End Try
```

下列范例将 **On Error** 错误捕捉改成 **Try...Catch** 处理，我们一样只要输入非数值型态的内容至

TextBox 中即可引发错误：

```
<Html>
```

```
<Form Runat="Server">
```



```
<ASP:TextBox Id="Text1" Runat="Server"/>X

<ASP:TextBox Id="Text2" Runat="Server"/>

<ASP:Button Id="Button1" Text="=" OnClick="Button1_Click"
Runat="Server"/>

<ASP:Label Id="Label1" Runat="Server"/><p>

<ASP:Label Id="Label2" Runat="Server"/>

</Form>

<Script Language="VB" Runat="Server">

Sub Page_Load(Sender As Object, e As EventArgs)

    Label1.Text=""

    Label2.Text=""

End Sub

Sub Button1_Click(Sender As Object, e As EventArgs)

    Try

        Label1.Text=CInt(Text1.Text)/CInt(Text2.Text)

    Catch ex as FormatException

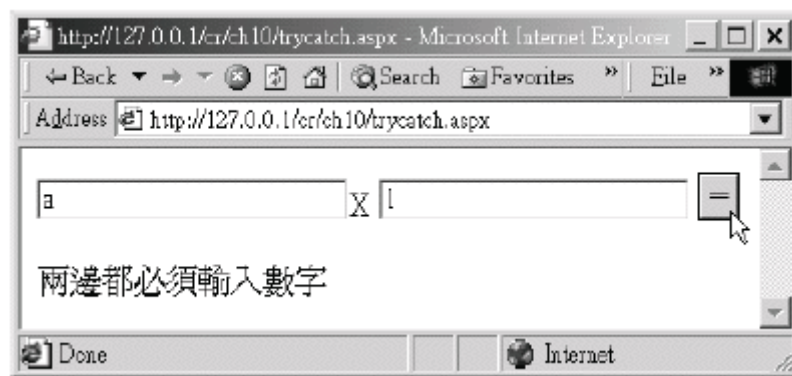
        Label2.Text="两边都必须输入数字"
```

```
End Try
```

```
End Sub
```

```
</Script>
```

```
</Html>
```



由于将非数值型态的内容传入 `CInt` 函数中态引发了 `FormatException` 类型的错误，因此就跳到

`Catch` 区块来处理例外。由于例外对象全部都是从 `Exception` 对象继承下来的，因此在使用多个

`Catch` 时要注意顺序。例如下面程序代码片段，我们若把 `Exception` 类别的对象放在

`FormatException` 类别对象之前，则侦测 `FormatException` 类型的例外处理将不会被执行：

```
Try
```

引发例外的程序代码

```
Catch err1 As Exception
```

例外处理的程序代码

```
Catch err2 As FormatException
```

例外处理的程序代码

```
End Try
```

Finally

Finally 区块不管有没有发生例外都会执行，我们可以使用 **Finally** 区块来做一些最后的处理。下

列范例当我们按下 **Button1** 程序执行到 **Try** 区块，无论例外有无发生都会执行 **Finally** 区块的中

的程序代码：

```
<Html>
```

```
<Form Runat="Server">
```

```
<ASP:TextBox Id="Text1" Runat="Server"/>X
```

```
<ASP:TextBox Id="Text2" Runat="Server"/>
```

```
<ASP:Button Id="Button1" Text="" OnClick="Button1_Click"
Runat="Server"/>

<ASP:Label Id="Label1" Runat="Server"/><p>

</Form>

<Script Language="VB" Runat="Server">

Sub Page_Load(Sender As Object, e As EventArgs)

    Label1.Text=""

    Label2.Text=""

End Sub

Sub Button1_Click(Sender As Object, e As EventArgs)

    Try

        Response.Write("Try 区块<Br>")

        Label1.Text=Cint(Text1.Text)/Cint(Text2.Text)

    Catch ex As FormatException

        Response.Write("Catch 区块<Br>")

    Finally

        Response.Write("Finally 区块")

    End Try

End Sub

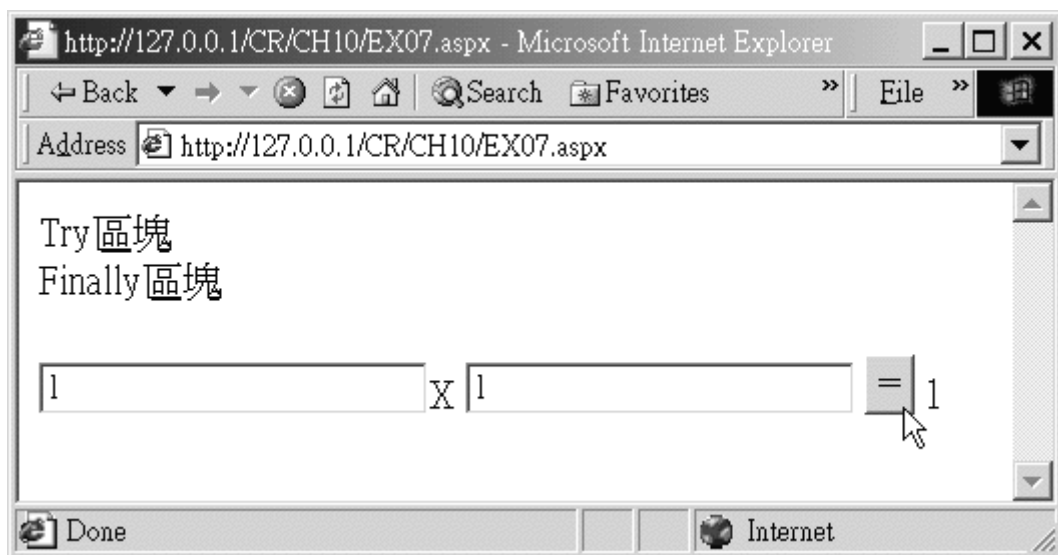
End Script
</asp:script>
</p>
</asp:label>
</asp:button>
</p>
</form>
</asp:form>
```

```
End Try
```

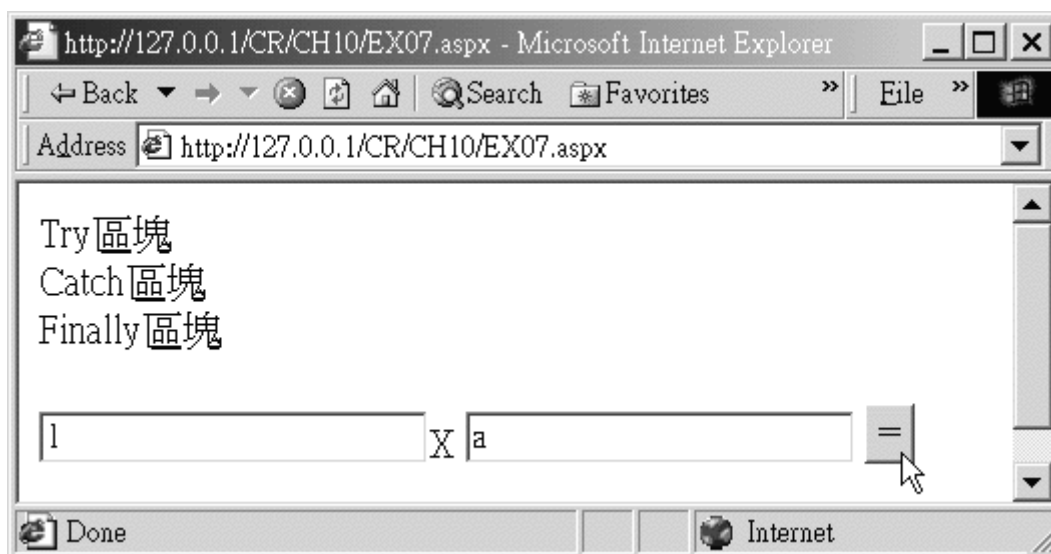
```
End Sub
```

```
</Script>
```

```
</Html>
```



没有发生例外时



发生例外时

Throw

在 **On Error** 结构中我们可以使用 **Err** 对象的 **Raise** 方法来自行引发一个例外，而 **ASP.NET** 新的

例外处理机制中也提供了 **Throw** 叙述，让我们引发一个自订的错误叙述；其使用语法如下：

```
Throw New 例外对象
```

下列范例中我们自行丢出一个 **Exception** 型别的例外，然后在 **Catch** 区块中印出 **Exception** 类别

对象的例外说明：

```
<Html>
```

```
<Form Runat="Server">
```

```
<ASP:Button Id="Button1" Text="产生例外" OnClick="Button1_Click"
```

```
Runat="Server" />
```

```
</Form>
```

```
<Script Language="VB" Runat="Server">
```

```
Sub Button1_Click(Sender As Object, e As EventArgs)
```

```
Try
```

```
Response.Write("自行丢出一个 Exception 例外<Br>")
```

```
Throw New Exception
```

```
Catch ex As Exception
```

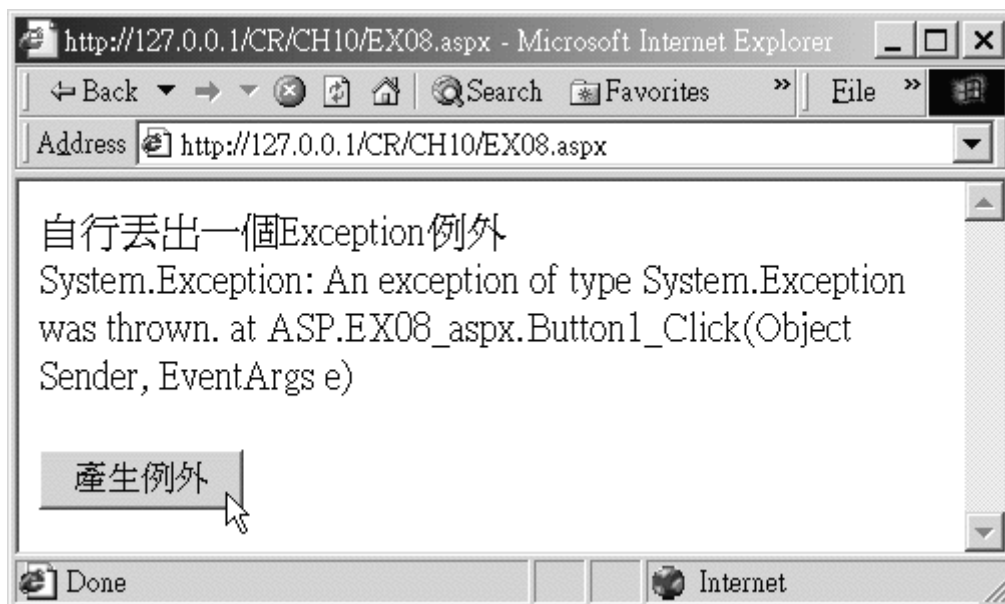
```
Response.Write(ex.ToString())
```

```
End Try
```

```
End Sub
```

```
</Script>
```

```
</Html>
```



Try....Catch 的结构化使得我们的例外处理变的相当有效率，我们可以在程序代码中使用

Try....Catch 结构来处里我们的例外，不用担心程序的执行流程被改变。使用 Try....Catch 要注

意不破坏原有的程序代码结构，例如下列程序代码片段是错误的示范：

```
Try
```

```
For shtI=0 To 9
```

```
    Response.Write(shtI)
```

```
    Catch ex As Exception
```

```
    ...
```


Next

巢状的 Try...Catch

我们可以在 **Catch** 区块或 **Finally** 区块中再使用 Try....Catch 叙述，如下列范例所示：

```
<Html>

<Form Runat="Server">

<ASP:Button Id="Button1" Text="产生例外" OnClick="Button1_Click"

Runat="Server"/>

</Form>

<Script Language="VB" Runat="Server">

Sub Button1_Click(Sender As Object, e As EventArgs)

    Try

        Response.Write("自行丢出一个 Exception 例外<P>")

        Throw New Exception

    Catch err As Exception

    Try
```

```
Response.Write(err.ToString() & "<P>")

Response.Write("再产生一个例外<P>")

Throw New IndexOutOfRangeException()

Catch err1 As IndexOutOfRangeException

Response.Write(err1.ToString())

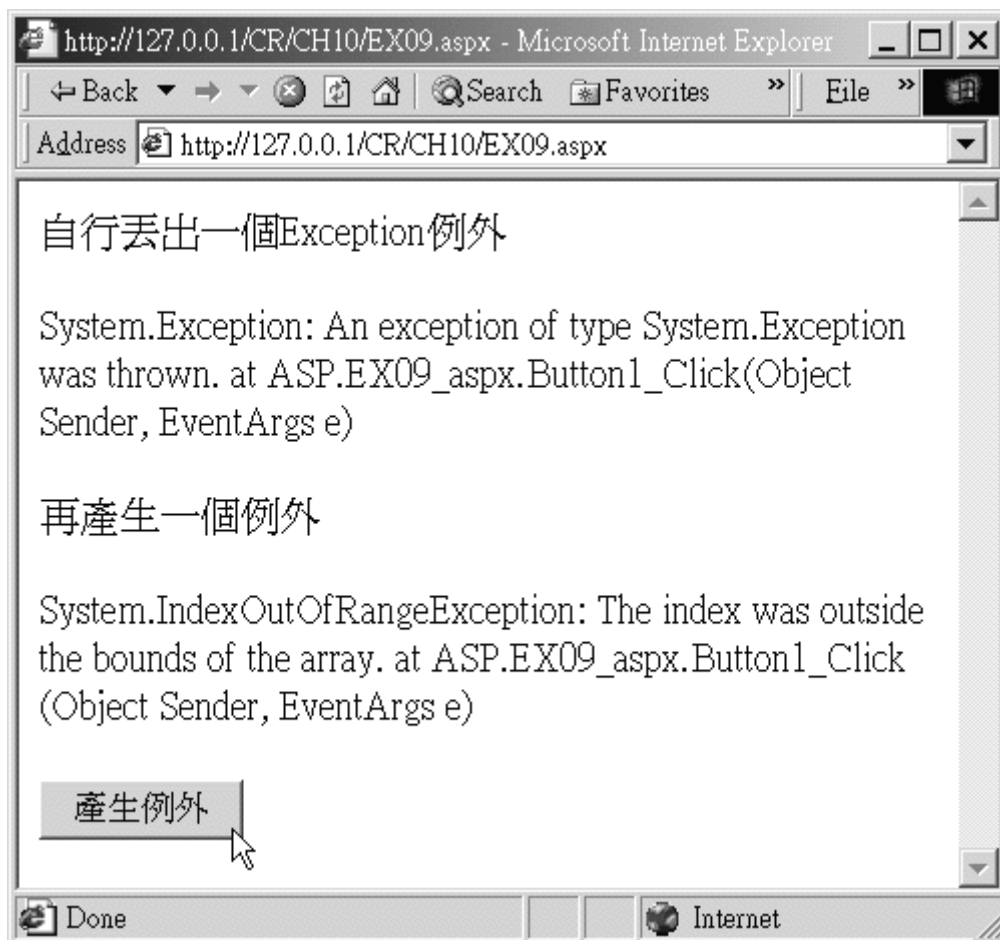
End Try

End Try

End Sub

</Script>

</Html>
```



常见的例外对象

下表为常见的例外对象：

例外对象	说明
ADOException	当使用 ADO 对象时若有错误时引发。
FormatException	变量的格式错误时引发。

IndexOutOfRangeException	存取数组时超出它的长度时引发。
NotSupportedException	当使用一个对象所不支持的方法时引发。
NullReferenceException	试图使用一个不存在的对象参考时引发。
OverflowException	当变量内容超过其所能存放的范围时引发。

各个 **Exception** 对象都有 **Message**，**Source**，**StackTrace** 属性，它们是从 **Exception** 对象继承

下来的；如下表所示：

属性	说明	型态
Message	例外的讯息。	String
Source	发生例外的来源。	String
StackTrace	传回错误的程序、行数	String

可视化的除错工具

.NET Framwork 中提供了一个相当友善好用的除错器（**Debugger**），这个图形化接口的工具能

帮我们更有效率的除错。

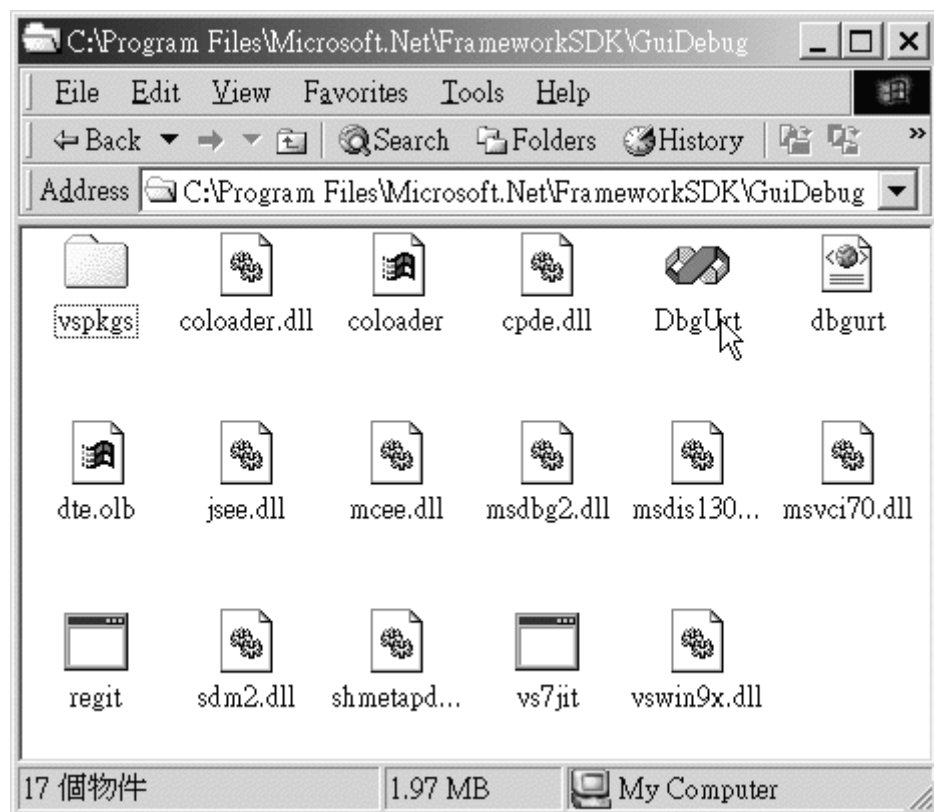
Microsoft .NET Framework SDK Debugger 简介

在使用除错器之前我们必须先在 **Config.web** 设定档中加入下列的设定，注意大小写：

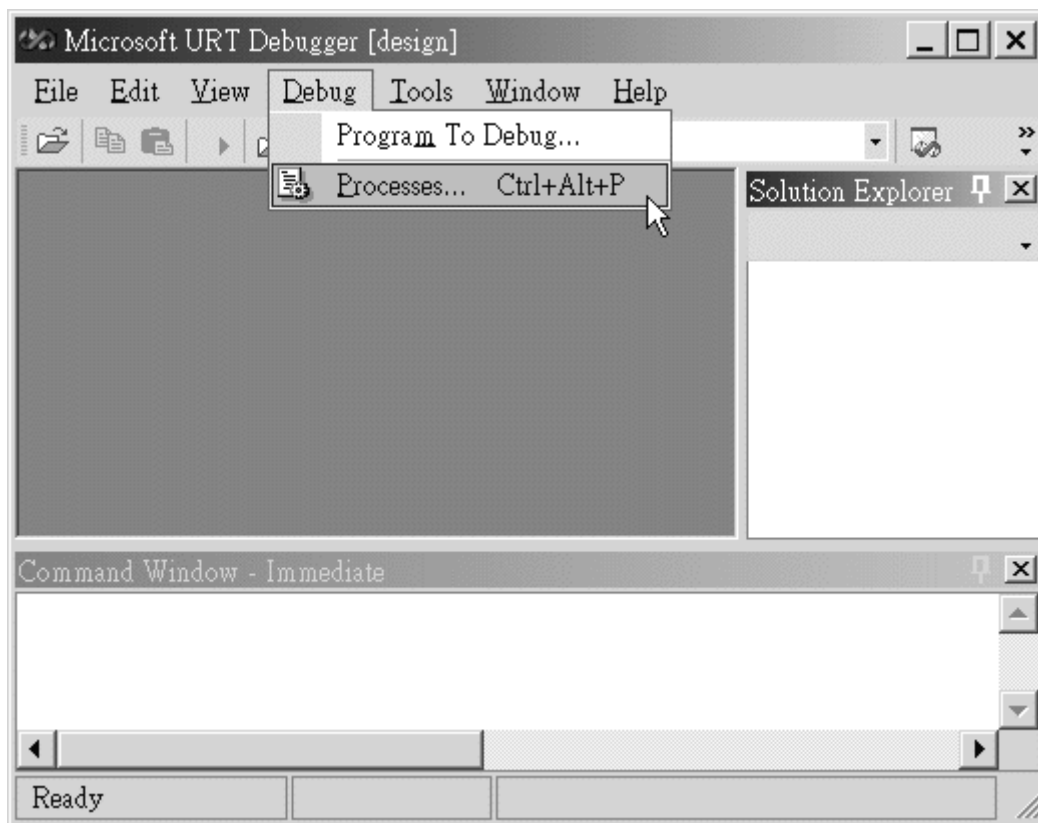
```
<compilation debugmode="true"/>
```

我们首先开启「C:\Program Files\Microsoft.NET\FrameworkSDK\GuiDebug」这个目录，接着

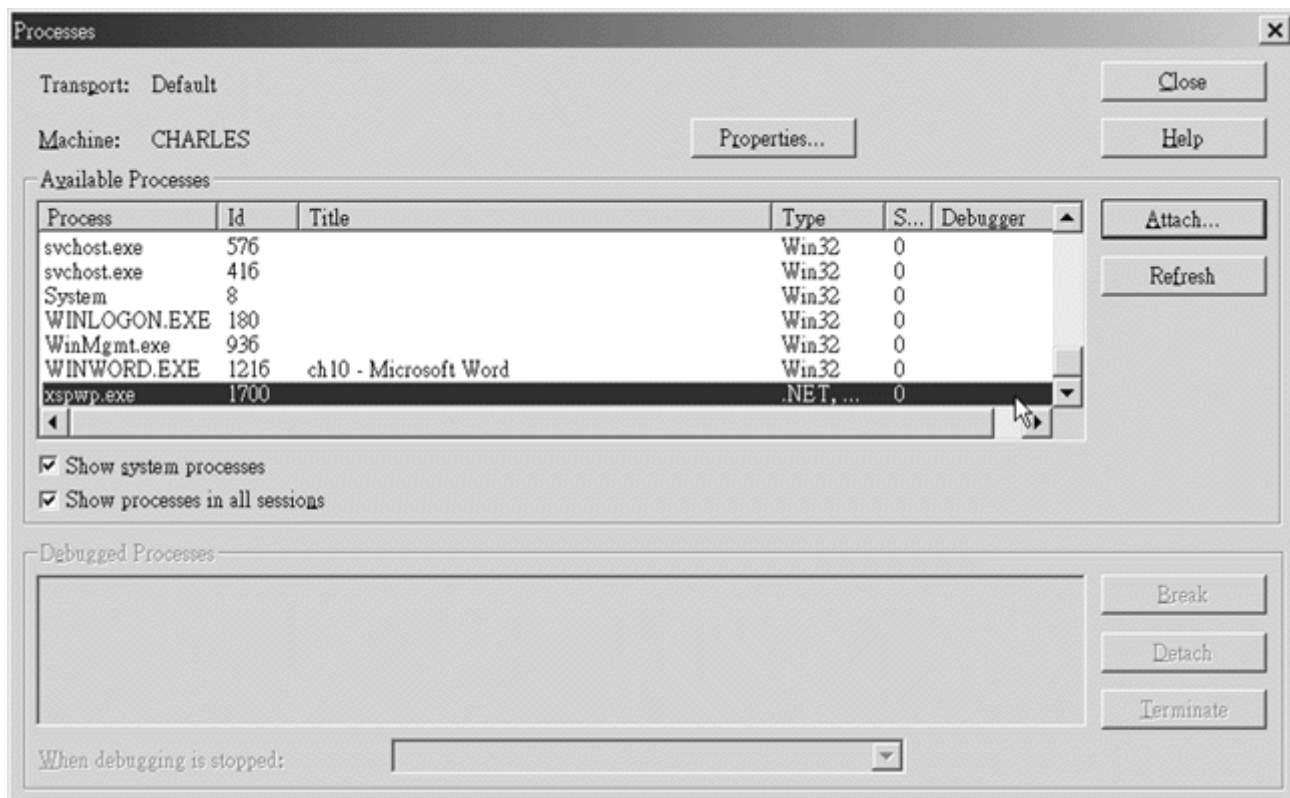
执行「DbgUrt.exe」。



然后我们在选单中选择「Debug」选单中的「Processes」选项。

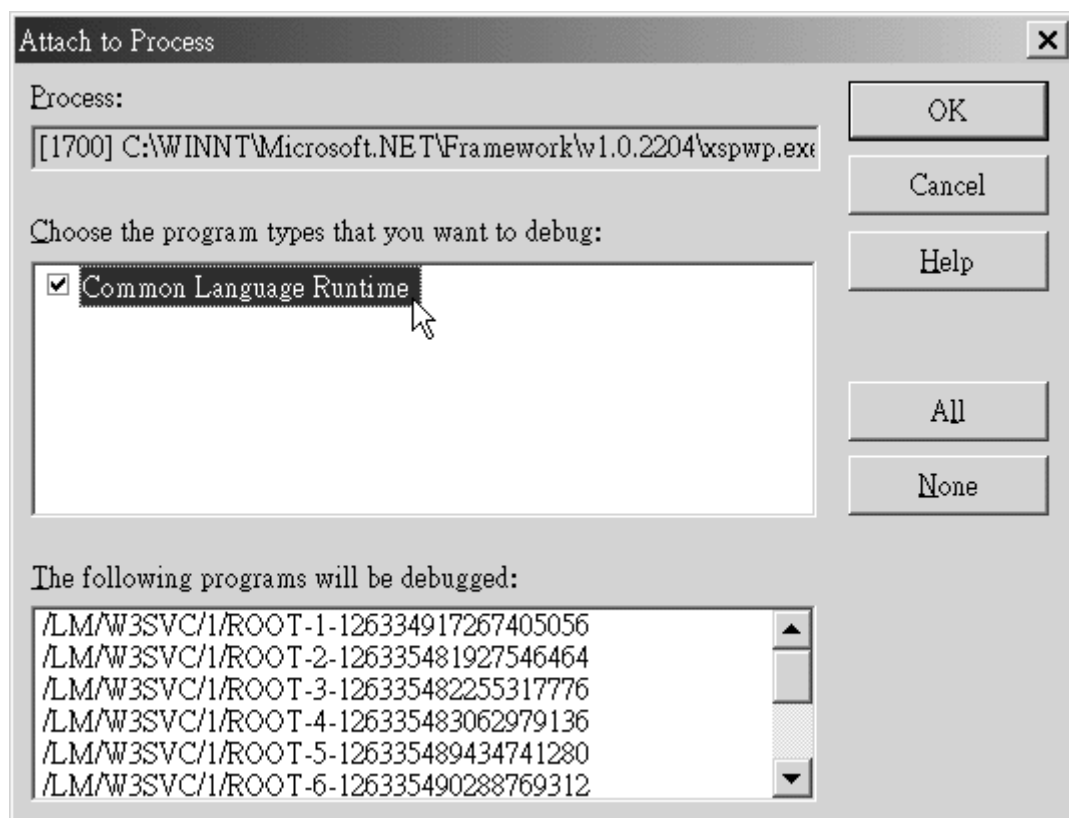


在 Processes 窗口中分别选取「Show system processes」、「Show processes in all sessions」，
并选择「xspwp.exe」这个选项。

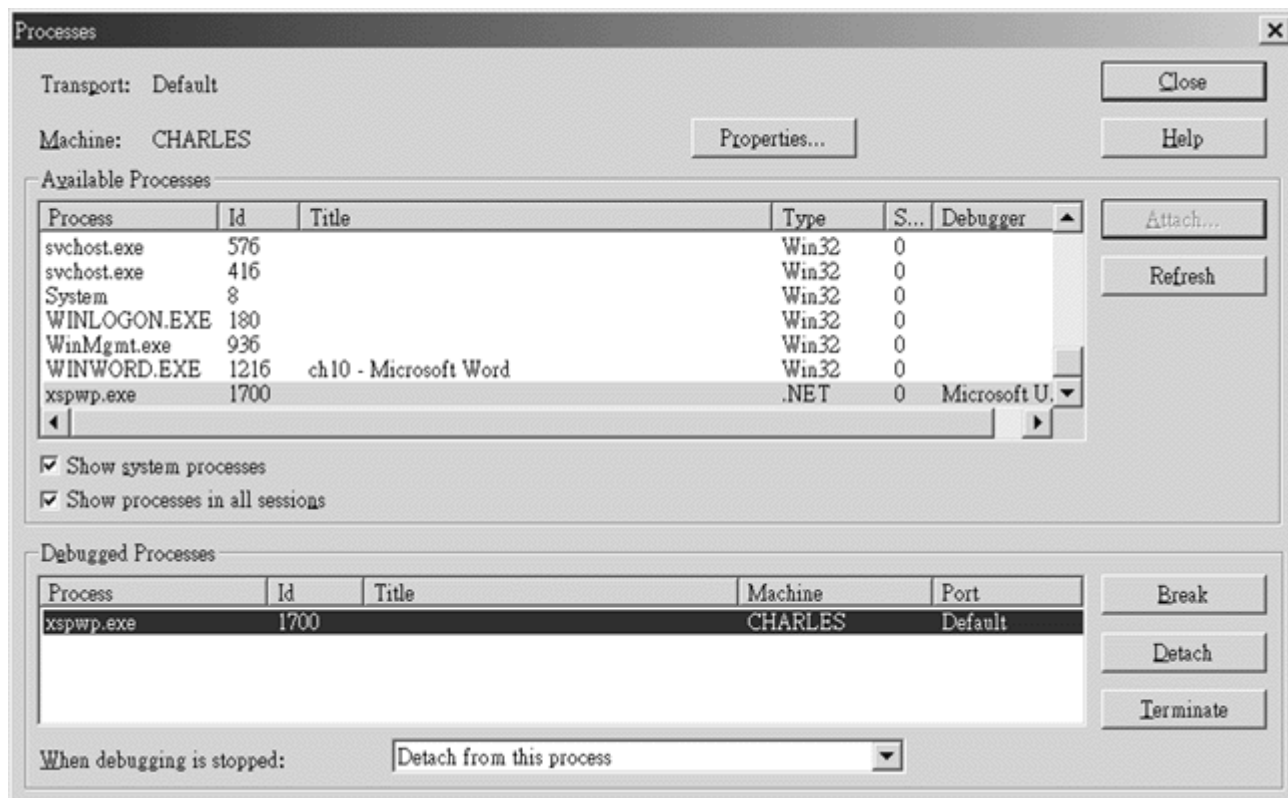


按下「Attach」按钮后会出现「Attach to Process」窗口，确认「Common Language Runtime」

选项被选取后按下 OK 按钮。



接着回到 **Processes** 窗口，我们可以发现 **Debugged Processes** 群组中多了我们刚刚加进去的 **xspwp.exe**。



xspwp.exe 是 ASP.NET 应用程序的工作程序，监视这个程序可以让我们在除错工具中，依序执

行程序中的每一个步骤、检视变量等。按下 Close 按钮后选择「File」选单中「Open」选项的

子选项「File」，来开启要进行除错的 EX10.aspx:

```
<Html>

<Form Runat="Server">

<ASP:Button Id="Button1" Text="确定" OnClick="Button1_Click"

Runat="Server"/>
```

```
</Form>

<Script Language="VB" Runat="Server">

Sub Button1_Click(Sender As Object, e As EventArgs)

    Dim I As Short

    Dim J() As Short={0,1,2}

    For I = 0 To 2

        Response.Write(J(I).ToString() & "<br>")

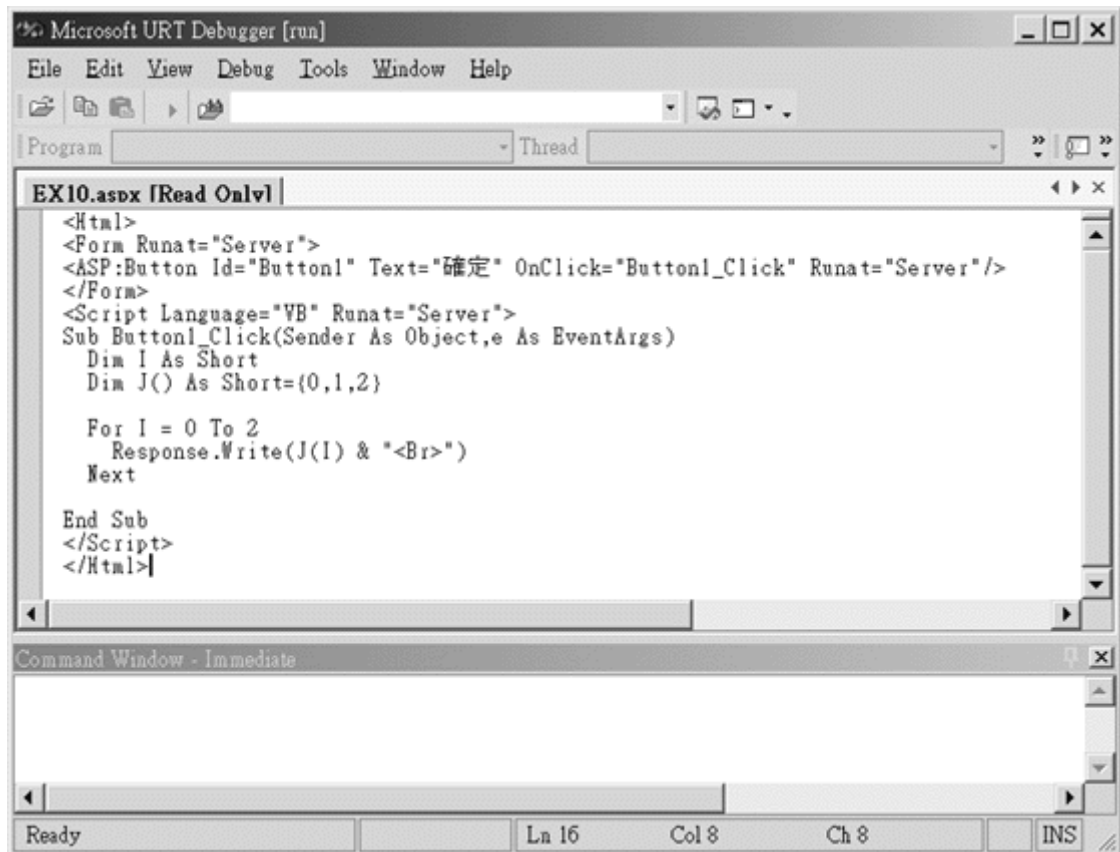
    Next

End Sub

</Script>

</Html>
```

在开启 EX10.aspx 后为了看的更清楚，以下的画面关掉 Solution Explorer 窗口。



断点是一种在除错模式时，告诉程序暂时停止执行的记号。接下来我们来为这个程序设定断点，

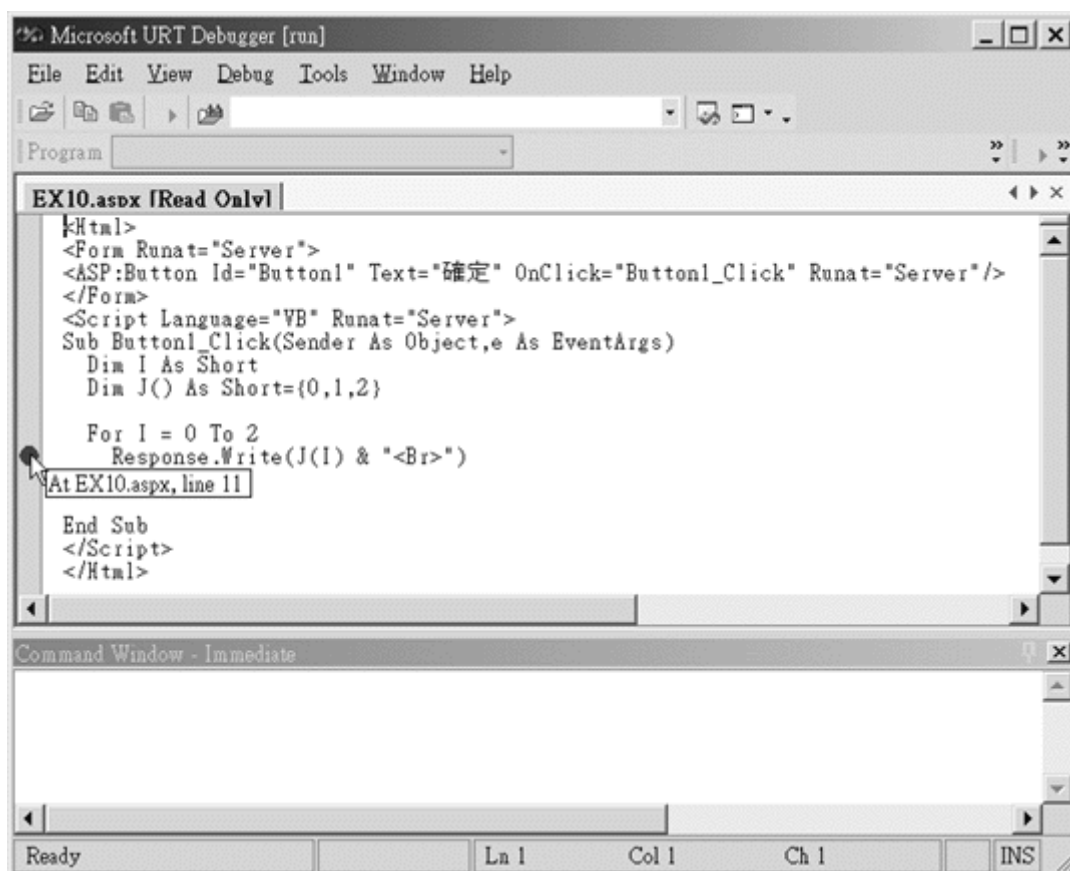
利用用除错工具来除错。首先我们先将键盘的光标停在程序代码中的「Response.Write(J(I) &

"
")」这一行，然后使用下列的任何方法都可以新增或移除断点：

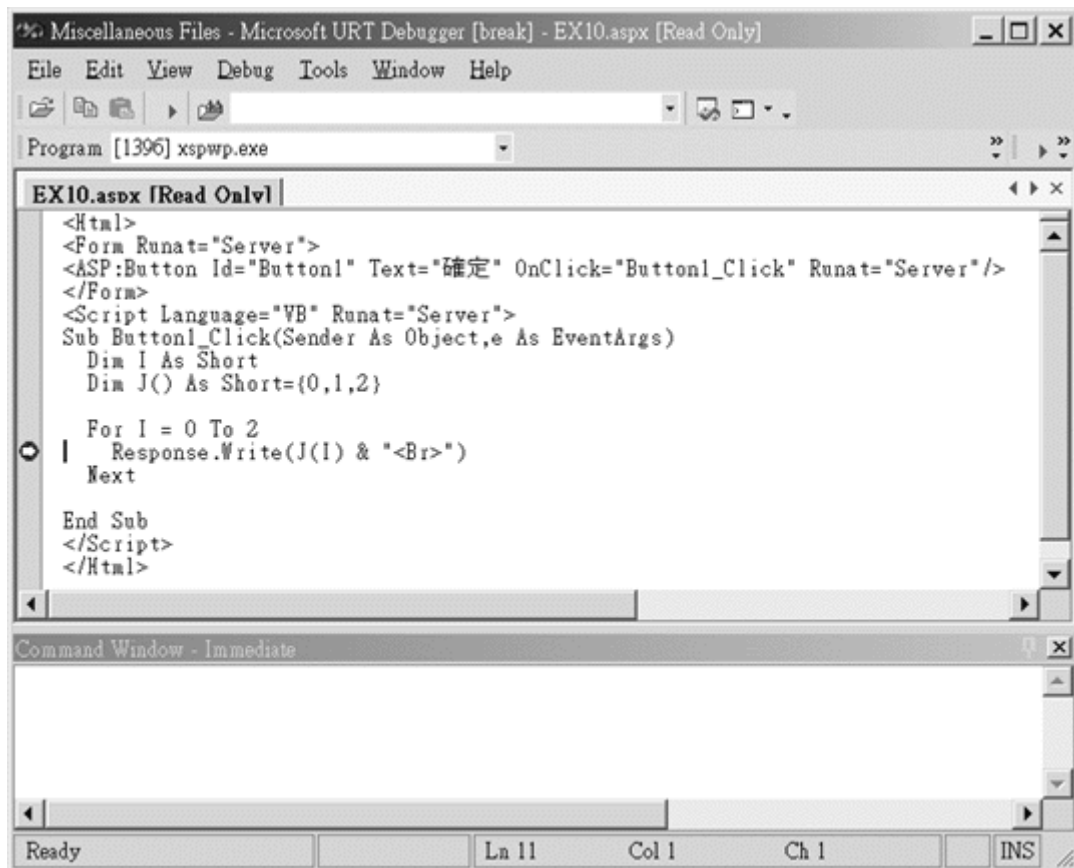
1. 窗口的左边界按下鼠标左键。
2. 按下「F9」键可新增或移除断点。
3. 选择工具列上的「Insert Breakpoint」按钮。

4. 使用鼠标右键再欲设定断点的程序代码按一下，再从弹出式菜单中选择 Insert

Breakpoint 选项。



我们发现利用 IE 浏览程序按下按钮后，便自动跳到除错程序中；而且在来断点上多了一个黄色
的箭头，程序停止之处还没被执行。



使用 Debug 工具列

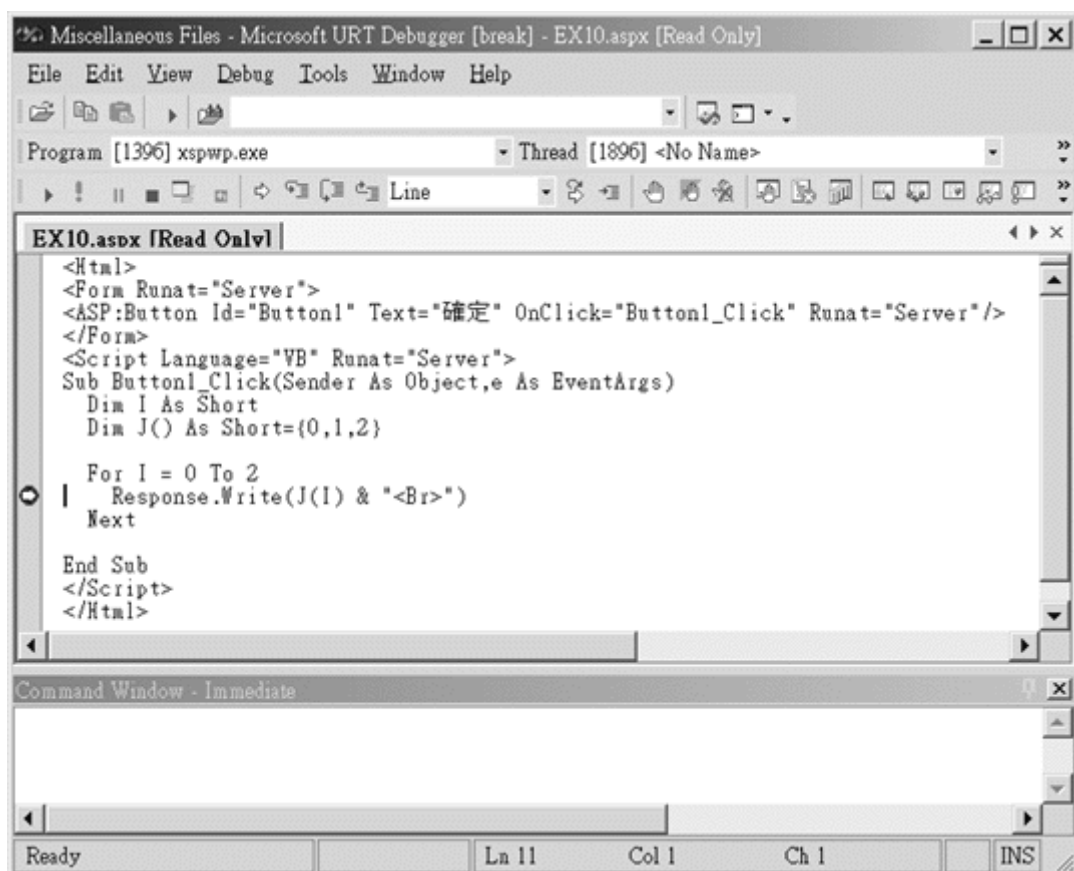
除错器的工具列上还有许多好用的功能。我们第一次进入中断模式时，**Debug** 工具列并不会出现；此时只在工具列的空旷处按一下鼠标右键，在弹出式选单选择「**Debug**」选项即可叫出 **Debug** 工具列，下图即为 **Debug** 工具列。



下表列出 **Debug** 工具列中常用按钮的功能。

按鈕圖形	按鈕名稱	說明
	Continue	當進入中斷模式時，按此鈕會讓程式繼續執行。
	Stop Debugging	停止除錯，若要再進入除錯模式需重設 Processes。
	Step Into	逐行追蹤程式，碰到其他程序或函數呼叫時會進入逐行執行。
	Step Over	逐行追蹤程式，碰到其他程序或函數呼叫時會進入執行，但不逐步執行。
	Step Out	執行完目前的程序或函數，回到呼叫目前程序或函數的程式碼中繼續執行。
	Insert Breakpoint 或 Remove Breakpoint	加入或移除中斷點。
	Disable Breakpoint	暫時關閉中斷點，但是在程式碼的 Margin Indicator 區會留下一個紅色的圓框。
	Clear All Breakpoint	清除全部的中斷點。
	Breakpoints	呼叫 Breakpoints 視窗並顯示目前哪幾行有設定中斷點。
	Exceptions	呼叫 Exceptions 視窗，可以設定當發生某種 Exception 時要做何種處理。
	Locals	列出整個網頁中變數的狀況。
	Me	列出目前頁面中物件的繼承情況。
	Watch	呼叫 Watch 視窗，並且可以將頁面中的物件或變數以拖放方式加入，以利我們及時監看內容。
	Immediate	開啓 Immediate Window。
	Modules	開啓 Modules 視窗，並列出目前所使用到的動態連結程式庫。
	Processes	開啓 Processes 視窗。
	Quick Watch	開啓 Quick Watch 視窗，這個視窗的內容是隨著鍵盤游標的位置而不同。

实时检查变量有许多方法，一个是直接将鼠标光标移到变量的位置上，然后会出现小提示，框内的内容就是目前变量的值。



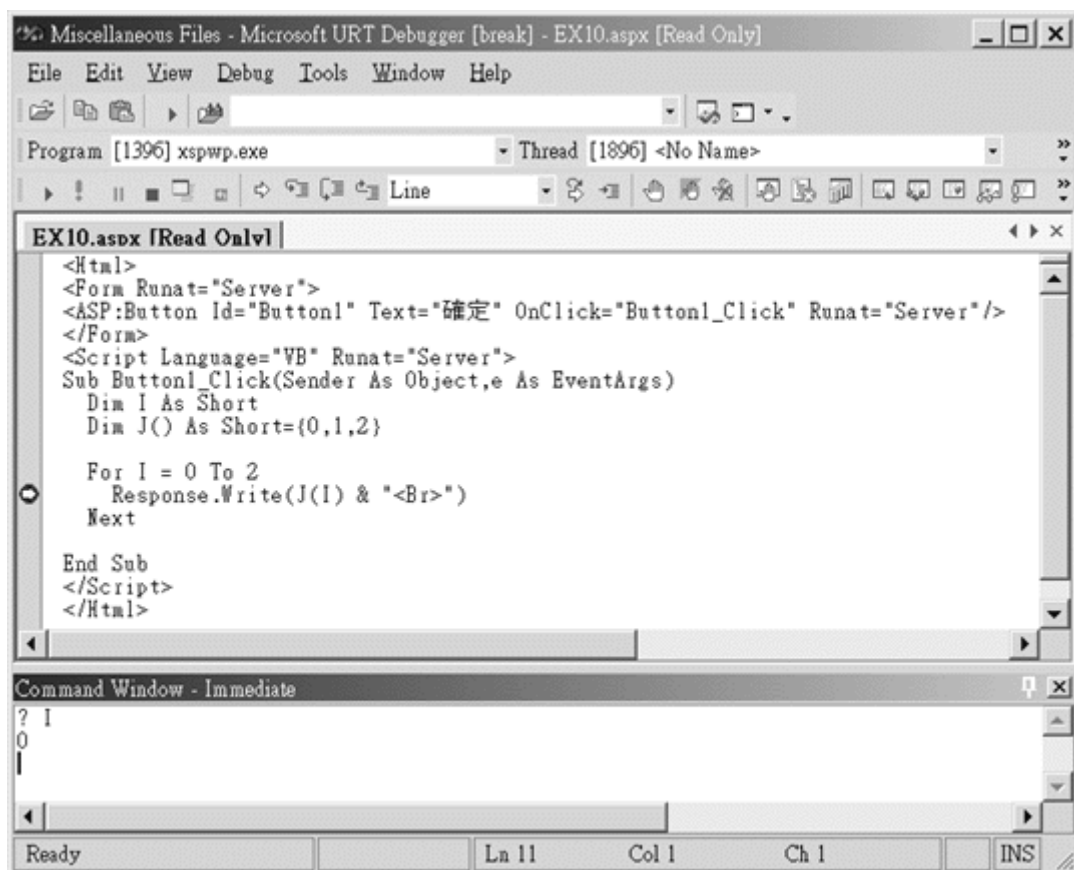
Immediate Window

Immediate Window（实时运算窗口）可以让我们执行任何合法正确的叙述，我们可以在立即运

算窗口内输入叙述，然后按「Enter」来执行。其中我们在立即运算窗口中最常用的叙述为「？」，

「？」表示印出的意思。例如我们输入「？」后按下「Enter」后，马上会在下一列显示出变量的

值：



在实时窗口中我们也可以实时的更改变量的内容。以这个程序为例，循环的次数是由 0 到 2 共执行 3 次，程序第一次进入时 I 的值为 0，但是我们可以输入 I=1 来马上改变它的值，对于需要执行大量数值的循环相当有用。



程序的追踪及检视

要让程序继续执行，只要按下「Continue」按钮就会执行目前断点以下的程序代码。由于我们这个例子在循环中执行，因此程序执行下一个 I 时又会停在这一行。由于刚才我们把 I 的值改成 1，所以只要再按两次 Continue 后 I 的值为 3，程序执行完毕；最后网页出现我们程序的结果：



Step...

要更详细的执行程序，可以使用 **Step Into**、**Step Over** 以及 **Step Out**。我们以下列程序为例：

```
<Script Language="VB" Runat="Server">  
  
Sub Page_Load(Sender As Object, e As EventArgs)  
  
    Dim I As Short  
  
    For I = 0 To 5  
  
        Test(I)  
  
    Next  
  
End Sub
```

```
Sub Test(ByVal J As Short)

    Response.Write("I=")

    Response.Write(J.ToString())

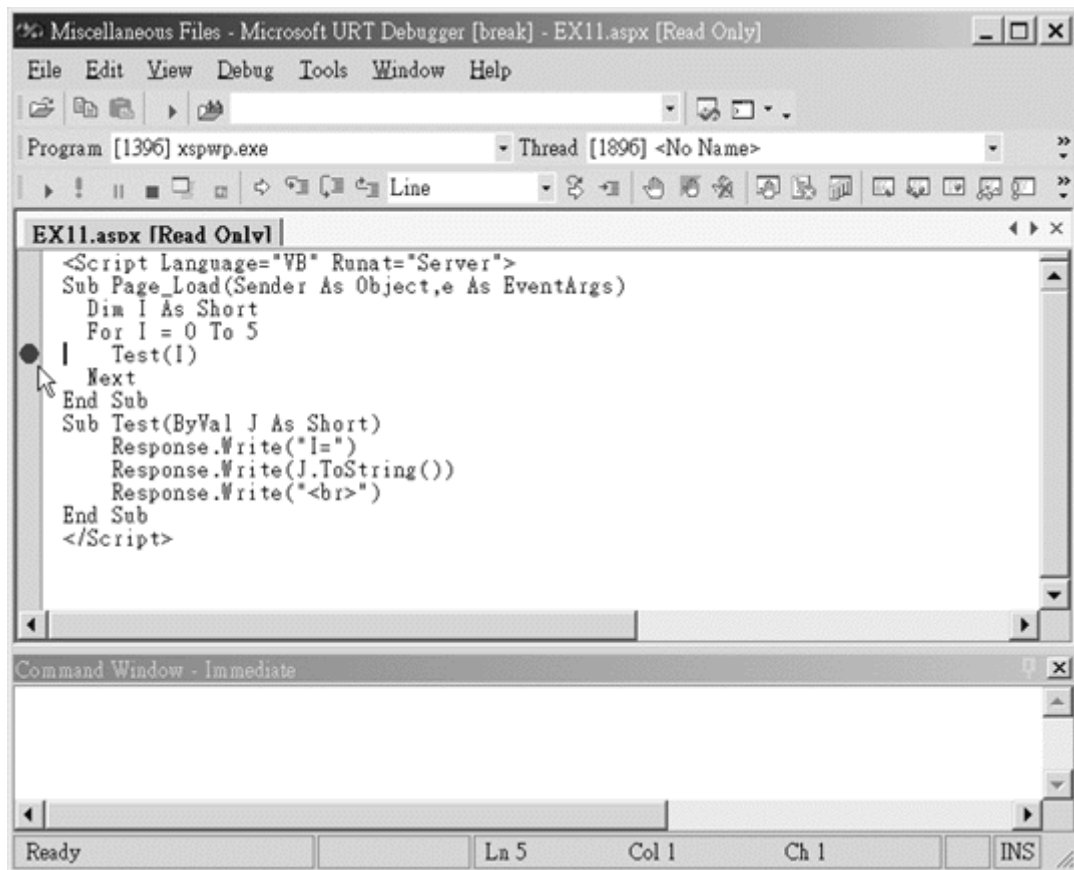
    Response.Write("<br>")

End Sub

</Script>
```

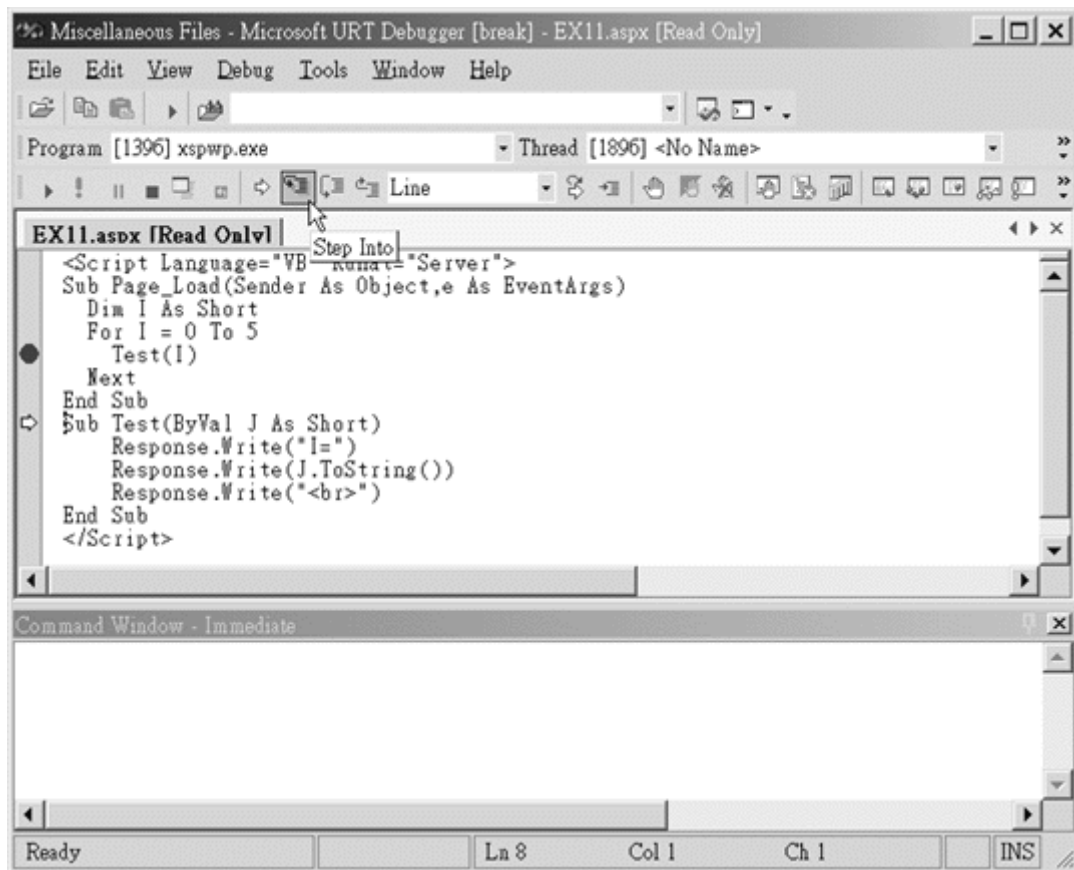
上述范例中我们在呼叫 **Test** 这个程序的叙述加入断点，然后浏览 **EX11.aspx**，即进入中断模式；

如下图所示：



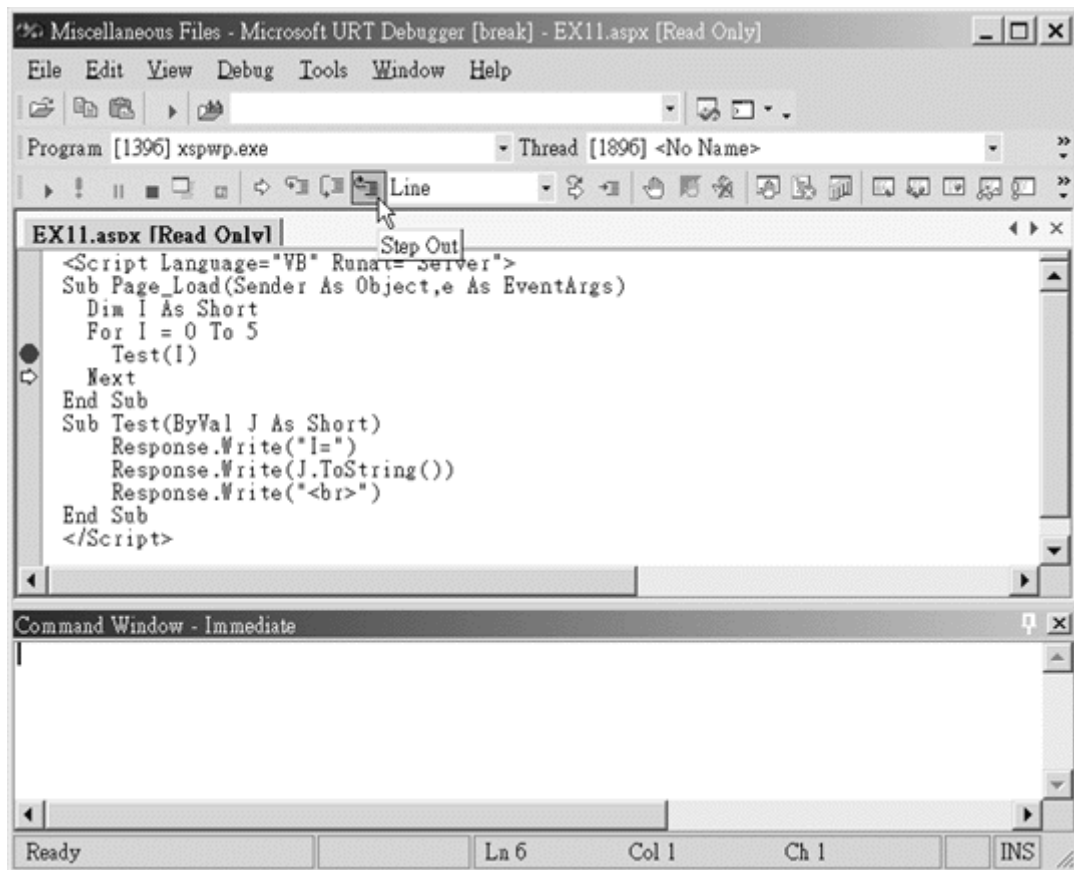
我们按下「Step Into」按钮，发现黄色箭头即往下移动一行，再按「Step Into」一次，程序即跳

入 Test 这个程序中执行：

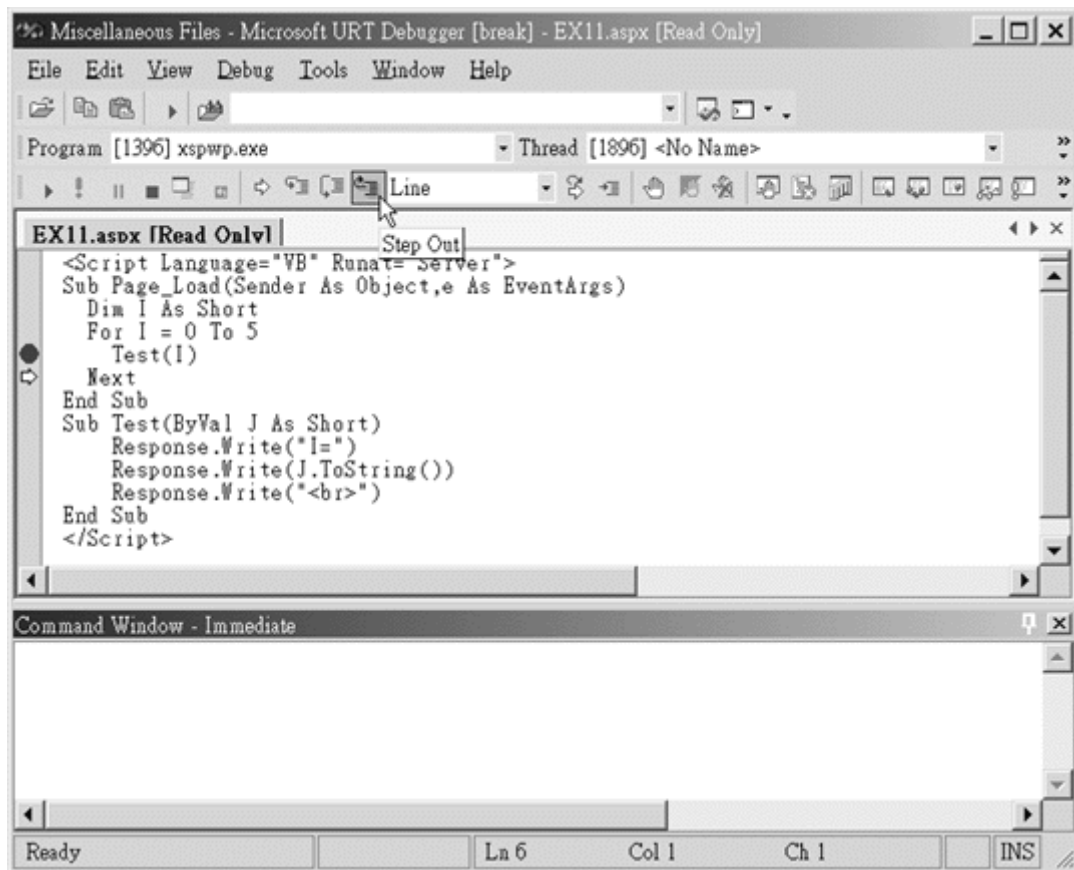


接下来我们直接按下「Step Out」按钮，发现程序的执行跳出 Test 这个程序，而停留在呼叫 Test

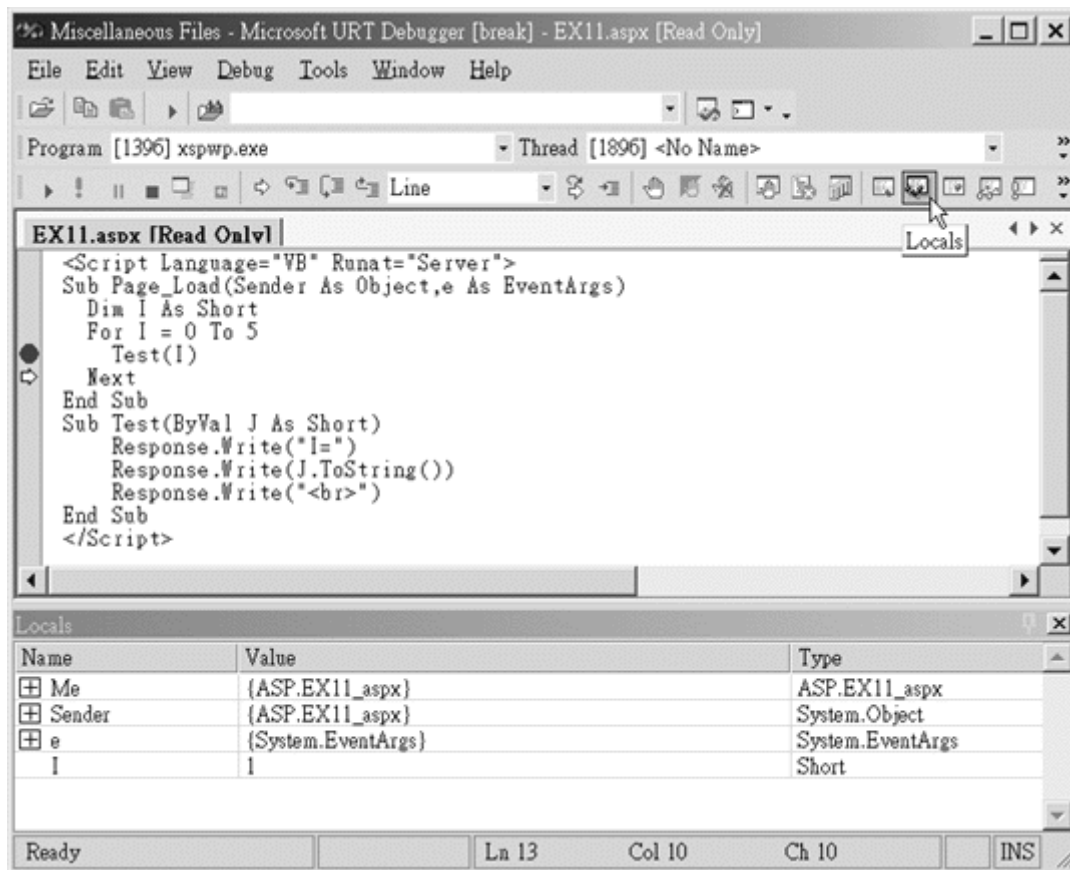
这个程序的下一行叙述：



「Step Out」不是逐行执行，而是一口气将程序内的所有程序代码执行完毕，再跳到呼叫该程序的下一行。接下来我们按下「Step Over」按钮，发现程序还是一次执行一行，而停在「Test(l)」这个叙述上。我们再按一次「Step Over」按钮时，发现程序并没有跳入 Test 程序中，而是停留在「Next」上面，如下所示：

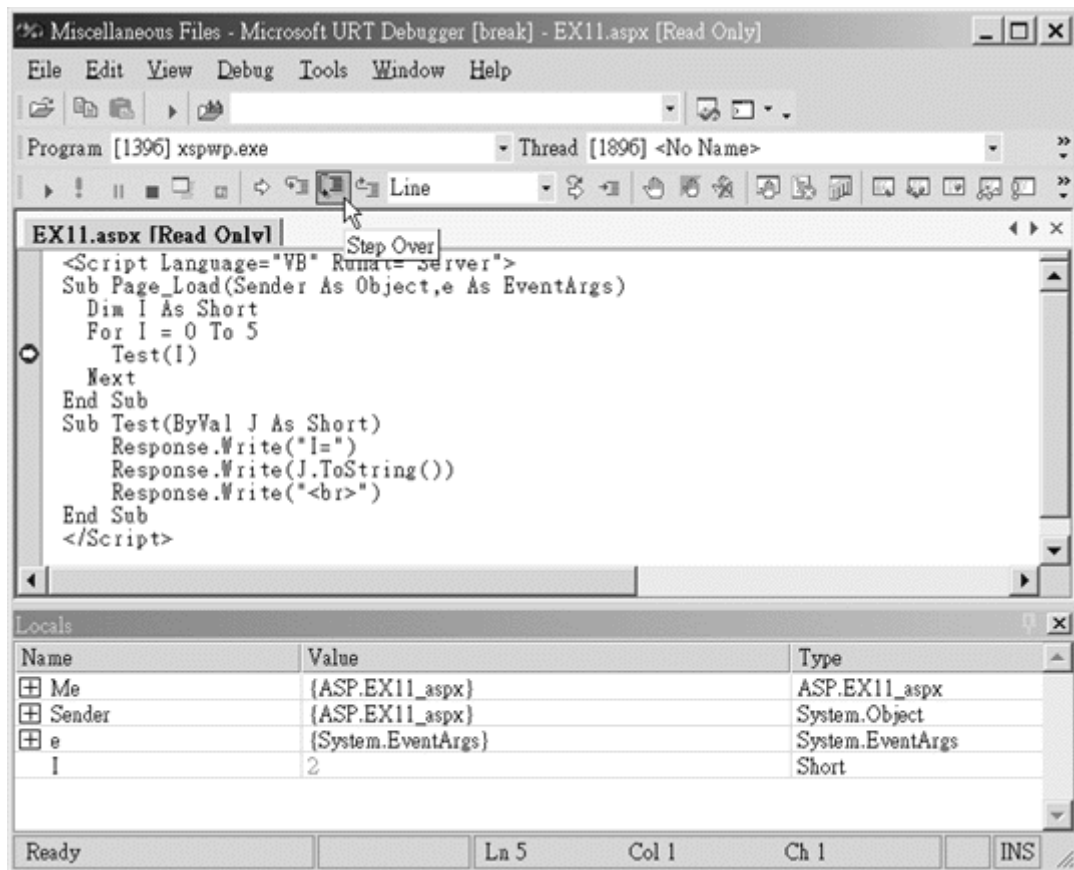


这是因为「Step Over」遇到过程调用时，一次就将程序内的程序执行完毕，并不一行一行的执行所致。



Locals Window

我们在上述例子追踪程序时，对于变量的检视还要跳到实时运算窗口输入「？」来观察，实在不太方便。此时我们可以利用 **Locals Window**（区域窗口），**Locals Window** 可以列出目前所在程序中的所程序阶层变量以及对象的值，如下图所示：

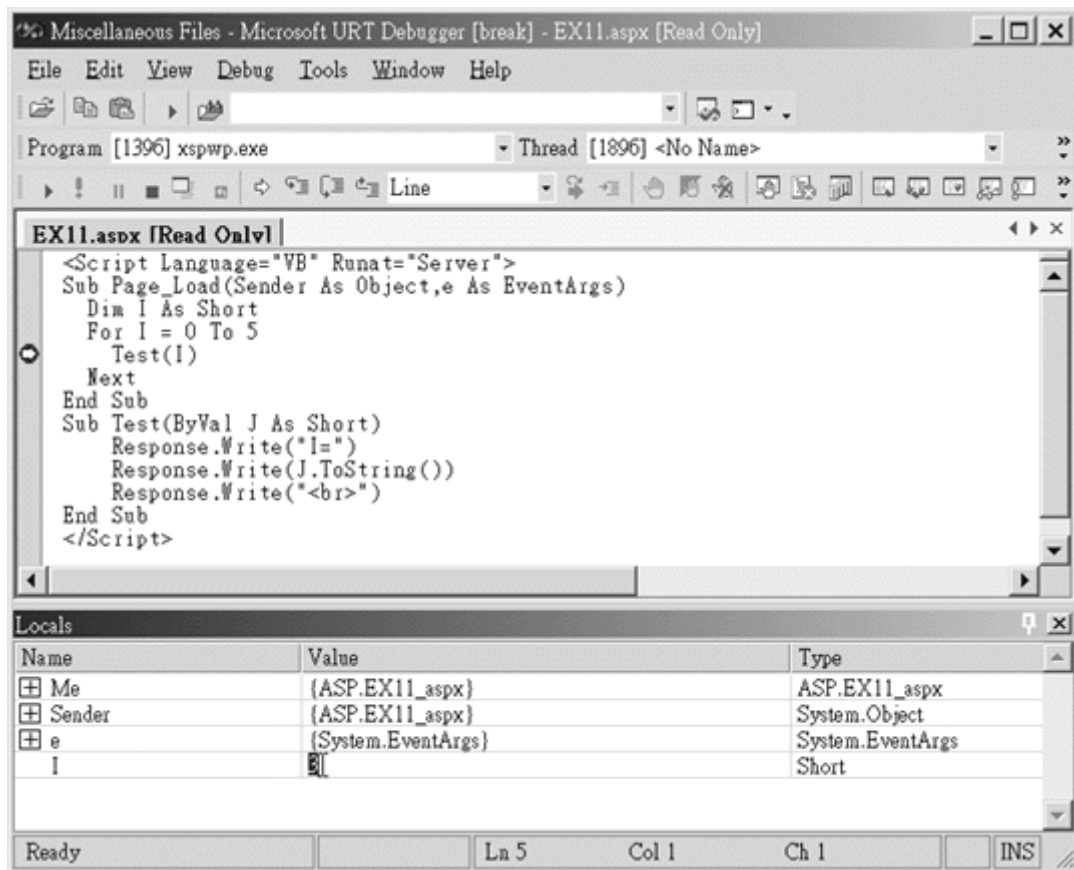


我们观察到变量 I 的内容为 1，而且型态为 Short。其中除了变量 I 之外，还一并列出了其它项目。

Me 代表目前的 Page 对象，我们可以展开来观察 Page 对象的所有成员，而 Sender 表示触发此

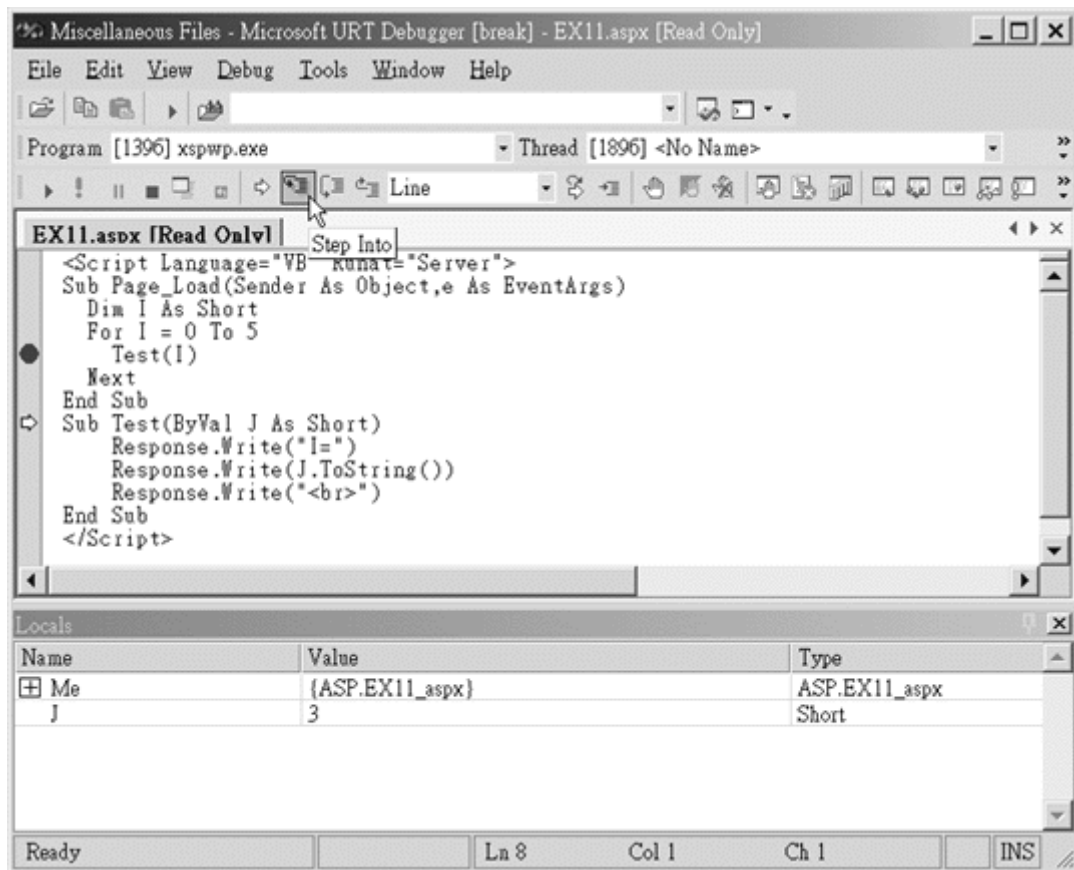
程序的对象为何，e 则是代表参数对象的内容。当我们再按一次「Step Over」时，发现 Local

Window 中的变量 I 自动由 1 更新为 2:



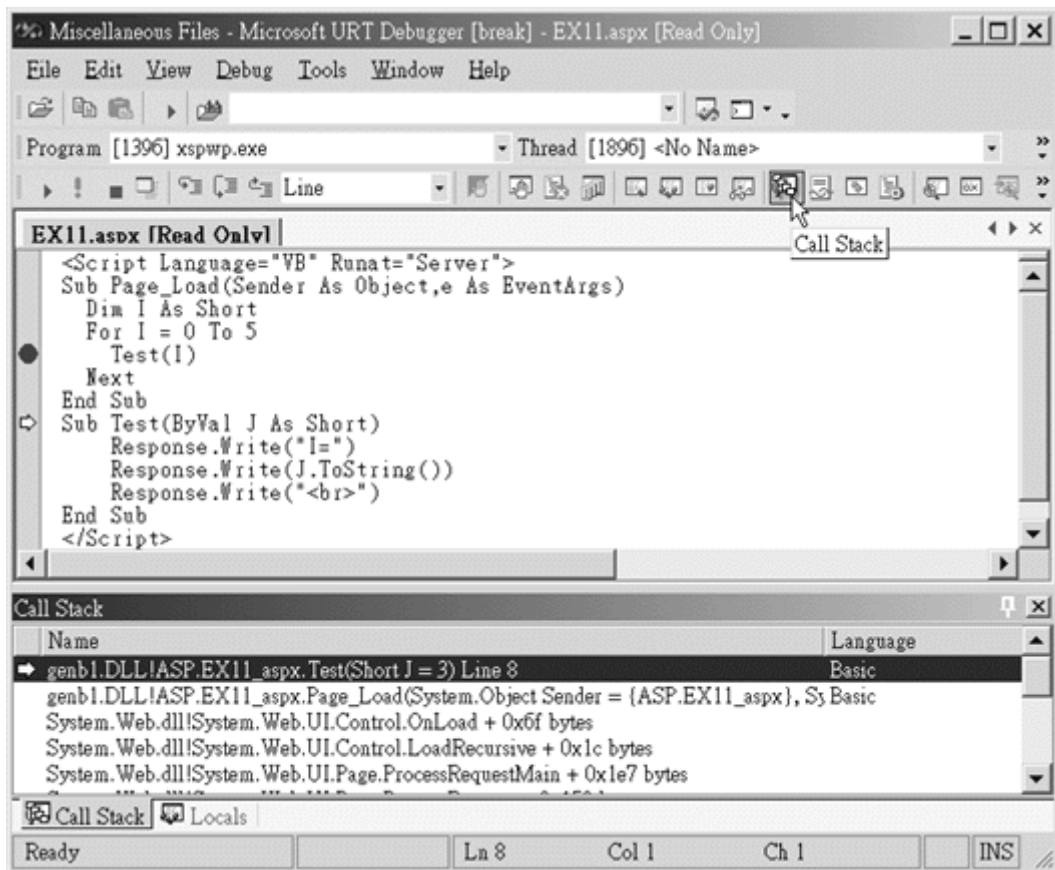
Locals Window 除了可以显示变量以及对象的内容外，在 Locals Window 中也可以直接修改变

量或对象的值。例如我们将 `I` 的值更改为 `3`，如下图所示：



接下来我们按下「Step Into」按钮跳入 Test 程序中，发现 Locals Window 会自动显示 Test 程

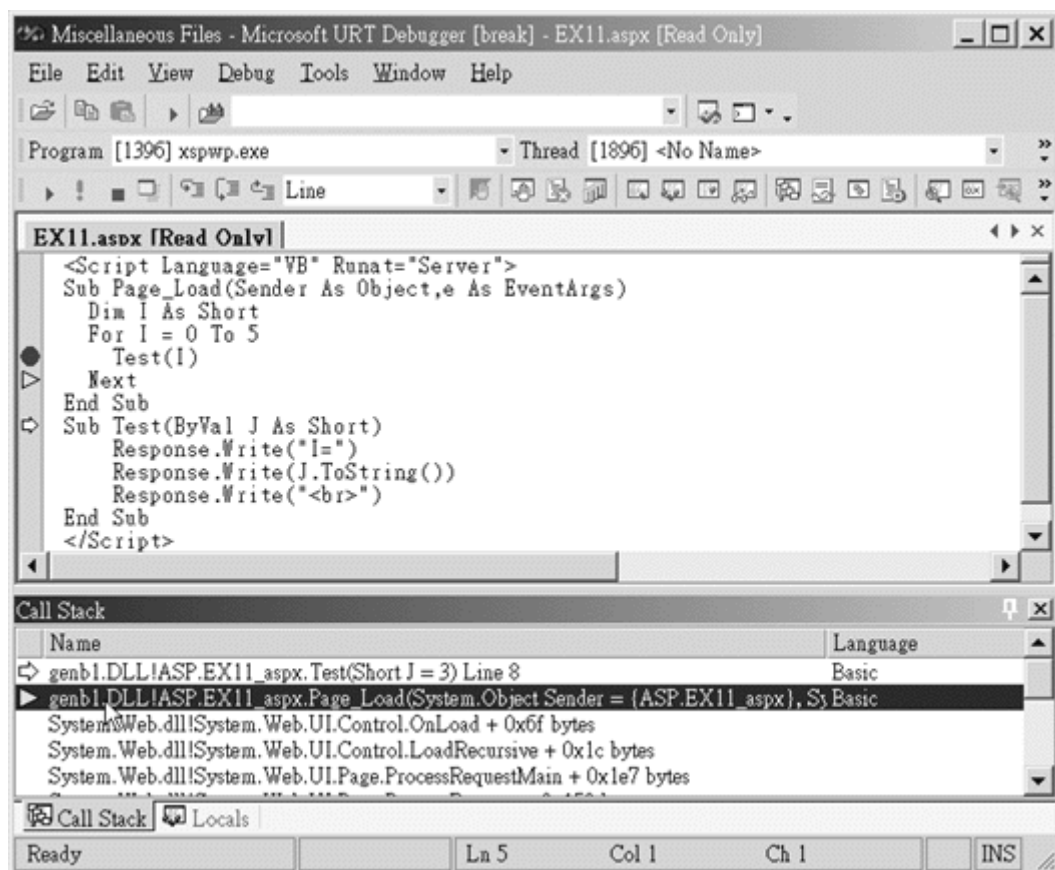
序中的变量 J，如下所示：



Call Stack

Call Stack 可以让我们追踪程序之间的呼叫关系。程序在执行的时候若遇到过程调用，原来的程序便会暂时停止执行而进入程序中执行；待程序执行完毕后，才又会回到原先的程序中执行。此时这些程序的执行就像堆积木一样，先执行的放下面，后执行的再堆上去；必须要等到后面堆上去的程序执行完毕后，才执行迭在下方的程序。这种执行的方式我们称为「先进后出」，也可以称为「堆栈」。一但程序比较复杂时，一定会有这种程序内呼叫其它程序的状况，此时要追踪这种我们称为「堆栈」的过程调用，就可以利用「Call Stack」。由于刚刚我们跳入了 Test 这个程

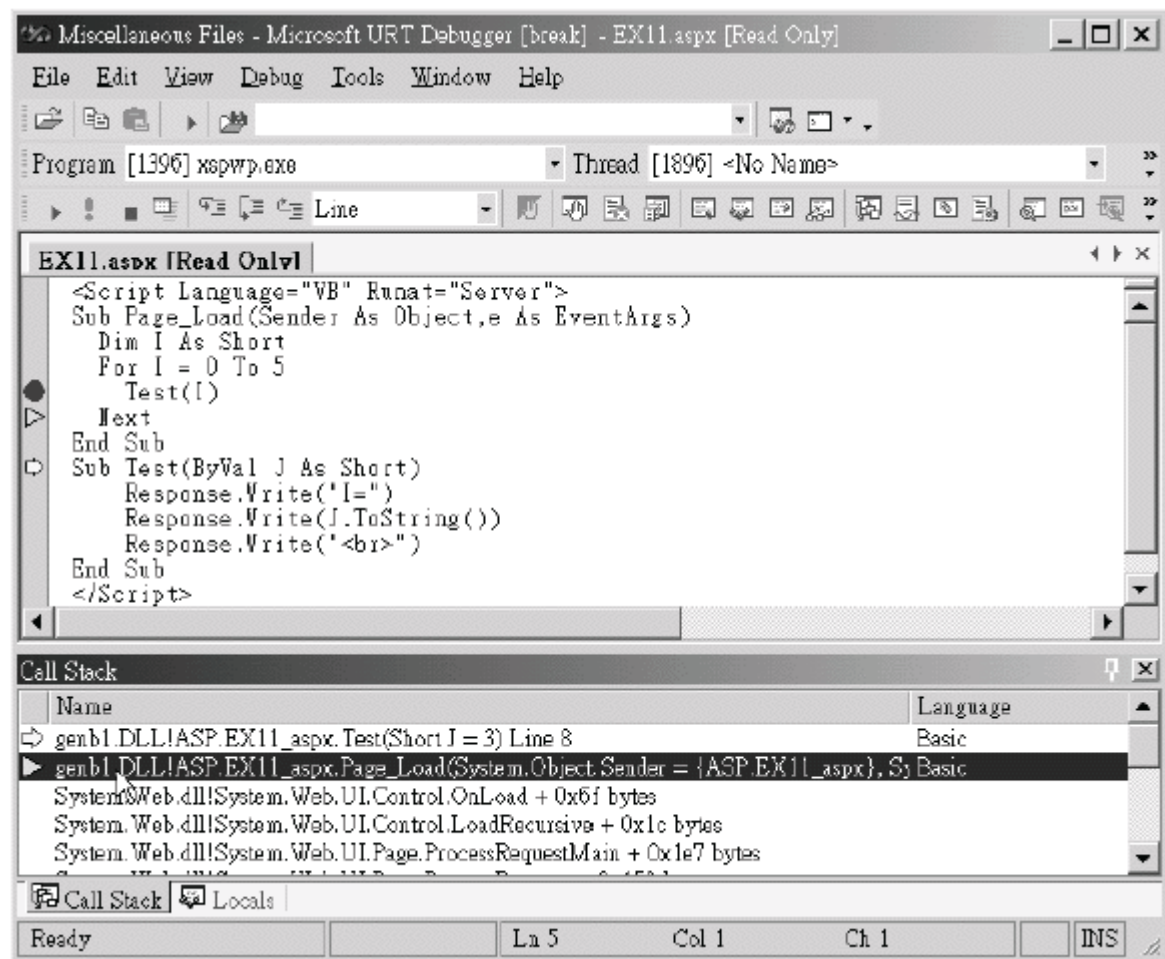
序内，我们知道 **Test** 程序是在 **Page_Load** 事件程序中被呼叫的，所以除了 **Test** 目前是最高层的程序之外，它的下一层就是 **Page_Load** 程序。我们按下「**Call Stack**」按钮来观察，如下图所示：



我们发现 **Call Stack** 窗口中的最上层对象果然是 **Test** 程序，并且在前面用黄色箭头标注，表示

目前程序在这个程序中执行。如果我们要追踪 **Test** 程序被上层所呼叫的地址，还可以在

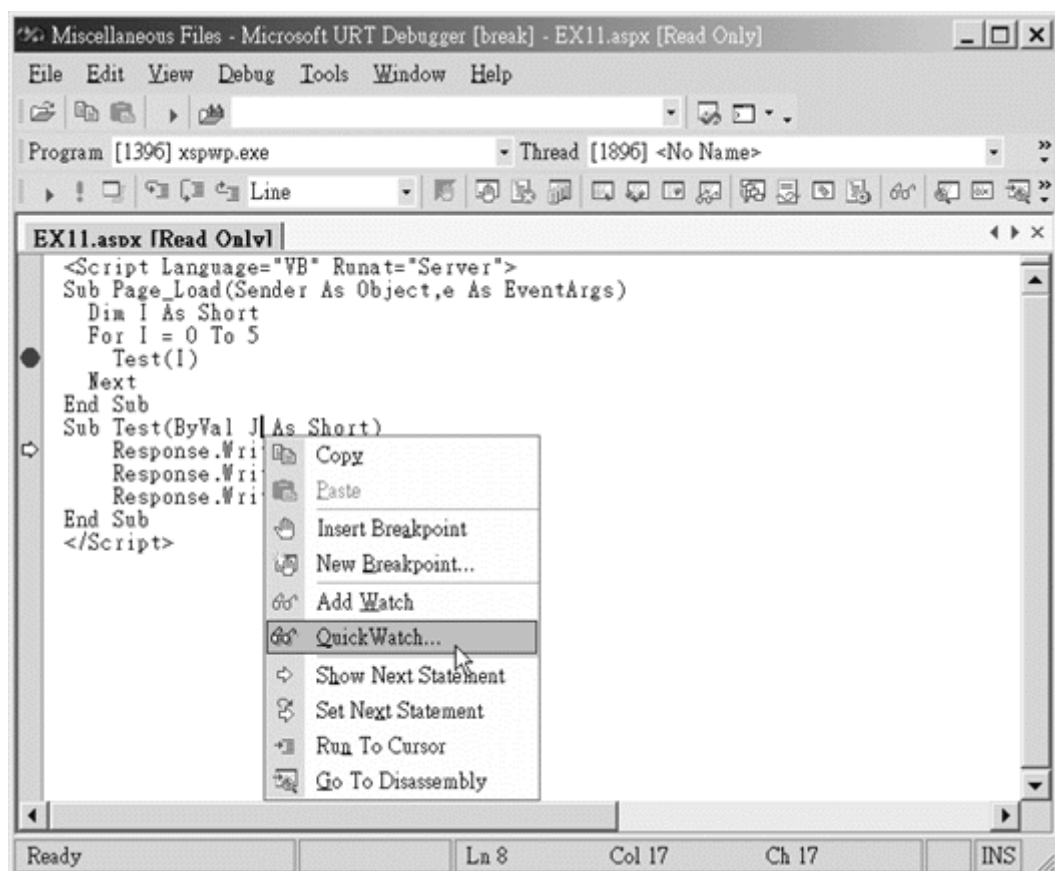
Page_Load 程序上双击鼠标左键，除错器便会以绿色箭头标出，如下图所示：



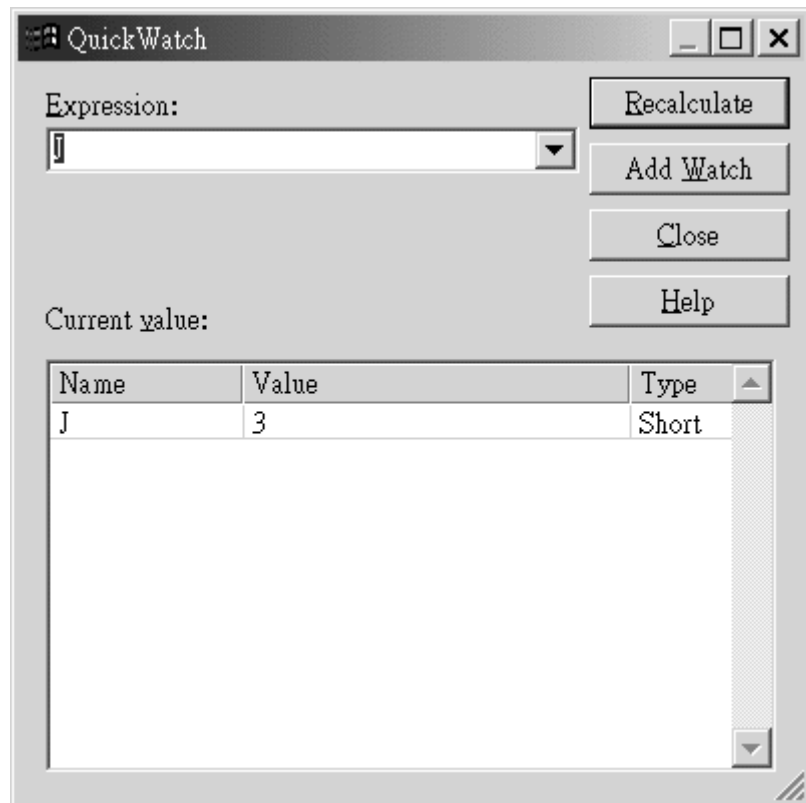
Quick Watch

Quick Watch（快速检视）是另外一种可以快速检视变量值的窗口。我们可以将键盘光标停在想

要观察的变量上，然后按下「Quick Watch」按钮或按鼠标右键选择，如下图所示：



选择「Quick Watch」后即出现变量的数据，如下图所示：

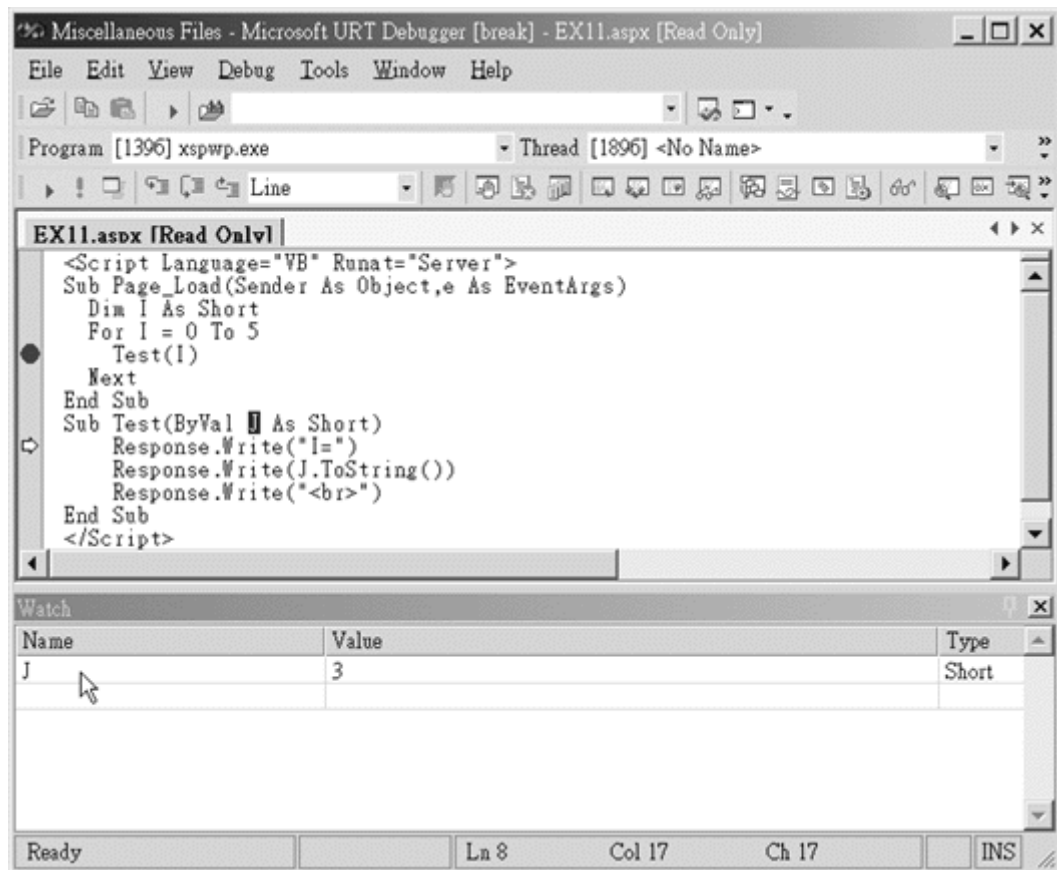


另外注意立即运算窗口以及 Quick Watch 可以检视的变量，只限制于目前所在的程序内。

Watch Window

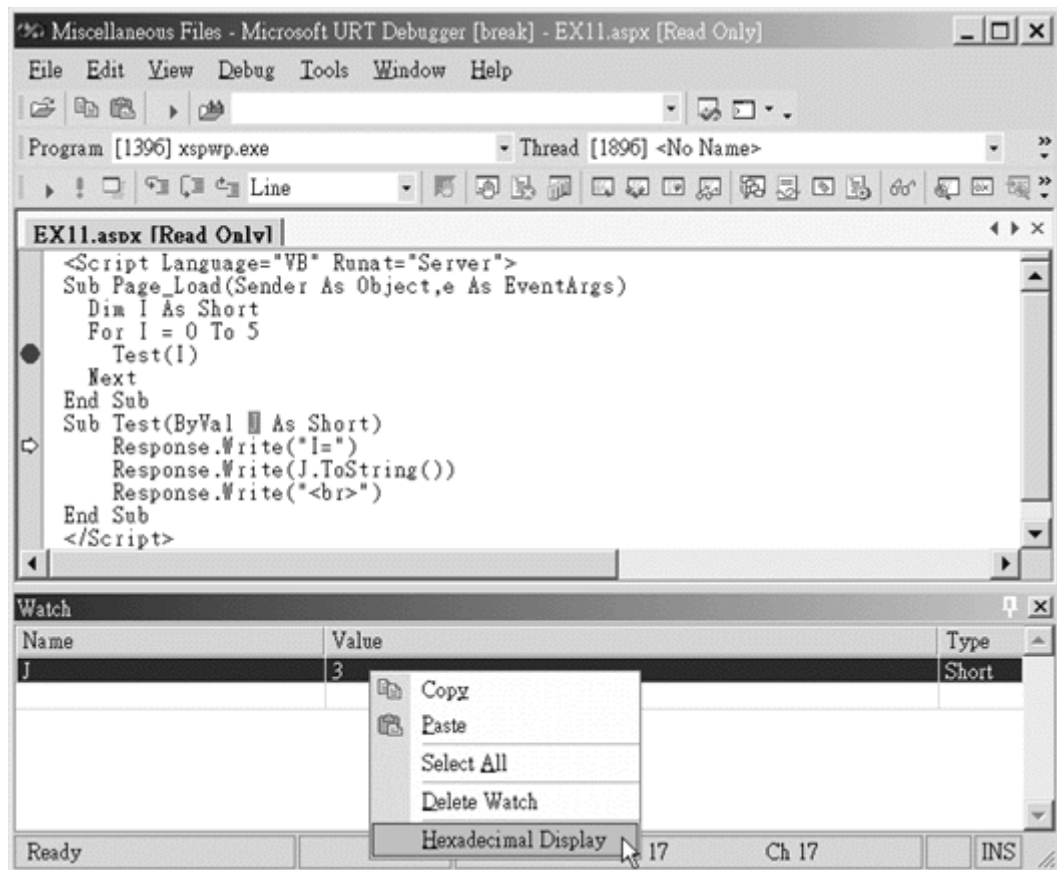
Watch Window（监视窗口）是另一个可以检视变量值的工具，我们点选「Watch Window」显

示 Watch Window，如下图所示：

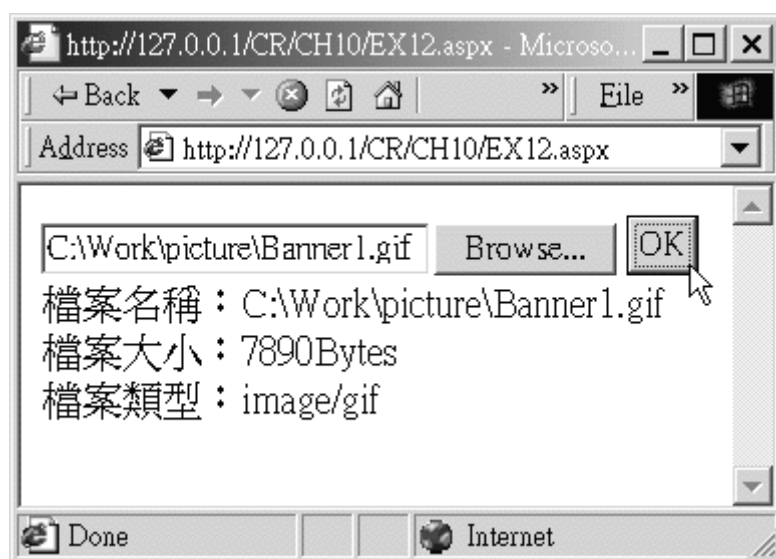


接下来我们可以在想要监视的变量上按鼠标右键，选择「Add Watch」；或是先将变量标记下来，

然后利用拖放的方式将变量拖到 Watch Window，如下图所示：



而 Watch 窗口中显示的值可以在 10 进制或 16 进制切换，只要在 Watch 窗口中按下右键并核取或取消「Hexadecimal Display」选项，即可切换显示的方式。



接着我们用 **Step Into** 按钮逐行执行，可以看到变量的内容变化。

档案上传处理

使用 **ASP.NET** 制作一个可以存放 **Client** 端档案的网页相当容易，因为 **ASP.NET** 里就有提供我

们将 **Client** 端档案传至 **Server** 端的对象，这个对象就是 **HtmlInputFile** 对象。**HtmlInputFile** 对象

必须存在窗体中，而且窗体 **<Form>** 标注中必须加入设定 **Enctype="Multipart/Form-Data"** 属

性才可使用。**HtmlInputFile** 对象的语法如下所示：

```
<Input Type="File" Id="被程序所控制的名称" Runat="Server">
```

当一个档案传送到 **Server** 端后，接收和处理的是 **HtmlInputFile** 对象的 **PostedFile** 属性。

PostedFile 属性的型态是 **HttpPostedFile** 对象类别，其常用属性如下表所示：

属性	说明	型态
ContentLength	传回上传的档案长度，单位是 Bytes。	Integer
ContentType	传回上传的档案的类型。	String
FileName	传回 Client 端上传的文件名称，具有完整的路径如 C:\Temp\test.txt。	String

,

其常用方法如下表所示：

方法	说明	语法
SaveAs	将 Client 端绍传的当按存至磁盘中。	SaveAs (ByVal filename As String)

基本档案上传

下列范例使用 **HtmlInputFile** 对象的基本用法，分别印出档案的名称、大小、类型：

```
<Html>

<Form Enctype="Multipart/Form-Data" Runat="Server">

<Input Type="File" Id="UpLoad" Runat="Server">
```

```
<Asp:Button Id="Button1" Text="OK" OnClick="Button1_Click"
```

```
Runat="Server" /><br>
```

```
<Asp:Label Id="Label1" Runat="Server" />
```

```
</Form>
```

```
<Script Language="VB" Runat="Server">
```

```
Sub Button1_Click(Sender As Object, e As EventArgs)
```

```
    Dim strContent As String
```

```
    If Upload.PostedFile.ContentLength<0 Then
```

```
        Label1.Text="上传失败"
```

```
    Else
```

```
        strContent="文件名称: " & Upload.PostedFile.FileName
```

```
        strContent=strContent & "<br>档案大小: " & _
```

```
Cstr(Upload.PostedFile.ContentLength) & "Bytes"
```

```
        strContent=strContent & "<br>档案类型: " &
```

```
Upload.PostedFile.ContentType
```

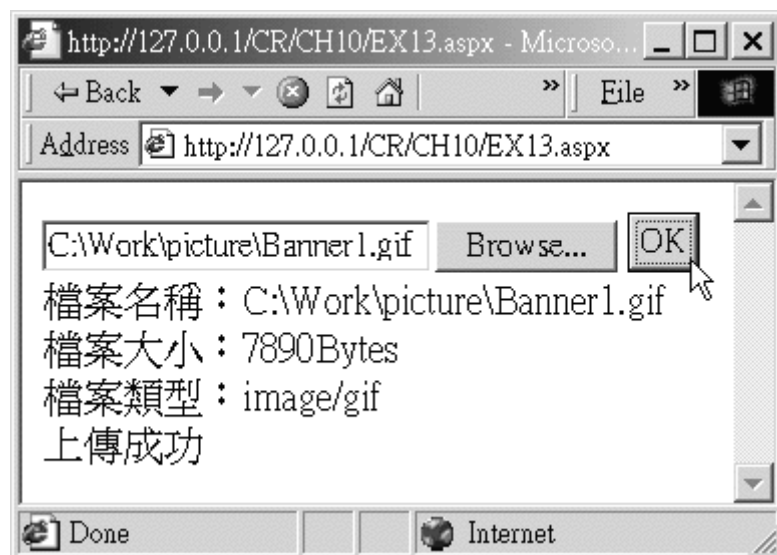
```
    End If
```

```
    Label1.Text=strContent
```

End Sub

</Script>

</Html>



存入磁盘

上传档案后若要写入磁盘中也相当容易，只要使用 **PostedFile** 的 **SaveAs** 方法即可。档案写入

时若直接将 **PostedFile.FileName** 属性来当文件名写入会有问题，因为 **FileName** 属性包含有档

案的路径名称，所以必须先将档案的路径分开。下列范例使用 **Split** 函数来取得文件名称，并将

档案存放到网页所在的目录：

```
<Html>

<Form Enctype="Multipart/Form-Data" Runat="Server">

<Input Type="File" Id="UpLoad" Runat="Server">

<Asp:Button Id="Button1" Text="OK" OnClick="Button1_Click"

Runat="Server" /><br>

<Asp:Label Id="Label1" Runat="Server" />

</Form>

<Script Language="VB" Runat="Server">

Sub Button1_Click(Sender As Object, e As EventArgs)

    Dim strContent As String

    If UpLoad.PostedFile.ContentLength>0 Then

        strContent="文件名称: " & UpLoad.PostedFile.FileName

        strContent=strContent & "<br>档案大小: " &

Cstr(UpLoad.PostedFile.ContentLength) & "Bytes"
```

```
        strContent=strContent & "<br>档案类型：" &  
Upload.PostedFile.ContentType  
  
        Dim Temp() As String = Split(Upload.PostedFile.FileName, "\" )  
  
        Dim Name As String = Temp(Temp.Length-1)  
  
        Upload.PostedFile.SaveAs(Server.MapPath(".") & "\" & Name)  
  
        Label1.Text=strContent  
  
    Else  
  
        Label1.Text="上传失败"  
  
    End If  
  
End Sub  
  
</Script>  
  
</Html>
```

Split 函数会将一个字符串以我们设定的条件为分隔，并将分段后的字符串存至一个字符串数组

中；所以我们只要以 "\" 为条件来分隔档案，在字符串数组中的最后一个元素即为文件名称。

多档上传

我们也可以一次上传多个档案。管理档案的对象是 **Request** 对象的 **Files** 属性，它的型别是

HttpFileCollection 集合对象类别，它的项目成员是 **HttpPostedFile** 类别对象。**HttpFileCollection**

型别变量的常用属性如下表所示：

属性	说明	型态
All	传回全部的 HttpPostedFile 型别对象。	HttpPostedFile ()
AllKeys	传回全部的 HtmlInputFile 控件的名称。	String ()
Item	以 HtmlInputFile 控件的名称传回	<div>1. Item(String)As HttpPostedFileHttp PostedFile 物件。</div> <div>2. Item(Integet)As HttpPostedFile</div>

其常用方法如下表所示：

方法	说明	参数
Get	以 HtmlInputFile 对象名称或索引传回	<div>1. Get(String)As</div>

	HttpPostedFile 对象。	<div>HttpPostedFile</div> <div>2. Get(Integer)As</div> <div>HttpPostedFile</div>
GetKey	以索引值在集合中传回 HttpPostedFile 对象。	<div>GetKey(Integer)As</div> <div>HttpPostedFile</div>

下列范例中我们放了两个 **HtmlInputFile** 对象，并上传两个档案：

```
<Html>

<Form Enctype="Multipart/Form-Data" Runat="Server">

<Input Type="File" Id="UpLoad1" Runat="Server"><br>

<Input Type="File" Id="UpLoad2" Runat="Server">

<Asp:Button Id="Button1" Text="OK" OnClick="Button1_Click"

Runat="Server"/><br>

<Asp:Label Id="Label1" Runat="Server"/>

</Form>

<Script Language="VB" Runat="Server">
```

```

Sub Button1_Click(Sender As Object, e As EventArgs)

    Dim shtI As short

    Dim strContent As String

    Dim Files As HttpFileCollection

    Files=Request.Files

    For shtI=0 To Files.Count-1

        If Files.Item(shtI).ContentLength>0 Then

            strContent=strContent & "第" & Cstr(shtI+1) & "个档案上传对象
<br>"

            strContent=strContent & "文件名称: " & Files.Item(shtI).FileName
& "<br>"

            strContent=strContent & "档案大小: " &
Cstr(Files.Item(shtI).ContentLength)_
& "Byte<br>"

            strContent=strContent & "档案类型: " &
Files.Item(shtI).ContentType & "<p>"

            Dim Temp() As String = Split( Files.Item(shtI).FileName, "\")

```

```
        Dim Name As String = Temp(Temp.Length-1)

        Files.Item(shtI).SaveAs(Server.MapPath(".") & "\" & Name)

    End If

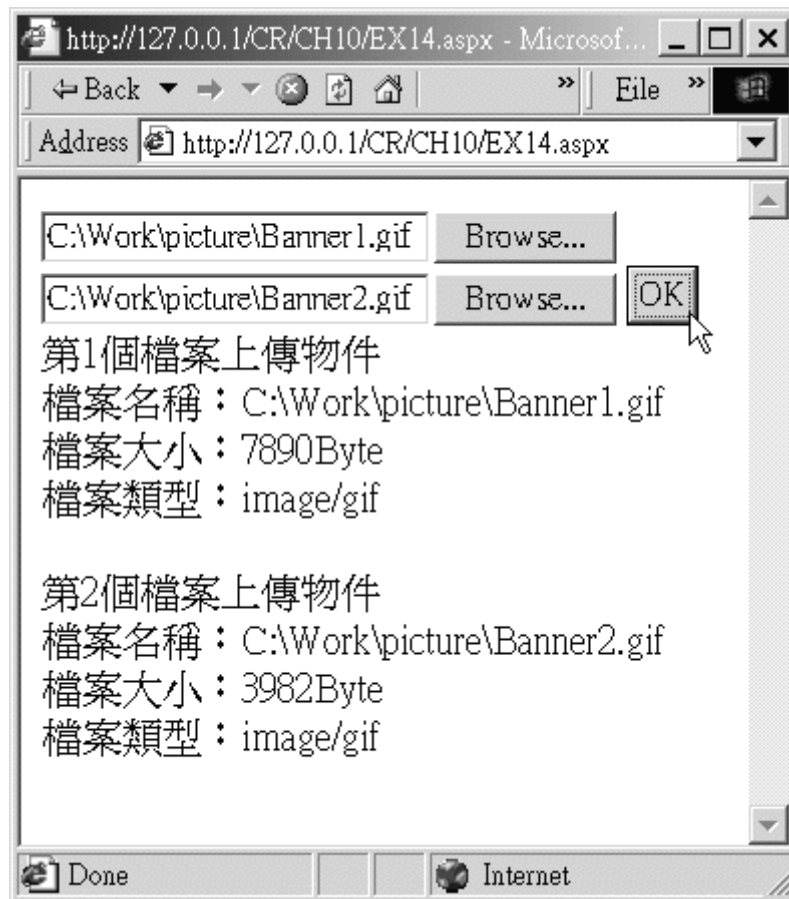
Next

Label1.Text=strContent

End Sub

</Script>

</Html>
```



上述范例中我们先定义一个 `HttpFileCollection` 类型变量 `Files`，并将 `Request.Files` 属性的内容放到 `Files` 变量中。再使用 `For...Next` 循环将档案信息加入 `strContent` 变量中，再以 `Split` 函数将档案路径和文件名称分开并且存到一数组，因数组的最后一个元素即为文件名称，因此个别取出后使用 `HttpPostedFile` 对象的 `SaveAs` 方法将档案写入磁盘。

E-Mail 传送

ASP.NET 可以轻松的制作出寄送 E-Mail 的网页。传送 E-Mail 使用到的对象有两个，分别是 **MailMessage** 对象以及 **SmtpMail** 对象。**MailMessage** 对象是用来设定信件内容，而 **SmtpMail** 对象则是将设定好的信件传送出去。因为这两个对象预设并没有被使用，所以使用 **MailMessage** 对象和 **SmtpMail** 对象之前要先宣告其名称地址，**MailMessage** 对象和 **SmtpMail** 对象使用的名称地址是 **System.Web.Util**。**MailMessage** 对象的常用属性如下表所示：

属性	说明	型态
Bcc	设定密件副本收信人 E-Mail 地址。收信人的姓名 及 E-Mail 地址不会出现在信件中。	String
Body	设定信件内容。	String
BodyFormat	设定信件的格式为纯文字或 HTML 格式。	MailFormat
Cc	设定副本收信人 E-Mail 地址。	String
From	设定送信者的 E-Mail 地址。	String
Subject	设定信件的标题。	String
To	设定收信者的 E-Mail 地址。	String

SmtpMail 对象只有一个方法，如下表所示：

方 法	说明	语法
Send	送出设定好的信件内容。	<ol style="list-style-type: none"> 1. Send(ByVal message As MailMessage) 2. Send(ByVal from As String, ByVal to As String, ByVal subject As String, ByVal messageText As String)

下列范例我们先用 **FrontPage** 画好表格及布置对象，当按下送出钮时，程序会产生 **Mail** 及

SendMail 对象。接着送出设定好的 **Mail** 对象后，在画面显示「信件传送成功」：

```

<%@ Import Namespace="System.Web.Util"%>

<Html>

<Form Runat="Server">

<Table Border="0" Width="100%">

    <Tr>

        <Td Width="100%" Valign="Middle" Align="Center">

            <Table border="0" Width="105%">

                <Tr>

```

<Td Width="28%" Valign="Top" Align="Right">收信人:</Td>

<Td Width="52%"><Asp:TextBox Id="txtReceiver" Runat="Server"

/></Td>

<Td Width="25%" RowSpan="2"></Td>

</Tr>

<Tr>

<Td Width="28%" Valign="Top" Align="Right">寄件人:</Td>

<Td Width="52%" Align="Left">

<Asp:TextBox Id="txtConsigner" Runat="Server" />

</Td>

</Tr>

<Tr>

<Td Width="28%" Valign="Top" Align="Right">主 题:</Td>

<Td Width="52%" Align="Left"><Asp:TextBox Id="txtSubject"

Runat="Server" /></Td>

<Td Width="25%"></Td>

</Tr>

```
<Tr>
```

```
<Td Width="28%" Valign="Top" Align="Right">内 容:</Td>
```

```
<Td Width="52%" Align="Left">
```

```
<Asp:TextBox Id="txtContent" TextMode="MultiLine"
```

```
Rows="5" Columns="36" Runat="Server"/>
```

```
</Td>
```

```
<Td Width="25%"></Td>
```

```
</Tr>
```

```
<Tr>
```

```
<Td Width="28%" Valign="Top" Align="Right">
```

```
<Asp:Button Id="btnSend" Text="送出" OnClick="btnSend_Click"
```

```
Runat="Server" />
```

```
</Td>
```

```
<Td Width="28%" Valign="Top" Align="Left">
```

```
<Input Type="Reset" Value="清除重写">
```

```
</Td>
```

```
</Tr>
```

```
</Table>

</Td>

</Tr>

</Table>

<Asp:Label Id="lblMessage" Runat="Server" />

</Form>

<Script Language="VB" Runat="Server">

Sub btnSend_Click(Sender As Object, e As EventArgs)

    Dim Mail As New MailMessage

    Dim SendMail As New Smtplib.SendMail

    Mail.To=txtReceiver.Text

    Mail.From=txtConsigner.Text

    Mail.Subject=txtSubject.Text

    Mail.Body=txtContent.Text

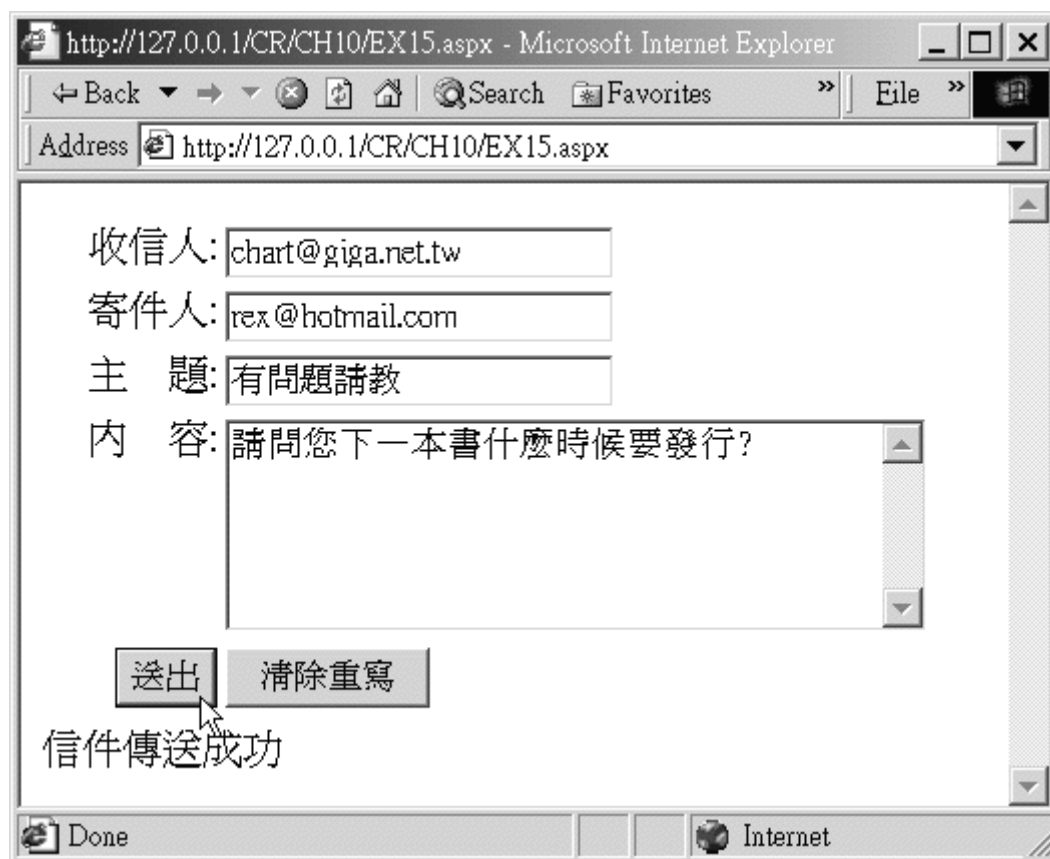
    SendMail.Send(Mail)

    lblMessage.Text="信件传送成功"

End Sub
```

</Script>

</Html>



若我們想要傳送一封圖文并茂或帶有超級鏈接的信件時，可以設定 **MailMessage** 對象的

BodyFormat 屬性。**BodyFormat** 屬性若是 **MailFormat.Text** 則代表信件內容是純文字，若是

Mail.Html 則代表信件內容包含 **HTML** 標註。如果同一封信同時要寄給很多人，則只要在設定

MailMessage 对象的 **To** 属性时，一次输入多个 **E-Mail** 地址，然后在各个 **E-Mail** 地址之间以逗号隔开，即可一次传送多人。

XML 的汇出与读取

XML 逐渐成为现今数据交换的一种标准格式，所以我们也要了解 **ASP.NET** 如何汇出与读取 **XML** 档。**XML** 档的汇出与读取主要是透过 **DataSet** 对象的 **WriteXML** 以及 **ReadXML** 方法，不过要执行档案的读写动作必需使用 **FileStream** 对象来进行档案的操作，这个档案的名称地址为 **System.IO**。

档案操作

要将 **XML** 汇出成档案要利用 **FileStream** 对象开启档案。**FileStream** 的使用语法如下所示：

```
Dim 变数 As FileStream  
  
变数=New FileStream(文件名称, FileMode, FileAccess)
```

其中 **FileMode** 的参数如下表所示：

参数	说明
FileMode.Append	如果档案存在的话开启档案，并移至到档案尾；若档案不存在则产生新档。使用本设定时，FileAccess 属性必需设定为 FileAccess.Write，若为其它值则产生错误。
FileMode.Create	产生一个新档案，若档案已经存在则会被覆写。使用本设定时，FileAccess 属性必需设定为 FileAccess.Write，若为其它值则产生错误。
FileMode.CreateNew	产生一个新档案。使用本设定时，FileAccess 属性必需设定为 FileAccess.Write，若为其它值则产生错误。
FileMode.Open	开启一个新档案。使用本设定时，FileAccess 属性必需设定为 FileAccess.Read，若为其它值则产生错误。
FileMode.OpenOrCreate	开启一个新档案或产生一个新档。
FileMode.Truncate	开启一个档案，若档案存在则删除档案的内容，所以档案的大小为 0Bytes。

其中 FileAccess 的参数如下表所示：

参数	说明
FileAccess.Read	档案只读。
FileAccess.Write	档案唯写。
FileAccess.ReadWrite	档案可擦写。

XML 档案的汇出

下列范例在使用者按下「汇出 XML」后，将第八章的会员数据表汇出成 XML 格式：

```
<%@Import Namespace="System.IO"%>

<%@Import Namespace="System.Data"%>

<%@Import Namespace="System.Data.ADO"%>

<Form Id="Form1" Runat="Server">

    <ASP:Button Id="btnOutPort" Text="汇出 XML"

OnClick="btnOutPort_Click"

    Runat="Server"/>
```



```
<ASP:Label Id="Label1" Runat="Server"/>

</Form>

<Script Language="VB" Runat="Server">

Sub btnOutPort_Click(Sender As Object, e As EventArgs)

Label1.Text="汇出中..."

Dim dscA As ADODatasetCommand=New ADODatasetCommand("Select * From
Members", _

    "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\InetPub\wwwroot\CR\Ch08\MyWeb.Mdb")

Dim dsDataSet As DataSet=New DataSet

dscA.FillDataSet(dsDataSet, "Members")

Dim fsA As FileStream=New FileStream(Server.MapPath("Members.xml"), _

    FileMode.OpenOrCreate,

FileAccess.Write)

dsDataSet.WriteXML(fsA)

fsA.Close()

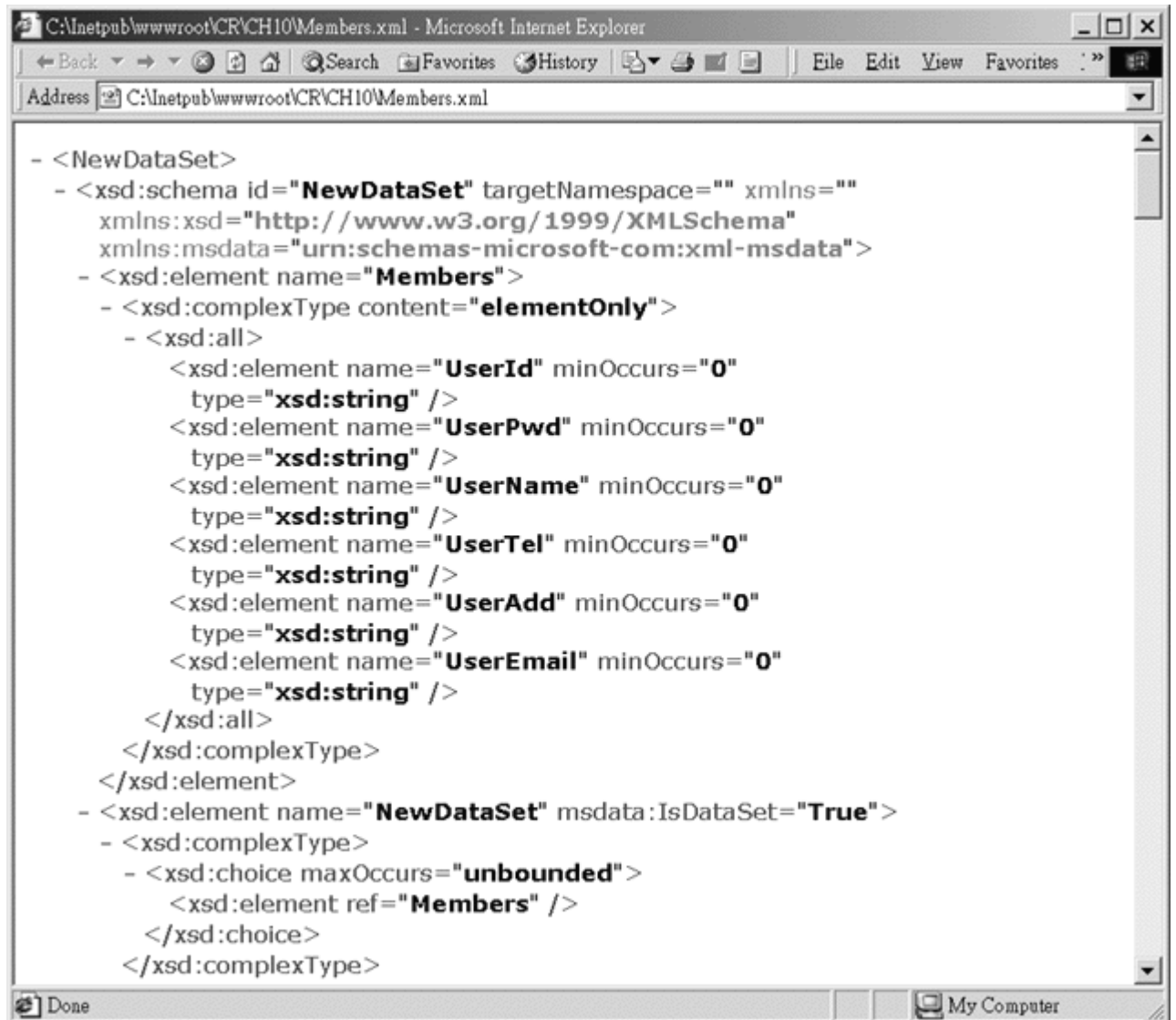
Label1.Text="汇出完毕"
```

```
End Sub
```

```
</Script>
```

```
</Html>
```

上述范例首先我们先将 **Members** 数据表填入 **DataSet** 对象中，然后再宣告一个 **FileStream** 对象并指明要在网页所在路径开启 **Members.xml** 档，如果有这个档案则开启否则就产生，而且这个档案是唯写。档案已经开启完成后，我们就可以利用 **DataSet** 的 **WriteXML** 方法将 **XML** 格式的数据写入档案中，最后将 **FileStream** 对象利用 **Close** 方法关闭即可。所汇出的 **XML** 档如下图所示：



XML 档案的读取

下列范例在使用者按下「汇入 XML」后,将刚刚所汇出的 XML 档案利用 DataSet 对象的 ReadXML

方法汇入:

```
<%@Import Namespace="System.IO"%>
```

```
<%@Import Namespace="System.Data"%>
```

```
<Form Id="Form1" Runat="Server">
```

```
    <ASP:Button Id="btnOutPort" Text="汇入 XML"
```

```
OnClick="btnOutPort_Click"
```

```
    Runat="Server"/>
```

```
    <ASP:Label Id="Label1" Runat="Server"/>
```

```
    <ASP:DataGrid Id="dgA" Runat="Server"/>
```

```
</Form>
```

```
<Script Language="VB" Runat="Server">
```

```
Dim dsDataSet As DataSet=New DataSet
```

```
Sub btnOutPort_Click(Sender As Object, e As EventArgs)
```

```
    Label1.Text="汇入中..."
```

```
    Dim fsA As FileStream=New FileStream(Server.MapPath("Members.xml"), _
```

```
        FileMode.Open,
```

```
        FileAccess.Read)
```

```
dsDataSet.ReadXML(fsA)

fsA.Close()

dgA.DataSource=dsDataSet.Tables("Members").DefaultView

dgA.DataBind()

Label1.Text="汇入完毕"

End Sub

</Script>

</Html>
```

上述范例首先我们宣告一个 **FileStream** 对象并指明要在网页所在路径开启 **Members.xml** 档，而且这个档案是惟读。档案已经开启完成后，我们就可以利用 **DataSet** 的 **ReadXML** 方法将 XML 格式的数据读入 **DataSet** 对象中，最后将 **FileStream** 对象利用 **Close** 方法关闭即可。所入出的 XML 资料如下图所示：

http://127.0.0.1/CR/CH10/EX17.aspx - Microsoft Internet Explorer

Back Forward Stop Reload Home Search Favorites History File

Address http://127.0.0.1/CR/CH10/EX17.aspx

匯入XML 匯入完畢

UserId	UserPwd	UserName	UserTel	UserAdd	UserEmail
jacky	5678	曾志遠	0933455001	台北市中正區	jacky@hotmail.com
jolin	abcd	林久翔	0939507000	桃園縣桃園市	jolin@hotmail.com
tina	6789	黃淑媛	0920086000	台北縣三重市	tina@hotmail.com
elvira	wxyz	鄧宜玲	0932123000	台北縣中和市	elvira@hotmail.com
vivi	1252	邱苑玲	0225810000	台北市中正區	vivi@hotmail.com
mary	fh43	饒惠玲	0939694000	台北縣板橋市	mart@hotmail.com
jonny	dkgf	李維華	0921951000	台北縣林口鄉	johhy@hotmail.com
kevin	23dg	鄭博文	0227908000	台北市內湖區	kevin@hotmail.com
steven	1985	廖富熒	0222013000	台北縣新莊市	steven@hotmail.com
gigi	kdji	鄭怡菁	0229354000	台北市福興路	gigi@hotmail.com
bear	1252	鄭智雄	0223922000	台北市金山路	bear@hotmail.com
chen	5952	陳正忠	0918452000	台北市大安區	chen@hotmail.com
huang	1522	黃博倫	0935979000	台北市富陽街	huang@hotmail.com

Done Internet

附录 A 函数索引

Abs(number) 取得数值的绝对值。

Asc(String) 取得字符串表达式的第一个字符 ASCII 码。

Atn(number) 取得一个角度的反正切值。

CallByName (object, procname, usecalltype,[args()]) 执行一个对象的方法、设定或传回对象的属性。

CBool(expression) 转换表达式为 Boolean 型态。

CByte(expression) 转换表达式为 Byte 型态。

CChar(expression) 转换表达式为字符型态。

CDate(expression) 转换表达式为 Date 型态。

CDbl(expression) 转换表达式为 Double 型态。

CDec(expression) 转换表达式为 Decimal 型态。

CInt(expression) 转换表达式为 Integer 型态。

CLng(expression) 转换表达式为 Long 型态。

CObj(expression) 转换表达式为 Object 型态。

CShort(expression) 转换表达式为 Short 型态。

CSng(expression) 转换表达式为 Single 型态。

CStr(expression) 转换表达式为 String 型态。

Choose (index, choice-1[, choice-2, ... [, choice-n]]) 以索引值来选择并传回所设定的参数。

Chr(charcode) 以 ASCII 码来取得字符内容。

Close(filenamelist) 结束使用 Open 开启的档案。

Cos(number) 取得一个角度的余弦值。

Ctype(expression, typename) 转换表达式的型态。

DateAdd(dateinterval, number, datetime) 对日期或时间作加减。

DateDiff(dateinterval, date1, date2) 计算两个日期或时间间的差值。

DatePart (dateinterval, date) 依接收的日期或时间参数传回年、月、日或时间。

DateSerial(year, month, day) 将接收的参数合并为一个只有日期的 Date 型态的数据。

DateValue(datetime) 取得符合国别设定样式的日期值，并包含时间。

Day(datetime) 依接收的日期参数传回日。

Eof(filename) 当抵达一个被开启的档案结尾时会传回 True。

Exp(number) 依接收的参数传回 e 的次方值。

FileDateTime(pathname) 传回档案建立时的日期、时间。

FileLen(pathname) 传回档案的长度，单位是 Byte。

Filter(sourcearray, match[, include[, compare]]) 搜寻字符串数组中的指定字符串，凡是数组元素中含有指定字符串，会将它们结合成新的字符串数组并传回。若是要传回不含指定字符串的数组元素，则 include 参数设为 False。compare 参数则是设定搜寻时是否区分大小写，此时只要给 TextCompare 常数或 1 即可。

Fix(number) 去掉参数的小数部分并传回。

Format(expression[, style[, firstdayofweek[, firstweekofyear]]) 将日期、时间和数值资料转为每个国家都可以接受的格式。

FormatCurrency(expression[, numdigitsafterdecimal [, includeleadingdigit]]) 将数值输出为金额型态。

numdigitsafterdecimal 参数为小数字数，**includeleadingdigit** 参数为当整数为 0 时是否补至整数字数。

FormatDateTime(date[,namedformat]) 传回格式化的日期或时间数据。

FormatNumber(expression[,numdigitsafterdecimal [,includeleadingdigit]]) 传回格式化的数值数据。Numdigitsafterdecimal 参数为小数字数，includeleadingdigit 参数为当整数为 0 时是否补至整数字数。

FormatPercent(expression[,numdigitsafterdecimal [,includeleadingdigit]]) 传回转换为百分比格式的数值数据。numdigitsafterdecimal 参数为小数字数，includeleadingdigit 参数为当整数为 0 时是否补至整数字数。

GetAttr(filename) 传回档案或目录的属性值。

Hex(number) 将数值参数转换为 16 进制值。

Hour(time) 传回时间的小时字段，型态是 Integer。

lif(expression, truepart, falsepart) 当表达式的传回值为 True 时执行 truepart 字段的程序，反之则执行 falsepart 字段。

InStr([start,]string1, string2) 搜寻 string2 参数设定的字符出现在字符串的第几个字符，start 为由第几个字符开始寻找，string1 为欲搜寻的字符串，string2 为欲搜寻的字符。

Int(number) 传回小于或等于接收参数的最大整数值。

IsArray(varname) 判断一个变量是否为数组型态，若为数组则传回 True，反之则为 False。

IsDate(expression) 判断表达式内容是否为 DateTime 型态,若是则传回 True,反之则为 False。

IsNull(expression) 判断表达式内容是否为 Null,若是则传回 True,反之则为 False。

IsNumeric(expression) 判断表达式内容是否为数值型态,若是则传回 True,反之则为 False。

Join(sourcearray[, delimiter]) 将字符串数组合并成一个字符串, delimiter 参数是设定在各个元素间加入新的字符串。

Lcase(string) 将字符串转换为小写字体。

Left(string, length) 由字符串左边开始取得 length 参数设定长度的字符。

Len(string) 取得字符串的长度。

Log(number) 取得数值的自然对数。

Ltrim(string) 去掉字符串的左边空白部分。

Mid(string, start[, length]) 取出字符串中 start 参数设定的字符后 length 长度的字符串,若 length 参数没有设定,则取回 start 以后全部的字符。

Minute(time) 取得时间内容的分部分,型态为 Integer。

Mkdir(path) 建立一个新的目录。

Month(date) 取得日期的月部分,型态为 Integer。

MonthName(month) 依接收的月份数值取得该月份的完整写法。

Now() 取得目前的日期和时间。

Oct(number) 将数值参数转换为 8 进制值。

Replace(expression, find, replace) 将字符串中 find 参数指定的字符串转换为 replace 参数指定的字符串。

Right(string,length) 由字符串右边开始取得 length 参数设定长度的字符。

Rmdir(path) 移除一个空的目录。

Rnd() 取得介于 0 到 1 之间的小数，如果每次都要取得不同的值，使用前需加上 Randomize 叙述。

Rtrim(string) 去掉字符串的右边空白部分。

Second(time) 取得时间内容的秒部分，型态为 Integer。

Sign(number) 取得数值内容是正数或负数，正数传回 1，负数传回 -1，0 传回 0。

Sin(number) 取得一个角度的正弦值。

Space(number) 取得 number 参数设定的空白字符串。

Split(expression[, delimiter]) 以 **delimiter** 参数设定的条件字符串来将字符串分割为字符串数组。

Sqrt(number) 取得一数值得平方根。

Str(number) 将数字转为字符串后传回。

StrReverse(expression) 取得字符串内容反转后的结果。

Tan(number) 取得某个角度的正切值。

TimeOfDay() 取得目前不包含日期的时间。

Timer() 取得由 0:00 到目前时间的秒数，型态为 **Double**。

TimeSerial(hour, minute, second) 将接收的参数合并为一个只有时间 **Date** 型态的数据。

TimeValue(time) 取得符合国别设定样式的时间值。

Today() 取得今天不包含时间的日期。

Trim(string) 去掉字符串开头和结尾的空白。

TypeName(varname) 取得变量或对象的型态。

Ubound(arrayname[, dimension]) 取得数组的最终索引值，**dimension** 参数是指定取得第几维度的最终索引值。

Ucase(string) 将字符串转换为大写。

Val(string) 将代表数字的字符串转换为数值型态，若字符串中含有非数字的内容则会将其去除后，合并为一数字。

Weekday(date) 取的参数中的日期是一个星期的第几天，星期天为 1、星期一为 2、星期二为 3 依此类推。

WeekDayName(number) 依接收的参数取得星期的名称，可接收的参数为 1 到 7，星期天为 1、星期一为 2、星期二为 3 依此类推。

附录 B 关键词

AddHandler	Alias	Append	As
Assembly	Auto	Binary	ByRef
ByVal	Case	Command\$	Compare
Date	Each	Else	Elseif
Empty	EndIf	Environ\$	Error
Explicit	ExternalSource	FALSE	For
Friend	Get	Handles	In
Input	Is	Len	Lib
Local	Lock	Loop	Me
Mid	Module	MustInherit	MyBase
MyClass	New	Next	Nothing
NotInheritable	NotOverridable	Null	Off
On	Option	Optional	Output
Overridable	ParamArray	Preserve	Print
Private	Property	Public	Random
Read	ReadOnly	RemoveHandler	Resume
Seek	Shared	Static	Step
String	Text	Then	Throw
Time	To	True	TypeOf
Until	WithEvents	WriteOnly	

附录 C 颜色成员

AliceBlue	AntiqueWhite	Aqua	Aquamarine
Azure	Beige	Bisque	Black
BlanchedAlmond	Blue	BlueViolet	Brown
BurlyWood	CadetBlue	Chartreuse	Chocolate
Coral	Cornflower	Cornsilk	Crimson
Cyan	DarkBlue	DarkCyan	DarkGoldenrod
DarkGray	DarkGreen	DarkKhaki	DarkMagenta
DarkOliveGreen	DarkOrange	DarkOrchid	DarkRed
DarkSalmon	DarkSeaGreen	DarkSlateBlue	DarkSlateGray
DarkTurquoise	DarkViolet	DeepPink	DeepSkyBlue
DimGray	DodgerBlue	Firebrick	FloralWhite
ForestGreen	Fuchsia	Gainsboro	GhostWhite
Gold	Goldenrod	Gray	Green
GreenYellow	Honeydew	HotPink	IndianRed
Indigo	Ivory	Khaki	Lavender
LavenderBlush	LawnGreen	LemonChiffon	LightBlue
LightCoral	LightCyan	LightGoldenrodYellow	LightGray
LightGreen	LightPink	LightSalmon	LightSeaGreen
LightSkyBlue	LightSlateGray	LightSteelBlue	LightYellow
Lime	LimeGreen	Linen	Magenta
Maroon	MediumAquamarine	MediumBlue	MediumOrchid
MediumPurple	MediumSeaGreen	MediumSlateBlue	MediumSpringGreen
MediumTurquoise	MediumVioletRed	MidnightBlue	MintCream
MistyRose	Moccasin	NavajoWhite	Navy
OldLace	Olive	OliveDrab	Orange
OrangeRed	Orchid	PaleGoldenrod	PaleGreen
PaleTurquoise	PaleVioletRed	PapayaWhip	PeachPuff
Peru	Pink	Plum	PowderBlue
Purple	Red	RosyBrown	RoyalBlue
SaddleBrown	Salmon	SandyBrown	SeaGreen
SeaShell	Sienna	Silver	SkyBlue
SlateBlue	SlateGray	Snow	SpringGreen
SteelBlue	Tan	Teal	Thistle
Tomato	Transparent	Turquoise	Violet
Wheat	White	WhiteSmoke	Yellow
YellowGreen			

< 全书完 >