DEV353
.NET Framework 更高效开发

李琪

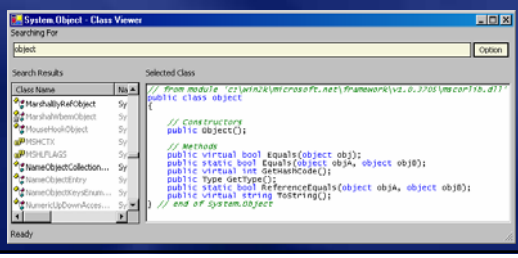
开发合作经理
平台及开发合作部
微软(中国)有限公司

Why This Talk?

- 大部分的文档和资源只关注技术
- 本课程关注如何更好的应用 .NET
  - 经验、技巧、工具、应用、设计指导
  - 目的在于增强你的开发效率
- 除特殊声明外 Microsoft Visual Basic 和 Visual C# 都适用

WinCV

- 类似头文件的类型信息
- C:\Program Files\Microsoft Visual Studio .NET 2003\SDK\v1.1\Bin\WinCV.exe

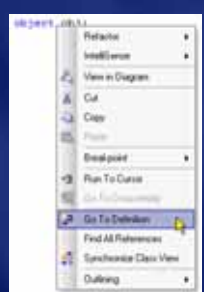


WinCV

- .NET 深藏的秘密
- 智能感知
  - 基本数据类型
  - 其他方法
- 快速参考
  - 节省在msdn上搜索的时间
  - 可以自定义的装配列表
  - WinCV.exe.config
  - 为大型项目节省时间

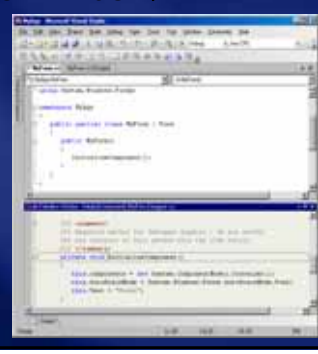
转到定义: Visual Studio 2005

- Visual Studio 2005内建WinCV式的类型信息
  - 公共成员
  - 包括注释
  - 包括属性



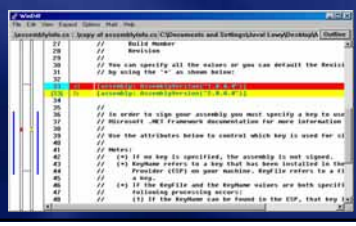
Visual Studio 2005代码定义窗口（Code Definition Window）

- 常量定义视图
- 保持当前的文档
- 只读



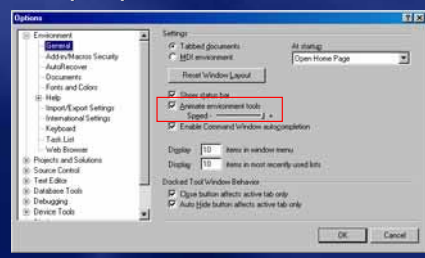
WinDiff

- 比较和分析文件的不同点
  - 单个文件或整个目录
  - 智能的
- 比 Microsoft Visual SourceSafe比较器更好



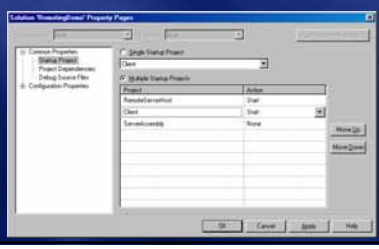
隐藏速度

- 自定义自动隐藏速度
  - 工具|选项|常规



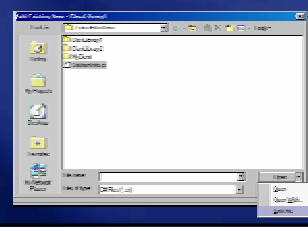
多工程启动

- 在一个解决方案中同时启动多个项目
  - 同一个 debug 会话 (Start button， F5)
- Solution|Properties|Startup Project



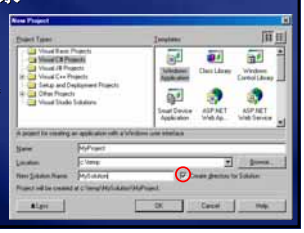
链接文件

- 链接方式开启文件
- 以引用方式编译，而不拷贝文件
- 例如：SolutionInfo.cs, 共享的文件:
  - snk 文件
  - 版本信息
  - 安全策略
  - 版权信息
  - 公司名等



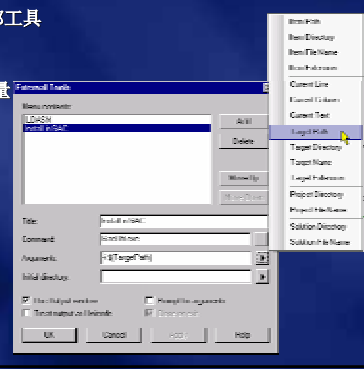
解决方案目录

- 通常一个解决方案包含多个项目
- 解决方案文件不应被包含在任何一个特定的项目中
- 更多|创建解决方案的目录
- 其他方法
  - 文件|新建|空白解决方案



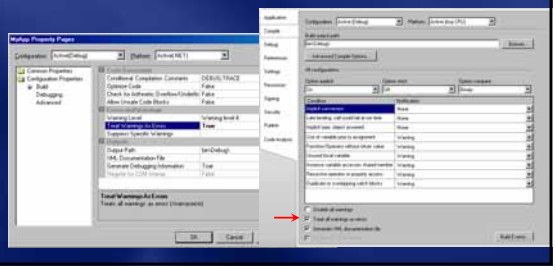
外部工具集成

- 自动化引用外部工具
  - ILDasm
  - GacUtil
- 传递VS的宏变量



将警告视为错误

- 项目属性|配置属性|生成
- 把警告等级设置成4
- 发布之前必经步骤
  - 对debug也非常有用



可拖拽的对象

- Windows Form, Web Form, Web Services 都可以接收多拽对象
  - SQL 表格/数据源
    - 生成adaptor, connection, command, dataset, binding 等对象
  - 组件
    - 事件日至，目录监视器，计时器，等等

可拖拽的对象

- 通过组件，为自定义类添加拖拽支持
  - System.ComponentModel
  - 用完后可去除继承
  - Visual Studio 显示不同的图标

```
Imports System.ComponentModel

Public Class MyDataAccess
             Inherits Component
End Class
```

矩形选择

- 选取前，先按住 ALT 键
  - 去除命名空间和重复的定义时特别有用

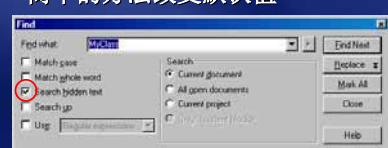
```
namespace MyClassLibrary
{
   public class MyClass : Component
   {
      private System.Data.SqlClient.SqlCommand sqlSelectCommand1;
      private System.Data.SqlClient.SqlCommand sqlInsertCommand1;
      private System.Data.SqlClient.SqlCommand sqlUpdateCommand1;
      private System.Data.SqlClient.SqlCommand sqlDeleteCommand1;
      private System.Data.SqlClient.SqlConnection sqlConnection1;
      private System.Data.SqlClient.SqlDataAdapter sqlDataAdapter1;

      private void InitializeComponent()
      {
```

文档大纲

- 按层次浏览对象
  - 支持 Microsoft ASP.NET forms, HTML, XML
- 对复杂的文件很有用
- 视图|其他窗口 |文档大纲

搜索隐藏文本

- 缺省情况下，搜索和查找替换并不针对收起的文本
- 简单的方法改变默认值



- Visual Studio 2005搜索和查找替换默认包括收起的文本

条件编译

- 编译时不执行的方法调用
  - 没有定义条件
  - System.Diagnostics

```
#Const MySpecialCondition = True 'usually DEBUG

Class SomeClass
   <Conditional("MySpecialCondition")> _
   public Sub SomeMethod()
   End Sub
End Class

'Client side code
Dim obj as SomeClass
obj = new SomeClass()
'This line is conditional
obj.SomeMethod()
```

事件访问器

- 使用添加/删除accessors, 而不是直接访问成员事件
  - 类似属性
  - 提倡封装和松耦合
- Visual C# 2002 - 2005
- 仅Visual Basic 2005

事件访问器

```
public class MyPublisher
{
   EventHandler m_MyEvent;
   public event EventHandler MyEvent
   {
      add
      {
         m_MyEvent += value;
      }
      remove
      {
         m_MyEvent -= value;
      }
   }
}
```

事件访问器

```
Public Class MyPublisher
    Event m_MyEvent As EventHandler
    Public Custom Event MyEvent As EventHandler
        AddHandler(ByVal value As EventHandler)
            AddHandler m_MyEvent,value
        End AddHandler

        RemoveHandler(ByVal value As EventHandler)
            RemoveHandler m_MyEvent,value
        End RemoveHandler

        RaiseEvent(ByVal sender As Object, ByVal ea As EventArgs)
            RaiseEvent m_MyEvent(sender,ea)
        End RaiseEvent
    End Event
End Class
```

事件访问器

- 使用 “正常的” 事件追加语法

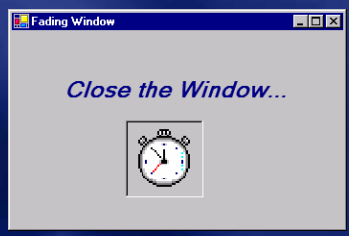
```
MyPublisher  publisher  = new MyPublisher();
publisher.MyEvent += new EventHandler(OnMyEvent);

void OnMyEvent(object sender,Eventargs args)
{...}

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Dim publisher As New MyPublisher()
AddHandler publisher.MyEvent, AddressOf OnMyEvent

Sub OnMyEvent(ByVal sender As Object, ByVal args As Eventargs)
...
End Sub
```

Windows 窗体透明

- 所有的可见控件都有透明属性
  - 0-100%
- 可视化效果

Windows 窗体透明

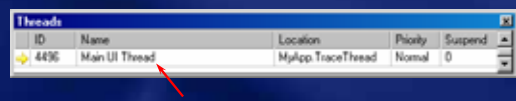
- 渐暗窗体
  - 组合 计时器 和Close 事件



Thread Name

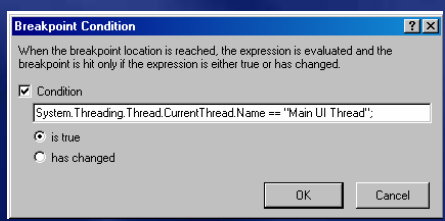
```
Imports System.Threading

Dim currentThread As Thread = Thread.CurrentThread

Dim threadName As String = "Main UI Thread"
currentThread.Name = threadName
```

- 线程命名不是必须，但是对于debug非常有用



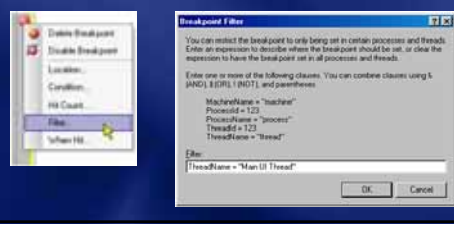
线程名断点 (.NET 1.1)

- 用线程名作为中断条件
  - 设置条件中断
  - 使用完全限定类型



线程名中断 (.NET 2.0)

- 设置断点过滤
- 默认禁用
  - 工具|选项|调试|常规|起用断点过滤



结束进程

- 不要调用Abort()
  - 线程需要被清除
  - Abort() 不能彻底清除
- 线程方法需要检查标志
  - 保护性的互斥标志
- Kill() 方法需要设置标志，并等待进程终结
- Abort() 还有一个缺陷
  - 线程可能在catch{}中进行未定义的处理

```
public class WorkerThread : IDisposable
{
   protected Thread m_ThreadObj;
   protected bool m_EndLoop;
   protected Mutex m_EndLoopMutex;
   protected bool EndLoop
   {
      set
      {
         m_EndLoopMutex.WaitOne();
         m_EndLoop = value;
         m_EndLoopMutex.ReleaseMutex();
      }
      get
      {
         bool result = false;
         m_EndLoopMutex.WaitOne();
         result = m_EndLoop;
         m_EndLoopMutex.ReleaseMutex();
         return result;
      }
   }
   public WorkerThread()
   {
      m_EndLoop = false;
      m_ThreadObj = null;
      m_EndLoopMutex = new Mutex();
   }
```

```
public class WorkerThread : IDisposable
{
   public void Start()
   {
      m_ThreadObj = Thread.CurrentThread;
      int i = 0;
      while(EndLoop == false)
      {
        //do work here
      }
   }

   //Kill is called on client thread - must use cached thread object
   public void Kill()
   {
      Debug.Assert(m_ThreadObj != null);
      if(m_ThreadObj.IsAlive == false)
         return;

      EndLoop = true;
      //Wait for thread to die
      m_ThreadObj.Join();

      if(m_EndLoopMutex != null)
         m_EndLoopMutex.Close();
   }
   //Rest of WorkerThread
```

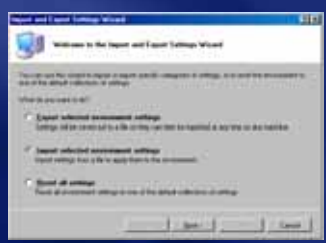
编码标准

- 务必要使用广泛接受的编码标准
  - 命名约定和风格
  - 编码实践
  - 工程设置和结构
  - Framework 特定的指导建议
- 少问为什么
  - 减少害处
- IDesign 编码标准
  - www.idesign.net

标准

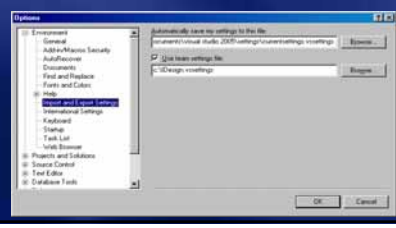
导入/导出设置

- 工具|导入/导出设置



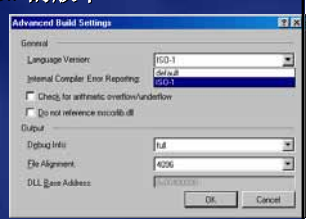
导入/导出设置

- 支持团队设置文件
- 模糊导入
  - 自动应用新版本的设置



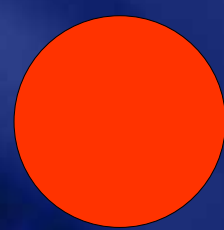
语言版本

- 可以限定 Visual C# 的版本
  - 生成|高级
- 默认
  - Visual C# 2.0
- ISO-1
  - Visual C# 1.0 (Visual Studio 2002)
  - 多平台



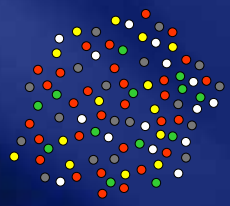
接口分解与设计

- 这是好的设计么？



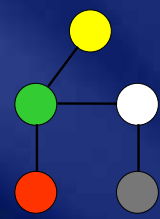
接口分解与设计

- 这是好的设计么？



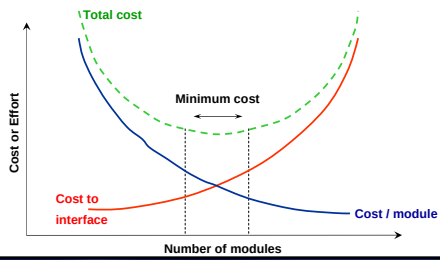
接口分解与设计

- 这是好的设计么？



接口分解与设计

- 平衡模块大小与集成度



接口分解与设计

- 从可重用的角度考虑接口分解问题
- 例如：“狗”接口
- 需求:
  - 叫
  - 捡
  - 户口号
  - 打过预防针

接口分解与设计

- 可以这样定义 IDog

```
public interface IDog
{
   void  Fetch();
   void  Bark();
   long  VetClinicNumber{ get; set; }
   bool  HasShots{ get; set; }
}
public class Poodle : IDog
{…}
public class GermanShepherd : IDog
{…}
```

这种设计并没有很好的分解接口

- Bark() 和 Fetch() 的关联性更强

接口分解与设计

- 更好的分解方式

```
public interface IPet
{
   long  VetClinicNumber{ get; set; }
   bool  HasShots{ get; set; }
}
public interface IDog
{
   void  Fetch();
   void  Bark();
}
public interface ICat
{
   void  Purr();
   void  CatchMouse();
}
public class Poodle : IDog,IPet
{…}
public class Siamese  : ICat, IPet
{…}
```

接口分解与设计

- 层次划设计逻辑上关联且重复的方法

```
public interface IMammal
{
   void ShedFur();
   void Lactate();
}
public interface IDog : IMammal
{
   void  Fetch();
   void  Bark();
}
public interface ICat : IMammal
{
   void  Purr();
   void  CatchMouse();
}
```

Interface Factoring Metrics

- 接口分解是其包含的方法更少
- 平衡对立面
  - 过于细腻的接口 vs. 过于粗糙的分解

Interface Factoring Metrics

- 可能出现只有一个成员的接口，但是应该尽量避免
  - Dull facet
  - 过多的参数
  - 过于粗糙: 应被分解为多个接口
  - 分解到多个已有接口
- 理想的成员数量
- 最多不要超过20(12)

Interface Factoring Metrics

- 方法、参数和属性的比例
  - 接口中的方法应比属性多
  - 搞好完全封装
  - 比值至少 2:1
  - 异常是只有属性的接口
    - 不应有方法
  - 避免定义时间

.NET Factoring Metrics

- 对300+接口设计统计
- 平均, 3.75 方法/接口
- 方法/属性比例 3.5:1
- 事件成员的比例小于3%
- On average, .NET interfaces are well factored

Resources

- *中文MSDN*
  - 文档库
  - 开发中心
  - Webcast
  - Express Center
- ISV Center
  - http://www.microsoft.com/china/msdn/ISV