适合开发者 项目经理 CTO&CIO 编程爱好者阅读收藏

Broadview
WWW.BROADVIEW.COM.CN

增值

程序员

2004合订本

Programmer 下

《程序员》杂志社 编

合订内容：

《程序员》全年12期精华
《开发高手》全年12期精华

超大增值内容：

新名词解析
流行工具手册
百本书评
程序员手册
2004开发年鉴
2004开发类网站精萃

含2 CD

光盘(2CD)：

《程序员》12期PDF及源码下载
《开发高手》12期PDF及源码下载
10本技术开发经典图书
[illegible]规范
[illegible]源软件介绍与源码下载
[illegible]员》全年技术沙龙视频
Sun公司技术社区技术培训课程

Csdn.net

電子工業出版社
Publishing House of Electronics Industry
http://www.phei.com.cn

杂志频道：http://mag.csdn.net/

程序员

2004 合订本（下）

《程序员》杂志社 编

总策划：蒋 涛

编委会：唐 琦 韩 磊 孟 岩 闫 辉
程 琰 霍泰稳 常 可 张浩祥
欧阳璟 罗景文 刘 婧 赵建凯
刘彦博

電子工業出版社
Publishing House of Electronics Industry
北京 · BEIJING

程序员2004合订本（下）

图书在版编目（CIP）数据

程序员2004合订本（下）/《程序员》杂志社编.—北京：电子工业出版社，2005.1
ISBN 7-121-00648-0

Ⅰ.程… Ⅱ.程… Ⅲ.程序设计－文集 Ⅳ.TP311.1-53

中国版本图书馆CIP数据核字（2004）第125478号

责任编辑：孟迎霞　熊妍妍
印　　刷：北京牛山世兴印刷厂
出版发行：电子工业出版社
　　北京市海淀区万寿路173信箱　邮编100036
经　　销：各地新华书店
开　　本：850×1168　1/16　印张：51　字数：800千字　彩插：8
印　　次：2005年1月第1次印刷
印　　数：35000册　　定价：45.00元（上、下册＋2CD）

读者订购或发现装订错误、缺损、破损，请与《程序员》读者服务部联系。
联系地址：北京市朝阳区北三环东路静安中心26层（100028）
电　　话：010-84480948/84540231～33转279
传　　真：010-84540263

程序员增值合订本内容简介

本着精心组织、全新策划、超值合订、强力推出等四大原则，《程序员》杂志社编辑部组织最强阵容全心投入，历时两个月整理编制出《程序员2004合订本》这一众多软件开发爱好者期待的书籍。《程序员2004合订本》全套共832页，分手册增值版、技术专题版上下两册，并附送两张配套光盘，相关具体内容介绍如下：

▶ 手册增值版

人物&报道：包括《程序员》的经典品牌栏目名人堂、特别策划、技术访谈、走向海外、软件创业及程序百味等。在这个栏目里可以品味软件业成功者的酸甜苦辣，程序员的感悟与梦想。

管理与实践：主要包括软件工程、交互设计、项目管理及企业应用等与软件开发相关的技术话题，2004年度最新的开发模式、测试模式及管理方法均集于此。

程序员手册：在这个栏目里我们全新组织了2004年最新名词，如何利用多种技术混合进行软件开发及国内知名技术网站精粹与热贴索引等内容，具有很高的实用价值。

2004年鉴：2004年的中国软件业有复苏的景象，在这一年中软件业究竟发生了哪些大事，在这个栏目中有一个全面的检索。而2004年百本好书与2004程序员薪资报告相信会给关心软件的朋友在技术书籍与业界薪资方面一个很好的参考。

▶ 技术专题版

《程序员》技术专题：每期的技术专题为把《程序员》打造成中国软件开发类最有影响力的专业杂志起了很大的作用，在这本合订本里我们完全重现了2004年《程序员》全年的技术专题，以飨读者。

《开发高手》技术专题：《开发高手》的定位是高校里热爱软件技术的学生，所以其文章都很有实用性，每期的技术专题也很有特色，可限于篇幅我们只是选择了几期有代表性的文章，但全部内容会在光盘里找到。

技术专区－专题：这一部分包容了Java、Open Source、.NET、REBOL、XML及数据库等软件开发技术内容，皆为《程序员》与《开发高手》的精华。而游戏开发、移动开发及网络与安全等三个小的专题栏目则将当前软件开发的热点和盘托出。

▶ 光盘介绍

CD－1：

本光盘内容主要包括了《程序员》12期PDF电子文件及全部源码下载，《开发高手》12期PDF电子文件及全部源码下载，还有CSDN网站推出的《Oracle专刊》电子版，常用开源软件介绍与安装文件等，特别适合开发者学习、查阅和收藏。

CD－2：

本光盘内容涵盖了著名技术专家2004年讲座实录及优秀技术培训课程实录，大容量的视频、音频文件给辛苦一年的软件开发者带来一顿精美的盛宴。

首先感谢您购买和阅读《程序员2004合订本（上、下）》，在阅读过程中如果您有好的建议与想法可通过参与“《程序员2004合订本（上、下）》有奖调查”活动反馈给我们，您的意见是我们前进的最大动力，我们会从所有参与调查的朋友中挑选出10名幸运读者，并送上我们准备的精美礼品。

调查链接：http://survey.csdn.net/inquiry/viewinquirystep.aspx?id=40

《程序员》技术专题

Whidbey——Visual Studio.NET 2004 之浮光掠影

C# Whidbey 值得期待的 n 个理由.....1
Lippman 谈.NET 2.0 中 C++的改变.....4
ASP.NET Whidbey 之少量代码实现 ASP.NET 应用安全.....7

MDA 的力量

MDA——从蜜蜂到工程师.....13
以正确的态度面对 MDA.....16
MDA 关键技术剖析：元建模与变换.....18
使用 ArcStyler 实现 PetStore.....22
OptimalJ 实现 MDA 的实践.....26

超越浏览器

战争与和平——纵观浏览器发展二三语.....31
Longhorn 时代，浏览器的终结？——关于 Avalon 和 XAML.....33
下一代 Windows（Longhorn）编程.....35
深入浅出 XAML.....36
Longhorn 和 Mozilla：同种羽毛的鸟.....39
XAML 之来龙去脉.....41

泛型技术和 Boost

C#、Java 和 C++中的泛型.....42
.NET 泛型与实现.....44
Boost 问答录.....45
Boost.Regex——C++正则表达式快速入门.....48

动态语言，隔岸观“火”

动静之变——裘宗燕教授访谈.....53
动态语言与 Java.....55
Why Groovy?.....57
动态语言与.NET.....58
中国不谈 Python.....59

“.NET 四周年”特别策划

.NET 四周年回顾.....60
.NET 渐入人心——CSDN 网上调查结果点评.....61
微软眼中的.NET.....64
众人眼中的.NET.....64
.NET：回首过去，展望未来.....67
在蹉跎中一路前行——谈 Microsoft .NET 战略.....68

来吧！移动开发

如何为 Microsoft Smartphone 开发应用程序.....78
J2ME 手机游戏开发入门指导.....81

代码安全

闲话代码安全.....85

缓冲区溢出攻击的原理与防范.....87
来自网络的攻击.....91
第三方程序代码安全检查技术.....93

ICE：夏天里的零度

Marc Laukien 访谈录.....98
新一代面向对象中间件.....100
大型多玩家游戏中间件.....105
Hello World——ICE 分布式应用开发入门.....108

会战 2005：J2SE 5.0 vs. .NET 2.0

锦上添花：J2SE 5.0 语言新特性.....111
元数据：J2SE 5.0 之 Annotation vs. .NET 2.0 之 Attribute.....113
匿名方法：.NET 2.0 的新利器.....114
殊途同归——J2SE 5.0 和.NET Framework 2.0 的虚拟机进展.....116
多线程：J2SE 5.0 略胜一筹.....117
数据访问：JSR114 比拼 ADO.NET.....119
从企业应用的核心价值看 Java 与.NET.....120
站在历史的河流里——写在 J2SE 1.5 与.NET 2.0 发表之际.....121
在较量中携手前行——写给 J2SE 5.0 和.NET 2.0.....123

C++/CLI：凤凰的涅槃

“C++/CLI 全景体验”开篇语.....124
聆听未来——Stan Lippman 谈 C++/CLI.....127
CSDN C++/CLI 调查报告.....129
C++/CLI 会冲击 C#吗？.....130
C++/CLI：鼎新革故.....130
山雨欲来风满楼——标准 C++及 C++/CLI 发展综述.....132
非典型 C++/CLI 教程.....134

2004 侧影

驻足观望的一年——我的 2004 思考.....136
变革前的思索——我的这一年.....137
2004 年，逝者如斯夫.....139
CTO 的第三只眼：从程序员说起.....140
元年移动 2004.....140
2004，我在写程序.....141
2004，最喜欢的一年.....142
我的 2004，我的林荫小路.....142
回顾 2004.....143

《开发高手》技术专题

Tiger 出山：Java 语言的又一次变革

探索 JDK1.5 的新特性.....144
JDK1.5 代码示例.....147
使用 Java 泛型创造更灵活的类——Pluto 中添加新的 Portal Page 和 Portlet.....150
对 JDK1.5 你是否满意？——专家讨论自动拆箱（Auto-unboxing）的危险.....153

Eclipse：最佳 IDE 工具的革命

关于 Eclipse 155
使用 Eclipse 进行 J2EE 项目开发 160
使用 Eclipse 环境开发手机应用程序 167
使用 EMF 开发基于模型的 Eclipse 插件 170

用.NET 打造自己的个人信息管理中心

系统分析与设计 174
开发数据库存取对象 178
开发自定义树控件 182
PersonalInfo 软件开发过程实录 187

基于 Java 的网上考评系统

系统分析与设计 190
环境配置篇 194
在考评系统中进行分层设计 197
模式的运用及代码重构 201

网络五子棋游戏开发

系统分析与设计 204
算法设计与网络通信 206
使用敏捷方法开发五子棋游戏 210
他山之石，可以攻玉——点评五子棋小游戏的开发 212

技术专区-专题

Java

JDK 1.5 的泛型实现 214
Java 多线程编程实例——优化 Cache 并发访问性能 218
J2EE 企业架构的参考模型和实现策略 220
剖析 EJB 224
EJB 3.0 先睹为快 228
OptimalJ 实现 MDA 的实践 231
Java 到.NET 的转换利器——JLCA 237
J2EE 报表开发技术基于 JReport 的报表实现 239

Open Source

蚀之韵——Eclipse 的敏捷开发实践 244
使用开源 Java 框架开发 Web 应用 247
Hibernate：一个简洁高效的 ORM 框架 252
PHP 开发环境的选择、建立及使用 255

.NET

.NET 新特性：ObjectSpaces 基础教程 262
一窥 Visual Studio .NET Whidbey 267
多线程实现并发访问——基于.NET 的服务器程序 269
深入.NET 控件开发——System.Windows.Forms 中的 Windows 窗口消息机制 272
.NET 环境下多层体系架构的实现 275
基于 Microsoft .NET 的加密与签名系统开发 279

基于组件的.NET 软件开发 284
COM 与.NET 互操作之实战与分析 288
解析.NET 的异步调用（Asynchronous Calls） 292

REBOL

初识 REBOL 296
Desktop、IOS、SDK、Plugin 299
REBOL 解释器与 Script 304
REBOL 的数据类型之一 309
REBOL 的数据类型之二 315
视觉接口方言（VID）之一 320
视觉接口方言（VID）之二 324

XML

X 档案——Java 持久化的另类解决方案 327
使用 XmlSpy 开发 XML 330
Xquery——XML 时代的数据查询语言 333
巧用 XML 实现票据精确打印 339

数据库

Oracle 面向对象特征新进展 342
应用 XML 数据库 344
利用 Oracle XML DB 存储 XML 数据 346
基于 XML DB 开发信息管理系统的新模式 349
SQL Server"Yukon" 中又新又酷的特性 354
彻底解决 MS SQL Server 2000 中最大流水号的生成问题 356

游戏开发

做一名游戏设计师 358
从《魔兽争霸 III》看 DataChunk 的应用 359
游戏开发中的脚本应用 363
我的游戏制作人面面谈 366

移动开发

短信应用开发基础 370
在 Pocket PC 中使用 Web Service 连接数据库 372
微软智能手机游戏开发经验谈 377
使用 Microsoft .NET 框架精简版编写移动游戏 379

网络与安全

利用验证密码拒绝非法访问 385
Windows 资源使用完全监视 386
浅谈网络实时监控系统的设计与实现 392
浅析木马服务端的生成技术 394

附录

《程序员》2004 年全年目录 397
《开发高手》2004 年全年目录 403

《程序员》2004.01

Whidbey——Visual Studio.NET 2004 之浮光掠影

“在每一个重要关头，先进的工具总是成为推动应用软件新浪潮的关键，而每一次应用软件浪潮又是推动计算领域迈向新水平的关键。”——比尔·盖茨

在即将来到的 2004 年，名为 Whidbey（美国一个风景怡人的小岛）的 Visual Studio.NET 2004 将以更加革命性变化出现在所有的 Windows 开发者面前。

改进是全方位的。

.NET framework 将会进化到 2.0，除了库的扩充之外，更多的还在于内部语言机制的改进。通过这些改进，.NET framework 2.0 对多语言的支持更加完善。而诸如泛型等语言机制也将得到很好的表现

Visual Studio.net 所支持的几门语言也将会有更多的改进。例如 C#对泛型的支持，对分布式类型的支持；而 C++中对语言的改进也是非常的出色，比如新的引用类型，使得 C++这种强类型的静态语言能够更好与.NET 模型配合；而 ASP.NET 对于安全性方面的改进也使得以前需要很多手工编写代码的工作现在也得以轻松实现。

Visual Studio 开发集成环境的进步同样引人瞩目。对重构的直接支持恐怕将会是这一版本最大亮点之一，而其它诸如智能感知技术、代码扩展技术也是使得程序员在开发的过程中感受到更多的乐趣。

Whidbey 的改进实在是太多了，我们在这里无法向你一一提供。何况 Whidbey 本身也处在变化之中。本期技术专题，我们也仅是挑选其中一些技术细节，试图让读者领略一番 Whidbey 的魅力，虽然并不全面。就算是管中窥豹吧。

C# Whidbey 值得期待的 n 个理由

韩磊 / 文

在.NET 社区中，对于 C# Whidbey 的讨论已经是甚嚣尘上，其势汹汹了。Whidbey 值得期待吗？从微软发布的白皮书和其它一些资料中，Whidbey 令人兴奋的一些新特性已经曝露出来。在阅读了所有这些资料之后，我们不能不得出一个结论——微软打算做开发领域的 Windows。也就是说：向开发人员提供生产效率更高、使用更方便的工具。我并无丝毫贬损 Unix/Linux 等其它操作系统的意思，但如果没有 Windows/Mac OS 这样的 GUI OS，恐怕计算机普及率不会达到今天的水平。那么，为什么不说微软打算把 Whidbey 做成 Mac OS 呢？很简单，.NET 是一个开放平台，与 Mac OS 的闭关锁国全然不同。下面我们将对 Whidbey 的新增/增强特性作走马观花式的检阅。

语言特性增强

泛型（Generic Type）

在语言方面，最可期待的就是对泛型的支持了。不管是代码重用、模型重用还是模式重用，都极度地反映了在开发中对“重用”的强烈需求。而泛型，无疑将使得我们的工作更具价值。

C#支持泛型，允许把 class、struct、interface、method 根据其储存和使用的数据类型进行参数化。来看看这个简单的例子：

```
public class Stack<ItemType>
{
        private ItemType[ ] items;
        public void Push(ItemType data) {...}
        public ItemType Pop() {...}
}
```

此例声明了一个 generic class（泛型类），接受名为 ItemType 的参数。很明显，我们没有固定对 ItemType 的数据类型，这意味着 client method 可以在构造 Stack 类时传入任何类型的参数。ItemType 本质上是一个 PlaceHolder，将根据参数填入不同数据。象这样：

```
Stack<int> stack = new Stack<int>();
stack.Push(3);
int x = stack.Pop();
```

传入 int 参数，实际上是创建了一个 constructed type。Stack 中所有 ItemType 出现的地方都被替代为 int。也可以象这样：

```
Stack<Customer> stack = new Stack<Customer>();
stack.Push(new Customer());
Customer c = stack.Pop();
```

传入自定义数据类型 Customer。你无需作数据类型转换等等令人烦躁的工作，但必须记住，泛型是强类型检查的，你不能在创建时传入

Customer，却试图使用这样的语句：

```
Stack.Push(3);
```

分布式类型（Partial Type）

如果让三个人同时写一个类/窗体，是不是 Mission impossible？在 C# Whidbey 中，利用 Partial Type 特性，可以轻而易举地做到这一点。所谓 Partial Type，就是同一个类的代码可以放到不同的.cs 文件里面。例如窗体 Form1，原来只应该对应 Form1.cs，而现在则可能是这样：

```
// Form1Designer.cs
public partial class Form1: System.Windows.Forms.Form
{
    // Designer code
    void InitializeComponent() { ... }
}

// Form1User.cs
public partial class Form1
{
    // user code
    void Mouse_Click(object sender, MouseEventArgs e)
    { ... }
}
```

好处显而易见。你可以做适当的分配，让一个人设计窗体、另一个人写事件响应代码、而三个人则负责用户方法实现。团队开发更为精致、专业化。现在可以让专业美工用 VS.NET 直接设计窗体界面了。

匿名方法（Anonymous Method）

如果要实现“点击鼠标后弹出消息框”，应该怎么做？你会毫不犹豫地写出以下代码：

```
this.button1.Click +=
        new   System.EventHandler(this.button1_Click);

 private void button1_Click(object sender, System.EventArgs e)
 {
        MessageBox.Show("Click");
 }
```

在Whidbey中，无需如此复杂，这样就可以了：

```
button1.Click += delegate { MessageBox.Show("Click"); };
```

这就是所谓 Anonymous Method，可以简单理解为接受代码块作为参数的方法。上例中，编译器从等号左边的语句得到“参数代码块应该返回 EventHandler 类型”的信息，并正确地作了处理。有时候我们需要指定返回类型，用下面的语句就可以了：

```
button1.Click += delegate(object sender, EventArgs e)
        { MessageBox.Show(sender.ToString()); };
```

别名限制符（Alias qualifier）

.NET 引入 namespace（命名空间）的概念，但这个概念尚有不足之处。例如，我以自己姓氏的拼音作为自定义控件的 namespace（当然，这并不是一个很好的例子，而且我也不鼓励读者这样做），然后如此实现这个类：

```
namespace Han
{
  namespace System
  {
      class Example
      {
          static void Main()
          {
              //在VS2003中，会在命名空间Han.System
              //中查找Console.WriteLine方法
              System.Console.WriteLine("Hello");
          }
      }
  }
}
```

读者一定会大叫起来，System.Console.WriteLine("Hello");这句有问题！问题在哪里呢？就在于在本例中，System.Console 会被编译器误认为归属于自定义 namespace han.System。而我的原意却是指根 namespace System。C# Whidbey 引入别名限制符“::”（双冒号）前缀，解决了这个问题。当你在 namespace 前面加上双冒号时，就是特指 root namespace。如下所示：

```
:: System.Console.WriteLine("Hello");
```

静态类（Static Class）

考虑这样的场景：我需要设计一个封装类（sealed class），其中有一个私有构造器，用来实现一些静态方法。由于目的在于实现静态方法，所以我并不希望用户创建类对象。正常的方式是：

```
public sealed class Environment
    {
      // 防止创建类对象
      private Environment() { }
    }
```

这种方式无疑是笨拙的。公布一个 class，然后又宣称它是私有的，感觉就像签订契约然后撕毁一样。在 C# Whidbey 中，可以用 static class 来解决问题：

```
public static sealed class Environment
    {
    }
```

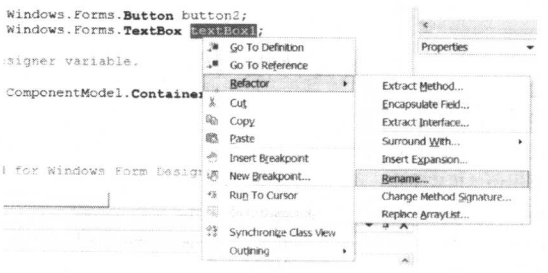
这回的契约比较完善，明确地告诉对方这个 class 只包括 static 方法，不可以创建类对象。

生产力提升

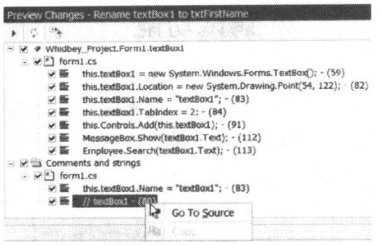
重构支持

对于有经验的开发者来说，重构（Refactoring）几乎是编码过程中随时可能进行的工作。我现在使用集成到 VS.NET IDE 的第三方重构工具，不免会感到有些不爽。在新版本 VS.NET 中，目前透露出来的消息是至少将集成支持以下重构式：

- Extract Method
- Rename
- Extract Interface
- Encapsulate Field
- Change Method Signature
- Replace Arraylist



图一　集成的重构支持



图二　利用重构工具重命名对象

很遗憾，微软文档中透露的重构支持，并没有超越我手边这个免费工具。当然我会毫不犹豫地使用集成的重构支持，尽管还是会安装目前使用那个工具（如果它继续支持 VS.NET 新版本的话）。

代码扩展（Expansion）

Delphi 开发者都会知道 CodeRush，一个 IDE 集成工具。我很喜欢 CodeRush 的代码模块功能。例如，键入 for 和空格，整个循环结构代码块就自动出来了。微软开发工具一向以代码编辑器易用性较好而傲视群雄，这个功能当然必须支持。类似这样一个循环：

```
for (int i=myList.Count-1; i>=0; i++){
        //do something
  }
```

我可以在 XML 文件中定义填充代码块：

```
<title>Reverse for-iteration by index</title>
<shortcut>forr</shortcut>
<description>Expansion snippet for reverse 'for' loop</description>
<category>Expansion</category>
<category>SurroundWith</category>
</header>
- <snippet>
  - <declarations>
    - <literal>
        <id>index</id>
        <default>i</default>
      </literal>
    - <literal>
        <id>max</id>
        <default>length</default>
      </literal>
    </declarations>
  - <code language="csharp">
    - <![CDATA[
        for (int $index$ = $max$ - 1; $index$ >= 0 ; $index$--)
                                    {
                                    $selected$ $end$
                                    }
      ]]>
    </code>
</snippet>
```

图三　定义填充代码块

然后在需要的地方使用右键菜单选择 forr，编辑器就会积极主动地写出整个结构，然后留出$index$、max的部分让我填写需要的内容：

```
List<int> myList = new List<int>();

myList.Add(1);
myList.Add(2);
myList.Add(3);

for (int i = myList.Count - 1; i >= 0; i--)
{

}
```

图四　Expantion 功能

增强的智能感知（IntelliSense）

微软的 IntelliSense 技术永远不会进化到电影《A.I.》中机器人小孩的水平。不过在 IDE 中放一个渴望爱的小孩，并不能解决手腕因为打字太久而酸软无力的问题，也无法医好程序员职业病——肩周炎。减少击键是问题的解决之道，而增强的 IntelliSense 技术，正好能让我们这些程序员看起来更加象脑力劳动者。在下图中，IntelliSense 正确感知到 myList 的元素类型，且在 hint 窗口中作了很好的解释。

```
static void Main(string[] args)
{

    List<int> myList = new List<int>();
    myList.Add(
    int List<int>.Add(int item)
```

图五　正确感知对象元素类型

在异常处理方面 IntelliSense 也有不俗表现。对于 try/catch 语句，IntelliSense 会列举出相关异常，“选择并回车”就万事大吉了。

```
try
{
    Employee c = new Employee();
}
catch(
{
}
try
{
```


图六　感知异常处理语句

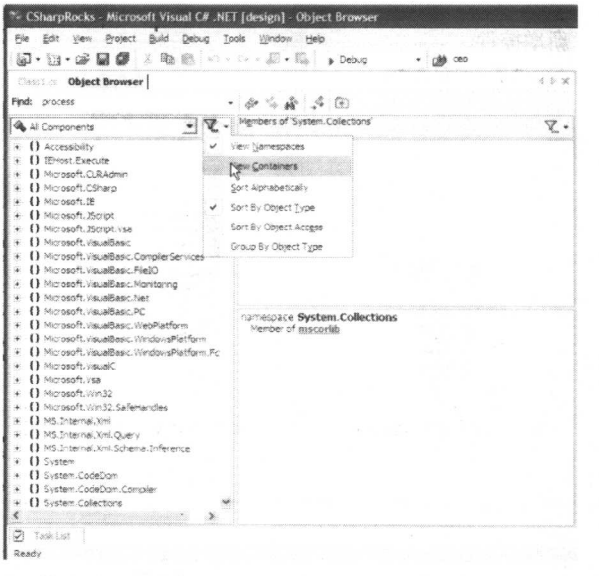
相应地，IntelliSense 也能很好地认知 Attribute：

```
namespace CSharpRocks
{
    [
```


图七　键入[后的 IntelliSense 菜单

对象浏览器

现在的对象浏览器比较鸡肋的是排序方式太少。而在 Whidbey 中，你甚至可以按照 namespace 排序。



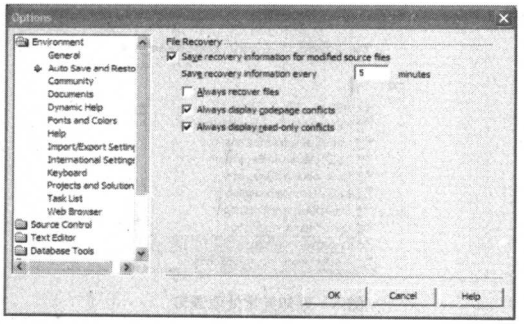
图八　排序方式更多的对象浏览器

自动保存

Palm 用户肯定会对这个 VS.NET IDE 新增加的特性嗤之以鼻——在 Palm OS 里，压根没有“保存”这个菜单项，因为系统会替你做保存工作。我有一位同事，在遥远的 DOS 年代，埋头打字数小时之后突然系统崩溃，真是叫天不应，叫地不灵啊。所以 MS Office 的“自动保存”功能一直为他所赞赏。

有没有试过埋头编码时突然断电？或者，有没有对时不时要记得保

存改动感到厌烦？看看下面的截图，下一版本 VS.NET IDE 已经能够为你按照指定时间间隔保存文件。



图九　自动保存选项

改动追踪（Change Tracking）

如果你对代码做了部分修改，系统将在编辑器左边中用黄色标识出未保存的新增代码，绿色则表示保存了的新增代码。Borland Delphi IDE 中，如果对代码做过修改，状态栏会显示 Modified，VS.NET 2003 则不会有任何提示，工具栏上的保存图标在任何时候处于可以点击的状态。

有了新的“改动追踪”功能，你可以随时知道自己做了哪些工作。

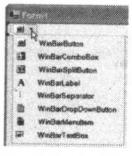
```
using System;
using System.Collections.Generic;

class Class1
{
    static void Main(string[ ] args)
    {
        //newly inserted code is highlighted
        List<int> myList = new List<int>();
        myList.Add(1);
    }
}
```

图十　一目了然的改动追踪

新增控件

不多说了，自己看图：



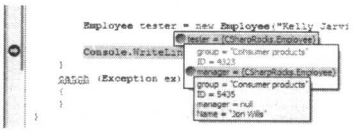
图十一　新增控件 WinBar

图十二　新增 Web Form 控件

调试功能

数据提示（DataTips）

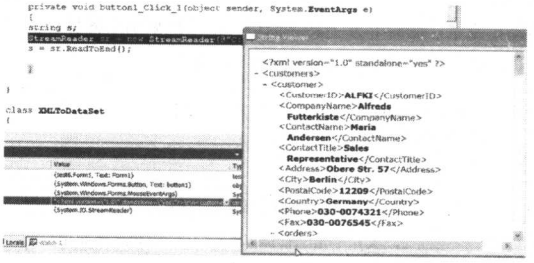
使用 VS.NET 2003 进行 debug 时，把光标放到某个变量上面，会显示该变量的当前值。在 Whidbey 中，此功能大大增强了。



图十三　增强的 DataTips

直接查看

如果在 VS.NET 2003 中作调试，基本上没有很好的方法去查看一个动态生成的 DataSet 或是 XML Stream。Whidbey 可以做到这一点。你可以即时查看数据集或是 XML Stream。



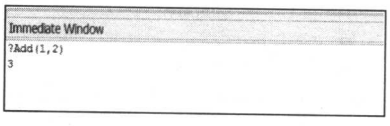
图十四　查看 XML Stream

设计时表达式计算

在 Whidbey 的 Immediate 窗口中可以做表达式运算。什么意思呢？假设我有这么一个方法，该方法简单计算两个参数之和。

```
class class1
{
        public static int Add(int a, int b)
        {
            return a+b;
        }
}
'
```

那么，我将可以在 Immediately 窗口中直接输入?Add(1,2)，并且得到“3”的反馈值。■



图十五　设计时表达式计算

Lippman 谈.NET 2.0 中 C++的改变

王昕 / 文

在即将来临的.NET 2.0 中，Microsoft 对它进行了很多的修改，其中也包括被广大程序员所关注的 C++。作为微软的 C++语言设计架构师，Lippman 向我们展示了在.NET 2.0 中许多新的变化，其中最引人瞩目之一就是一种全新声明引用类型。

这是原来 C++中的写法：

```
// original language
```

```
Object * obj = 0;
```

而在.NET2.0 中将会被要求修改成这样：

```
// revisied language
Object ^ obj = nullptr;
```

这样做的意义何在呢？我们为什么要使用^，以及为什么我们需要在C++中新增一个语法条目呢？

Stan Lippman 说到：

尽管我们可以在.NET 中使用 C++来开发应用程序，但这两者在最基本的对象模型之间就存在着很大的不同。

因此他决定对此进行一些修改，具体的做法如下：

众所周知，C++是构建于面向具体机器的系统观点（System View）上的。即便它可以支持高层的类型系统，但我们还是有办法绕过这个类型系统。这些回避机制常常将我们导入到有关机器的细节中去——我们可以对类型进行转换、抑止虚拟机机制起作用、甚至直接将程序中的名字和具体的地址联系起来。C++中支持着一种静态的（即编译期中）规则——即使我们是在运行期中对它们进行封装和转换，在编译期中它们就被严格地定义好了："它们"包括我们上面所谈到的虚拟机机制，以及我们尚未谈到的异常处理机制。当程序员觉得无法从正常的途径得到他们所希望的结果时，他们就会绕过程序的抽象模型，转而使用诸如"地址"和"偏移"这些和具体机器相关的类型。还记得在最初的星战中，Luke 决定不使用计算机辅助系统，转而依赖于一些熟练的 C 程序员来进行其中的动画处理。你能相信他们可以很好地做到这一点吗？你能发现在电影中的宇宙危机吗？

.NET 的基础是一个已经构建好的软件组件层，通常我们也称它为通用语言运行时（Common Language Runtime，或者 CLR）。这是一个用来解决复杂问题的基础设计。

还记得当我（此处指 Stan Lipmman）开始编程时，我在 Bell 实验室所得到的最好的忠告就是：如果你不知道如何解决一个问题时，向里面添加一个指针（实质上也就是一个间接层）就可以让你得到足够的空间来很好地解决它。

在 C++中相应的做法就是定义一个新类——也就是说，新增一层抽象。这也就是我们在 STL 中为什么要使用迭代器（iterator）来对 vector（其中空间是连续的）和 list（非连续空间）进行透明①的遍历。这样做给我们带来的权衡包括两部分：一是这个新增的抽象层会给程序的复杂性带来一定的影响；另外一个则涉及到可能会发生效率的问题。这两点也通常被认为是这种做法所带来的缺点。另外，该做法的优点在于：这样做会扩展程序的弹性，使得程序更容易被我们所修改。这也就是 .NET 所希望做到的，不过.NET 中的抽象层是位于软件的运行（runtime）时之上的。.NET 支持的是一种动态的（即运行时中的）机制。当程序员碰到问题时，他们会向执行环境反映（reflect）这一点，从而从字面上来查找、编码、以及创建对象。

如果将 C++中的静态对象模型称为 Kansas，把.NET 中的动态对象模型称为 Oz；那么为了将它们联系起来而进行的语法设计将会同时涉及到 C++语言本身和一个庞大的 C++社区。要做到这两点中的无论哪点都不是一件轻松的事情。

现在，让我们来考虑一个具体的例子吧。假设我们在程序中碰到如下的代码，它的意思会是什么呢？

```
T t;
```

在 ISO C++中，不管 T 的本质是什么，我们都可以确信：（1）对于 t 来说，在编译期中我们会为它分配 sizeof(T)字节的内存；（2）在 t 所处的上下文中，它所占据的内存空间不会和其他对象所冲突；（3）t 的状态/值被直接地保存在该内存空间中；（4）在 t 的生命周期内，该内存空间将会被一直保持。

从上面的结果我们可以得到如下的推论：条目（1）告诉我们 t 不能被用于多态（polymorphic），也就是说，我们不能在编译期中为多态类型进行内存分配（当然，如果我们的派生类型所需的内存空间和其基类型一样的话，这一点是允许的）。只有当类型被一个可以改变其含义的记号（token）所修饰时（如使用 T*来代表一个指针，T&来代表一个引用），在 C++中使用多态类型才是可能的，而这样做得到的结果只能是一个用来代表 T 对象的间接引用对象。

将值和引用从一种简单的符号类型系统中分隔开来的做法是 Bjarne Stroustrup 在 70 年代末期所做的经过了深思熟虑的设计决策，该决策来自于 Stroustrup 在剑桥大学读博士时使用 Simula-68 的经验。在 Simula-68 中，所有的对象都分配自运行期的堆中，所有对对象进行的访问都是通过一个透明的句柄来进行的。

为了将资源分配推迟到运行期中，C++中采用了如下两种间接方式：

1. 指针形式：T *pt = 0;
2. 引用形式：T &rt = *pt;

而这两种形式都不适合于传统的 OO 语言模型。

指针可以很好是适合 C++的对象模型，在如下的语句中：

```
T *pt = 0;
```

pt 所容纳的只是一个类型为 size_t 的值，它的长度和作用范围都是固定的。对指针的直接引用和对它所指的对象的间接引用，我们都是从字面上来区分它们的。有时，对于如下的用法：*pt++，我们也可能会到底是哪种模式被使用产生混淆。

使用引用可以从语法上减轻由于使用指针所带来词法方面的复杂性，并且仍然可以保持指针所带来的高效性。如：

```
Matrix operator+(.const Matrix&, const Matrix& );
Matrix m3 = m1 + m2;
```

对于引用，不存在着所谓的直接和间接使用。但我们必须记住如下两点：（1）在初始化过程中，它们是被直接使用的；（2）在后续的操作中，引用则是透明的。

从某种意义上来说，引用给 C++对象模型带来了很多的不规则做法：（a）除临时对象外，它们会占据内存空间，但它们确不具有和其他对象相区别的实质内容；（b）在赋值时，它们会使用深拷贝（deep copy），而在初始化时，它们使用的则是浅拷贝（shallow copy）；（c）和 const 不同的是，它们是真正的不可被修改的对象②。

虽说在 ISO C++中，除了用在函数参数上之外，引用没有很大的用处。但它对于我们在.NET 2.0 中产生的语言修订起了一个启发的作用。

任何一个关于C++中用于.NET的扩展特性的问题到最后都可以被简化为如下的这个问题：

我们应该如何将 CLR 中的这个（或那个）特性整合到 C++中，以使得：（a）C++程序员对它不会觉得突兀；（b）它可以像在.NET 中那样被方便地使用。

说到这，Stan Lippman 给我们提了一个问题用来刺激我们进行更深层次的思考：

我们应该如何声明和使用.NET 中的引用类型呢？请注意，.NET 中的对象模型和 C++中的对象模型有着很大的不同：不同的内存模型(.NET 中支持垃圾收集)；不同的复制语义（浅拷贝）；不同的继承模型（单根继承，.NET 中有着一个公共基类 Object，此外.NET 中还支持"接口"这个语言特性。）

对于.NET 来说，其核心的特性就是引用类型，因此将它和已有的 C++语言整合到一起也可以用来代表一个概念上的证明。那么，我们该如何得到一个通用的评判标准呢？我们需要一种方法来表示.NET 中的引用类型，使得它既独立于已有的类型系统之外，又和该系统相类似。这将使得用户在不熟悉它的独特特性之前也可以向其他他们已经熟悉了的特性来使用它。我们所采用的类似物来源于 Stroustrup 在 C++中所发明的引用。也就是说，我们的通用形式就应该表示为如下的形式：

```
Type TypeModToken Id [ = init ];
```

此处，TypeModToken 是一个将会被用于其他上下文中的记号。

对于上面的语句，我们期望 TypeModToken 能够在对它进行操作符函数调用时表现出和对象一样的特性——而这是以往的语法所不能支持的。这种特性也被 Lippman 称为弹性引用（flexible reference），以示于和 C++中已有的引用相区别：

1．它能够不指向任何的对象。而在 C++中，这一点显然是不能直接做到的（虽然我们可以通过使用 reinterpret_cast 来用 0 来初始化一个引用）。将一个引用“指向”空对象的常用做法是提供一个显式的 singleton，用该对象来表示一个空对象。这种做法通常都用于函数的缺省参数中。

2．它可以不需要初始值，但它的生命周期也可以始于指向空对象的时候。

3．它可以用来指向另外的对象。

4．在缺省的情况下，对它的初始化和赋值引起的将只是浅拷贝。

通过对与原有的 C++引用（而不是.NET 中受控的引用）的比较，Lippman 把这种新的类型称为句柄而不是指针或引用。句柄是一个用来表示封装的术语，它所表示的意思就是：我们正在操作的对象可能会在我们所不清楚的情况下消失。在.NET 中，这种消失的原因可能是：在一次垃圾清理中，我们对于被引用的对象进行了内存空间的重新分配。至于这种重分配的过程中所发生的操作，对于用户来说是透明的，用户只能通过句柄（它的值会自动更新为指向被重新分配的对象上）来访问该对象。在静态语言（如 C++）中，这种做法十分复杂，并且开销也很大，有时甚至还会破坏程序运行时的堆。这也就是为什么.NET 中没有大力支持指针的原因。

下面给出的是三个用来表示全局对象的句柄声明：

```
Object^ obj;
Object^ poly = gcnew Foobar;
Object^ obj2 = poly;
```

obj 是一个指向“空”对象的句柄。如果不是用于表示全局句柄，那么它的等价形式应该是：

```
Object^ obj = nullptr; // 局部对象不能被缺省地初始化
```

poly 是一个指向类型为 Object 的句柄，它被初始化为指向分配于受控堆上的 Foobar 对象。由于我们现在支持两种不同的动态堆内存——本地堆（不会有垃圾收集）；以及受控堆——我们还为从不同的堆中分配内存设计了不同的语法规则。在修订的语言设计中，我们新增了一个关键字：gcnew，用于表示从.NET 的受控堆中分配空间。例如，下面是对 System:: Windows::Forms::Button 对象进行分配的老新版本：

```
Button __gc *button1 = __gc new Button(); // 使用了显式形式
Button^ button1 = gcnew Button;
```

最后，我们用来对句柄 obj2 进行初始化的操作并没有导致逐位拷贝（bit-wise copy）的产生，而在 ISO C++中是必须的。我们在此所进行的拷贝是一个浅拷贝，它只是简单地将 obj2 和 poly 指向了同一个对象。

我们对类型表示的改变所产生的一个副作用就是：现在我们需要用一种明确的形式来表示引用没有指向任何对象。用 0 来初始化或对引用进行赋值再也不能表示将引用指向空地址。例如：

```
obj = 0; //会导致一个对0的隐式装箱操作，而不是将obj指向“空”对象
```

在将已有的受控 C++代码移植到新修订的语言中时，这会导致一些难以捉摸的问题产生。例如，我们来考虑如下的用于表示值的类声明：

```
// the original language syntax
__value struct Holder
{
  Holder(Continuation* c, Sexpr* v) {
        cont = c;
        value = v;
        args = 0;
        env = 0;
  }
private:
  Continuation* cont;
  Sexpr* value;
  Environment* env;
  Sexpr* args __gc [];
};
```

由于 args 和 env 都是受控引用，在构造函数将它们初始化为 0 在新的语法中将不会再保持它们在已有代码中的意思，我们必须将它们改为 nullptr（有专门的工具用来自动进行这种转换）才可以保证上面的代码不会产生什么问题：

```
// the revised language syntax
value struct Holder
{
  Holder( Continuation^ c, Sexpr^ v ) {
        cont = c;
        value = v;
        args = nullptr;
        env = nullptr;
  }
private:
  Continuation^ cont;
  Sexpr^ value;
  Environment^ env;
  array<Sexpr^>^ args;
};
```

与之类似的是，将这些成员和 0 相比的操作也必须改为和 nullptr 相比较。下面是原有语法下的代码：

```
// the original language syntax
Sexpr* Loop (Sexpr* input)
{
  value = 0;
  Holder holder = Interpret(this, input, env);
  while (holder.cont != 0) {
        if (holder.env != 0) {
              holder = Interpret(holder.cont,
                    holder.value, holder.env);
        }
        else if (holder.args != 0){
              holder = holder.value->closure()->
                    apply(holder.cont, holder.args);
        }
  }
  return value;
}
```

下面的代码是用新的语法所改写的：

```
  // the new revised syntax
Sexpr^ Loop ( Sexpr^ input )
{
  value = nullptr;
  Holder holder = Interpret( this, input, env );
  while ( holder.cont != nullptr ) {
        if ( holder.env != nullptr )
        {
                    holder = Interpret(
              holder.cont, holder.value, holder.env );
        }
        else
        if ( holder.args != nullptr )
        {
              holder = holder.value->closure()->
                    apply( holder.cont, holder.args );
```

```
        }
    }

return value;
}
```

最后，关于这种新的引用语法，我们还需要注意在于它的成员选择操作符（member selection operator），虽说 operator.和 operator->这两种形式都不令人满意，但我们还是从不同的层面来讨论了引用的使用方式：

```
// the pointer no-brainer
T^ p = gcnew T;
// the object no-brainer
T^ c = a + b;
```

这样的话，引用就可以同时在对象和指针这两种程序上下文之间使用。我们的成员选择操作符选用的是原有语言中的 operator->。

除了上面我们所谈论的新增的这种引用类型声明语法之外，Stan Lippman 还在其 blog 中给出了其他的有关 C++语言修订的文章，具体可以参考 Stan Lippman 的 blog：http://blogs.gotdotnet.com/slippman/default.aspx?date=2003-12-17T00:00:00 ■

① “透明”的意思就是用户意识不到他所遍历的到底是连续的 vector，还是非连续的 list。

② 我们可以通过间接的方式来对 const 进行修改。

ASP.NET Whidbey 之少量代码实现 ASP.NET 应用安全

Michele Leroux Bustamante / 文　韩磊 / 译

我曾经搭建过许多 ASP.NET 应用：客户端程序和原型、自己的站点、朋友们的站点、文章站点、培训课程站点等等。我常发现自己在每个应用中重复着一些工作，其中尤以认证模块最为常见。ASP.NET 1.x 提供简单、安全的，基于窗体的验证过程，这使得问题较易得到解决，但你仍需自行实现角色管理。假使每做一个新的登录窗就得 1 分钱，那我口袋里肯定装了不止 10 美元——算一下，有多少个 form。将发布的 ASP.NET 版本、Whidbey，会提供一些新的配置工具、控件和组件，用以支持完整的用户认证和资源保护管理系统。这些新特性如此直观，甚至于祖母们都可以在一天之内创建安全的站点。

传统的认证模块往往按照以下步骤构建：

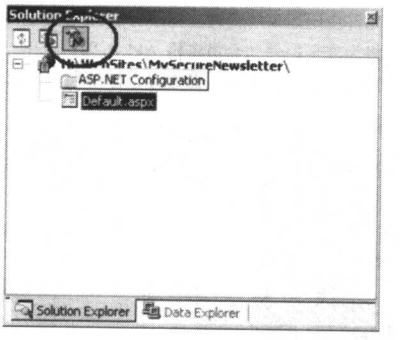
1. 收集需保护的资源和活动，定义应用角色和权限；
2. 设计关系数据库表，储存用户、角色和相关权限；
3. 设计登录页；
4. 撰写用户认证、相关角色和权限等代码；
5. 添加基于角色的站点资源保护配置；
6. 撰写基于角色、权限的页面内容控制代码。

即便你拥有一些封装这些重复任务的可重用组件，仍然会感到肩负重任。上述五个步骤仍然得一一实现，但 ASP.NET Whidbey 提供了新的组件来解决问题。利用基于 Web 的管理向导，可以自动生成一系列表，用来处理用户、角色和权限。可以通过同样的管理接口来配置应用的登录、用户资格和角色管理灯。在一秒钟（取决于你把 Login 控件拖到 Web Form 的速度）之内，无需撰写一行用户认证、角色联系代码，即可设计出登录窗，这一切都是自动的。菜单和页面内容会随用户角色、登录状态的不同而变化，同样也无需写代码来保证这个。

听起来就像美梦成真？来看看怎么做的。

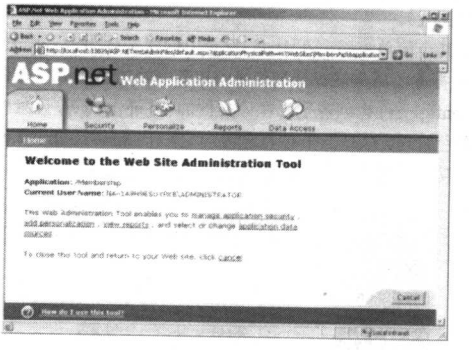
管理接口

ASP.NET Whidbey 包括一个新的基于 Web 的配置工具，运行于特定应用上下文中，所以它能互动地修改应用的 web.config 文件。这个工具里有几个向导，其中一个帮助你实现安全配置。在 Visual Web Developer 中创建新站点之后，可以从 Solution Explorer 中点选 ASP.NET Configuration 图标或是从 Website 菜单选择 ASP.NET Configuration，运行配置工具。



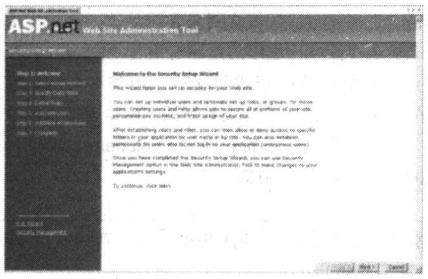
图一　从解决方案资源管理器或主菜单运行基于 Web 的配置工具

在运行配置 Web 界面时，配置的目标应用将被作为参数传入。我创建一个名位 MySecureNewsletter 的新应用，并进入管理界面：



图二　在我机器上管理界面

Security 框有一个选项，运行 Security Setup Wizard。对于新应用，应该先运行这个向导。向导界面左栏显示了将经历的步骤，并指示当前步骤：

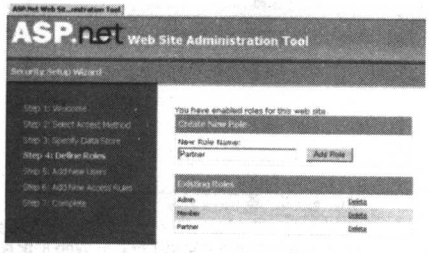


图三　Security Setup Wizard 显示了 ASP.NET Whidbey 其它一些酷特性，如导航向导

这些步骤相当直观，所以我只简单提一下我自己的做法。首先，向导询问站点是 intranet 还是 Internet 站点。前者默认值为 Windows 验证，而后者（我选择的那种）则会配置 forms 验证。下一步，选择是否创建保存用户、角色的数据库。如果是，则需要在 Microsoft Access 和 Microsoft SQL Server 数据库间做出选择，工具会带你进行后续步骤。我没有选择这一项，所以在应用的\Data 目录下将会创建缺省的 Access 数据库。在 machine.config 文件中，添加了<connnectionString>元素，指明该 Access 数据库的缺省位置：

```
<connectionStrings>
        <add name="LocalSqlServer" connectionString="data
          source=127.0.0.1;Integrated Security=SSPI" />
        <add name="AccessFileName"
          connectionString="~\DATA\ASPNetDB.mdb" />
</connectionStrings>
```

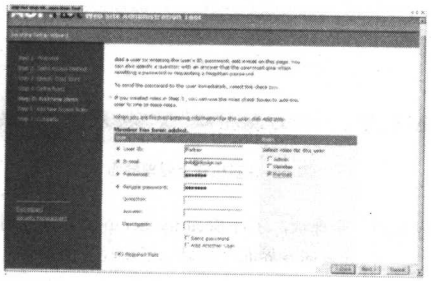
下一步，允许应用的角色管理，在缺省 Access 数据库中一系列表被创建，这些表将为安全控件和会员资格 API 所用。这里，可以选择性地创建角色和用户。对于每个角色，需要设定简单的字符串值。



图四　向导的左栏指出目前所处步骤

在创建用户时，系统假定你需要收集某些用户信息，特别是用户名、密码、E-mail、描述和密码问题等等。如果设定了角色，则角色选择也会出现（如图五）。

经过几个简单步骤，在\Data 目录中创建了名为 AspNetDB.mdb 的 Access 数据库，用来储存会员信息。刷新解决方案资源管理器，可以看到 web.config 文件也创建了，在该文件的<roleManager>元素中，设定是否允许角色管理。这是一个必要选项，因为 machine.config 文件缺省地禁止了这一功能。新的 web.config 文件大致如下所示：



图五　E-mail 字段值将用于密码找回等等互动过程

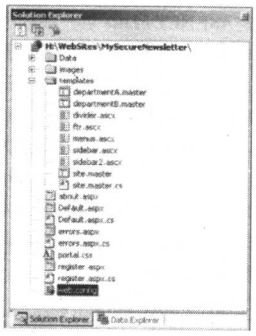
```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
    <system.web>
        <roleManager enabled="true">
            <providers />
        </roleManager>
    </system.web>
</configuration>
```

在上述简单步骤中，我创建了标准的会员管理数据库，角色、用户也被创建，完成了必要的 Web 配置，万事具备，只待搭建一个安全应用了。

如果在上述配置过程中没有创建角色，你还可以在任何时候调用配置工具，进行设置修改，不过我也会展示如何创建自己的管理页面。

拖放安全

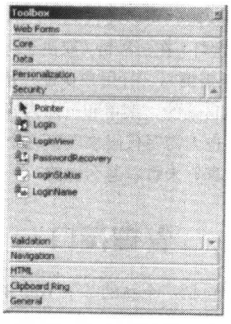
在本小节结束之时，你还没有输入任何代码。下面我要展示如何利用 ASP.NET Whidbey 安全控件来搭建一个基于角色的认证系统。我将会从一个范例新闻邮件列表应用 MySecureNewsletter 开始阐述。除了第一步中所述操作外，目前并无任何安全模块可用。另外，该应用还使用了 Whidbey 的 Master Pages 特性（请参见本期相关文章）。在解决方案资源管理器中，可以看到/templates 目录，该目录包括了可重用控件和 master page 模板；还有/images 目录，该目录保存了所有用到的图片；当然还可以看到根目录下的应用页面。



图六　范例应用程序中的所有*.aspx 页面文件都用了 master page 做模板。Content page 与其它 Web Forms 页面没有区别，但它们指定了 master page 模板，所有内容都放到 content 控件中。

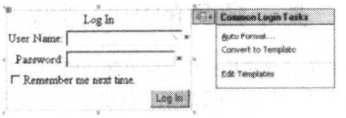
登录

添加一个 content page，命名为 login.aspx，正式开始我们的乐趣之旅。在 Whidbey 工具箱中增加了 Security 页，用来放置新的安全相关控件。



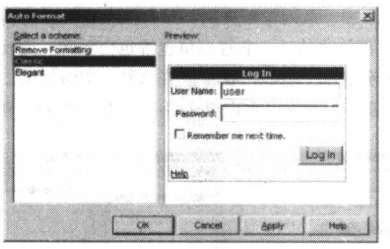
图七　工具箱中新增的安全控件专页

向 login.aspx 页面拖放一个 Login 控件，从这个控件开始，我们一个一个浏览所有控件：



图八　Login 控件提供了交互设计界面，用来编辑控件的缺省布局

通过选择 AutoFormat 选项，可以为控件指定外观：



图九　现阶段有两种可选格式，但如果应用指定了缺省样式表，则可以在属性窗口中为控件的每个元素指定样式。

属性窗口同样提供了对控件 label, value, 验证错误信息等元素的访问功能。你可以自定义 login 按钮的外观，添加一个“新用户注册”链接等等。我决定采用喜欢的样式表和控件缺省状态。控件创建的源代码看起来像这个样子：

```
<asp:login id="Login1" runat="server">
</asp:login>
```

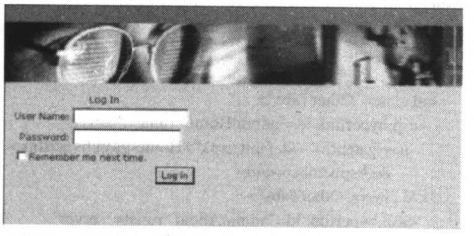
就这样，登录页面已告完成。由于之前已经配置了用户和角色，所以现在只需要修改 web.config 文件，把 authentication mode 设置为 Forms（缺省值为 Windows）。还有，为了测试登录过程，我禁止了匿名用户登录：

```
<authentication mode="Forms"/>

<authorization>
        <deny users="?" />
</authorization>
```

注意　上文中我提到过，Security Setup Wizard 按照 Forms 验证配置应用，具体做法是创建一个储存认证信息的数据库。Wizard 没有修改 web.config 文件中的 authentication 方式（至少在 Whidbey Alphi 中如此），所以需要我们自己做。

现在匿名用户将被导往登录页面。托 Master Pages 和样式表的福，虽然只是简单放置 login 控件，页面看起来也还过得去：



图十　无需修改任何属性，这就是 login 控件的缺省外观，再加上我的样式表，在我的页面模板中显示的样子

什么是登录？

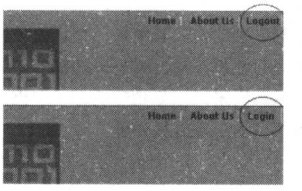
输入正确的用户名和密码，Log In 按钮就会自动验证用户，并重定向导原始请求页面。当然，在登录成功后，也可以显示个性化欢迎信息，并提供注销登录途径。我将在 menus.ascx 用户控件中实现这两个特性，并把该控件显示道每个页面。LoginName 控件显示用户名字，LoginStatus 控件提供方便的链接，根据当前身份验证状态让用户可以登录或注销。下面是两个新控件的 HTML 代码：

成功登录为：

```
<asp:loginname id="LoginName1"
  runat="server"></asp:loginname>

<asp:loginstatus id="LoginStatus1" runat="server">
</asp:loginstatus>
```

看，现在我们做好了一个登录页面，在用户登录成功后能显示个性化欢迎信息，还提供注销登录的途径，而且根本没写代码。



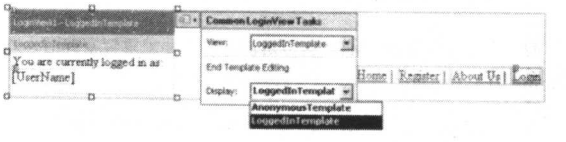
图十一　在登录/注销间转换

LoginName 控件仅用来显示登录成功用户的名字。LoginStatus 控件自动在登录/注销之间转换，但可以利用模板做更多自定义动作。该控件也封装了注销过程。

唯一的问题是在用户登录之前，欢迎信息中用户名是空白的，这也许需要写一些代码来解决。再一次地，我们不用代码来实现。

控制内容访问

在以前版本的 ASP.NET 中，要显示登录成功的用户名，得用代码访问当前上下文的用户身份认证（context's user identity）。在用户通过验证后，需要编写更多代码来检查用户状态。在 ASP.NET Whidbey 中，LoginView 控件也许是最有意思的安全控件之一，因为它让我们能够根据用户验证状态和权限控制页面内容。为了尝试一下，我将拖放一个 LoginView 控件到 menu.ascx 文件。你可以与设计环境互动，编辑向匿名用户（未登录用户）和已验证用户显示的不同信息。



图十二　Common LoginView Tasks 允许你在视图之间转换，或点击 Edit Templates 链接编辑模板

你可以随时从 Source 视图修改 HTML 代码。下列代码显示了在添加 LoginView 控件后对 menus.ascx 文件中个性化欢迎信息的修改

```
<asp:loginview id="LoginView1" runat="server">
    <anonymoustemplate>
        尚未登录。
    </anonymoustemplate>
    <loggedintemplate>
    成功登录为：<asp:loginname id="LoginName1"
      runat="server">
                </asp:loginname>
    </loggedintemplate>
</asp:loginview>
```

在用户未登录时，LoginView 控件显示<anonymoustemplate>段中的内容，用户登录后显示<loggedintemplate>段中的内容。就这样，不用写一行代码，我们已经利用安全控件实现了登录页面、注销途径、个性化欢迎信息和根据登录状态显示不同页面内容的功能。