

实验项目列表

序号	实验项目名称	成绩	指导教师
1	ACME 软件体系结构描述语言应用		
2	SOA 实践		
3	MDA 实践		
4	MVC 实践		
5	产品族实践		
6	软件体系结构风格实践		
平均成绩			

实验 1：ACME 软件体系结构描述语言应用

一、实验目的

- 1)掌握软件体系结构描述的概念
- 2)掌握应用 ACMESTUDIO 工具描述软件体系结构的基本操作

二、实验学时

2 学时。

三、实验方法

由老师提供软件体系结构图形样板供学生参考，学生在样板的指导下修改图形，在老师的指导下进行软件体系结构描述。

四、实验环境

计算机及 ACMESTUDIO。

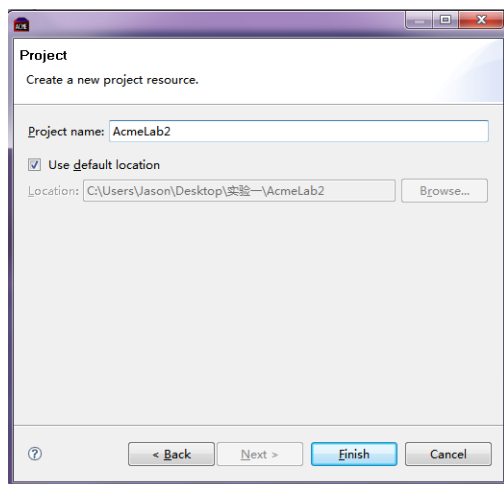
五、实验内容

利用 ACME 语言定义软件体系结构风格，修改 ACME 代码，并进行风格测试。

六、实验操作步骤

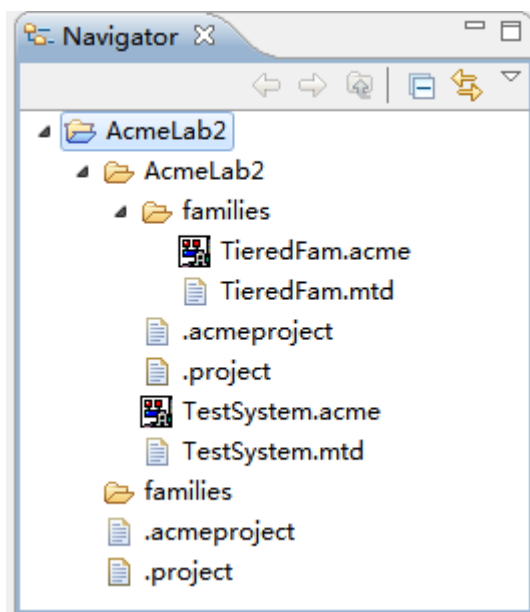
一、导入Zip文档

建立的一个Acme Project, 并且命名为AcmeLab2。如下图：



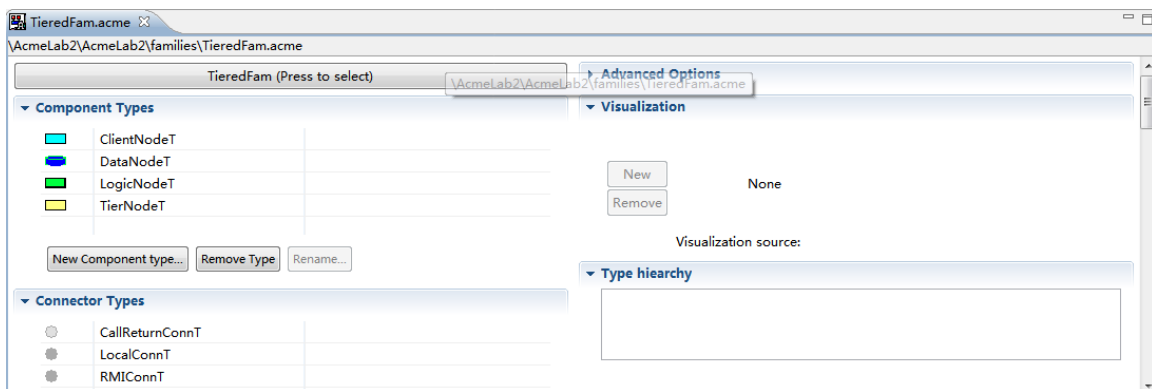
的如下图：

接着导入ZIP文档，导入完ZIP文档后显示



二、修改风格

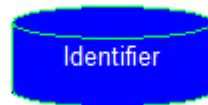
在AcmeLab2项目中,打开families下的TieredFam.acme.如下图:



修改组件外观

1. 在组件类型中, 双击DataNodeT; 在其右边的编辑器中, 将产生预览; 选择 Modify按钮, 将打开外观编辑器对话框。
 2. 首先改变图形: 找到Basic shape section, 在Stock image dropdown menu中选择 Repository类型.
 3. 在Color/Line Properties section修改填充颜色为深蓝色。
 4. 在颜色对话框中选择深蓝色, 并单击 [OK].
 5. 修改图形的边框颜色为绿色
 7. 单击Label tab, 在Font Settings section, 设置字体颜色为白色,单击[OK]
- 产生的图形如下图:

Preview

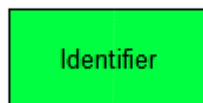


添加新元素类型

1. 在Component Types section选择New按钮
2. 在对话框中, 类型名称输入LogicNodeT .
3. 选择TierNodeT 为父类型.
4. 单击 [Finish].
5. 按照修改外观的步骤, 修改LogicNodeT的外观: 填充颜色为浅绿色, 边框颜色为黑色, 大小为2, 其他默认。

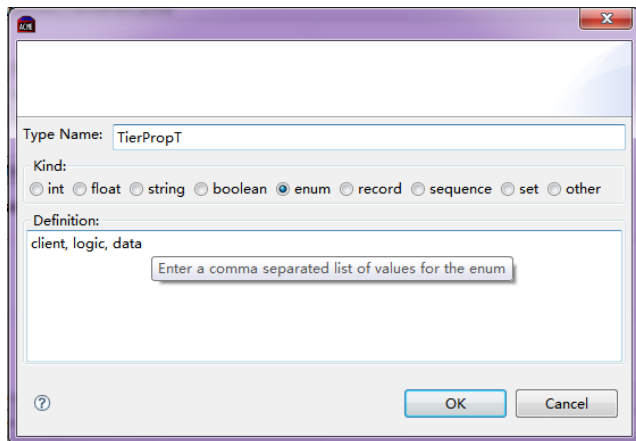
产生的图形如下图:

Preview



添加新属性类型

1. 选择Property Types
2. 选择New按钮
3. 在对话框中, 类型名称为TierPropT
4. 类型选择enum
5. 值为: client, logic, data
6. 单击[OK].



添加属性

1. 激活属性视图页
2. 双击TierNodeT
3. 选择Properties tab
4. 右击空白位置，或者单击图标，选择新建属性
5. 属性名为tier.
6. 类型为TierPropT(找不到，则直接输入TieredFam.TierPropT)
- 7.单击 [OK].

添加规则

1. 单击Family editor中的TieredFam (Press to select).
2. 选择属性视图中的规则页
3. 单击  生成新规则
4. 规则名为hostCheck, 选择invariant单选项
5. 在规则框中输入(直接粘贴过去)

```
Forall t1 : TierNodeT in self.Components |
    !t1.allowShareHost -> (Forall t2 : TierNodeT in self.Components |
        t1 != t2 -> t1.host != t2.host)
```

6. 单击 [Parse Rule] 以确认无语法错误，有错误，要重新写。
7. 在标签中输入 “Tier nodes respect host assignment.”
8. 在出错标签中输入 “Two nodes that cannot share a host must not reside on the same host.”
9. 单击[OK] 。若前面有语法错误，按钮是灰色的
10. 保存.

Acme代码编辑

1. 查看编辑器底端的区域, 有3个区: Overview, Acme Source and Family - TieredFam. 选择Source
2. 在代码中找到TierNodeT
3. 复制其中的属性内容
4. 定位至ClientNodeT, 粘贴刚才复制的内容

```
Component Type TierNodeT = {  
  Property host : string;  
  Property allowShareHost : boolean << default : boolean = true; >> ;  
  Property tier : TieredFam.TierPropT;  
}
```

5. 把tier属性的值赋值为client.
Component Type ClientNodeT extends TierNodeT with {
 Property tier : TierPropT = client;
}
6. 同样把复制的内容粘贴在LogicNodeT和 DataNodeT中, 前者的tier值为logic, 后者的tier值为data.

```
Component Type LogicNodeT extends TierNodeT with {  
  Property tier : TieredFam.TierPropT = logic;  
}
```

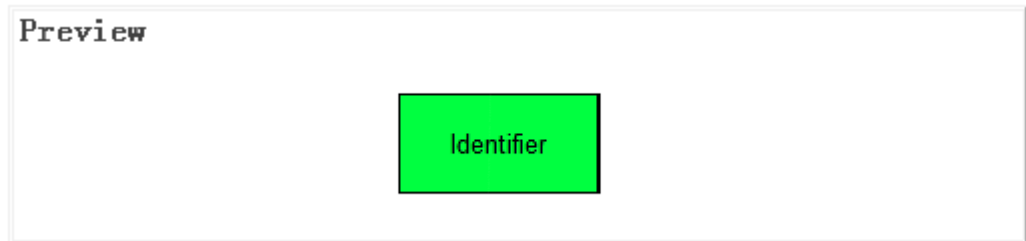
```
Component Type DataNodeT extends TierNodeT with {  
  Property tier : TieredFam.TierPropT = data;  
}
```

7. 单击TieredFam editor tab, 若有错, 将指出错误

添加可视变量

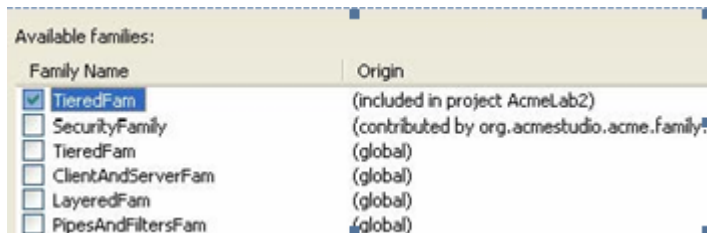
1. 编辑LogicNodeT 组件的外观
2. 选择Variants tab.

3. 单击 [New...] 创建新的变量
4. 名为:Not logic tier.
5. 选择基于条件的单选按钮
6. 下拉菜单选择allowSharedHost, 条件选择==, 值输入false
7. 设置填充颜色为深绿色
8. 单击 [OK]



测试风格

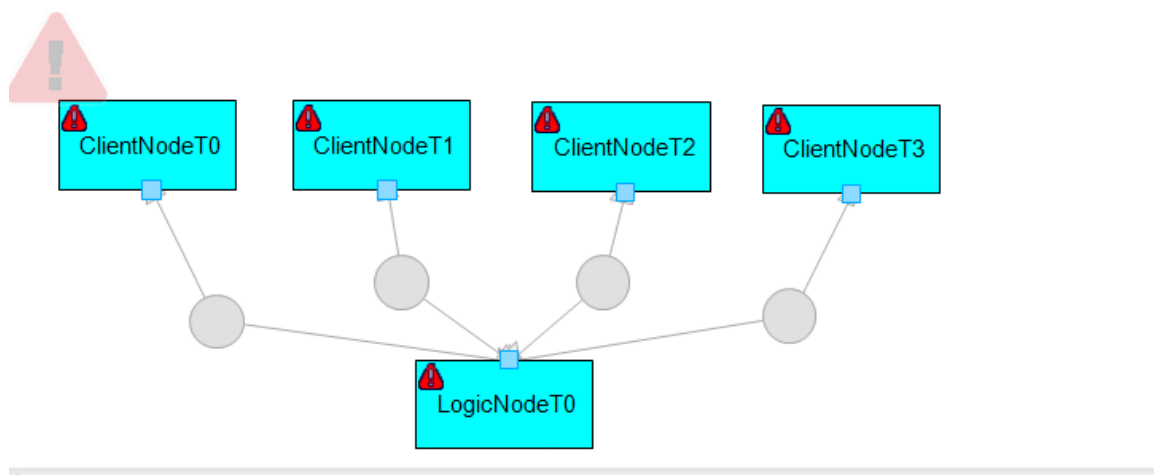
- 1.在Navigator的AcmeLab2中,单击右键, 在弹出式菜单中选择New Acme System.
2. 系统名为TestSystem.
4. 单击[Next].
5. 在风格类型中, 选择的内容如下图所示。



- 6.单击[Finish]

Notice the palette of types on the left from which you can drag and drop an element to create an instance.

- 7.在右边的palette面板中, 拖入 4个 client nodes和1个logic node , 并具有4个ports , 同时拖入4个connectors , 实现客户端和逻辑层的连接。



七、总结

通过这次实验, 我已经初步掌握了 Acme□Studio 这个软件的使用, 也初步对于三层体系结构风格的理解有了一定的认识, 让我们初步对于软件体系结构的构造有了一定的了解, 也使得我们在软件体系构建的时候有很多好的软件可以进行应用。对我们进行软件体系结构的做法有了一定的参考。以上就是我对这次实验的总结。

实验：SOA 实践

一、实验目的

- 1) 初步了解 SOA 的体系结构
- 2) 掌握用 Web service 技术实现 SOA

二、实验学时

2 学时。

三、实验方法

根据实验指导书, 实现 web service。

四、实验环境

计算机及 VS2005。

五、实验内容

Web service 实现。

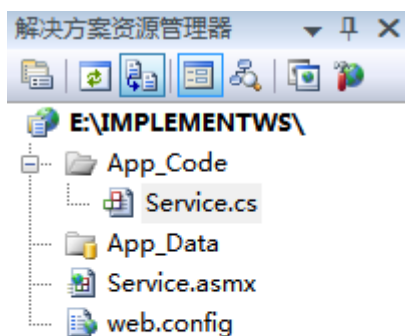
六、实验操作步骤

内容一：Web Services实现

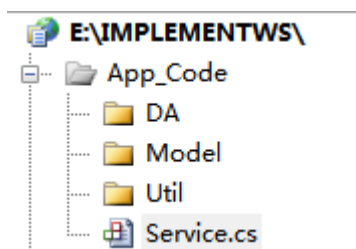
步骤：

1 创建WEB服务

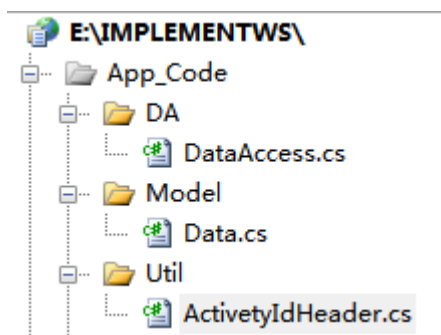
- 1) 打开VS.NET2005, 新建一个网站, 在对话框中选择“ASP.NET WEB服务”, 选择好位置, 把Website1改为“IMPLEMENTWS”, 单击确定。



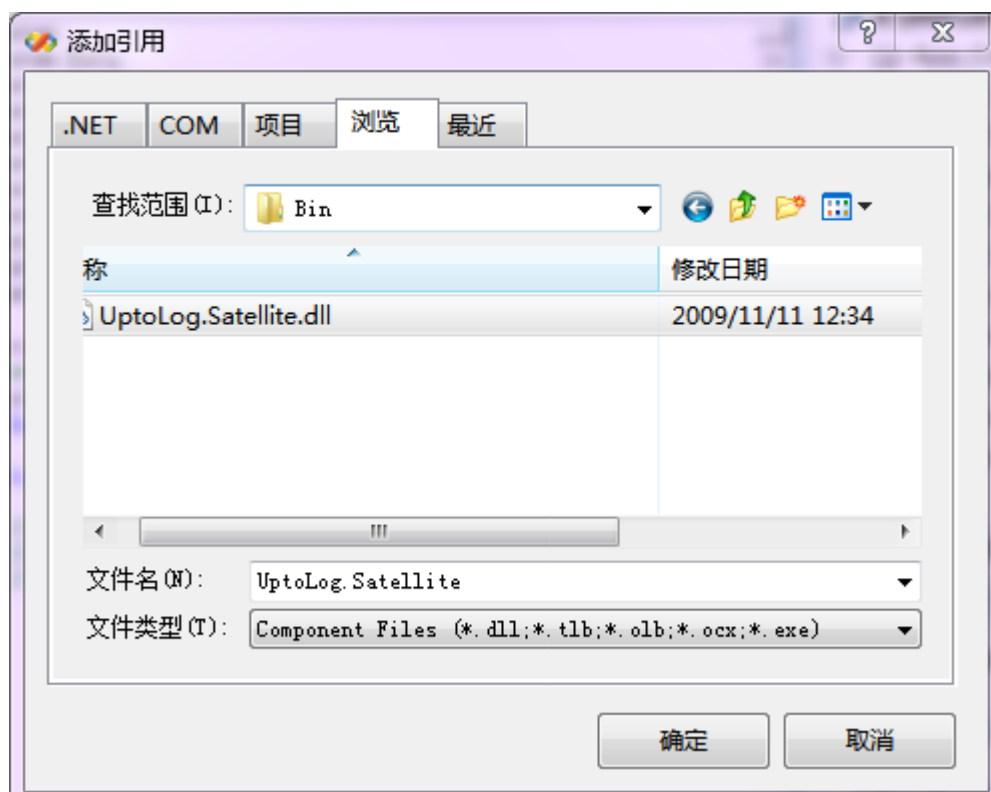
2) 展开解决方案资源管理器，在“APP_CODE”下创建3个文件夹，依次命名为：DA，Model，Util，见下图：



3) 右击DA文件夹，选择添加新项，新增一个类：DataAccess.cs，按同样的方法，在MODEL和UTIL文件夹中，添加入下图所示的类。



4) 右击项目，选择添加引用（注意不是Web引用），选择浏览页，定位拷贝的DLL，见下图



5) 完成DATA.CS的代码

```

1 using System;
2 using System.Data;
3 using System.Configuration;
4 using System.Linq;
5 using System.Web;
6 using System.Web.Security;
7 using System.Web.UI;
8 using System.Web.UI.HtmlControls;
9 using System.Web.UI.WebControls;
10 using System.Web.UI.WebControls.WebParts;
11 using System.Xml.Linq;
12
13 /// <summary>
14 /// Data 的摘要说明
15 /// </summary>
16 public class Data
17 {
18     public Data()
19     {
20         //
21         //TODO: 在此处添加构造函数逻辑
22         //
23     }
24     private string someData;
25
26     public string SomeData
27     {
28         get { return someData; }
29         set { someData = value; }
30     }
31 }
32
33

```

6) 完成DataAccess.CS的代码

```

1 using System;
2 using System.Data;
3 using System.Configuration;
4 using System.Linq;
5 using System.Web;
6 using System.Web.Security;
7 using System.Web.UI;
8 using System.Web.UI.HtmlControls;
9 using System.Web.UI.WebControls;
10 using System.Web.UI.WebControls.WebParts;
11 using System.Xml.Linq;
12
13 using System.Threading;
14 using UptoLog.Satellite;
15
16 /// <summary>
17 /// DataAccess 的摘要说明
18 /// </summary>
19 public class DataAccess
20 {
21     public DataAccess()
22     {
23         //
24         //TODO: 在此处添加构造函数逻辑
25         //
26     }
27
28     public Data LoadData()
29     {
30         using (Log.BeginTimeMeasure("Load data from DB"))
31         {
32             Log.LogTraceItem("Load data from DB");
33             Data data = new Data();
34
35             //Load data...
36             data.SomeData = "The data...";
37             Thread.Sleep(328);
38
39             Log.LogTraceItem("Data loaded from DB");
40             return data;
41         }
42     }
43 }

```

7) 完成ActivityIdHeader.cs

```

1 using System;
2 using System.Data;
3 using System.Configuration;
4 using System.Linq;
5 using System.Web;
6 using System.Web.Security;
7 using System.Web.UI;
8 using System.Web.UI.HtmlControls;
9 using System.Web.UI.WebControls;
10 using System.Web.UI.WebControls.WebParts;
11 using System.Xml.Linq;
12
13 using UptoLog.Satellite;
14 using System.Web.Services.Protocols;
15
16 /// <summary>
17 /// ActivityIdHeader 的摘要说明
18 /// </summary>
19
20 public class ActivityIdHeader : SoapHeader
21 {
22     public string ActivityId
23     {
24         get
25         {
26             return Log.CurrentActivityId.ToString();
27         }
28         set
29         {
30             Log.CurrentActivityId = new Guid(value);
31         }
32     }
33 }
34
35

```

8) 双击Service.CS文件，完成代码

```

1 using System;
2 using System.Linq;
3 using System.Web;
4 using System.Web.Services;
5 using System.Web.Services.Protocols;
6 using System.Xml.Linq;
7
8 using UptoLog.Satellite;
9
10 [WebService(Namespace = "http://tempuri.org/")]
11 [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
12 // 若要允许使用 ASP.NET AJAX 从脚本中调用此 Web 服务，请取消对下行的注释。
13 // [System.Web.Script.Services.ScriptService]
14 public class Service : System.Web.Services.WebService
15 {
16
17     private ActivityIdHeader activityIdHeader;
18     public ActivityIdHeader ActivityIdHeader
19     {
20         get { return activityIdHeader; }
21         set { activityIdHeader = value; }
22     }
23     public Service () {
24
25         //如果使用设计的组件，请取消注释以下行
26         //InitializeComponent();
27     }
28
29     [WebMethod]
30     [SoapHeader("ActivityIdHeader", Direction = SoapHeaderDirection.In)]
31     public Data LoadSomeData()
32     {
33         try
34         {
35             DataAccess da = new DataAccess();
36             return da.LoadData();
37         }
38         catch (Exception e)
39         {
40             Log.LogFatalError(e);
41             throw;
42         }
43     }
44 }
45
46

```

9) 编译，并运行，查看效果。

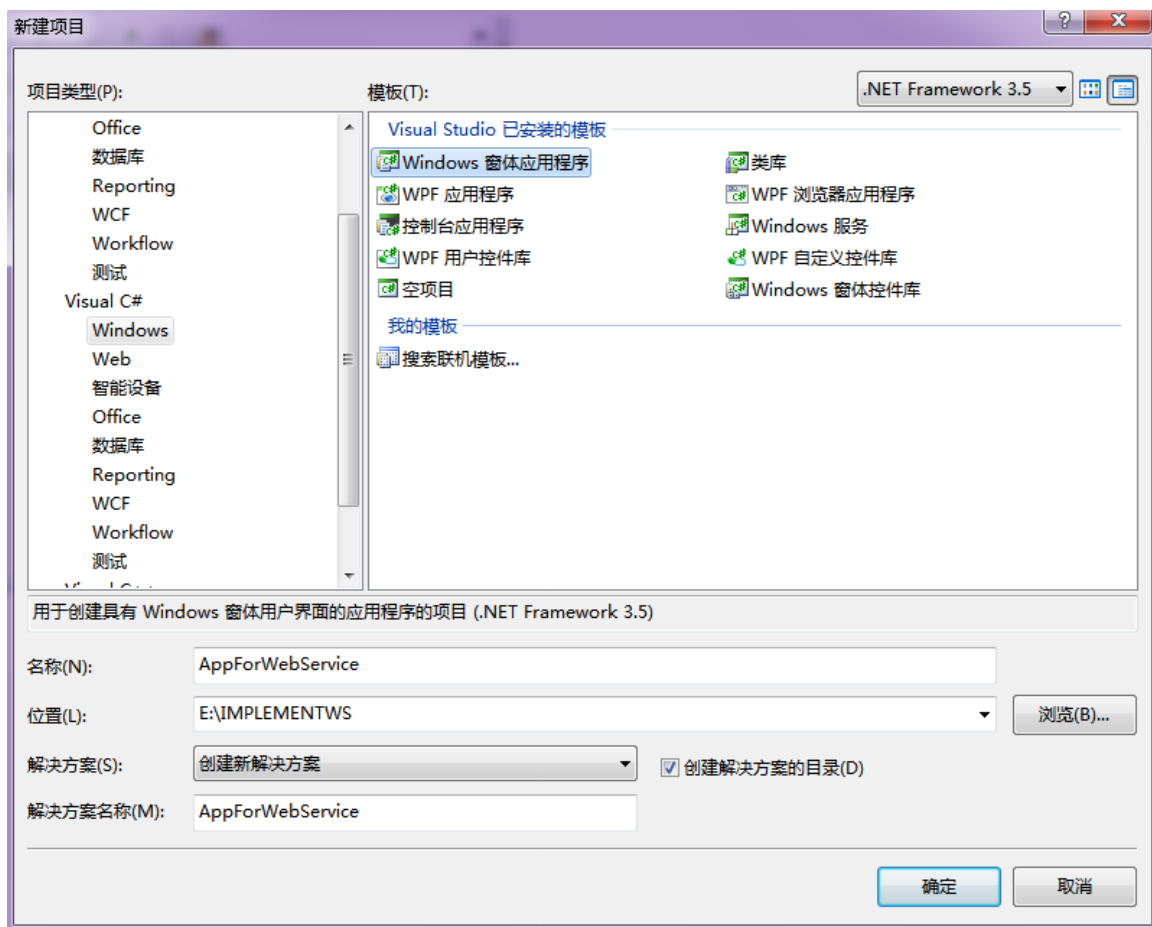


一个WEB服务编写完成。

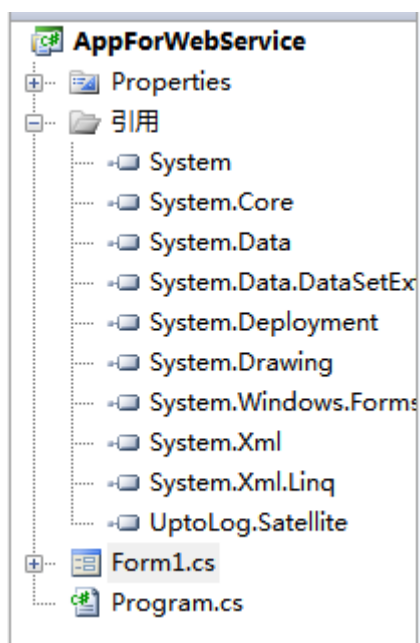
2 创建WINDOWS应用

1) 右击解决方案，选择添加-》新项目

2) 按下图完成：



3) 右击项目，选择添加引用（注意不是Web引用），选择浏览页，定位拷贝的DLL，见下图



4) 右击项目，新建文件夹，并在其下新增一个类，并完成代码，如下图所示



```

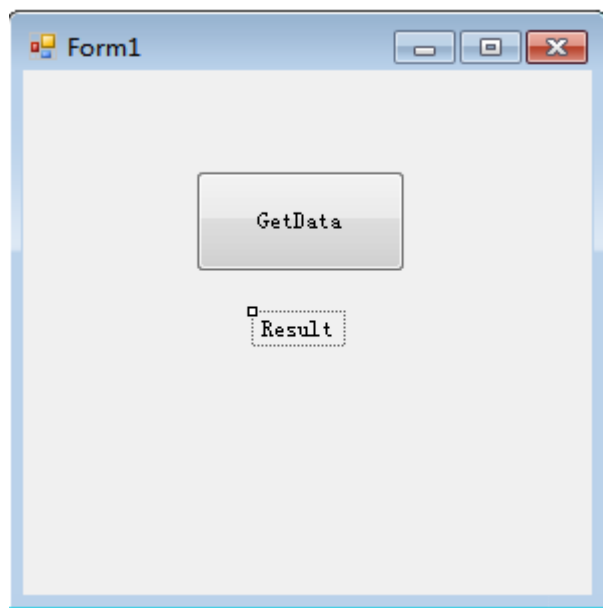
1 using System;
2 using System.Collections.Generic;
3 using System.Text;
4
5 using UptoLog.Satellite;
6 using AppForWebService.MYSERVICE;
7
8
9 namespace AppForWebService.SA
10 {
11     class ServiceAgent
12     {
13         public Data LoadData()
14         {
15             using (Log.BeginTimeMeasure("Load data", "Load data into the client"))
16             {
17                 Log.LogTraceItem("Load data into the client");
18                 Service ss = new Service();
19                 ss.ActivityIdHeaderValue = new ActivityIdHeader();
20                 ss.ActivityIdHeaderValue.ActivityId = Log.CurrentActivityId.ToString();
21                 Data data = ss.LoadSomeData();
22                 Log.LogTraceItem("Data loaded");
23                 return data;
24             }
25         }
26     }
27 }
28

```

5) 添加Web引用，右击项目，选择添加WEB引用，在URL中输入，WEB服务的地址，然后单击前进，可以修改引用名，单击“添加引用”按钮



6) 双击FORM1.CS，在窗体上放置一个按钮和一个标签，其中标签命名为Result，按钮的text为“get data”



7) 双击按钮，完成以下代码

```

1 using System;
2 using System.Collections.Generic;
3 using System.ComponentModel;
4 using System.Data;
5 using System.Drawing;
6 using System.Text;
7 using System.Windows.Forms;
8 using UptoLog.Satellite;
9 using System.Threading;
10 using AppForWebService.MYSERVICE;
11 using AppForWebService.SA;
12
13 namespace AppForWebService
14 {
15     public partial class Form1 : Form
16     {
17         public Form1()
18         {
19             InitializeComponent();
20         }
21         private void Button1_Click(object sender, EventArgs e)
22         {
23             Log.StartNewActivityId();
24
25             using (Log.BeginTimeMeasure("Show data"))
26             {
27                 Log.LogTraceItem("Show data");
28                 ServiceAgent sa = new ServiceAgent();
29                 Data data = sa.LoadData();
30
31                 using (Log.BeginTimeMeasure("Handle data"))
32                 {
33                     Log.LogTraceItem("Handle data");
34                     Result.Text data.SomeData;
35                     Thread.Sleep(126);
36                 }
37             }
38         }
39     }
40 }

```

8) 右击项目，新增一个类：UptoLogEvents.cs

```

using System;
using System.Collections.Generic;
using System.Text;
using UptoLog.Satellite;
using System.Windows.Forms;

namespace AppForWebService
{
    internal class UptoLogEvents
    {
        public static void SetupUptoLogEvents()
        {
            Log.LogException += LogException;

            //Log.PreFatalErrorLog += PreFatalErrorLog;
            //Log.PreErrorLog += PreErrorLog;
            //Log.PreWarningLog += PreWarningLog;
            //Log.PreTraceLog += PreTraceLog;
            //Log.PreTimeMeasureLog += PreTimeMeasureLog;

            //Log.FatalErrorLogged += FatalErrorLogged;
            //Log.ErrorLogged += ErrorLogged;
            //Log.WarningLogged += WarningLogged;
            //Log.TraceLogged += TraceLogged;
            //Log.TimeMeasureLogged += TimeMeasureLogged;

        }

        private static void LogException(object sender, LogExceptionEventArgs e)
        {
            MessageBox.Show(e.ExceptionObject.ToString(), "UptoLog error");
        }
    }
}

```

9) 双击PROGRAM.CS, 完成代码

```

using System;
using System.Collections.Generic;
using System.Windows.Forms;
using UptoLog.Satellite;
using System.Threading;
namespace AppForWebService
{
    static class Program
    {
        /// <summary>
        /// 应用程序的主入口点。
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);

            UptoLogEvents.SetupUptoLogEvents();

            // Add the event handler for handling UI thread exceptions to the event.
            Application.ThreadException += new ThreadExceptionEventHandler(Application_ThreadException);

            // Set the unhandled exception mode to force all Windows Forms errors to go through our handler.
            Application.SetUnhandledExceptionMode(UnhandledExceptionMode.CatchException);

            // Add the event handler for handling non-UI thread exceptions to the event.
            AppDomain.CurrentDomain.UnhandledException +=
                new UnhandledExceptionEventHandler(CurrentDomain_UnhandledException);

            Application.Run(new Form1());
        }
    }
}

```

```

    }

    static void Application_ThreadException(object sender, ThreadExceptionEventArgs e)
    {
        // Logged as a fatal error.
        Log.LogFatalError(e.Exception);
        if (e.Exception != null)
        {
            MessageBox.Show(e.Exception.Message, "Fatal Error (in ThreadException)");
        }
        Application.Exit();
    }

    static void Application_ThreadException(object sender, ThreadExceptionEventArgs e)
    {
        // Logged as a fatal error.
        Log.LogFatalError(e.Exception);
        if (e.Exception != null)
        {
            MessageBox.Show(e.Exception.Message, "Fatal Error (in ThreadException)");
        }
        Application.Exit();
    }

    static void CurrentDomain_UnhandledException(object sender, UnhandledExceptionEventArgs e)
    {
        MessageBox.Show("Hallo!!!, feilen: " + e.ExceptionObject.ToString());

        if (e.IsTerminating)
        {
            // Logged as a fatal error.
            Log.LogFatalError(e.ExceptionObject as Exception);
            if (e.ExceptionObject != null)
            {
                MessageBox.Show(((Exception)e.ExceptionObject).Message, "Fatal Error (in UnhandledException)");
            }
        }
        else
        {
            // Logged as an error.
            Log.LogError(e.ExceptionObject as Exception);
            if (e.ExceptionObject != null)
            {
                MessageBox.Show(((Exception)e.ExceptionObject).Message, "Error (in UnhandledException)");
            }
        }
    }
}
}

```

10) 设置APPFORWEBSERVICE为启动项目，运行，查看结果

七、总结

通过这次试验，我了解了应用VS2008进行我们基于web服务的编程 的做法。也使得我对于VS这个系列的软件都认识有了一个比较深刻的理解。也使得我初步了解SOA的体系结构。并且基本上掌握用Web service技术实现SOA。虽然在制作中还是有很多问题，但是都经过自己查阅和一系列的指点都完成了。并且在相关科目的一起协作下，发现这个实验对我的帮助很大。以上就是我对这个实验的总结。

实验：MDA 实践

一、实验目的

- 1) 初步了解 MDA 的框架。
- 2) 了解 PSM 到 CODE 的过程。
- 3) 掌握应用 ECO 实现 MDA 过程。

二、实验学时

6 学时。

三、实验方法

根据实验指导书, 实现 Staruml、ECO。

四、实验环境

计算机及 STARUML 和 VS2005。

五、实验内容

(1) StarUML 实践

六、实验操作步骤

实验内容一：StarUML 实践

1. 安装：

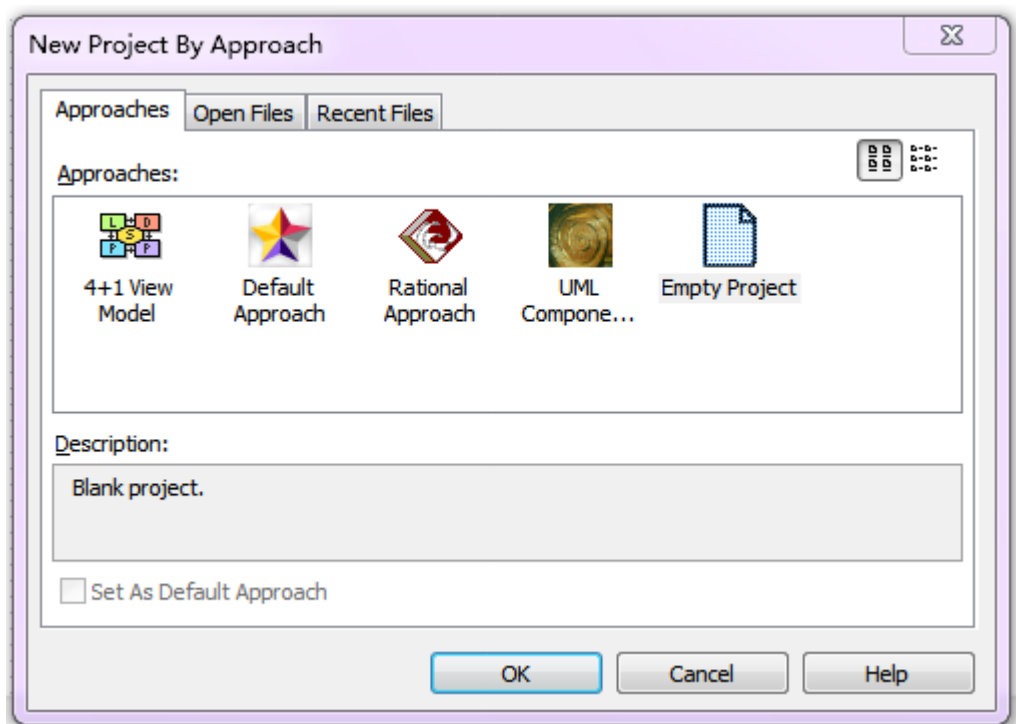
首先，我们必须先安装将要使用的软件。StarUML，是一个开放源码软件，遵循 GPL 协议许可（GNU 公共许可证），并免费提供下载。

2. 启动

安装以后就可以启动该程序。

3. 添加新工程

在 New Project By Approach 的对话框会弹出。选择“Empty Project”并且按下“确定”。



4. 选择模块

在右边的“Model Explorer”框中选定“Untitled”模块。

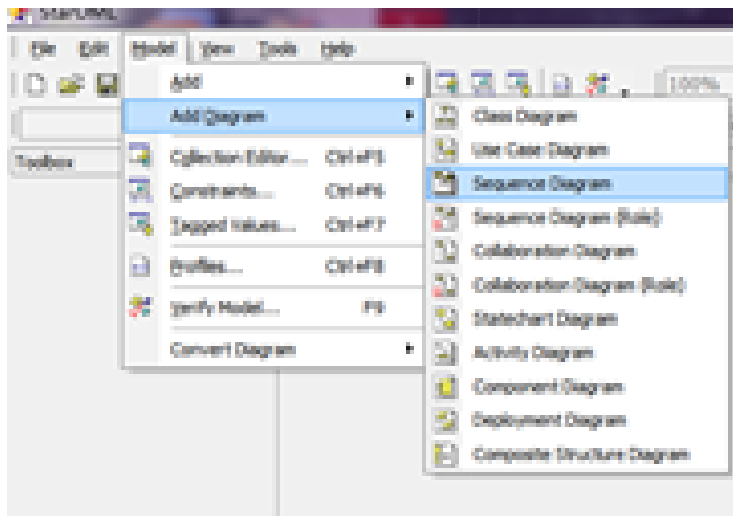
5. 添加模块

通过“Model”主菜单，或右击选定的模型，可以“Add/Model”



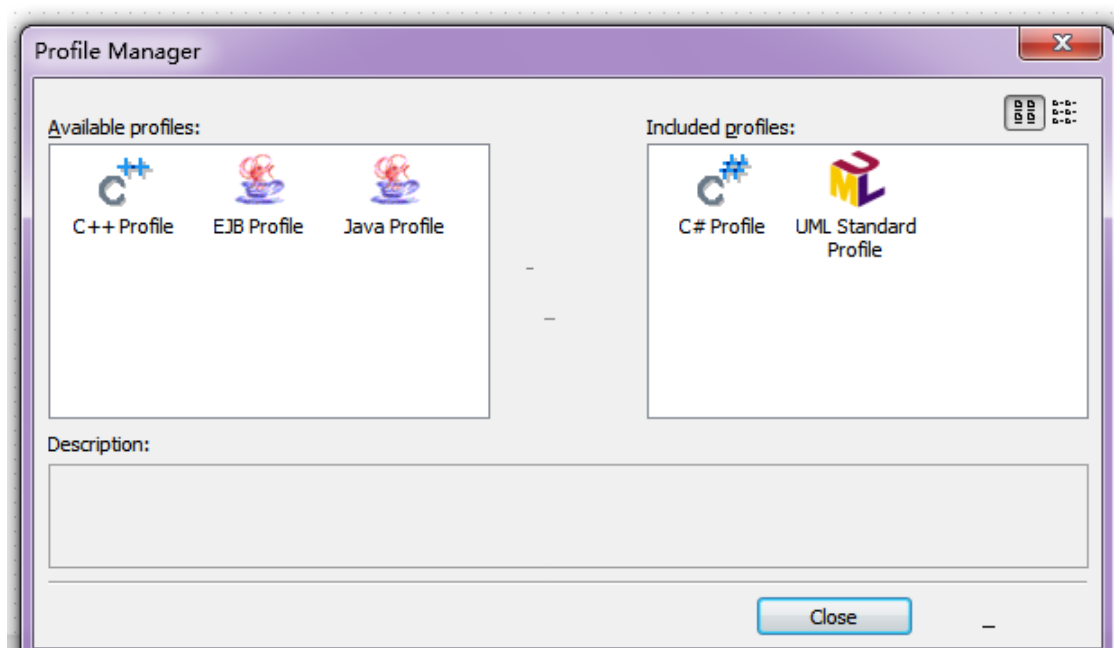
6. 添加类图

通过“Model”主菜单，或右击选定模型，可以“Add Diagram/Class Diagram”：



7. 设置 profile

通过“Model/Profile...”菜单去设置工程所需的 profile。这决定了工程所使用的规则和约定。根据语言，选择不同的 Profile，比如 JAVA，一定要包含“JAVA Profile”这一项目；比如 C#，一定要包含“C# Profile”这一项目。



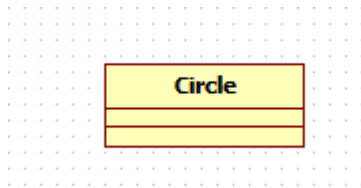
8. 保存工程

保存工程，命名为 LAB3. UML



9. 创造图表

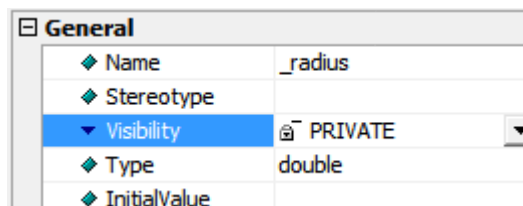
从默认就在屏幕的左边的“Toolbox”选择“类”图标，然后左键单击 diagram 窗口的某处。这样就使用通用名字创造了一个新的类。双击，将类改名为 Circle。



10. 添加属性

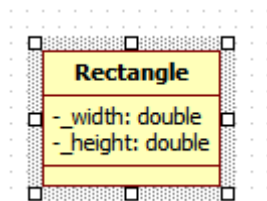
右击图中的目标，在弹出菜单中选择“Add”中的“Attribute”（被标示为绿色），为其添加一个属性(或者域)，填入期望的名字“_radius”。

- 具体的数据类型，在属性面板（右下侧的窗口），由双打字，在“类型”时段。在窗体右下边的 Properties 面板中，找到“Type”输入框，输入 double 作为 _radius 属性的类型。
- 类的内部数据（域/属性）都是私有的，因为他们是严格由类内部使用的。所以，在 Properties 面板中将 _radius 设置为“私有”



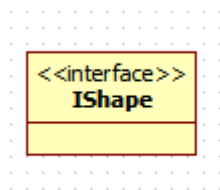
11. 继续进行设计

重复同样的过程，添加所谓的名字叫做 Rectangle 的类和 double 型的私有成员 _width 和 _height。

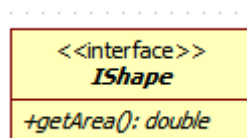


12. 创建 IShape interface

- 从 toolbox 中, 选择 “Interface”, 并点击图表的某处。将其改名为 IShape。创建以后, 选中它。
- 在顶部工具栏, 选择 “Stereotype Display” 下拉按钮, 将值改变为 “None”。这将改变以往的圆形形状, 使其变为成长方形。
- 还是在顶部工具栏, 取消选中 “Suppress Operations”。这将使我们能够看到接口所拥有的方法。



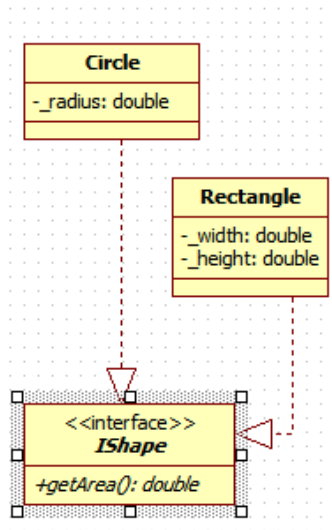
- 向 IShape 接口添加返回值为 double 的 getArea 方法。
 - 可以通过右击 interface 的图标, 在弹出菜单中点击红色的 “Operation” 按钮, 然后输入 getArea。
 - 设定返回值类型。在 “Model Explorer” 中展开 IShape 节点, 右击你刚刚创建的 getArea 方法, 并选择 “Add Parameter”。在 “Properties” 框中, 将参数的名字变为空, 将 “DirectionKind” 变为 “RETURN”, 将 “Type” 变为 double。
- 将 IShape 和 getArea 的 IsAbstract 属性框打上勾, 他们在图标上的名字将变为斜体。这是 UML 的标准, 表示这是接口或者其他纯虚实体。



13. 添加类和接口的关系

- 可以通过从 toolbox 中选择表示 “Realization” 的箭头, 并从 Circle 拖拽向 IShape, 使 Circle 实现接口 IShape。重复同样的过程, 为 Rectangle 添加实现关系。这是添加了 Circle 和 Rectangle 对于 IShape 接口的实现关系。

- 如果想使连接线表现为直角的方式，右击连接线，并选择"Format/Line Style/Rectilinear"菜单。你通过这种方式，使箭头重叠在一起，可以使你的图看起来更整洁。

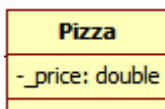


14. 添加类基于接口的行为

- 由于 **Circle** 和 **Rectangle** 类都实现了 **IShape** 接口，就必须有同样的行为(方法)。
 - 在“Model Explorer”面板中，复制 `getArea` 法(按 Ctrl-C 或者右键点击并选择 Copy 菜单)，并粘贴到 **Circle** 和 **Rectangle** 类。
 - 这些实现了的方法在 **Circle** 和 **Rectangle** 类中都不是抽象的，而是具体的。这是因为他们实际上是执行一些特定行为（例如，为一个圆形和长方形分别计算面积），所以不要勾选 `IsAbstract` 框。

15. 添加 Pizza 类

- 向 **Pizza** 添加 `double` 型的私有域 `_price`。
- 添加返回 `double` 类型的共有操作 `getPrice`。



以上内容仅为本文档的试下载部分，为可阅读页数的一半内容。如要下载或阅读全文，请访问：<https://d.book118.com/036050022213010133>