```
                    //检查文件是否存在
        if(!File.Exists(this.txtFileName.Text.Trim()))
        {
                MessageBox.Show("文件不存在！");
                return;
        }
        //连接数据库
    string strConnection="Provider = Microsoft.Jet.OLEDB.4.0
;Jet OLEDB:Database Password=;Data Source ="+    Application.
StartupPath.ToString().Trim()+"\\mydatabase.mdb" ;
        OleDbConnection myConnect=
           new OleDbConnection(strConnection);
        OleDbCommand myCommand=
            new OleDbCommand("select * from 版本 ",
myConnect);
        OleDbDataAdapter myDataAdapter=
           new OleDbDataAdapter();
        myDataAdapter.SelectCommand=myCommand;
        OleDbCommandBuilder myCommandBuilder=
            new OleDbCommandBuilder(myDataAdapter);
        myConnect.Open();
        //获取已有的数据
        m_DataSet=new DataSet();
        try
        {
        myDataAdapter.Fill(m_DataSet,this.m_TableName);
        //如果是首次上传,则增加一条记录
        if(m_DataSet.Tables[m_TableName].Rows.Count==0)
        {
                DataRow  newrow=m_DataSet.Tables
[m_TableName].NewRow();
                newrow["序号"]="1";
                m_DataSet.Tables[m_TableName].Rows.Add
(newrow);
        }
        DataRow row=m_DataSet.Tables[m_TableName].Rows[0];
        //填入去掉路径的文件名称
        row["文件名称"]=this.GetFileNameFromPath(this.
txtFileName.Text.Trim());
        //填入版本号
        row["版本号"]=this.txtVersion.Text.Trim();
        //将实际文件存入记录中
        FileStream fs=new FileStream(this.txtFileName.Text.
Trim(),FileMode.Open);
        byte [] myData = new Byte [fs.Length ];
        fs.Position= 0;
        fs.Read (myData,0,Convert.ToInt32 (fs.Length ));
        row["文件内容"] = myData;
        fs.Close();//关闭文件
        //更新数据库
        myDataAdapter.Update(this.m_DataSet,this.m_TableName);
        myConnect.Close();
        MessageBox.Show("文件更新成功！");
        }
        catch(Exception ee)
        {
                MessageBox.Show(ee.Message);
        }
        }
        else
        {
                MessageBox.Show("请输入文件名");
        }
}
```

至此，上传工具制作完成，通过该程序，可以上传主程序文件。当然，该工具是给软件开发供应商用于发布新软件用的，千万不要给用户哦。

最后是编写登录程序，按照编写上传工具的方法添加一个项目，项目名称为 Login，设置输出路径为 D:\Output，并设置该项目为启动项目。

添加一个组合框(cmbUserName)，设置 DropDownStyle 为 DropDownList，用来选择已有的用户名；添加一个用于输入密码的文本框(txtPassword)，设置 PasswordChar 属性为“*”，并在前面加入相应的文字标签；再添加确定（btnOK）和取消(btnCancel)按钮，并将确定按钮的Enable属性设置为false，目的是如果新软件没有下载完成，不准登录。布置如下图：

切换到代码窗口，添加引用：

```
using System.Data.OleDb;
using System.Threading;
using System.IO;
using Microsoft.Win32;
```

再添加如下变量：

```
    /// <summary>
    /// 存放操作员及密码的DataSet
    /// </summary>
    private DataSet m_DataSet;
    /// <summary>
    /// 本功能用到的数据库表
    /// </summary>
    private string m_TableName="操作员";
    private DataTable m_Table;
```

为了避免每次都下载主程序，我们将当前主程序的版本号要保存下来，采用的办法是保存到注册表中，为此，写两个函数，用于读取/写入注册表，如下：

```
    /// <summary>
    /// 定义本软件在注册表中software下的公司名和软件名称
    /// </summary>
    private string m_companyname=
                "lqjt",m_softwarename="autologin";
    /// <summary>
    /// 从注册表中读信息;
    /// </summary>
    /// <param name="p_KeyName">要读取的键值</param>
```

```
        ///<returns>
///读到的键值字符串,如果失败(如注册表尚无信息),则返回""
///</returns>
        private string ReadInfo(string p_KeyName)
        {
                RegistryKey SoftwareKey=Registry.
LocalMachine.OpenSubKey("Software",true);
                RegistryKey CompanyKey=SoftwareKey.
OpenSubKey(m_companyname);
                string strValue="";
                if(CompanyKey==null)  return "";
                RegistryKey SoftwareNameKey=CompanyKey.
OpenSubKey(m_softwarename);//建立
                if(SoftwareNameKey==null) return "";

                try
                {
                        strValue=SoftwareNameKey.GetValue
(p_KeyName).ToString().Trim();
                }
                catch
                {}
                if(strValue==null) strValue="";
                return strValue;
        }
        ///<summary>
        ///将信息写入注册表
        ///</summary>
        ///<param name="p_keyname">键名</param>
        ///<param name="p_keyvalue">键值</param>
        private void WriteInfo(string p_keyname,string
p_keyvalue)
        {
        RegistryKey SoftwareKey=Registry.LocalMachine.
OpenSubKey("Software",true);
        RegistryKey CompanyKey=SoftwareKey.CreateSubKey
(m_companyname);
        RegistryKey SoftwareNameKey=CompanyKey.CreateSubKey
(m_softwarename);
                //写入相应信息
        SoftwareNameKey.SetValue(p_keyname,p_keyvalue);
        }
```

再写一个函数,用户来获取用户名/密码和更新主程序版本:

```
///<summary>
///获取操作员情况,同时更新主程序版本
///</summary>
private void GetInfo()
{
        this.m_DataSet=new DataSet();
        this.combUsers.Items.Clear();
        string strSql=string.Format("SELECT * FROM ²Ù×÷Ô±
ORDER BY 姓名");
        //连接数据库
        string strConnection="Provider=Microsoft.Jet.OLEDB.
4.0;Jet OLEDB:Database Password=;Data Source="+Application.
StartupPath.ToString().Trim()+"\\mydatabase.mdb";
        OleDbConnection myConnect=
                new OleDbConnection(strConnection);
        OleDbCommand myCommand=
                new OleDbCommand(strSql,myConnect);
        OleDbDataAdapter myDataAdapter=
                new OleDbDataAdapter();
        myDataAdapter.SelectCommand=myCommand;
        try
        {
        myConnect.Open();
        //获取操作员信息
        myDataAdapter.Fill(this.m_DataSet,this.m_TableName);
        //将查询到的用户名填充到组合框供用户选择
        this.m_Table=this.m_DataSet.Tables[this.m_TableName];
        foreach(DataRow row in m_DataSet.Tables[m_TableName].Rows)
        {
        this.combUsers.Items.Add(row["姓名"]).ToString().Trim();
        }
        //检查是否有新的版本
        DataSet dataset=new DataSet();
        string tablename="tablename";
        //为减少数据传送时间,不获取文件内容
        strSql="select 文件名称,版本号 from 版本";
        myCommand=new OleDbCommand(strSql,myConnect);
        myDataAdapter=new OleDbDataAdapter();
        myDataAdapter.SelectCommand=myCommand;
        myDataAdapter.Fill(dataset,tablename);
        if(dataset.Tables[tablename].Rows.Count==1)//有文件
        {
        string filename=dataset.Tables[tablename].Rows[0]["
文件名称"].ToString();
        string version=dataset.Tables[tablename].Rows[0]["
版本号"].ToString();
        //读入本机主程序的版本号
        string oldversion=this.ReadInfo(filename);
        if(oldversion.Length==0)//不存在
                oldversion="0";
        //有新的版本出现
        if(Decimal.Parse(version)>Decimal.Parse(oldversion))
        {
          //取回文件内容
          dataset=new DataSet();
          strSql="select * from 版本";
          myCommand=new OleDbCommand(strSql,myConnect);
          myDataAdapter=new OleDbDataAdapter();
          myDataAdapter.SelectCommand=myCommand;
          myDataAdapter.Fill(dataset,tablename);
          //将文件下载到本地
          DataRow row=dataset.Tables[tablename].Rows[0];
          if(row["文件内容"]!=DBNull.Value)
          {
          Byte[] byteBLOBData =  new Byte[0];
          byteBLOBData = (Byte[])row["文件内容"];
          try
          {
            FileStream fs=new FileStream(Application.
StartupPath+"\\"+filename,FileMode.OpenOrCreate);
        fs.Write(byteBLOBData,0,byteBLOBData.Length);
        fs.Close();
        //写入当前版本号,供下次使用
        this.WriteInfo(filename,version);
        }
        catch(Exception ee)
        {       MessageBox.Show(ee.Message);  }
        }

        }//有新版本
        }//有文件
        //关闭连接
        myConnect.Close();
```

```
}
            catch(Exception ee)
            {
                  MessageBox.Show(ee.Message);
                  return;
            }
            //允许登录
            this.btnOK.Enabled=true;
      }
```

为了不让用户等待太久,在启动时通过一个线程,让获取用户信息和更新在后台完成,即在窗口Load事件中,通过线程调用上面的GetInfo的函数,故窗口Load代码如下:

```
private void Form1_Load(...)
{
      //为加快显示速度,将数据库连接等放到另外一个线程中去
      Thread thread=new Thread(new ThreadStart(GetInfo));
      thread.Start();
}
```

有了上述准备,我们来编写确定按钮的响应代码如下:

```
private void btnOK_Click(...)
{
      //根据组合框的选择,得到当前用户在DataSet中具体物理位置
      if(this.combUsers.SelectedIndex<0)//没有选择
            return;
      DataRow rowNow=null;
      foreach(DataRow row in this.m_DataSet.Tables[this.
m_TableName].Rows)
      {
            if(row["姓名"].ToString().Trim()==this.
combUsers.Text.Trim())
            {
                  rowNow=row;
                  break;
            }
      }
      if(rowNow==null)      return;
      //获取当前正确密码
      string strPassword=rowNow["密码"].ToString().Trim();
      this.txtPassword.Text=this.txtPassword.Text.Trim();
      if(this.txtPassword.Text==strPassword)//密码正确
      {
            //主程序名称
         string filename=
            Application.StartupPath+"\\"+"MainPro.exe";
            //参数名称
       string arg=this.combUsers.Text+" "+this.txtPassword.Text;
       //运行主程序
       System.Diagnostics.Process fun=
          System.Diagnostics.Process.Start(filename,arg);
      //关闭登录框
      this.Close();
      }
      else
      {
            MessageBox.Show("密码错误!如果你确信密码输
入正确,\n可以试着检查一下大写字母键是否按下去了。","警告
",MessageBoxButtons.OK,MessageBoxIcon.Warning);
      this.txtPassword.Focus();
      this.txtPassword.SelectAll();
      }
}
```

取消按钮的代码非常简单,就是关闭登录窗口:

```
      private void btnCancel_Click(...)
      {
            this.Close();
      }
```

把Login和MainPro软件连同其他相关文件打包成安装程序,将Login以快捷方式放到桌面或开始菜单中供用户使用(当然,快捷方式名称可以随便取了),用户运行Login后,会自动更新软件。

本例中所有代码请到ftp://qydn.vicp.net/下载,文件名为update.rar,解压缩后别忘了在D:\创建一个output文件夹,并将mydatabase.mdb复制到该文件夹中。

说明:本文只起抛砖引玉的作用,通过该思路进行扩展可以完成许多功能,如通过修改上传/登录程序,不仅可以实现对主程序的更新,而且可以实现对任何要用到的资源文件进行更新。本例中不能实现对登录框本身的更新,我采用的办法是在主程序的Closing事件中更新登录窗口,因为此时登录窗口已经关闭了。在登录窗口中,可以放一个"保存密码"的复选框,如果用户选中该组合框,可以将用户名和密码保存到注册表中,下次登录时直接读入,用户只要点确定按钮即可,免去了每次都选用户名和输密码的烦恼。

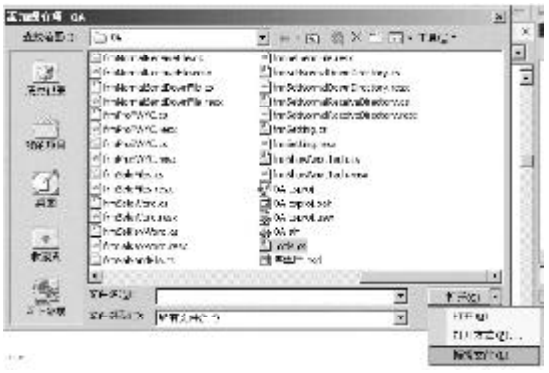
在本例中,我们可以看到,数据库的连接、查询等工作是重复性劳动,且三个个项目中用到的数据库、公司名称等是一样的。在实际工作中,我们可以单独新建一个cs文件,不妨取名为MyTools.cs,将一些常用函数和变量(如数据库连接、公司名称等)做成静态的,各具体项目中链接本文件,然后直接使用。我们只需修改MyTools.cs中的相关变量或函数而不必在每个项目中都去改,既方便又不会遗漏,MyTools.cs参考如下:

```
///<summary>
///预编译选项,如果定义了NETWORKVERSION,,表示是网络版,使
用SQL2000数据库,否则,使用ACCESS2000数据库
///</summary>

//#define NETWORKVERSION

using System;
using System.Drawing;
using System.Collections;
```

```
using System.ComponentModel;
using System.Windows.Forms;
using System.Drawing.Imaging;
using System.IO;
using System.Data;

#if NETWORKVERSION
using System.Data.SqlClient;
#else
using System.Data.OleDb;
#endif
using System.Reflection;
using Microsoft.Win32;
namespace OA
{
      public class Tool
      {
            public Tool()  {      }
            ///<summary>
            /// 主程序的文件名
            ///</summary>
            public const string FileName="OA.exe";
            public const string g_TitleName="丽汽集团办公自动化系统";
            public static string g_UserName;
            public static void WriteInfo(string p_keyname,string p_keyvalue)
            {    ……    }
//其他类似代码略……
}
}
```

如果一个项目中要用到MyTools中的内容,可以按如下方式进行:

在“解决方案资源管理器”窗口中选择该项目,选择菜单“项目”→“添加现有项”,此时弹出打开文件对话框,文件类型设为所有文件(*.*)。找到MyTools.cs,不要直接点打开按钮,看到了打开按钮后面的“↓”了吗?单击它可以弹出一个菜单,选择“链接文件(L)”,这样插入的文件只是一个链接,不会生成副本(如下图)。



使用时,添加MyTools的应用,再使用Tool类中的公共函数,如:

```
      using OA;
      private void myFun()
      {  string s=Tool.FileName;  }
```

如果单位名称变了,我们只要修改MyTools.cs中的变量就可以了,不必到每个项目中都去修改。

我们还注意了一个细节:

```
///<summary>
///预编译选项,如果定义了NETWORKVERSION,,表示是网络版,使用SQL2000数据库,否则,使用ACCESS2000数据库
///</summary>
//#define NETWORKVERSION
```

我们知道,对于Access或SQL Server等,除了连接方式外,其余操作几乎完全一样,因此,我们定义了一个选项,如果#define NETWORKVERSION,表示是网络版,使用SQL Server数据库,否则(将#define NETWORKVERSION注释掉)就是单机版。使用Access数据库,在MyTools中我们将两种连接方式有区别的地方分别编写,就可以通过是否注释掉#define NETWORKVERSION这一行分别生成单机版和网络版软件,参考代码如下:

```
/// <summary>
/// 根据SQL语句返回一个查询结果,主要用于只要求返回一个字段的一个结果的情况
/// </summary>
/// <param name="p_Sql">查询用到的SQL语句</param>
/// <returns>查询到的结果,没有时则返回空""</returns>
public static string GetAValue(string p_Sql)
{
string strResult="";
Tool.OpenConn();
//设计所需要返回的数据集的内容
try
{     //打开指向数据库连接
      #if NETWORKVERSION //网络版
      SqlCommand aCommand = new SqlCommand ( p_Sql , m_Connect ) ;
      SqlDataReader aReader =aCommand.ExecuteReader ( ) ;
      #else  //单机版,注意变量名aCommand和aReader在两个版本中都是一样的,有利于编程
      OleDbCommand aCommand = new OleDbCommand ( p_Sql , m_Connect ) ;
      OleDbDataReader aReader = aCommand.ExecuteReader ( ) ;
      #endif
      // 返回需要的数据集内容这里就不分单机版还是网络版了,反正变量名一样
      if(aReader.Read()) strResult=aReader[0].ToString();
      aReader.Close () ;
      }
      catch(Exception ee)
      { MessageBox.Show(ee.Message);}
      return strResult;
      }
```

以上类似的小技巧还很多,注意总结,定会收益良多。

投稿信箱:tougao@csdn.net

SWT和JFace的可视化编程

▶撰文/温晓晖　郝聃

Java语言目前在服务器端表现出强大的开发能力，特别是J2EE规范的日趋丰富和完善，更加推动了Java企业级的应用。但是，使用Java开发桌面应用，特别是高度交互的客户机应用程序，总是不能达到满意的效果。虽然Sun公司为了加强Java的界面功能而不断更新和增加Swing的功能，然而采用Swing或者AWT（Abstract Window Toolkit）开发出来的Java应用程序总是同操作系统的其他应用程序显得很不一样，无论是速度还是外观上，都不尽人意。那么我们是否能够使用Java开发出与其运行操作系统其他应用程序界面一致的程序吗？随着Eclipse项目的出现，这种想法已经成为现实。

SWT和JFace

SWT是Eclipse的一个窗口构件集和图形库，它集成于本机窗口系统但有独立于OS的API。JFace是用SWT实现的UI工具箱，它简化了常见的UI编程任务。JFace在其API和实现方面都是独立于窗口系统的，它旨在使用SWT而不隐藏它。虽然SWT和JFace是为Eclipse开发的，但是我们完全能够在自己的Java应用中使用它们，从而能够建立高效的Java桌面应用程序。事实上，这样开发出来的应用程序拥有同操作系统其他应用完全一样的界面，并且能够跨平台运行。

目前最新的Eclipse版本是3.0，可以从www.eclipse.org网站上下载发布包及源代码。

可视化编程工具：SWT Designer

Eclipse的标准版本并不包含任何可视化的界面设计模块。然而对于开发一个桌面应用的用户来说，如果采用了所见即所得（WYSIWYG, What You See Is What You Get）功能，那么不但界面的设计可以得到保证，而且能够极大提高开发效率，这也正是VB、VC等微软开发工具流行的原因之一。

在Eclipse下通过一个插件——SWT Designer能够可视化地建立基于SWT和JFace部件的程序。完整版本的SWT Designer包括UI设计器和菜单设计器，而免费版本则不包括菜单设计器和UI设计器的部分功能，因此，要可视化地进行菜单和其他一些方面的界面设计，只能够通过代码实现了。免费版本的SWT Designer可以从www.swt-designer.com网站下载。

操作系统：Windows 98/ME2000/XP or Linux +GTK2。

运行：任何基于Eclipse 2.1以上或者3.0 (M3)以上的开发环境，包括：

- ❑ Eclipse 2.1.1 or Eclipse 3.0 M3；
- ❑ WebSphere Studio Application Developer (WSAD) 5.1；
- ❑ WebSphere Studio Site Developer (WSSD) 5.1；
- ❑ WebSphere Studio Enterprise Developer (WSED) 5.1。

安装：将下载的压缩包解压到Eclipse目录下。

在Eclipse 3.0中，如果SWT Designer安装正确，则可以通过“Windows-Preferences”菜单显示出Designer节点，能够进行配置。

设计和使用

我们将使用SWT、JFace建立记事本程序，同时利用SWT Designer的UI设计功能进行可视化界面设计。

3.1. 建立新工程

建立一个新的Java工程“Java Project”，设置构建路径如下：

添加 Eclipse 安装目录下 plugins 中的：

```
    $ECLIPSE$/plugins/org.eclipse.jface_3.0.0/jface.jar"
    $ECLIPSE$/plugins/org.eclipse.swt.win32_3.0.0/ws/win32/swt.jar"
    $ECLIPSE$/plugins/org.eclipse.core.runtime_3.0.0/runtime.jar"
    $ECLIPSE$/plugins/org.eclipse.ui.workbench.
texteditor_3.0.0/texteditor.jar"
    $ECLIPSE$/plugins/org.eclipse.core.boot_3.0.0/boot.jar"
```

如果是2.1.1版本，则使用2.1.1替代3.0.0。如果不是Windows平台，则需要到以下位置找到swt.jar：

```
    gtk: $ECLIPSE$/plugins/org.eclipse.swt.gtk_3.0.0/ws/gtk/
    motif: $ECLIPSE$/plugins/org.eclipse.swt.motif_3.0.0/ws/motif/
    photon: $ECLIPSE$/plugins/org.eclipse.swt.photon_3.0.0/ws/photon/
    macosx: $ECLIPSE$/plugins/org.eclipse.swt.carbon_3.0.0/ws/carbon/
```

为了能够独立启动采用SWT开发出来的应用程序，我们配置运行时环境。在Windows下，将$ECLIPSE$\plugins\org.eclipse.swt.win32_3.0.0\os\win32\x86\下的DLL拷贝到系统目录下，在Windows2000/XP就是windows/system32目录。其他操作系统的运行环境如下：

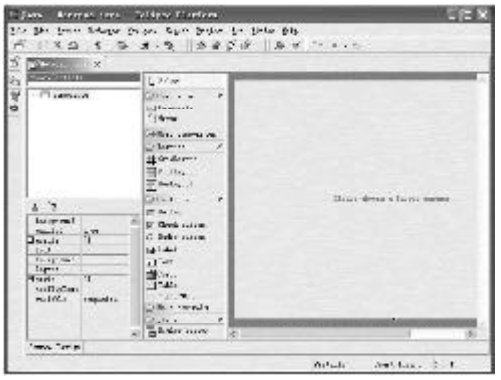
```
    Linux GTK: $ECLIPSE$/plugins/org.eclipse.swt.gtk_3.0.
0/os/linux/x86/
    Linux Motif: $ECLIPSE$/plugins/org.eclipse.swt.motif_3.
0.0/os/linux/x86/
```

完成了以上步骤后，我们可以开始使用设计器了。

3.2. 使用设计器

在当前工程中新建文件，选择“SWT”，可以发现目前可用的几种应用。

由于JFace将SWT最常用的一些功能进行包装，因此，我们首先建立一个“JFace ApplicationWindow”作为记事本程序的主窗口。完成后打开源代码窗口，在代码窗口底部能够进行代码和设计器的切换。切换到设计器显示：



我们可以看到内容区、工具区、属性窗口和控件树。为了保证在多个平台上运行的程序界面比较一致，我们需要首先对程序设置一个布局。目前，免费版本支持三种类型的布局管理：GridLayout、FillLayout和RowLayout。一般使用较多的是GridLayout，它能够以网格的形式对其中的对象进行管理。

记事本的菜单和状态条可以从代码添加：

```
    protected MenuManager createMenuManager() {
        MenuManager result = new MenuManager("menu");
        buildMenu(result);
        return result;
    }
        private void buildMenu(MenuManager menu) {
        MenuManager menufile = new MenuManager("文件(&F)");
        MenuManager menuedit = new MenuManager("编辑(&E)");
        MenuManager menuformat = new MenuManager("格式(&O)");
        MenuManager menuview = new MenuManager("查看(&V)");
        MenuManager menuhelp = new MenuManager("帮助(&H)");
                menu.add(menufile);
                ......//帮添加其它菜单
                buildFileMenu(menufile);
                ......//其它略
        }
    protected StatusLineManager createStatusLineManager()
        {
    StatusLineManager status = new StatusLineManager();
    StatusLineContributionItem sub = new
StatusLineContributionItem("calc");
                sub.setText("1 : 1");
                status.add(sub);
                return status;
        }
```

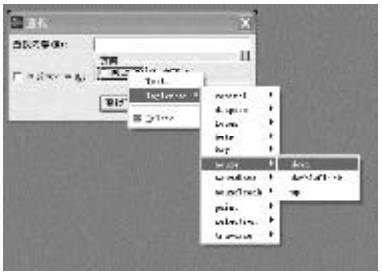
现在，我们再添加另外一个窗口，显示查询功能。新建一个“JFace dialog”，拖动工具区中的控件直接到界

面中,同时进行调整和在属性窗口中编辑控件的属性,界面设计如下:

下一步,我们将处理界面操作的事件。

3.3. 事件处理

右键点击某个希望处理事件的控件,在弹出的对话框中选择'Implement'就可以出现各种可能事件的子菜单,直接点击某个菜单项将进入对该事件具体实现的代码窗口。



到这儿,我们似乎也可以多少体验到一些可视化编程的优势了,尤其是对于这种界面事件处理不是太熟悉的用户。

某些默认按钮的事件并不能够从界面进入编辑,如"取消"按钮,代码如下:

```
protected void cancelPressed() {
        this.close();
}
```

因此,界面的事件选择和直接代码编写事件结合起来才能够获取最佳的程序运行状态。

独立运行

当我们完成应用程序的开发时,重要的是使其在用户的系统中能够正确运行。而这样做的工作并不复杂,当然首先我们还是需要Java运行环境已经安装。

在Windows下运行方式和结果:

```
SET CLASSPATH=.;lib/boot.jar;lib/jface.jar;lib/runtime.
jar;lib/texteditor.jar;
lib/notepad.jar;lib/win32/swt.jar;

start %JAVA_HOME%\bin\javaw -cp "%CLASSPATH%" -Djava.
library.path=lib/win32 com.winer.notepad.Note
```

在Linux下运行方式:

```
DIRNAME=`dirname $0`
USER_HOME=`cd $DIRNAME/..; pwd`
JAVA_HOME="$USER_HOME/j2re1.4.2_02"
NOTE_HOME="$USER_HOME/notepad"
LIB_HOME="$NOTE_HOME/lib"

# Setup the JVM
JAVA="$JAVA_HOME/bin/java"
CLASSPATH="$LIB_HOME/notepad.jar:$LIB_HOME/boot.jar:
$LIB_HOME/jface.jar:$LIB_HOME/runtime.jar:$LIB_HOME/
texteditor.jar:$LIB_HOME/linux/swt.jar:$LIB_HOME/linux/swt-
gtk.jar:$LIB_HOME/linux/swt-mozilla.jar:"
export LD_LIBRARY_PATH=$LIB_HOME/linux:LD_LIBRARY_PATH
LC_ALL=zh_CN.GB2312;export LC_ALL
LANG=zh_CN.GB2312;export LANG

# Execute the JVM
"$JAVA" -classpath "$CLASSPATH" -Djava.library.
path=$LIB_HOME/linux com.winer.notepad.Notepad
```

评价

SWT Designer对于本身不具有界面设计器的Eclipse来说确实是一种非常实用的插件。虽然目前来说可以实现的功能并不多(对于免费版本更是限制颇多),但是随着Eclipse项目的发展和SWT及JFace开发的普及,可视化的界面设计必然成为一种趋势,尤其是在Sun公司提供的Java语言界面设计方面功能不是太强的条件下,这种充分利用操作系统资源的应用模式必然能够得到更好的应用。

参考资料

- 有关Eclipse的资源可以从www.eclipse.org下载。
- 有关SWT Designer的资源可以从www.swt-designer.com下载。

利用 Turbine 的事件映射机制来扩展 Struts 的功能

▶撰文 / 刘冬

Turbine 和 Struts 的简介

Turbine是一个基于Java小服务程序(Servlet)的Web应用程序框架,它使得Java开发者可以快速、安全的构建自己的Web应用程序。与Struts不同的是Turbine的设计思路决定了它是一个Web应用的完整的解决方案,它是一个完整的实现了MVC模式的应用框架。但是对众多Web开发者来讲Turbine框架过于烦杂难懂,同时由于Struts的广泛应用而且有众多的参考的资料,使得Turbine的很多优点都被埋没。但是Turbine有一个简单但是非常有用的东西。我们今天要介绍的就是Web表单事件的映射机制。同时我们也会把这机制带入到Struts框架中,使Struts应用的开发更加方便快捷。

Turbine主要由五个部分组成,如下图所示。其中我们要介绍的表单事件映射属于Action这一部分,我们主要了解一下Action的基本结构,其他的部分不进行描述。

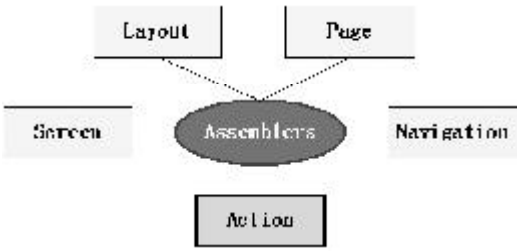


图 1

当用户提交一个HTML表单的时候,通过一些隐含的字段就可以包含将要被执行的Action的信息。Action机制使得Java开发者更容易处理用户提交的数据。例如,对于“Logout”这个事务,在系统的多个地方都可能被调用;因此,将Logout的事务处理流程写成一个可重用的模块,使得这个事务可以更方便地被调用。这个可重用的模块就是一个Action。通过系统中多种多样的Action,每个Action处理用户数据中不同的信息,这样,整个系统就显得更加简单明快,更易编写、扩充与维护。并且,Turbine通过Action机制,还可以使程序流程更加灵活多变。例如,在Page的处理过程中,可以通过执行特定的Action,帮助判断其后将要显示哪个Screen。这时,Action的执行结果就可以作为以后程序的判断依据。这些特性看起来好像跟Struts的Action差不了多少。但是我们回想一下Struts,我们是不是每次都需要给不同的表单提交动作开发不同的Action的类?当然我们也可以使用一个Action类来处理多种不同的请求,但是这就要求我们必须在Action的实现方法中加入一大堆的if...else....的条件判断来转到不同的方法去处理,这样做使得代码判断分支过于庞大,而且隐讳难懂。

Turbine通过使用Java的反射技术很好的解决了这个问题,也就是将不同的表单请求动作与某个Action类的某个方法联系起来,避免开发者自己编写大堆的条件判断语句。如果开发者需要将事件和方法联系起来,那么需要提供一些Turbine规定的特殊标识来进行处理。例如,如果我们要将某个表单的提交按钮点击后运行指定Action类的doDelete方法,那我们可以给这个按钮命名为 eventSubmit_Delete,其中名称前缀 eventSubmit_ 是Turbine中规定的一个标识符。框图如下所示:

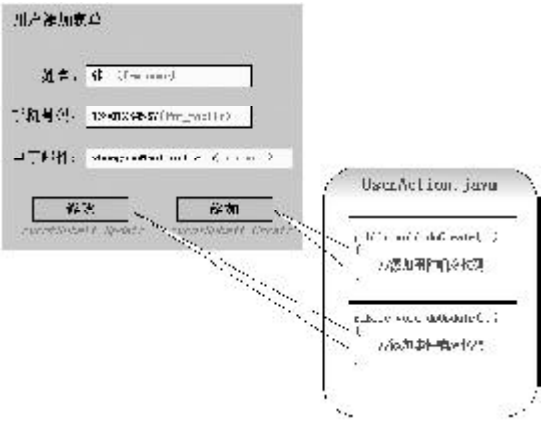


图 2

上图中,同一个表单中的两个提交按钮点击后触发同一个类的两个不同的方法,这是由Turbine内部进行处理的,开发者只要按照它提供的命名规范即可完成该功能。本文的目的就是引入这个机制来扩展Struts的功能,为了方便Turbine的开发者,我们使用的命名约定与Turbine的要求一致。

具体实现细节

a. 约定

首先我们先给出表单域名称与Action类方法名之间的关系,下表是一些例子:

表单域名称	方法名
eventSubmit_Create	doCreate
eventSubmit_UpdateUser	doUpdateUser

表单域名称和方法名是大小写敏感的。

doXXX的方法原型有多种类型,如下表所示:

1	public ActionForward doXXXX(ActionMapping map, ActionForm form, HttpServletRequest request, HttpServletResponse response) throws IOException, ServletException
2	public ActionForward doXXXX(ActionMapping map, ActionForm form, HttpServletRequest request, HttpServletResponse response, PT pt) throws IOException, ServletException

其中类型PT可以是Java中几种基本类型,包括String类型,这个参数的类型主要取决于eventSubmit_XXXX表单域对应的值。如果用户需要获取该域的值,那么可以多加一个PT参数,在方法体内通过访问PT参数来获取表单域的值。

b. 结构

开发一个类FormAction使它直接是Struts的Action类的子类,在FormAction类中重载Action类的execute方法以实现事件的映射功能。Action类的开发者如需要事件的映射功能,那么他的Action类需要扩展FormAction类而不是Action类。建议把FormAction类中重载的execute声明为final以防止被其他Action类重载。

c. 流程

类FormAction的execute方法体中首先遍历所有浏览器提交上来的参数判断是否有命名以eventSubmit_开头参数,如果有,则解析出包含在名称中的方法名,并增加do的方法前缀;然后查找实例中是否存在该名称的方法。如果存在,则触发并终止遍历,否则继续判断下一个参数。

d. 关键代码

```
/* Created on 2003-11-29 by Liudong
 */
package lius.struts;

import java.io.IOException;
import java.lang.reflect.*;
import java.util.*;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;

/**
 * 支持事件映射的Action
 * 该类的所有方法和属性都定义成final或者private以防止子类修改
 * @author Liudong
 */
public abstract class FormAction extends Action
{
   public final static String SUBMIT_BUTTON_PREFIX =
   "eventSubmit_";
   public final static String METHOD_PREFIX = "do";
   public final ActionForward execute(
      ActionMapping mapping,ActionForm form,
      HttpServletRequest req,HttpServletResponse res)
            throws IOException, ServletException {
            String method = null;
            Enumeration enum = req.getParameterNames();
            while (enum.hasMoreElements()) {
            String param = (String) enum.nextElement();
if (param.startsWith(SUBMIT_BUTTON_PREFIX)) {
method=param.substring(SUBMIT_BUTTON_PREFIX.length());
if (!method.startsWith(METHOD_PREFIX))
method = METHOD_PREFIX + method;
   try {
    return callDoMethod(method,req.getParameter(param),
            mapping,form,req,res);} catch (Throwable t)
            {
            if (t instanceof IOException)
            throw (IOException) t;
            if (t instanceof ServletException)
            throw (ServletException) t;
            throw new ServletException(t);
                  }
            }
   }
   return null;
}
   /**
    * 调用doXXXX方法
    * @param methodName   要调用的方法名
```

```
     * @param paramValue    eventSubmit_XXXX对应的参数值
     * @param mapping
     * @param form
     * @param req
     * @param res
     * @return
     * @throws Throwable
     */
private ActionForward callDoMethod(
      String methodName,
      String paramValue,
      ActionMapping mapping,
      ActionForm form,
      HttpServletRequest req,
      HttpServletResponse res)
      throws Throwable {
        Method m = null;

      //获取当前实例类的所有定义的方法
      Method[] ms = getClass().getDeclaredMethods();
        for (int i = 0; i < ms.length; i++) {
            if (isActionMethod(ms[i])) {

      //找到合法的Action方法
            Object[] params = null;
                  try {

      //准备方法所需的参数
      Class[] paramTypes = m.getParameterTypes();
            params = new Object[paramTypes.length];
            params[0] = mapping;
            params[1] = form;
            params[2] = req;
            params[3] = res;
      for (int pc = 4; pc < paramTypes.length; pc++)
      params[pc] = getParamObject(paramValue, paramTypes[pc]);
      return (ActionForward) ms[i].invoke(this, params);
             }catch (InvocationTargetException e) {
                  throw e.getTargetException();
                  }
            }
          }
      return null;
     }

      /**
      * 将字符串value转化成用户在Action方法中定义的参数类型
      * @param value
      * @param objType
      * @return
      * @throws Throwable
      */
 private Object getParamObject(String value, Class objType)
            throws Throwable
      {
            if(String.class.equals(objType))
                  return value;
            //将字符串value转换相应类型的对象,限于篇幅此处不再重复
            return null;
      }

      /**
      * 判断某个类是否为JAVA的基本类型
      * @param type
      * @return
      */
      private boolean isBaseType(Class type) {
            return (
                  String.class.equals(type)
                        || int.class.equals(type)
                        || long.class.equals(type)
                        || double.class.equals(type)
                        || float.class.equals(type)
                        || byte.class.equals(type)
                        ||boolean.class.equals(type)
                        || short.class.equals(type)
                        || char.class.equals(type));
      }

      /**
      * 判断指定的方法是否是合法的Action方法
      *
      * @param m
      * @return @throws
      * Exception
      */
 private boolean isActionMethod(Method m) throws Exception
  {
      //判断方法参数是否符合要求
      Class[] paramTypes = m.getParameterTypes();
      if (paramTypes.length < 4 || paramTypes.length > 5)
            return false;

      //前四个参数必须与Action类的execute方法定义的参数一致
            for(int i=0;i<4;i++)
      if(!paramTypes[i].equals(g_paramTypes[i]))
            return false;

      //后面的一个参数必须是基本类型
      for(int i=4;i<paramTypes.length;i++)
                  if(!isBaseType(paramTypes[i]))
            return false;

      //判断返回值是否符合要求
      Class returnType = m.getReturnType();
      return ActionForward.class.isInstance(returnType.
            newInstance());
      }

      //基本参数
      private final static Class[] g_paramTypes =
            new Class[] {
                  ActionMapping.class,
                  ActionForm.class,
                  HttpServletRequest.class,
                  HttpServletResponse.class };
 }
```

e. 对象的转换

由于篇幅的问题,上面代码中关于对象的转换也就是getParamObject方法并没有写完。该方法的作用就是将字符串转换为几个Java的基本类型,例如int等。如果仅仅是将String转换成int那就没有什么好讲的了。但是大家疑惑的是getParamObject方法返回值是Object, Object类型怎么能跟int对应起来？返回int的

封装类Integer行不行？行不行,我想你一试就知道了。执行时将抛出 ClassCastException异常，那岂不是需要将doXXXX方法中自己定义的参数改为例如Integer而不是int吗？其实并没有这个必要,我们完全有办法用一个Object来替代int类型,下面代码完成了这个转换过程:

```
    /**
     *将一个整数转换为对象
     * @param i
     * @return
     */
public static Object getObject(int i) {
   Integer I = new Integer(i);
   Method m;
   Object v = null;

   try {
      m = I.getClass().getDeclaredMethod
                  ("intValue", null);
      v = m.invoke(I, null);
    }
    catch (Exception excp)
     {
     }
    m = null;
    I = null;
    return v;
}
```

这是我实在没有其它办法的时候采用的方法,我不知道是否还有其他更简单直观的方法可以使用,至少目前还没有找到这样的方法,希望有更好方法的读者可以给我指点迷津。

第一段代码唯一需要完善的地方就是针对每个基本类型完成一个转换方法，然后就可以完成getParamObject函数了。

结束语

好了,现在你可以随便根据自己的需要想在Action里处理多少事件就处理多少。不过扩展这样的功能并不是要我们把所有的事件响应方法都集中在一个Action类中,而是建议把同一类的事件处理放在同一个类中,这样可以避免过多的Action类,导致代码管理混乱。例如我们可以把所有关于用户维护操作所产生的事件放到一个叫UserAction的类里进行集中处理,这种方法感觉起来就非常的直观,而且类文件也很简洁。

当然了，Turbine还有其他非常棒的功能，只不过这不是我们文章的内容。

本文仅仅是介绍了事件方法的映射,如果能结合属性的映射也就是说将表单中的域自动与Action类中的属性关联起来,那开发Action类又多了一样宝物了,属性的映射需要用到Java的反射技术,有兴趣的朋友可以致信编辑部探讨！

参考资料

1 http://jakarta.apache.org/turbine/,Turbine官方站点。
2 http://jakarta.apache.org/struts/,Struts官方站点。

小知识

Velocity vs JSP; Turbine vs Struts观点荟萃

（不代表本刊观点）

bwwlpnn 观点：

Turbine vs. Struts:两者都实现了 MVC 中的 view 和 controller的要求。只不过 Turbine在 view的实现上主要采用了 Velocity，Turbine能支持JSP，但是支持的不好，且不推荐使用 JSP（详见下文）；Struts在 view的实现上只采用JSP。如果要实现controller，两者不能同时使用。即在一个 Web Application中只能用Turbine或Struts两者之一来实现 controller。当然，对于view的实现，两者应该可以同时使用。

Velocity vs. JSP: 两者都是实现同一功能的template languages。JSP是 Sun的标准，Velocity是开源社区开发的系统。网上有几篇详尽的文章 (http://jakarta.apache.org/velocity/ymtd/ymtd.html) 比较两者的异同,由于是开源社区写的,当然偏向Velocity。不过我详细读过一遍,文章写得有理有据,全都是开发者的经验,并没有推销Velocity之嫌。每篇文章后都有一句:You make the decision。在开源社区看来，JSP是 Sun开发的一个“ 不好用 ”的产品。

“ 不好用 ”主要体现在：

难于Debug;没有必要把简单问题复杂化，如JSP的先编译再运行;不能防止用户将Java code嵌入JSP,从而破坏MVC的设计。

重构及其在 JBuilder 下的应用

▶撰文/陈华　陈铁英

重构介绍

假如你目前正参与改善一个设计混乱的项目，面对糟糕的上万行代码，也许你会觉得无法着手，希望有好的方法能帮助你解脱出来。如果是这样，请看看《重构》（Refactoring）吧。

什么是重构？《Refactoring》一书中有这样的定义，重构（Refactoring）：对软件内部结构的一种调整，目的是在不改变软件可观察之行为的前提下提高其可理解性、降低其修改的成本。

重构的目的就在于改善现有代码设计质量，提高软件项目的可维护性，并尽可能少的引入新的Bug。简单的说，传统的软件项目观点是先设计后编码，重构的思想就相反，讲究在编码过程中逐步改善现有设计。

举个简单的例子，假设你现在阅读的代码中一个方法有几千行，你想拆分代码为几个方法，工作量一般较大而且容易出错，如果你有一个自动化的重构工具，使用重构中的方法拆分(Extract method)，可以在几秒内达到你的目的。或者是面对有几千个源代码文件的大型系统，你想给一个方法新增一个参数，由于调用这个方法的地方也许有几百个，难道需要每个文件逐一去手工修改么，如果你使用重构中的修改参数（Change Parameters）方法，可以很迅速地完成你的工作。

上面两个例子就是比较简单的重构应用了，但很多个这些小的代码重构就可以最后引起质的飞跃。

目前支持重构的工具很多，比如JBuilder、eclipse等开发工具都引入了重构功能，在本文中，以目前比较流行的Java IDE工具JBuilder 9.0为例来介绍怎样利用重构来改善现有程序。

重构在JBuilder下的运用

JBuilder目前只支持5种常见的重构功能，如迁移重构（Move refactoring）、改名重构Rename refactoring、引入变量（Introduce variable）、方法拆分（Extract method）、改参重构（Change parameters），也许在将来的版本中会引入更多的功能。

一 迁移重构(Move refactoring)

我们知道Java的class文件都是以包的方式存在的，但这里也有一个问题，就是在项目开发过程中你发现各个class文件的组织方式很混乱，需要重新整理。

比如这里有一个Account类，存放在com.demo.bank包下，但后来发现这个Account应该是属于com.demo.company包的，所以需要整理代码。

```
//Account.Java
package com.demo.bank;

public class Account {
...
}

//Test.Java
package com.demo.bank;
public class Test {
    Account account  = new Account();
}
```

典型的传统实现方法就是首先修改Account.Java文件中的包名，并另存到同新包名对应的目录下；接着你需要查找所有应用到Account类的Java文件，比如上面的Test.Java文件，然后需要修改其中用到Account类的代码。如果有成百上千个文件引用到Account类，那修改工作量是十分可观的。

接着我们看看使用JBuider中的重构功能来实现我们的目的。

1. 你需要编译你的整个工程为最新版本，通过Rebuild可以实现；

2. 打开Account.Java文件，鼠标右键点击public

class Account {中的"Account",在弹出菜单中选择Move Class "Account";

3.弹出窗口如下:

修改 To Package 为 com.demo.company, View references before refactoring选项的意思就是是否允许在重构前查看引用到这个类的所有文件,在这里我们选择它,点击OK按钮;

4. 在JBuilder的底部会出现一个Refactoring面板,里面以树形结构列出了所有引用到这个类的地方,点击相应的引用可以定位到相应的引用行;

5.点击面板下方的按钮,整个项目的代码就被重构了,点击按钮可以撤销刚才的重构操作。

再看看原来的两个Java文件代码:

```
//Account.Java
package com.demo. company;

public class Account {
...
}

//Test.Java
package com.demo.bank;
import com.demo.company.*;
public class Test {
Account account = new Account();
}
```

上面代码中Account类的包名已经被修改为了com.demo.company,并且Account.Java文件也被移到了目录com/demo/company下,而为了保持整个项目同重构前一致,Test.Java文件中自动新增了一个import com.demo.company.*语句,使得它可以同原来一样直接引用 Account类。

几步简单的操作就可以完成原来较大工作量的繁杂杂劳动,重构的优势不言而喻。

二 改名重构(Rename refactoring)

改名重构常常用于我们在开发的过程中修改变量名、包名、类名、接口名、方法名等,但它不同于我们常用的查找替换功能,而是更加智能化更加方便。比如现在项目中有10个Java文件在com.demo.bank包下,通过使用改名重构,所有Java文件中的包名都会替换成com.demo.company, 同时这10个文件会自动移动到新的com/demo/company目录下。

假设现在有两个Java文件

```
//Account.Java
package com.demo.company;

public class Account {
  String Account = "";
}

//Test.Java
package com.demo.bank;
import com.demo.company.*;

public class Test {
   Account account = new Account();
}
```

上面代码中多处出现了Account,有的地方代表类名,有的地方代表变量名,我们的目的是修改所有的类名Account,对于变量account不能修改,显然我们只能查找后一个一个的替换,每次替换前需要确认一下。

看看JBuilder的改名重构吧:

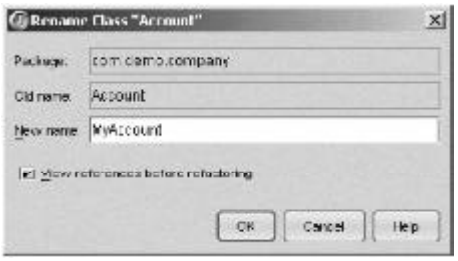
1. Rebuild一下你的整个工程,确保是最新编译的版本;

2. 鼠标右键点击Account.Java文件中的Account类名, 在弹出菜单中选择 Rename class "Account";

3.弹出窗口如下页左上图所示:

在 New name处输入新的类名 MyAccount, 选取 View references before refactoring, 点击 OK按钮;

4.点击面板下方的按钮,整个项目的代码就被



重构了，点击 按钮可以撤销刚才的重构操作。

重构后的代码如下：

```
//MyAccount.Java
package com.demo.company;

public class MyAccount {
  String Account = "";
}

//Test.Java
package com.demo.bank;
import com.demo.company.*;

public class Test {
   MyAccount account = new MyAccount();
}
```

可以看到所有的类名 Account都已经被替换为了MyAccount，而变量名保持不变，并且在文件目录下Account.Java 文件已经不存在了，而是 MyAccount.Java，这一切都是JBuilder自动完成的。

其它的修改方法名、修改字段名、修改包名的操作都是类似的，在这里就不一一介绍了，大家可以自己试试看。

三 引入变量(Introduce variable)

引入变量的功能在于使用一个变量替换原有重复的表达式，从而提高程序的可维护性同性能。

这里引入一个假想计算个人所得税的例子，一般个人所得税根据应纳税额度的不同使用不同的税率来进行计算，逻辑很简单就不介绍了，请看代码：

```
public float GetTax(int cash){
    if(cash - 1200< 0)
      {
       return 0;
      }
    else if(cash - 1200 < 2000 )
      {
       return cash * 0.1f;
      }
    else if(cash - 1200< 5000)
     {
      return cash * 0.15f;
     }
   else{
      return cash * 0.2f;
    }
  }
```

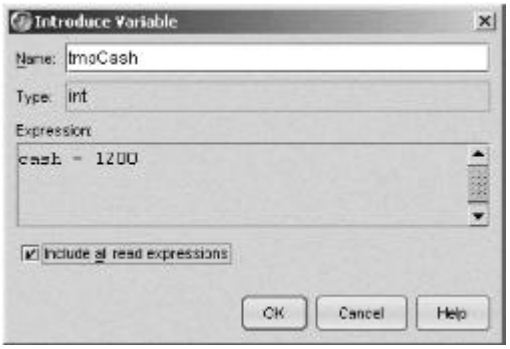
可以看出，程序多处引用了cash-1200，这样的代码有两个弊病，第一，不方便维护，如果现在需求1200变为了1600，修改代码就需要找到这几处代码进行替换；第二，效率较低，因为程序每次运行到cash-1200时，都需要重新计算一遍。加入我们引入一个变量tmpCash=cash-1200，后面所有的cash-1200都替换为tmpCash，问题就解决了。

使用JBuilder提供的引入变量重构方法如下：

1.Rebuild一下你的整个工程，确保是最新编译的版本；

2.在JBuilder中用鼠标选取代码行cash - 1200；

3.点击鼠标右键，弹出菜单中选择Introduce Variable，弹出窗口如下：



4.输入变量名tmpCash,，因为我们目的是替换所有的代码行，所以选择Include all read expressions，点击OK按钮。

5.重构完成。

重构后的代码如下：

```
public float GetTax(int cash){
  int tmpCash = cash - 1200;
  if( tmpCash < 0){
    return 0;
  }else if( tmpCash < 2000 ){
    return cash * 0.1f;
  }else if( tmpCash < 5000){
    return cash * 0.15f;
  }else{
    return cash * 0.2f;
  }
}
```

可以看到重构后引入了一个新的本地变量tmpCash,赋值为cash—1200,以前的重复代码都被替换为了新的变量 tmpCash。

四 方法提取(Extract method)

也许你目前面对的是挺难维护的一个老系统,因为其中一个方法可能就有几千行代码,需要重新提取方法,JBuilder中的方法拆分重构功能就可以实现这样的需要。在重构的时候,JBuilder会自动检测代码块中的代码并生成新的方法,新方法中有经过分析后相应的参数、返回值等。

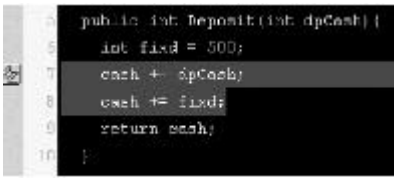
重构前的代码如下:

```
//Account.Java
package com.demo.company;

public class Account {
  private int cash;
  public int Deposit(int dpCash){
    int fixd = 500;
    cash += dpCash;
    cash += fixd;
    return cash;
   }
  }
```

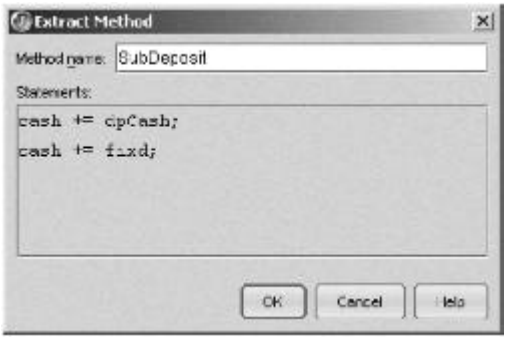
使用JBuilder重构过程如下:

1.Rebuild一下你的整个工程,确保是最新编译的版本;

2.在JBuilder中用鼠标选取代码行



3.点击鼠标右键,弹出菜单中选择Extract Method,



弹出窗口如左下图所示:

4.输入方法名 SubDeposit, 点击OK按钮;

5.重构完成。

看看重构后的代码:

```
//Account.Java
package com.demo.company;

public class Account {
  private int cash;
  public int Deposit(int dpCash){
    int fixd = 500;
   SubDeposit(dpCash, fixd);
    return cash;
  }

  private void SubDeposit(int dpCash, int fixd) {
    cash += dpCash;
    cash += fixd;
  }
}
```

可以看到重构后多了一个方法 private void SubDeposit(int dpCash, int fixd) ,以前的 Deposit 方法中的那两行代码已经被移到新方法中,取而代之的是一个调用 SubDeposit(dpCash, fixd)。

五 改参重构(Change parameters)

在开发过程中我们常常碰到这样的问题,就是在开发一个新模块的时候调用原来的某个方法,结果发现需要新增几个参数,怎么办呢,逐一给所有调用这个方法的代码行添加一个参数。聪明的人当然选择重构了,JBuilder中的改参重构功能提供了新增方法参数同修改原来参数顺序的功能。

为了简单起见,这里只引入一个Java文件

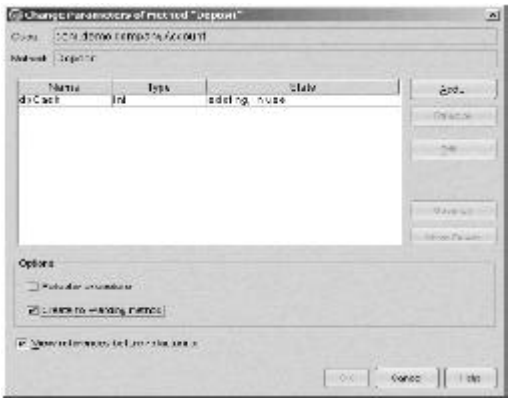
```
//Account
package com.demo.company;

public class Account {
  private int cash = 0;
  public int Deposit(int dpCash){
    cash += dpCash;
    return cash;
  }

  public static void main(String[] args) {
    Account account1 = new Account();
    account1.Deposit(1000);
  }
```

现在的需求是需要传入一个boolean的标志位,根据

这个标志位决定cash是否增加,看看JBuilder的功能吧。

1.右键点击Deposit方法名,弹出菜单中选择Change Parameters for "Deposit";

2.弹出窗口如下图所示:



最上面是class同方法的名字,中间部分就是方法参数的列表,包括参数名、参数类型、参数状态,通过右边的几个按钮我们就可以实现参数的修改以及顺序调整功能。

下方有两个选项,Refactor ancestors代表是否重构当前类的祖先类(包括父类、父类的父类……),如果选择它而且当前方法是一个重载(Overriding)方法的话,则所有的祖先类也会被重构。Create forwarding method选项代表是否创建一个前转方法,在这里我们选择它。

选取View references before refactoring,点击Add按钮;

3.弹出新窗口如下:

这里选择参数的Type为boolean,Name为aBoolean,Default value为false,点击OK按钮;

4.点击步骤2窗口中的OK按钮;

5.点击面板下方的按钮,整个项目的代码就被重构了,点击按钮可以撤销刚才的重构操作。

重构后的代码如下:

```
//Account.Java
package com.demo.company;

public class Account {
  private int cash;
  public int Deposit(int dpCash, boolean aBoolean){
    cash += dpCash;
    return cash;
  }

 /**
   * Forwarding method
   * @param dpCash
   * @return
   */
  public int Deposit(int dpCash) {
    return Deposit(dpCash, false);
  }
  public static void main(String[] args) {
    Account account1 = new Account();
    account1.Deposit(1000, false);
  }
```

对比重构前的代码,可以看到现在新增加了一个Deposit(int dpCash, boolean aBoolean)方法;而原来的方法Deposit(int dpCash)成了一个前转方法,而它只是简单地调用新的Deposit方法,新参数使用我们设置的默认值false;最后main中调用Deposit方法的地方也自动新增了一个参数,变为了account1.Deposit(1000, false)。

结束语

通过前面的例子,相信大家都对重构有了一个初步的了解,运用重构可以把一个普通的设计重构为一个完美的设计。实际运用中重构常与单元测试工具如JUnit一起运用,以保证重构后的代码不会影响原来程序的正确性。

想深入了解重构的朋友可以看一看Martin Fowler写的这本关于重构的经典著作《Refactoring: Improving the Design of Existing Code》,里面深入阐述了重构的原则,重构的过程以及众多的重构方法,所幸的是目前电力出版社也出版了中文版《重构——改善既有代码的设计》以及原书的影印版本。

作者简介

本文作者陈华,华中科技大学控制系硕士,就职于深圳黎明网络教育中心,从事软件开发方面教学培训研究工作。

2004 Java官方开发工具总览

▶撰文 / 王森

关于作者：

王森

Sun Microsystems教育训练中心技术顾问/讲师。Run PC专栏作家，CSDN专栏作家。

擅长：

- 语言：C/C++、Java、C#等程序语言之应用
- 技术：J2ME(Java 2 Micro Edition)、J2SE(Java 2 Standard Edition)、Palm OS程序设计、Windows CE程序设计、Symbian OS程序设计。

著作：

- 深入浅出KJava/2001年3月(繁体)
- Java手机程序设计入门/2001年9月(繁体)

前言

过去几年，Sun Microsystems最让同业“津津乐道”的事情，莫过于Sun运用其优越的技术能力研发了Java技术，可是却从来没有能从Java身上获取庞大的利益。最具市场价值的企业级应用程序服务器(Application Server)，分别由Bea System的WebLogic与IBM的WebSphere独大；而开发工具的市场，则以Borland的JBuilder为大宗。曾经有朋友这样问笔者：“Sun在Java这项技术上，到底靠什么赚钱呀？”，其实，Sun在Java这个技术上，除了赚取微薄的技术授权费，或者协助其它厂商做技术咨询之外，还真的从未在Java技术上有过什么丰厚的营收。或许这是一家过去皆以硬件为基础的公司都会有的毛病 - 认为软件只是硬件的附属品。现在，Sun要开始全力以“Java元祖”的身分，告诉大家，不管是企业级的服务器或是开发工具，只有Sun所出品的，才是正宗的Java产品。

Sun在Java的品牌策略

Sun将旗下的所有软件产品，区分成Java与Java System两大类。(见上表)

品牌	网址
Java	Java开发者入口网站 http://java.sun.com/ Java应用入口网站 http://www.java.com (有繁体中文) Java开发社群入口网站 http://www.java.net
Java System	http://wwws.sun.com/software/javasystem/index.html

其中，Java这个品牌，所代表的是具有公开业界规范(JCP, Java Community Process, http://www.jcp.org)以及开放原始码的免费开发工具与产品。而Java System所代表的是符合企业需求，具有效能以及稳定性的付费开发工具与产品。

Java品牌

在Java的品牌之下，区分成J2EE(Java 2 Enterprise Edition)、J2SE(Java 2 Standard Edition)、J2ME(Java 2 Micro Edition)、Java Card四种平台。

J2SE(http://java.sun.com/j2se/index.jsp)是所有Java技术的基础，细部区分成Core Java (http://java.sun.com/j2se/corejava/index.jsp)与Desktop Java (http://java.sun.com/j2se/desktopjava/index.jsp)。

不管开发人员要开发任何平台的Java应用程序，都必须先从J2SE的Core Java学起，因为Core Java所代表的是Java技术的核心知识。当开发者对Core Java所包含的技术有所认识之后，接下来如果要开发桌面平台的应用程序，就可以学习Desktop Java技术。Desktop Java包括底下几种技术：

技术属性	技术名称
组件技术	JavaBeans
GU程序开发	AWT(Abstract Window Toolkit)

	JFC(Java Foundation Class, 包括 Swing、Java 2D、Accessibility、Internationalization)
多媒体	Java Sound API JMF(Java Media Framework) Java 3D
其它	JAI(Java Advance Imaging) Java Speech API Java Help System

在J2SE部分，Sun提供免费的开发工具Java 2 SDK，此工具目前提供Windows 32-bits/64-bits、Java Desktop System(Linux) 32-bits/64-bits、Solaris Sparc 32-bits/64-bits、Solaris x86 32-bits八种平台的版本。

> 注意
> 其它平台也会提供自己的Java 2 SDK，例如IBM的AIX、HP的HP-UX、MacOS X、FreeBSD等，都会有非官方提供，但是却兼容于标准Java 2 SDK的Java 2 SDK。

由于Java 2 SDK提供的是命令行环境下的开发工具，对大多数的开发者来说，使用难度太高。因此，需要免费高级开发工具的开发者，可以采用NetBeans(http://www.netbeans.org)这套开放原始码的高级开发工具，目前提供Windows、Solaris x86、Solaris Sparc、OpenVMS、MacOS X、Java Desktop System(Linux)六种平台的版本。

> 注意
> NetBeans使用纯Java撰写而成。上述所列的平台，NetBeans官方网站都提供各平台的安装程序。但是并不是只有上述平台才能执行NetBeans。基本上只要该平台提供与标准Java 2 SDK兼容的Java 2 SDK，就可以执行NetBeans。

Sun官方网站也提供NetBeans与Java 2 SDK合并在一起的版本。以方便开发者。

J2EE(http://java.sun.com/j2ee/index.jsp)技术着重于企业端(服务端)的应用，所包含的技术包括：

技术属性	技术名称
网络服务	JAXP(Java API for XML Processing);JAXR(Java API for XML Registries);JAX-RPC(Java API for XML-based RPC);SAAJ(SOAP with Attachments API for Java)
展示层	Servlet;Java Server Pages;Java Server Faces
企业逻辑层	Enterprise JavaBeans
管理	J2EE Deployment;J2EE Management;J2EE Client Provisioning;J2EE Authorization;Contract for Containers
其它	Java Message ServiceJ2EE Connector Architecture

Sun官方提供的标准J2EE开发工具为J2EE SDK，官方提供Windows、Java Desktop System(Linux)、Solaris Sparc三种版本。

如果您需要开发符合J2EE 1.3规格的J2EE应用程序，那么您可以使用J2EE 1.3 SDK，或是下载Sun的企业级应用程序服务器Java System Application Server 7 Platform版。如果您需要开发符合J2EE 1.4规范的J2EE应用程序，那么您可以使用J2EE 1.4 SDK。过去J2EE SDK与Java System Application Server是两套独立的产品，造成Sun需要花费两倍的人力来维护，从J2EE 1.4开始，J2EE 1.4 SDK指的就是Java System Application Server 8 Platform版。如果您对效能和稳定度的需求更高，可以另外下载需要收费的Java System Application Server 8 Standard/Enterprise版。

J2ME(http://java.sun.com/j2me/index.jsp)主要针对嵌入式系统，包括手机、PDA等。J2ME细分成CDC(Connected Device Configuration)与CLDC(Connected Limited Device Configuration)两种。CLDC针对CPU较弱、内存较少、电力资源较少的硬件装置而设计。CDC则使用在硬件条件较佳的装置上。Sun针对J2ME的每项技术都提供简朴的SDK，针对手机应用程序的开发，则提供免费的J2ME Wireless Toolkit(http://java.sun.com/products/j2mewtoolkit/index.html)，J2ME Wireless Toolkit与其它标准SDK都可以在Windows、Java Desktop System(Linux)、Solaris Sparc三种平台下执行。

> 注意
> J2ME Wireless Toolkit提供繁体中文版。

Java Card(http://java.sun.com/products/javacard/index.jsp)技术用来开发可以植入在IC智能卡之中的应用程序(Card Applet)。Sun官方的开发工具只提供简单的开发工具与仿真工具,如果要开发真正的Card Applet,开发者最好购买IC卡制造商所提供的商业开发套件。

NetBeans

NetBeans(http://www.netbeans.org)是一套完全以Java撰写而成、并且开放原代码的开发工具。就算NetBeans官方网站不提供某个操作系统的NetBeans安装程序,只要在操作系统中有Java 2 SDK(不管官方版本的或是其它厂商的兼容版本),就能够执行NetBeans。NetBeans还提供超过11国语言的版本(有简体中文版,没有繁体中文版)。(如图1)

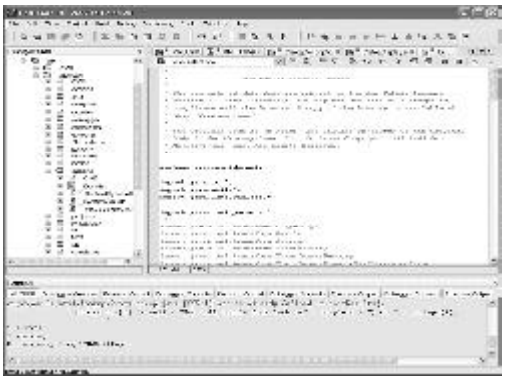


图 1

对大多数来说,NetBeans是一套集合开发环境(IDE),但实际上NetBeans分成两大部份,分别是NetBeans Platform与NetBeans IDE。NetBeans Platform是一套函数库,里面提供了各种开发IDE所需要的底层基础建设,NetBeans的开发团队都是先开发NetBeans Platform,然后才根据这个基础建设来发展NetBeans IDE。所以我们可以说,NetBeans Platform是一个用来开发整合开发环境的平台,任何人都可以使用这个基础建设来发展自己的整合开发环境,而一般程序设计师所使用的NetBeans IDE,只是根据NetBeans Platform所开发出来的一个集成开发环境实况而已。(如图2)

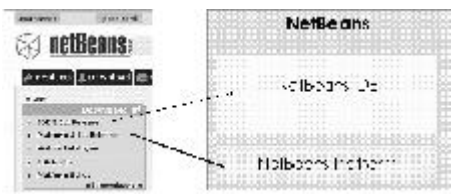


图 2

有许多不习惯使用NetBeans的人质疑为何Sun选择NetBeans作为官方Java标准开发工具的基础,而不是其它厂牌的开发工具。其实Sun的研发团队在选择官方开发工具的时候,花了很多心思研究到底哪一个开发工具最能展现Java的本质,只有NetBeans赢得了Visual Java 2 SDK(可视化的Java 2 SDK)的美名,**这是因为NetBeans的开发完全是以Java的观点来看世界,而其它开发工具还是留有Visual Basic、Delphi等开发工具所遗留下来的操作观念。**因此选用NetBeans有好有坏,坏处是过去习惯于Visual Basic、Delphi等开发工具的人,很难容入NetBeans所造出来的开发世界,也因而比起Borland JBuilder,NetBeans的普及程度稍为弱了些。但是好处是,NetBeans强化了"Java是一个平台"、"Java本身就是一个操作系统"的观念。当开发者看到NetBeans左上方的Explorer窗格,再看到"Mount"这个动词时,有没有更深刻的体会"Java是一个全新开发平台"这样的想法呢?(如图3)

图 3

对NetBeans有兴趣的朋友,请参考O'Reilly所出版的NetBeans: The Definitive Guide一书。

Java System品牌

在Java System的品牌之下,商用的J2SE平台叫做Java Desktop System。而商用J2EE平台则称作Java Enterprise System。J2ME的商用对应产品是Java Mobility System。Java Card的商用对应产品则命名为Java Card System。在撰写本文时,只有Java Desktop System、Java Enterprise System有具体的产品。

其余两者都还在酝酿之中。这些商用Java平台的官方开发工具，统称Java Studio，Java Studio针对J2SE、J2EE、J2ME都有对应的版本。整个Java System的架构如图4所示。

图 4

接下来，我们分别针对现有的Java Desktop System、Java Enterprise System以及Java Studio做讨论。

Java Desktop System

Java Desktop System(http://wwws.sun.com/software/javadesktopsystem/index.html)是一套以Suse Linux与Gnome桌面系统为基础的操作系统。其主要目的是要取代Windows，成为企业里稳定、低价的操作系统。(如图5)

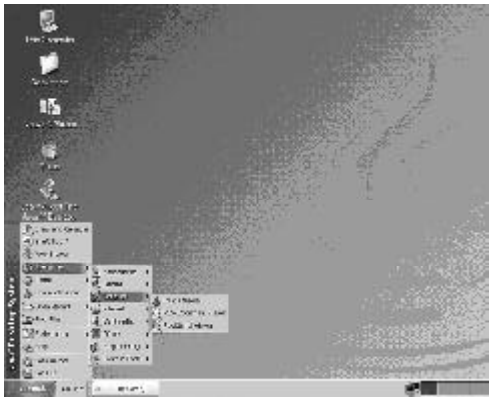


图 5

在Windows上有的杀手级软件，在Java Desktop System上面一样有。包括内建Java 2 SDK，让终端使用者不用再另外安装Java执行环境就能执行Java应用程序；内建Mozilla浏览器，让习惯使用IE浏览器的使用者可以享受速度更快的浏览器；内建StarSuite 7(也称作StartOffice)，提供一个MS Office的替代品。

Java Enterprise System

Sun的企业级软件产品，过去叫做iPlanet，后来再改名为Sun ONE(Sun Open Network Environment)，现在全面统一名称，在所有的软件产品前都加上Java System两个字。然而，这些产品之前有一些相依性，使得在过去要安装与使用这些软件，都必须有一定的基础知识。

现在，Java Enterprise System(http://wwws.sun.com/software/javaenterprisesystem/index.html)整合了过去Sun旗下所有的服务器产品，变成一个单一的软件产品。所有的企业级软件都有一致的安装接口，安装者只要顺着安装程序的指示，就能一次装好所有的软件，并设定好这些软件之间的相依关系。

Java Enterprise System十几种服务器软件，我们整理如下：

分类	软件名称
网络识别服务	Java System Directory Server (5.2) Java System Identity Server (6.1) Java System Directory Proxy Server (version 5.2) Java System Administration Server (version 5.2)
Web应用程序服务	Java System Application Server (7 update 1, PE and SE)Java System Web Server (version 6.1)Java System Message Queue (version 3.0.1 SP2)
整合与通讯服务	Java System Messaging Server (version 6.0)Java System Calendar Server (version 6.0)Java System Instant Messaging (version 6.1)
Portal服务	Java System Portal Server 6.2 (S1AS7 or Java System Web Server. WebLogic and WebSphere)Java System Portal Server, Secure Remote Access
集群服务	Sun Cluster Server (version 3.1.0) Sun Cluster Directory Agent

目前最新Java Enterprise System的版本为2004Q4版(也叫做Release 1)，只提供Solaris Sparc与Solaris

x86 两种版本。未来Release 2版将会提供Linux版，并整合Portal Server，Mobile Access这套产品进来。到了Release 3版，将加入Java System Application Server Enterprise Edition与Active Server Pages服务。以后的版本也将提供Windows平台的版本。欲了解未来的发展，请参考http://wwws.sun.com/software/javaenterprisesystem/roadmap.html

高级Java开发工具

Sun官方的Java开发工具，统称Java Studio (http://wwws.sun.com/software/javasystem/javastudio/index.html)。

这套开发工具换过很多名称，最早以前叫做Forte for Java，后来改名叫Sun ONE Studio，一直到现在终于统一名称叫做Java Studio。目前最新的版本为Java Studio 5，是直接从Sun ONE Studio 5而来，如果读者看到Sun ONE Studio 7与8这两个版本，请不要搞混了，Sun ONE Studio 7、8是用来开发C/C++应用程序的版本，与Java无关。往后与Java相关的官方开发工具只有一个名称，就是Java Studio。Sun官方分别提供Java Studio的Solaris/Linux/Windows三种版本。

目前，Java Studio共分成几种层级，分别是：Java Studio Micro Edition(提供Solaris Sparc、Linux、Windows三种版本)(http://wwws.sun.com/software/sundev/jde/studio_me/index.html)

Java Studio Standard Edition(提供Solaris Sparc、Linux、Windows三种版本)(http://wwws.sun.com/software/sundev/jde/index.html)

Java Studio Enterprise Edition(提供Solaris Sparc、Solaris x86、Windows三种版本)(http://wwws.sun.com/software/javastudioenterprise/index.html)

Java Studio Creator(提供Solaris Sparc、Linux、Windows三种版本)(http://wwws.sun.com/software/products/jscreator/)

四种版本。Java Studio Creator就是传说中的Project Rave。

Java Studio Micro Edition大致上是把NetBeans IDE和J2ME Wireless Toolkit结合在一起的产品。可以方便J2ME应用程序的开发者更容易追踪问题与调试。至于Standard、Enterprise、Creator三种版本，都能够用来开发J2SE与J2EE应用程序，这三者最大的差别就在于所针对的开发族群以及对于Sun企业级Java产品的支持程度。

Creator和Standard两种版本是同一种等级的产品，除了J2SE应用程序的开发之外，它们与NetBeans IDE的最大差别就在于J2EE应用程序开发的支持。Creator和Standard两种版本在安装后都会附带安装J2EE SDK (Java System Application Server)，并提供专门的Plug-in，让开发者可以直接在整合环境中进行开发、调试、集成、部署、测试这些工作。而Enterprise版所针对的目标更大，除了上述两种版本所拥有的功能之外，Enterprise版针对需要开发Java Enterprise System应用程序的开发人员所设计，所以不只涉及了J2EE应用程序的开发，还包括了如何开发Portal Server上的应用程序、集成Web Server、Directory Server、与Identity Server。所以三者所针对的开发范围如下图6所示。

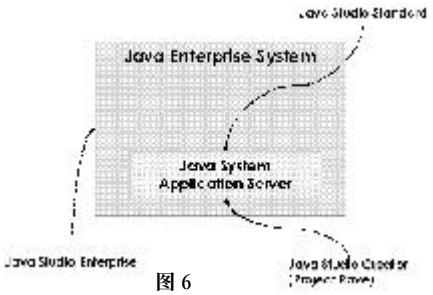


图 6

虽然Creator和Standard属于同一种等级，所能做到的功能类似，但是他们针对的开发者族群不同。Standard针对的是传统的Java开发者；Creator则是针对过去使用过Visual Basic或Delphi这类开发工具的使用者，冀望Creator可以降低Java程序开发的门槛，并提供一个更简单的开发环境。

我们把Java Studio的四种版本做个整理，如图7。

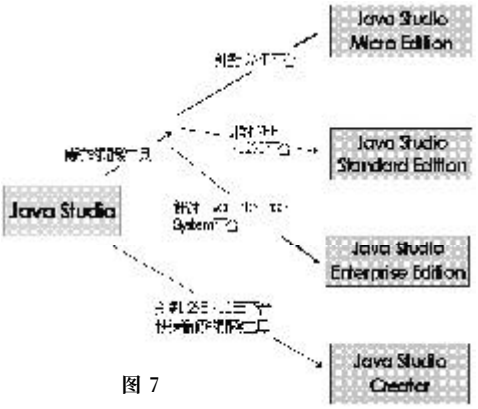


图 7

接下来我们针对Standard、Creator、与Enterprise三种版本做详细说明。

Java Studio Standard Edition

Java Studio Standard Edition根植于NetBeans IDE，并在J2EE应用程序的开发上与以加强，如图8。

图8

Java Studio Standard Edition在J2EE的加强方面，包括加入了可以与Java System Application Server整合的Plug-in，让J2EE开发者不用离开集成环境，就能够直接部署、调试与测试J2EE应用程序。如图9、10。

另外，Java Studio Standard Edition还加入了对Sun ONE Application Framework的支持。如右下图11。

Sun ONE Application Framework (http://wwws.sun.com/software/products/application_framework/home_app_framework.html)是一套企业级的J2EE应用程序框架，其目的是为了让系统开发者能够用更简单快速的方法融入J2EE Design Pattern于所开发的系统之中。因此Java Studio Standard Edition对于Sun ONE Application Framework所提供的可视化组件有非常良好的支持。对Sun ONE Application Framework有兴趣的朋友可以参考文章：

Introducing Sun ONE Application Framework in Sun Java Studio Standard(http://developers.sun.com/tools/javatools/articles/afintro.html)

Java Studio Creator

Java Studio Creator并非直接使用NetBeans IDE，而是采用NetBeans Platform为基础，所发展出来的一套全新集成环境。因此在外观上，不会像Java Studio Standard Edition一般与NetBeans IDE极为相似。其架构如下页图12、13、14所示。


图9

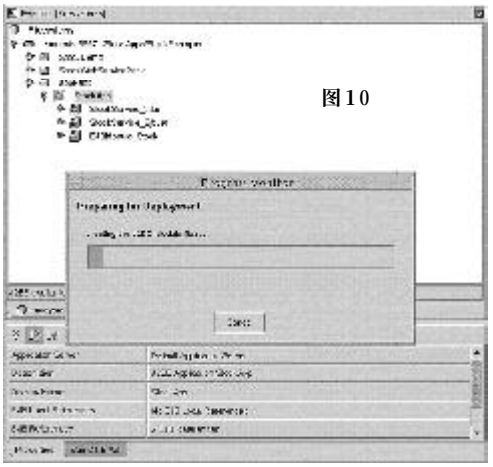


图10

图11

为了让过去习惯于Visual Basic、Delphi开发环境的开发者能够快速进入Java程序设计领域，Java Studio Creator除了加入了对J2EE SDK的集成之外，在Web应用程序的开发上，特别加强对JavaServer Faces(http://java.sun.com/j2ee/javaserverfaces/index.jsp)的支持，让开发者可以用随拖即放的方法来开发Web应用程序，如下页图15、16、17。

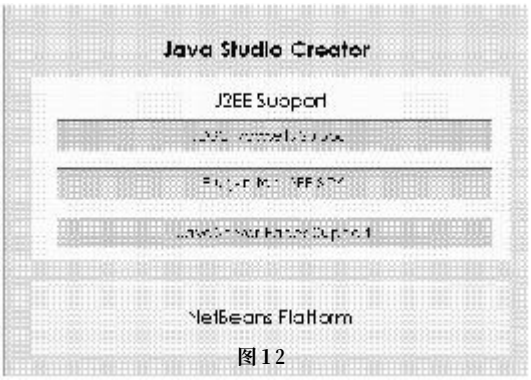


图12

图13

图14

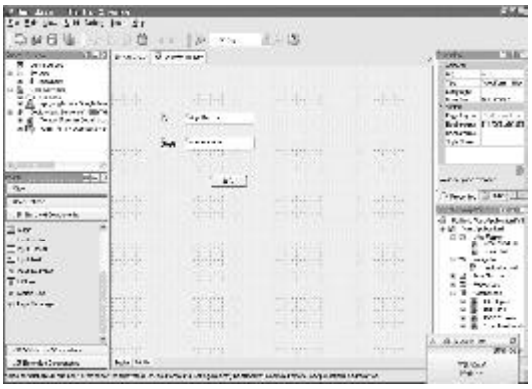


图15：拖放Web应用程序外观的开发画面


图16：根据所设计的画面，Java Studio Creator自己所产生的JSP原代码


图17：填写事件处理函数的开发画面

另外一个开发者最重视的问题，就是如何能够让应用程序与数据来源快速地结合，毕竟数据才是应用程序的核心，在.NET的世界里，存取数据最重要的技术就是ADO.NET，在Java的世界里，对等的技术为JDBC Rowsets (http://java.sun.com/products/jdbc/index.jsp)。Java Studio Creator加强了对JDBC Rowsets的支持。如下页图18、19。

Java Studio Creator将使得Java应用程序的开发迈向一个新的里程碑。

Java Studio Enterprise Edition

Java Studio Enterprise Edition根植于Java Studio Standard Edition，其目的是为协助开发者开发以Java Enterprise System为平台的应用程序。除了具有Java Studio Standard Edition所有的功能之外，Java Studio Enterprise Edition将Java System Connector

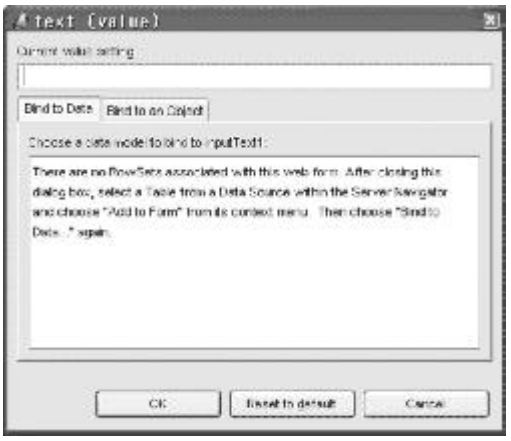


图18

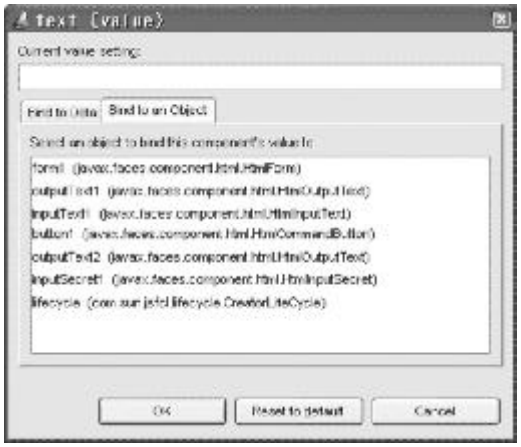


图19

Builder与Java System Portlet Builder两项开发工具集成进来，其架构如图20所示。

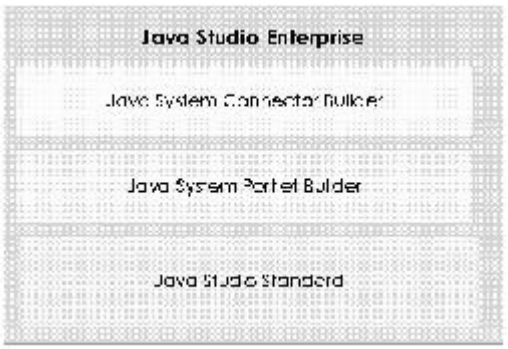


图20

在安装Java Studio Enterprise Edition时，除了既有的Java System Application Server之外，还会自动帮我们安装

Java System Directory Server

Java System Identity Server

Java System Web Server

Java System Portal Server

这几项服务器软件。Java Studio Enterprise Edition的安装程序提供我们一个集成的安装程序，开发者不必额外安装这些重要的服务器软件。如图21。

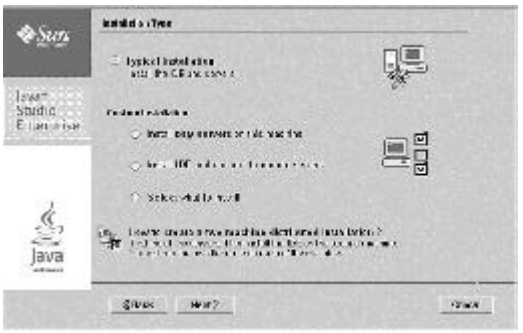


图21

Java System Connector Builder(http://wwws.sun.com/software/products/connector_builder/home_connector_builder.html)的主要功能是拿来建立以SOAP为基础、并符合J2EE Connector Architecture (J2EE CA)的Connector软件。而Java System Portlet Builder则用来开发符合JSR-168 Portlet规格之Portlet的开发工具。Java Studio Enterprise Edition可以让开发出来的Portlet轻易地部署到Java System Portal Server之中，并进行测试工作。

对Java Studio Enterprise Edition有兴趣的朋友可以参考文章:Sun Java Studio Enterprise(http://developers.sun.com/prodtech/javatools/reference/techart/jseintro.html)

结语

目前Sun对Java的既定政策之一，就是Java程序更容易开发(Easy of Development)。为了破除一般工程师普遍认为“Java进入门槛过高”、“Java只有高阶开发人员才能够使用”的迷思，Sun将在开发工具上更加努力，为了顾及已经习惯传统开发方式的使用者，Sun持续维护Java Studio Standard/Enterprise这条产品线；而针对过去习惯Visual Basic与Delphi这类开发工具的开发者，Sun硬是卯足全力开发Java Studio Creator。相信2004年开始，将是Java开发工具有重大突破的元年。请大家拭目以待。

大话工厂方法设计模式

▶撰文 / 徐锋

人 物

行者：一个普通的程序员，聪明好学。在超过2000小时的项目开发中，积累了一定的编程经验，水平逐渐提高，但他开始讨厌自己所写的代码，因为总觉得有“坏味道”，带着这样的问题，他找到了无名……

无名：从汇编到C，又从C到C++和JAVA，原本视不同“语种”之士为异类，经历数个春秋的感悟，终于领会“武之大成，飞花逐叶即可伤人”之精髓。近来，研习OO之法，颇有心得，对编程之悟又进一层。

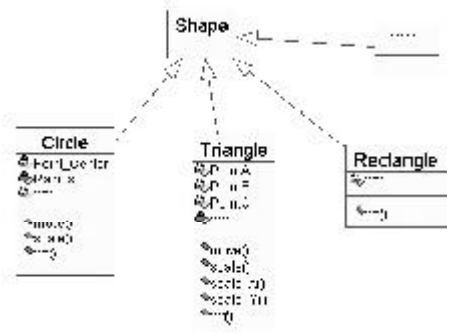
夜幕缓缓降临，窗外的车灯越来越亮，刚完成手头任务的无名伸了伸懒腰，收拾了一下桌面，准备离开办公室。而这时，他突然发现行者还端坐在自己的位子上，紧锁着双眉，全神贯注地望着屏幕。由于在前几次交流中，好学的行者给无名留下了很好的印象。因此，一向冷漠的无名也破例走向前，十要关切地问道：“年轻人，今天可是周末呀，怎么还这么有定力，是什么吸引了你？”

一看到无名，行者马上高兴起来，回答说：“噢，我遇到了一个难题，一直没有找到一个好的设计方案。”

“什么难题，能够将我们一向聪明伶俐的行者难倒，快说来听听，看看我能不能帮你。”，无名饶有兴趣地追问道。

“我准备自己动手实现一个 画图 程序。在这个程序中将提供一些常见图形工具，使用时可以像Rose一样，选中相应的图形工具后，就可以在画布上绘制出相应的形状，然后还可以对它进行缩放、移动等操作……”

“有什么问题吗？”，说着无名又露出了那招牌式的微笑。



图一 原始构思

“我本来的想法是这样的，就是将形状的显示、控制等操作都作为每个形状类的一个方法，而这些形状均继承于抽象类Shape”，行者边说，边拿过一张纸，画上了一个类的示意图。“但是，这样有一些问题，一是将显示与控制放在一个类里，我感觉违反了上次您在飞机上和我说起的SRP原则，二是抽象类Shape并不能够很好地定义，因为每种形状所拥有的方法有很大的区别，例如三角形的缩放可以是等比例缩放、沿X轴缩放、沿Y轴缩放，而圆形则只能够等比例缩放，否则不成圆形了，难道总提取共性的方法部分。”

“我有两点建议：一是控制部分不根据结果，而根据事件；二是运用工厂方法模式。”

“什么是工厂方法模式？”，行者听完后感到颇为迷茫，话音刚落马上又补了一句：“对了，你当时说过设计原则只是指导思想，而模式则是具体的解决方案。工厂方法模式就是其中一种吧！”

听到这里，无名露出了欣慰的笑容：“我说的你还都记得，真是难得！”

“对了，你说的不根据结果而根据事件是什么意思？”

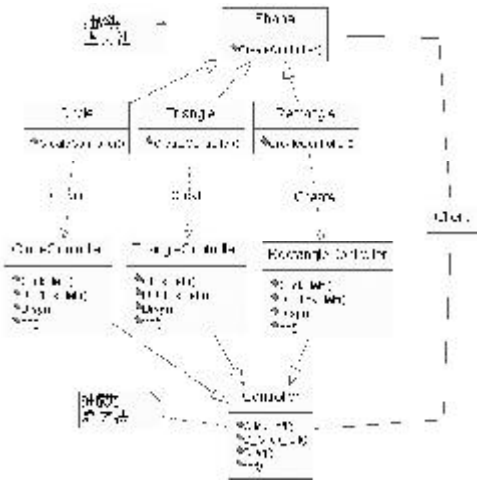
“其实，当你将图形绘制出来后，就可以对其进行控制，而控制的办法通常无外乎包括单击鼠标左键、单击鼠标右键、双击鼠标左键、按下左键移动……等等，你可以根据这个来编写控制部分的代码。”，无名解释到。

“嗯，这是一个不错的方法，那工厂方法模式是怎么回事？”

“你别这么急，看看这幅图”，说着无名就提笔在纸上画了一个新类图，并且在边上附了一个源代码片段。

```
public class Circle
{
    …… //定义成员变量
```

```
        public Controller CreateController()
    {
            new CircleController();
    }

    …… //定义其它成员方法
}
```



图二　修正后的类图

“你想一下，Triangle和Rectangle两个类的成员方法CreateController的代码应该如何写”，无名轻声地问到。行者想了想，也在边上写了几行源代码：

```
Triangle:
        new TriangleController();
    Rectangle:
        new RectangleController();
```

“嗯，完全正确，你能够参透其中的道理吗？”，无名追问。

“有点明白，在这个设计中，调用这些类的任何Client类，都可以创建Shape类型的对象，并且可以通过调用CreateController方法，将自动地创建出相应的Controller。”

“完全正确，无名补充说：“而且这样的设计，要添加一个新的形状，Client类无需做相应的改动，这也是符合OCP原则的好设计。”

“虽然我已经理解了这个设计，但我还是不明白什么是工厂方法模式，你能说说吗？”，行者的兴趣又起来了。

无名笑而不答，在纸张上写上一行代码：

```
    Rectangle R1=new Rectangle(5,8);
```

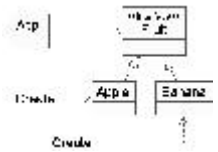
“这样的代码，你熟悉吗？应该十分常用吧！”

“当然，可以说是太经常了”，行者应声答到，“这有问题吗？”

“在通常情况，应该是没有什么问题的，但是如果Rectangle类的实现发生了变化的话，那么就会影响到使用这个类的其它类。而我记得曾经和你说过，在写程序时，应该优先依赖于抽象类，而避免依赖于具体类。那么这就存在一定的问题。要解决这个问题很简单，只需要将创建具体类的工作搬移出来。”

“那如何搬移呢？”，行者自言自语的嘀咕着，突然大声说到：“对了，你刚才的这个设计中已经将创建具体类的工作从Client类中搬移出来，当需要创建时，只需调用Shape类型的CreateCotroller方法就可以了！我知道了，就是使用工厂方法模式来完成，工厂方法模式就是用来转移具体类的创建工作的。”

“工厂方法模式还有两个兄弟，一个弟弟叫简单工厂模式，一个哥哥叫抽象工厂模式，它们统称工厂模式，一个比一个厉害。其中简单工厂模式最为简单、能力也最小，工厂方法模式则要比简单工厂要复杂一些，能力也强一些。而且它们之间还是一种演化关系呢？下面我就结合一个具体的场景，一步步地告诉你它们是如何应用和如何解决问题的。”无名一边说着，一边又画出了一个类图：“如果我们直接在具体的应用类中用new操作创建类，那就会呈现出这样的情况。”



图三　创建具体类的应用程序APP

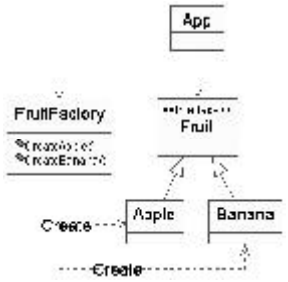
“从上面的图中，我们可以看出应用类App，还是与具体的Apple、Banana类发生了关系，也就产生了依赖关系，因此需要改进它。最简单的办法就是将创建具体类的工作转移到另一个类中，由于这个类的主要工作是制造对象，因此形象地称为工厂”，无名边说，边将原来的类图改成为了图四所示。

“也就是引入了个新类FruitFactory，其中包括两个创建具体类的方法CreateApple和CreateBanana……”

“我想这两个成员方法的代码是这样的吧”，行者打断无名的话，在纸上写出了：

```
    return new Apple();
    return new Banana();
```

“完全正确，但这样做还存在一些问题，如果我们需

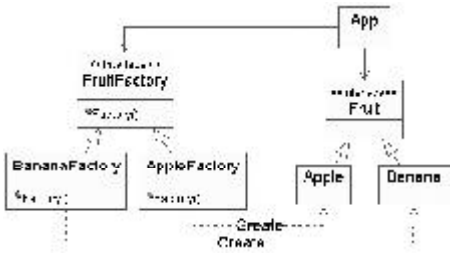


图四　采用简单工厂模式

要新增一个Orange类，那么虽然App类无需改动，但我们还是需要修改FruitFactory类，再加上CreateOrange才行”，无名又引出了一个新问题。

听到这里，行者十分高兴地说道：“哈哈，我知道了，用工厂方法模式来解决嘛，我来画设计类图。”

话音刚落，一幅工整的类图就跃然纸上：



图五　采用工厂方法模式

“是这样吧，我理解了”，行者略微显得有几分得意，但又马上问：“不过，我觉还有两个问题，一是这样不是会造成类数量增长，提高了复杂性吗；二是工厂方法模式在哪些场合应用更好呢？”

“我先回答你的第一个问题。使用工厂模式的确会带来复杂性，但它同时也使系统的可扩展性更好。因此我们需要有选择地根据场合来使用，其实可以总结成为以下两句话：对系统中所有易变类使用工厂模式（也就是从简单工厂模式开始考虑），如果引入新类的变化可能性大则需使用工厂方法模式。例如我们在使用String类时，就没有必要引入工厂模式，因为String是一个相对很稳定的类，无需多此一举。”

“嗯，我明白了。”

“而关于工厂方法模式通常应用于三种情况：第一种情况是用来将一系列类的创建工作从客户类中转移出来，降低客户类对这些类的依赖；第二种情况是用来构建两个相同层次结构的类体系，就像我们所举的例子；第三种情况则是常用于公共框架（Frameworks）的开发，因为这些框架处于一个抽象级别，其自身并无法知道将创建哪种对象，COM和EJB的支持框架都采用了工厂方法模式。说了这么多，你能否再举出一两个可以应用工厂模式的实例呢？”

“对了，上次那个需要支持Oracle和SQL Server的那个项目中，我可以使用工厂方法模式来重构代码。根据需要创建不同的连接、组建不同的SQL语句，而且如果需要再支持一种新的数据库，改起来也会觉得代码清晰得多，这个应用应该可以吧。”

“还有，我觉得在编写网络通讯方面的程序时，可以用工厂方法模式来动态创建各种的数据包，这样数据包格式变化时，修改起程序来就简单多了”，行者越想越高兴，突色脸色一变：“但有一点我没有想明白的是，工厂只能创建了它，但如何去填充包体，完成整个工作呢？每一种包体的填充方式可是不一样的呀。”

时间已经不早，无名无心恋战，和行者说：“我不和你多说了，饿坏了，要走了。不过呀，你的疑问正巧是另外一个模式要解决的问题，你可以先自己找找解决方法，找个空闲时间，我再和你细细说来。要不我的肚子可会不高兴的哟。”

“好，我也得去过周末了”，行者显得有些不好意思，快速地收拾了东西一起离身走向电梯口。突然，行者抛出一句：“对了，你还没有告诉我工厂方法模式的大哥抽象工厂模式有什么特殊的地方呢？”

“哈哈，来日方长嘛，下次一定给你全部补上。”说着说着，无名拦下一辆的士，很快消失在夜幕之中。

笔者注：

1）由于篇幅所限，文中所引用的例子均是在真实项目的基础上做了一些简化；

2）从本期开始，大话系列将开始讨论具体的模式，也希望有兴趣的读者与我交流，提供您宝贵的建议，联系信箱：tougao@csdn.net

关于作者：

徐锋，高级程序员、系统分析员、CSAI顾问、《csdn开发高手》专栏作家。自1992年开始就被电脑神奇的魅力深深吸引，一发不可收拾。从Apple II的汇编语言开始入门，一直就在计算机软件领域孜孜不倦、兢兢业业，不敢有一分懈怠。十余年来，涉猎面广泛，只叹自己仍博而不精。现致力于需求工程、系统分析与设计、过程改进等领域的研究与探索。

Java编写ACDSee类软件

▶撰文/周松奕

让我们开始吧！

这是一个和往常一样的早晨，却被一阵电话铃声吵醒。原来是某软件公司的哥们黄。电话里说有一个类似AcdSee的软件需要嵌入他现在所做项目的客户端中，因此需要我来帮忙。

“叮咚”门铃响了。

“这么快？”我开门，黄风风火火地闯进来。

“要赚钱就要快，这才是搞IT的速度。嘿嘿”黄往沙发上一坐，就把他的笔记本电脑拿出来。“我先大致说一下项目的思路：这个项目是某行业解决方案，涉及到一些图像预览处理的客户端，也就是克隆一个类似AcdSee的界面。但是，需要和我们的系统紧密结合。你只要帮我把这个框架搭起来，我回头让我的程序员把数据集成进去就可以了。不难吧？”黄看着我。

“不难，我帮你搭框架和主要功能。就做成类AcdSee的界面？用什么语言来实现？”

“是啊，和AcdSee界面大概差不多就可以，那上面花哨的界面就不要了。我们项目是用Java来做的，所以我希望这个也用Java来写。不过听说Java写界面不太好？”

“Java编写GUI，主要使用的是AWT类库，这个库用来开发比较复杂的界面时，确实显得有点力不从心。因此，Sun在J2SE中AWT类库的基础上，实现了Swing类库。使得用Java编写具有漂亮UI的软件不再是难事了。”我边说边把我的笔记本电脑打开了。

图1

“我们先来分析一下AcdSee的界面。”我打开了AcdSee（如图1所示）。

“可以看出整个AcdSee的界面其实是可以分成几个部分的，菜单、工具栏、资源浏览区域、图片预览区域、状态栏。”

“是的。其实这里的菜单、工具栏、状态栏都不难，最难的是资源浏览区域和图片预览区域。”黄问。

“没错。”我说：“这里面有三个难点，资源浏览区域，图片预览区域，还有就是多线程，也就是在图片预览的同时，还能够变动资源浏览区域，并选择其它目录。既然已经分析出界面的几个区域，那就可以大概设计出系统框架了。”

“是的，其实，按OO（面向对象）的概念，大概可以知道有几个界面类了。”黄说。

“好的，我们来画出大概的类图。”我拿出了一张A3的纸和笔，“从这个界面上可以得到以下几个类，菜单类：MenuBar，工具栏类：ToolsBar，状态栏类：StatusBar，资源浏览区域我们可以作为一个类：Explorer，图片预览区域，它是个容器，那么我们可以这么命名：ImageListPanel，因为在Java中，可以用JPanel作为容器。对了，我们这个小软件的名称应该叫什么？JcdSee吧，那我们多定义一个JcdSee的类，作为整个系统入口。OK！还有别的么？”我边问黄，边在纸上画着。

黄作出沉思状：“我想，这些类是差不多够了。但是图片预览区域是个容器，那么里面的图片缩略图就是一个一个的实体，是容器中的个体，那么我们增加一个类：ImageUnit。”

“另外我们可以通过定义事件来实现自动更新。当Explorer产生一个变化的时候，就触发一个事件，这个事件告诉容器，Explorer已经改变了，你也应该相应的改变。那么为了实现这个事件，需要定义一个事件类和一个接口：ExplorerEvent和ExplorerListener。这样，我们可以通过实现事件监听器来监听事件是否发生。”我边说边在图中加了一个类和一个接口。“至于多线程处理，是必要的，但是不需要另外做一个类，我们可以在JcdSee这

投稿信箱：tougao@csdn.net

个类中实现Java的Runnable接口。我想这个线程应该实现的一个主要功能是:在屏幕上显示这些缩略图。

“我们更深入分析一下每个类吧,从Explorer这棵树开始。”我看了看AcdSee的界面说:“它可以用Java的Swing提供的JTree来实现。这些节点是比较特殊的节点,因为它得处理一些事情,包括:判断从文件列表读取目录,和这个目录结构有变动的时候,需要进行相应的变动。我们需要为这个节点设计一个类:FileNode,让他继承DefaultMutableTreeNode。这样需要在我们的类图里加上。让这棵树列出系统的资源,首先得能够取得系统资源。这个可以用Java提供的FileSystemView类实现。它用来实现和操作系统相关的系统资源信息,包括根目录、文件类别信息、系统图标等。Explorer类,应该差不多就这些需要细化的了。”

“是的,那么接下来是容器了,好像容器没什么需要细化的,不过里面的缩略图类是不是需要细化?另外,我只需要实现JPG格式图片的浏览。对了,还有必须标明每个文件的名称。”

“对,缩略图可以分成两部分:显示区域和文件名区域。文件名区域其实就是个Label,这个容易实现。缩略图区域,我想就是一个显示图像的地方。同时再定义一个容器,ImageUnitPanel,这个容器中包含一个类ImagePanel,还有你要增加的其他东西。ImagePanel的功能就是用来实现图片预览。至于,只需要实现JPG格式图片的预览,这个可以通过过滤器控制,滤出JPG文件就可以了。这个过滤器,我们就命名为:PicFileFilter吧。”我在纸上画着。“另外在缩略图生成过程中,我们需要知道它的进度,可以用JProgressBar实现。进一步的类图,大概就是这样。”(如图2所示)

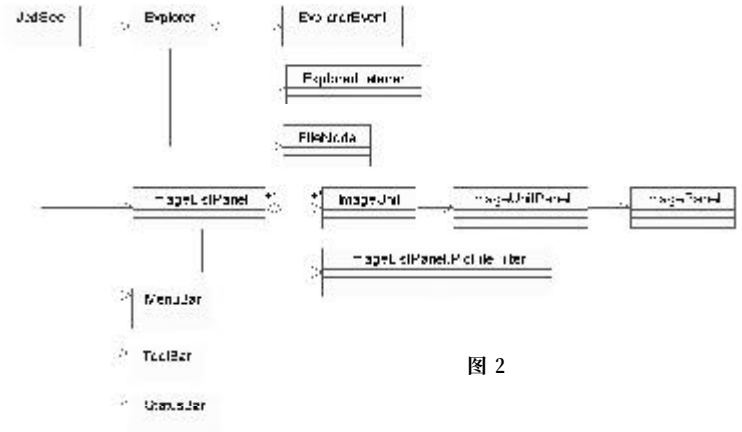


图 2

黄仔细看了几遍,说“差不多了,具体设计编码过程中,可能还会修改。毕竟开发都是迭代的。”

“我们还没有为每个类找到属性和方法呢!先把这个图画到Together里面去,让它把程序框架生成出来。你等一下。”我打开Together,开始在那里面画起了类图。

界面区域	类	描述
主界面	JcdSee	实现系统的入口
菜单	MenuBar	实现菜单栏
工具栏	ToolBar	实现工具栏
资源浏览区域	Explorer	实现资源浏览栏
资源树节点	FileNod	实现资源浏览栏中的文件节点
图片预览区域	ImageListPanel	实现缩略图的容器
进度条	JProgressBar	实现缩略图显示的进度
缩略图区域	ImageUnit	实现缩略图的显示的区域
缩略图区域中图形区域	ImagePanel	实现缩略图显示
状态栏	StatusBa	实现状态栏
	ExplorerEvent	实现资源浏览鼠标点击事件
	ExplorerListener	实现资源浏览鼠标点击事件的接口
	ImageListPanel .PicFileFilter	实现文件过滤,只要JPG文件

克隆原来如此简单!

“好了,画完了。”我在Together画完了类图。(如下页图3所示)

“我分了三个Package:JcdSee、Explorer、ImagePreview-Container。三个Package里分别包含了刚才分析出来的类。至于MenuBar和ToolBar还有StatusBar我没有做在这上面,因为那些功能比较简单,你可以以后自己加进去。”我喝了口茶。

“嗯,那几个类我可以让程序员加进去。现在我们开始为这些类找

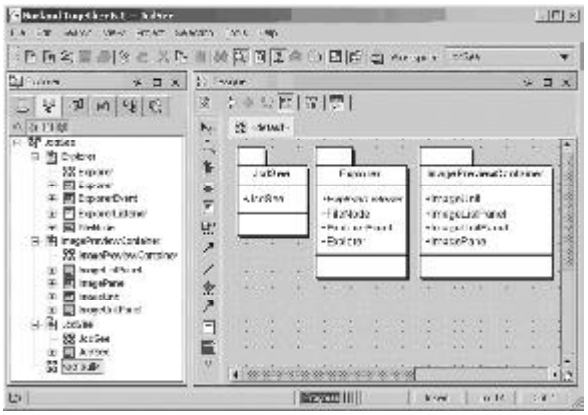


图 3

出属性和方法吧。”

“从Explorer这个Package开始吧,这里面包含了三个类,一个接口,包括:Explorer、FileNode、ExplorerEvent和ExplorerListener。我先整理一下,然后你看看有什么要补充的吧。”我说完就开始在Together里面画了起来。不到半小时,我就画好了这些类。(如图4所示)

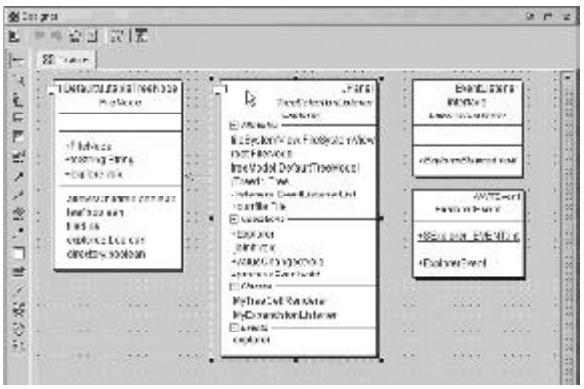


图 4

“我看看你画的,告诉我你的思路。”黄说。

“我从Explorer这个类说起。”我用鼠标点了一下Explorer的类图。“首先,Explorer是一个容器,我让它从JPanel继承。

```
Public class Explorer
    extends JPanel
    {
            ……
    }
```

“其次,需要列出的是系统资源,正如前面分析的,我们可以用Java的FileSystemView这个类来获取系统资源列表。所以我实例化了一个FileSystemView。

```
FileSystemView fileSystemView = FileSystemView.getFileSystemView();
```

“第三,Explorer这里面包含一棵树,必须有一个根节点root。我们前面分析了,这棵树的节点是比较特殊的文件目录节点,所以我定义了一个FileNode的类,这里的根节点,也就是FileNode类的一个实例。

“第四,这个根节点,必须是文件系统的根目录。可以通过FileSystemView的getRoots方法取得根节点。

```
FileNode root = new FileNode(fileSystemView.getRoots()[0]);
```

“第五,开始生成这棵树。先用DefaultTreeModel生成一个树的样板,这棵树的根节点是root,并且对于任何节点都可以有子节点的。然后按树的样板生成一个JTree的实例。

```
DefaultTreeModel treeModel = new DefaultTreeModel(root);
JTree jTree1 = new JTree(treeModel);
```

“第六,那么既然是棵树,则必然要去捕获树的变化。可以通过实现TreeSelectionListener接口捕获树中当前节点变化。该接口有一个方法要实现,也就是valueChanged。

```
public class Explorer
        extends JPanel
        implements TreeSelectionListener
        {
                public void valueChanged(TreeSelectionEvent e) {……}
        }
```

“第七,因为树的节点是我们自己写的类,因此我们必须处理树的展开和收回的事件,可以通过addTreeExpansionListener方法和实现TreeExpansionListener接口来实现,我用MyExpandsionListener类来实现该接口。

“第八,除此之外,还需要显示的节点图标是系统默认的图标,而不是JTree的图标,那就需要通过JTree类的setCellRenderer方法设置图标重画的DefaultTreeCellRenderer类。在这里,我用类MyTreeCellRenderer扩展了DefaultTreeCellRenderer,并重载了getTreeCellRendererComponent方法。同时通过FileSystemView的getSystemIcon方法取得。

```
public class Explorer
extends JPanel
implements TreeSelectionListener {
        private void jbInit() throws Exception {
                ……
                jTree1.addTreeExpansionListener(new
                        MyExpandsionListener());
```

```
                jTree1.setActionMap(null);
                jTree1.setCellRenderer(new MyTreeCellRenderer());
                jTree1.addTreeSelectionListener(this);
                ……
        }

class MyExpandsionListener
        implements TreeExpansionListener {
                public MyExpandsionListener() {}
                public void treeExpanded(TreeExpansionEvent event) {
                        ……
                }
                public void treeCollapsed(TreeExpansionEvent event) {
                ……
        }
}

  class MyTreeCellRenderer
        extends DefaultTreeCellRenderer {
        public MyTreeCellRenderer() {}
         public Component getTreeCellRendererComponent(JTree tree, Object
                        value,boolean sel,boolean expanded,
                        boolean leaf,int row,boolean hasFocus)
                {
                super.getTreeCellRendererComponent(tree, value, sel,
                        expanded, leaf, row,hasFocus);
                setIcon(fileSystemView.getSystemIcon( ( (FileNode)
                        value).getFile()));
                return this;
                }
        }
……
}
```

“最后，我为Explorer定义了一个事件和相应的事件监听器的接口，因此我们必须在Explorer类中增加一个EventListenerList，用来存储监听器列表。还有，我们在这个类中添加一个Event。这个Event包括两个方法：addExplorerListener和removeExplorerListener。有了Event，就必须有一个processEvent方法来处理事件。

“那么这个事件在什么情况下会触发呢？应该是在树中的当前节点发生改变的时候，触发这个事件。也就是，触发事件的代码应该放在valueChanged方法中。这就是我对于这个类的设计思路。”我说完，朝黄看了一眼。

```
public class Explorer
        extends JPanelimplements
        TreeSelectionListener {
        private EventListenerList listeners;
        ……
        public void valueChanged(TreeSelectionEvent e) {
                Object obj = jTree1.getLastSelectedPathComponent();
                if (obj == null) {
                        return;
                }
                else {
                        currfile = ( (FileNode) obj).getFile();
//触发ExplorerEvent事件 ExplorerEvent event = new ExplorerEvent(this);
                        EventQueue queue = Toolkit.getDefaultToolkit()
                                        .getSystemEventQueue();
                        queue.postEvent(event);
                }
        }
//增加一个事件监听器
        public void addExplorerListener(ExplorerListener listener) {
        listenerList.add(ExplorerListener.class, listener);
        }
//删除一个事件监听器
        public void removeExplorerListener(ExplorerListener listener) {
        listenerList.remove(ExplorerListener.class, listener);
        }
//事件处理程序
         public void processEvent(AWTEvent event) {
                if (event instanceof ExplorerEvent) {
                        EventListener[] listeners = listenerList
                                        .getListeners(ExplorerListener.class);
                        for (int i = 0; i < listeners.length; i++) {
                          ( (ExplorerListener) listeners[i]).
                            ExplorerElapsed((ExplorerEvent)event);
                        }
                }
                else {
                        super.processEvent(event);
                }
        }
        ……
}
```

“嗯，没有什么意见了。只是currfile的属性，是用来干嘛的？”黄指着屏幕问道。

“这个属性是用来存放树中当前选择的文件目录的。用来把当前文件夹传递给缩略图容器。剩下一个类和一个接口，也没什么好说的，这是Java中做事件驱动的一般方法。”我说，“我接下来把两个包的类都细化一下，你再来看吧。”

经过了将近一个小时的工作，我终于完成了两个包的细化。如图5所示。

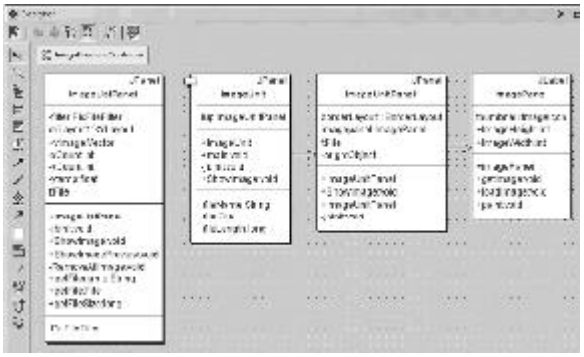


图5

“这里面关键点有三个：第一，要把缩略图读出来；第二，要把文件目录下所有图形的缩略图按照一定的排列顺序显示在桌面上；第三，要对文件名进行过滤。”我

边泡茶边说。“首先，读缩略图。在ImagePanel中实现，我是采用了Java提供的ImageIcon类，这个类是实现把图像提取出来显示为图标功能的。但是这个类有个不好的地方就是，只支持JPG和GIF格式。如果以后不够用，可以继续扩展这个类，使之支持更多的图形甚至图像格式。”

```
public class ImagePanel extends JLabel {
……
ImageIcon thumbnail = null;
public void loadImage(File f) {
……
thumbnail = new ImageIcon(
tmpIcon.getImage().getScaledInstance(iwidth,iheight, Image.SCALE_FAST ));
……
}
public void paint(Graphics g) {
    if(thumbnail != null) {
        int x = getWidth()/2 - thumbnail.getIconWidth() /2;
        int y = getHeight()/2 - thumbnail.getIconHeight() /2;
        ……
        thumbnail.paintIcon(this, g, x, y);
    }
}
```

“其次，要把文件目录下所有图形的缩略图按照一定的排列顺序显示在桌面上。这里我们需要注意几点：第一、我们需要定义一个集合来装这些缩略图。我这里选用的是Vector（向量）；第二、由于图形读取时间较长，因此我们需要把ImageUnit的显示和图形的显示分开，先把ImageUnit按照一定的排列顺序在容器中显示出来，然后再一个一个地去显示图形缩略图。所以我定义了两个方法：ShowImage 和 ShowImagePreview。ShowImage负责生成ImageUnit集合，然后按照排列顺序排在容器中；ShowImagePreview则负责显示缩略图。”

```
public class ImageListPanel
extends JPanel {
        //用来装所有的图形文件的缩略图
        public Vector vImage = new Vector();
        ……
        //ShowImage负责生成ImageUnit集合，然后按照排列顺序
        //排在容器中，而不显示缩略图
        public void ShowImage(File f) {
                ……
        }
        //ShowImagePreview负责显示缩略图
        public void ShowImagePreview(int i) {
                ( (ImageUnit) vImage.elementAt(i)).ShowImage();
                this.updateUI();
        }
        ……
}
```

“再次，对文件名进行过滤，因为要求只显示JPG格式文件，那么必须对文件名过滤，把不是JPG的文件过滤掉。可以实现FileFilter这个接口。

```
class PicFileFilter implements FileFilter {
        public boolean accept(File f) {
        return (f.isDirectory() || f.getName().endsWith(".jpg") ||
                f.getName().endsWith(".JPG") || f.getName().
                endsWith(".jpeg"));
        }
        public String getDescription() {
       return ("Jpeg 文件(*.jpg) 选择文件夹打开可以加入文件
              夹下的所有图形文件");
        }
}
```

“ImagePreviewContain包基本上就这样实现了。还有什么有问的地方？”我问黄。

“有一个疑问，因为显示缩略图占用时间长，需要用到多线程，但是你这里面没有有关于线程的地方啊？”黄不解的问。

“确实需要用到多线程，但是多线程不能在这里实现，而应该在JcdSee类中。”我打开JcdSee包的类图。如图6所示。

图 6

“这个包里面，我们应该注意几个要点：第一、必须实现多线程，并注意多线程的同步，同时，要实现线程的控制，包括有可能出现在图形显示的过程中会切换到另外一个目录去，这时候要能够控制住线程；第二、要捕获Explorer的产生的事件；第三、要注意界面显示内容有可能超出界面的大小范围，因此必须有滚动条；最后，要实现Explorer和ImageListPanel之间的分割栏，因为有时候需要改变它们的大小。

“第一、在Java中实现多线程只要实现Runnable接口就可以了，Runnable接口有一个run方法。在run方法中实现缩略图的显示。同步问题，我通过定义一个互斥量来实现。

```
public class JcdSee extends JPanel implements Runnable{
        Thread thread = new Thread(this);
```

```
        Object threadlock = new Object();
        //定义一些变量实现对线程的控制
        boolean stopflag, continueflag, showflag, threadflag;
        public JcdSee() {
                //开始线程
                thread.start();
                ……
        }
        ……
        public void run() {
                ……
                while (threadflag) {
                        //将threadlock锁住,同步线程
                        synchronized (threadlock) {
                                ……
                        }
                        ……
                }
        }
}
```

“第二、通过Explorer类的addExplorerListener方法把监听器类注册到事件监听器列表中。这样,当Explorer触发ExplorerEvent事件的时候,就可以捕获,并处理。

```
public class JcdSee extends JPanel implements Runnable{
        Explorer se = new Explorer();
        private void jbInit() throws Exception {
                ExplorerAction listener = new ExplorerAction();
                se.addExplorerListener(listener);
                ……
        }
        private class ExplorerAction
        implements ExplorerListener{
                public void ExplorerElapsed(ExplorerEvent event) {
                        ……
                }
        }
        ……
}
```

“通过JScrollPane类可以实现滚动条。另外,Explorer和ImageListPanel之间的分割栏则可以通过JSplitPane实现,这些工作就交给你的程序员去完成吧。”

大功告成!

现在整个系统的框架已经搭好了,实现一些处理细节,就比较容易了。

……

“Yeah!终于完成了。”我看了看表,花了两个小时。“在Swing中,引入了一个UIManager类。这个类中有个方法setLookAndFeel,这个方法就是用来设置界面的表现形式的。UIManager类中还有一个方法getSystemLookAndFeelClassName,用来取得当前运行系统的界面表现形式的。使用了这个方法后,软件在不同的操作系统中,就会表现为和该系统相匹配的界面了。”

```
public static void main(String[] args) {
        try {
        //取得当前的系统UI
        UIManager.setLookAndFeel(UIManager
                .getSystemLookAndFeelClassName());
        Font font = new Font("宋体", Font.PLAIN, 12);
        String names[] = {"Label", "CheckBox", "PopupMenu",
                        "TextPane","MenuItem",
                        "CheckBoxMenuItem",
                        "JRadioButtonMenuItem","ComboBox",
                        "Button", "Tree", "ScrollPane",
                        "TabbedPane","EditorPane", "TitledBorder",
                        "Menu", "TextArea", "OptionPane",
                        "MenuBar", "ToolBar", "ToggleButton",
                        "ToolTip", "ProgressBar","TableHeader",
                        "Panel", "List", "ColorChooser",
                        "PasswordField","TextField", "Table",
                        "Label", "Viewport",
                        "RadioButtonMenuItem","RadioButton"};
                for (int i = 0; i < names.length; i++) {
                        UIManager.put(names[i] + ".font", font);
                }
                UIManager.put("Label.foreground", Color.black);
                UIManager.put("Border.foreground", Color.black);
                UIManager.put("TitledBorder.titleColor", Color.black);
                ……
        }catch (Exception e) {
                ……
        }
}
```

“你看怎样?”(如图7所示)

图7

“好极了!这就是我想要的!太棒了!”黄兴奋得直搓手。

“好了,交工了,你拿回去有什么问题再来找我吧,这个应该很容易扩展的。”我说。

“OK!先走了,bye”黄复制完文件,急匆匆地走了。

妙用中间表

▶撰文 / 唐荣辉

公元2000年夏天，我所在的公司承担了一个这样的任务：开发某项费用的代扣代缴系统。我负责此项任务，下面给大家讲讲这个故事，希望对各位有所帮助。

因为涉及到商业秘密，所以我不能向大家公开这个系统。可以将客户方看成某电信局，银行则是该市信用联社。当然描述中涉及到的东西不一定和电信局、银行相符，但是可以作为这个项目的示范。

我们要做的系统运行于电信局内，主要业务是这样：该系统要在数天内完成全市固定电话用户（一个发达的东南沿海地级市，试想有多少固定电话用户）电话费缴纳任务。他们预先在帐户内存入电话费，系统从电信局系统中获取代扣话费数额，并把扣款信息发送到信用联社；再由信用联社把话费从电话用户户头拨到电信局户头。业务看起来似乎简单，但是面临的难点却是数据量大、且需要集中在几天内完成，还不能出差错。因为无论是电信局还是电话用户都不允许出现错误的情况。

我和信用联社信息中心的人开始了合作，在2个月的时间里，大家的脑细胞阵亡了不少，总算达到了预计目标，得到电信局大老爷的认可。公司拿到了钱，我们拿到了奖金，项目圆满结束了，但项目中发生的一些事情还是有值得回味的地方。

我们开发的系统以及银行方面都采用多线程开发，否则没有办法完成任务（也许有高手会说把所有数据打包一个文本发送给信用联社，联社再把扣款结果打包一股脑发送回来，这样不是又快又好？这样确实挺好，但要声明一点：我们的实际商业客户并非电信局。这里只是打个比方，我们的客户有其它理由要采用这样实时的方式运作）。双方约定保证各自的系统最少能承受10个线程，也就是说我这里有可能同时发送10条扣款信息给信用联社，联社的系统则必须能够同时处理这些信息，并且准确无误。我们这边后台是Windows 2000 Advanced Server＋MS SQL Server 7.0；联社后台是Solaris下的Sybase。双方通过128K点对点专线相连，并基于TCP/IP的ASCII字节流交互（关于如何开发这种通讯程序和本文没有多大关系，所以不能赘言，大家可以到网上搜索一把，相信有很多的文章保证让你看个不亦乐乎）。按照理想的情况，每扣一笔款，时间应该少于2秒，对我们系统处理时花费的时间要求尤其很高，约在200毫秒左右（数据事先由其他程序采集，运行时只要Select扣款数据，组合成报文发送给联社，当然还要记录扣款流水。至于回填扣款结果到电信局系统中，则有其它程序实现），关键的消耗几乎都在联社边。熟悉内情的人都知道，银行系统很复杂，规矩也很多，不绕那么几下是不会轻易完成操作的。

很快，这个多线程系统就完成了，而且初步联调完成的小数据量测试中，没有发现任何问题。两方面都准确无误地完成了操作，并且时间控制得也非常理想。既然这么顺利，接下来的工作就是提高数据量了。先采用5个线程进行测试，没有发现问题。当数据量达进一步提高到采用10个线程时，等待时间就越来越长，有时一笔扣款甚至要十几秒之多。完了，这个样子在几天内绝对完不成扣款任务，那不是要受电信局的白眼？

联社的弟兄们很着急，但光上火也没有用呀！于是邀请我们去他们那里吃免费午餐，顺便参考解决这个问题的方法。俗话说：三个臭皮匠顶一个诸葛亮。我们一堆臭皮匠，可以顶好几个，司马徽说伏龙凤雏得一可安天下，这么多诸葛亮难道还有办不成的事？

紧张的调试马上就开始了。先跟踪，看问题出在什么地方，这也是遇到问题程序员所做的第一件事。四处加调试语句（这里没有什么秘密，无非就是记录时间或者程序运行标志到调试日志文件）。不一会就有发现，问题的症结在Oracle后台执行上。线程处理几乎都堵在数据库的位置，时间都浪费在等待上了。代码没有问题，人家的数据库可是金字招牌，你要说有问题，人家Oracle肯定跟你急。那看来是数据库设计有问题，导致执行不动了。

问题基本上找到了，马上分析原因。联社同仁有这么一张表放置所有帐户的余额，为了保护商业秘密和简单起见，下面只列出了这个表的简化版本，真正银行的系统可远比这个严谨，需要很多校验一致的条件才可能

进行转帐操作,否则大家的钱谁放心存到银行?

表名:ACCOUNTINFO

列 名	类 型	备 注
ACCOUNT_NO	VARCHAR(32)	帐号, 主键
ACCOUNT_NAME	VARCHAR(80)	户名
ACCOUNT_BALANCE	FLOAT	帐户余额

两个帐户之间发生金额交易一般是这么操作(假定钱从帐户A到帐户B):先检查A是否足额,如果不足额,则不扣款,如果足额,则起一个事务,减少A帐户的金额,把减少的金额加到B帐户上,然后提交事务完成扣款。假设电话用户帐号为"1234",本月代扣电话费为100,电信局帐号为"9999",那么SQL中的指令基本如下(判断帐户足额的SQL代码没有在这段代码中实现):

```
BEGIN TRAN TRA    --启动事务
UPDATE ACCOUNTINFO   --扣钱
SET ACCOUNT_BALANCE= ACCOUNT_BALANCE-100
WHERE ACCOUNT_NO='1234'
--影响行不为1,说明发生了错误,表明需要回滚事务
IF @@ROWCOUNT<>1
  BEGIN
    ROLLBACK TRAN TRA
    RETURN
  END
--加钱到电信局帐户
UPDATE ACCOUNTINFO
SET ACCOUNT_BALANCE= ACCOUNT_BALANCE+100
WHERE ACCOUNT_NO='9999'
--影响行不为1,说明发生了错误,表明需要回滚事务
IF @@ROWCOUNT<>1
  BEGIN
    ROLLBACK TRAN TRA
    RETURN
  END
--提交事务,扣款完成
COMMIT TRAN TRA
```

看起来似乎没有什么问题了,而且这套机制也一直运行得很不错。银行方面每天业务量也很大,全部采用同样的模式,为什么这次却行不通?看来还得继续分析。

经过进一步跟踪发现,几乎大部分等待的时间都浪费在把金额转移到电信局帐户这一步上。问题找到了,最终的问题是多线程竞争对电信局帐户的操作。大家知道,一般我们操作数据库,都采用READ COMMITTED模式,即人家没有提交的数据我们是不去读的,只有得到确认的数据才会操作。现在大家都要对电信局帐户进行增加金额的操作,在A线程提交事务前,其他线程为了得到确认数据,不得不等待A线程完成操作。这好比一条双向十车道的高速公路,车开得挺快,结果经过一条小河时,河上只有一座仅容一辆车通过的小桥,大家只得接受交警的指挥慢慢通过小桥。这里的小桥就是电信局帐户,线程快车本来是并行,到它这里只能串行。

问题找到了,就要解决,于是臭皮匠开始了讨论。最终还是我更臭一点,想出来一个办法,不采用他们的现行模式,加一个中间表就搞定了,下面是中间表结构:

表名:ACCOUNTTRANSFER

列 名	类 型	备 注
ACCOUNT_NO	VARCHAR(32)	帐号, 主键
TRANSFER_MONEY	FLOAT	转帐金额
TRANSFER_TIME	DATETIME	转帐日期, 默认 SYSDATE()
HANDLE_FLAG	CHAR(1)	处理标志,未处理为'0',正在处理为'1',处理好了为'9',默认'0'

我设想的是:扣款操作不立即完成转帐,假如现在操作电话用户甲,他需要缴纳100元电话费,而系统检测到他帐户上有200元,足额,于是可以扣款。这时启动一个事务,先插入一条扣款金额信息插入到中间表,然后把甲帐户扣去100元,提交事务。大家一看就知道,钱还没有到电信局户头,而且帐也不平了,存款发生了丢失。没有关系,我另外使用一个监视线程,监视中间表的变动情况。这样看来,事务的处理并不是实时的,但由于时间很短,也就近似于实时了。我在中间表里加了一个处理标志,如果处理过,该标志为一位字符,插入数据时为0,正在处理为1,处理好了为9(以后还可以再定义其他标志含义)。监视线程启动一个事务,先执行一条SQL,把标志为0的全部打上正在处理的标志1(也可以针对时间段操作);然后,用一条简单SQL求和语句,获取所有标志为1的金额总和;最后对电信局帐户进行存钱操作。存入的金额当然就是前面中间表中转出金额的综合,存好后,再将中间表里标志1的记录变成9,提交事务,操作完成。

下面是扣款对应的SQL查询语句(假设电话用户帐号为"1234",本月代扣电话费为100,电信局帐号为

“9999”，参见图1）：

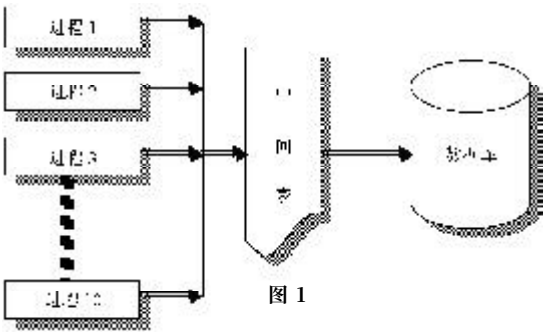


图 1

```
BEGIN TRAN TRA    --启动事务
UPDATE ACCOUNTINFO  --扣钱
SET ACCOUNT_BALANCE= ACCOUNT_BALANCE-100
WHERE ACCOUNT_NO='1234'
--影响行不为1,说明发生了错误,表明需要回滚事务
IF @@ROWCOUNT<>1
  BEGIN
    ROLLBACK TRAN TRA
    RETURN
  END
--插入扣款记录到中间表
INSERT INTO ACCOUNTTRANSFER
(ACCOUNT_NO,TRANSFER_MONEY,TRANSFER_TIME)
VALUES('1234',100,GETDATE())
--影响行不为1,说明发生了错误,表明需要回滚事务
IF @@ROWCOUNT<>1
  BEGIN
    ROLLBACK TRAN TRA
    RETURN
  END
--提交事务,插入扣款记录完成
COMMIT TRAN TRA

下面是监视线程的处理SQL:
DECLARE @TOTALMONEY FLOAT
BEGIN TRAN TRA    --启动事务
UPDATE ACCOUNTTRANSFER    --扣钱
SET HANDLE_FLAG='1'
WHERE HANDLE_FLAG='0'
--没有待处理记录,回滚事务,退出
IF @@ROWCOUNT<1
  BEGIN
    ROLLBACK TRAN TRA
    RETURN
  END
--获取总金额
SELECT @TOTALMONEY=SUM(TRANSFER_MONEY)
FROM ACCOUNTTRANSFER
WHERE HANDLE_FLAG='1'
--把钱存到电信局帐户
UPDATE ACCOUNTINFO
SET ACCOUNT_BALANCE= ACCOUNT_BALANCE+@TOTALMONEY
WHERE ACCOUNT_NO='9999'
--影响行不为1,说明发生了错误,表明需要回滚事务
IF @@ROWCOUNT<>1
  BEGIN
    ROLLBACK TRAN TRA
    RETURN
  END
--打完成标志
UPDATE ACCOUNTTRANSFER
SET HANDLE_FLAG='9'
WHERE HANDLE_FLAG='1'
--没有更新到标记,不正确,回滚事务,退出
IF @@ROWCOUNT<1
  BEGIN
    ROLLBACK TRAN TRA
    RETURN
  END
--提交事务,插入扣款记录完成
COMMIT TRAN TRA
```

联社同行马上行动，修改数据库、主程序，然后联调。这次果然没有再出现问题，而且处理速度和单线程一样快。电信局及时收到钱，谁也没有损失。

总结：资源竞争是多线程系统要解决的一个关键问题。当资源有限时，诸多进程如果不加爱惜相互竞争资源操作的全力，毫无疑问将导致所有进程的“温饱问题”都得不到解决。只有想出合理的办法，降低对资源的使用和竞争，你省一口，他省一口，大家才能共同吃饱吃好。任何现象的出现一定有它的原因，我们的责任就是找到它，不要退缩，同时根据以往的经验，开动脑筋，总是可以解决。自己实在不行，还是那句话：三个臭皮匠，顶一个诸葛亮，多找几个人，集思广益，虚心请教，问题也终将得到解决。，年深月久，你也可以成为牛人，到时候到处享受免费午餐和丰盛的大餐，衷心祝福所有的程序员日精月进！

小知识　临 界 区

临界区是保证在某一个时间只有一个线程可以访问数据的方法。使用它的过程中，需要给各个线程提供一个共享的临界区对象，无论哪个线程占有临界区对象，都可以访问受到保护的数据，这时候其它的线程需要等待，直到该线程释放临界区对象为止，临界区被释放后，另外的线程可以强占这个临界区，以便访问共享的数据。临界区对应着一个对象，当线程需要访问保护数据时，调用临界区对象的Lock()成员函数；当对保护数据的操作完成之后，调用临界区对象的Unlock()成员函数释放对临界区对象的拥有权，以使另一个线程可以夺取临界区对象并访问受保护的数据。当若干进程在等待同一个资源的使用时，就造成了多线程对资源的竞争。中间表就是可以用来解决多线程竞争资源问题的一种方法。利用各线程对资源进行同一种操作的特性，使用一个中间表来记录各个线程对资源需要做的访问，然后再一次性执行所有的操作，这样一来，问题就得以解决。

JDeveloper10g：数据库里长出的咖啡豆

▶撰文 / 郑涛

卡布奇诺苦涩中蕴含着淡淡的奶香，爱尔兰咖啡浓郁的香里飘散着威士忌的甘醇，每一种咖啡都有其独有的芳香。看一下世界顶级数据库厂商Oracle给我们准备的是怎样一种咖啡？这就是Oracle JDeveloper10g，让我们细细品味这杯来自Oracle的咖啡吧。

在2003年秋天，Oracle伴随Database10g的发布给我们带来了他的Oracle JDeveloper10g，一个端到端(end to end)的J2EE开发环境，支持建模、开发、调试和发布J2EE应用以及Web Services。JDeveloper系列是一个曾经获得'最佳Java数据库开发工具'奖的Java IDE，它在一个开发环境中集成了所有设计、开发、测试、调试、性能调整、发布、版本控制等等开发J2EE和Web Services应用所必需的功能，甚至同时还是一个完整的SQL和PL/SQL开发环境，可以使开发人员在一个开发环境下开发不同层次的应用。JDeveloper10g允许开发者自己从零开始创建一个J2EE应用；同时，也允许采用已有的J2EE框架来创建应用，无论何种方式，JDeveloper10g都提供了高效的工具简化应用开发的复杂性。这些工具覆盖了从UMI建模、可视化的编辑器、向导、对话框到代码编辑器等诸多方面。JDeveloper10g还提供了一个公共的扩展SDK（Extension SDK），用以扩展和定制开发环境以及无缝的集成其它外部产品。

面对J2EE应用开发日益增长的复杂性，Oracle的解决方案就是Oracle ADF(Application Development Framework)——Oracle应用开发框架。这一框架基于MVC（Model-View-Controller）模型-视图-控制器架构，可以使开发者精力集中在业务范畴而不是放在底层的支持技术上。通过可视化、向导式的编码技术，这一框架使得开发者不必是个J2EE专家就可以开发出高效的应用；另一方面，框架完全基于工业标准，因而用ADF框架开发的应用可以被发布到任何J2EE应用服务器，连接到任何SQL数据库。

除了ADF框架，JDeveloper10g还在J2EE和Web Services、IDE、建模和数据库开发等几个方面提供了吸引人的特性，下面我们将慢慢的逐一品尝。

ADF框架——Oracle的“银弹”

为了对付复杂性——这一软件开发中的“妖魔”，Oracle为我们提供了ADF框架这一“银弹”。在使用Oracle ADF框架开发应用之前，我们先看一下什么是ADF框架？

Oracle ADF是为了开发J2EE应用而提供的一个全面高效的框架，它通过把复杂应用的创建简化为创建一系列用于Web、无线应用（Wireless）、富客户端接口（Rich Client Interfaces）等等应用的业务服务(business services)来降低开发的复杂性。ADF框架提供了可用的J2EE设计模式实现（Design Pattern implementation）和元数据驱动组件(metadata-driven component)来加速开发过程，这些组件和模式实现都是不需要你再花时间编码、测试和调试，因为这些模式实现都是基于标准，这些组件特性中蕴含了Oracle自己的商用产品开发人员多年的开发经验。因此，你的基于ADF框架开发的应用将是一个高性能、结构良好和轻便的应用。

Oracle JDeveloper10g中内置了用于ADF框架的集成的可视化设计工具，可以在一个单一的环境中针对你的J2EE应用的各层建模、开发、测试、调试、性能调整、维护、发布以及版本控制。Oracle ADF提供了一个灵活的、端到端（end to end）的应用开发基础框架，在这个基础框架的每一个层面上你可以选择适合自己的实现技术。Oracle JDeveloper 10g则提供了一个贯穿整个开发周期的“拖拉”（drag and drop）操作开发方式。Oracle ADF框架可以帮助开发者借助可视化的工具，高效开发J2EE应用，同时允许开发者选择任何他们希望使用的技术，例如，ADF业务组件（Business Components）、Apache Struts和JSP等技术。

ADF框架是基于MVC模式架构基础之上的集成的应用开发框架，结构上分为：业务服务层、Model层、View层和Controller层（如下页图1）。

在业务服务层，ADF业务组件（Business Components）技术提供了可声明的“构建块”(building-blocks)，你可以用来构建遵循业务规则和处

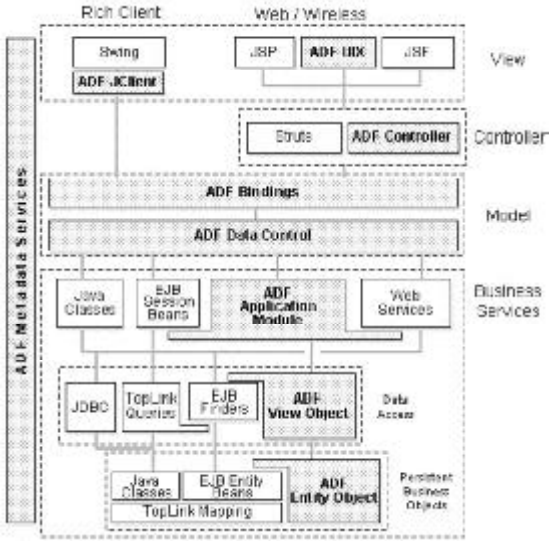


图1

理数据持久化的可升级业务服务组件、数据存取对象和业务对象等。

在Model层，ADF框架针对多种后台业务服务层技术提供了一致的、可声明的“数据绑定”（data-binding），来适应以ADF应用模块（ADF Application Modules）、JavaBean、EJB和Web Services等多种技术实现的业务服务。

在View层，ADF UIX技术提供了“声明式”的页面定义方式以及针对HTML和无线UI接口提供了一个“富”UI组件集，还提供了对JSP、JSTL（JSP标记库）、Jakarta Struts和ADF DataTags的支持。JClient和Swing的结合简化了创建“富”客户端用户接口的难度。

在Controller层，集成了对Jakarta Struts的支持，允许“数据绑定”（data-binding）和业务服务层与流行的Controller层框架无缝的集成在一起，同时，ADF Controller层提供了页面流程控制的另外一种选择。

Oracle ADF框架的所有层为我们的开发工作提供了多种选择方案，它们可以适应各种编程工作的需要，并且由XML元数据来配置。在你的应用开发中，你可以选择部分或者整个使用ADF框架。

相对于传统的开发框架，Oracle ADF由于是基于工业标准，所以不仅使应用开发更加轻松高效，而且更具灵活性和扩展性。针对model、view和controller模式结构，Oracle ADF提供了可插入技术（plug-able technologies），允许开发者在结构的不同层次选择不同的实现技术。

J2EE和Web Services——企业软件开发中的时尚

J2EE已经被证明是一个强健的、灵活的企业级应用开发和发布平台，已成为企业软件开发中的时尚。但是，当我们享用这一平台带来的轻便快捷和标准化等等好处的同时，J2EE开发的复杂性以及RAD开发工具对J2EE支持的缺乏却阻碍了许多IT企业采用J2EE平台。

现代的Java IDE都提供了许多提高开发效率的特性，但它们主要还是针对于代码驱动的开发者，例如更加“聪明”的编辑器可以帮助开发者自动完成代码等等。

小知识　在JDeveloper10g中使用HttpUnit

随着XP方法的流行，单元测试越来越被普遍采用，JUnit在Java编程中引入了单元测试技术，当然也包括运行在JUnit之上的针对某项具体技术的其他单元测试技术，例如：针对XML和HTTP的测试技术。下面这段文字简单介绍一下如何在JDeveloper10g中构建运行在JUnit之上的HttpUnit单元测试环境。

下载相应的软件

HttpUnit可以从httpunit.sourceforge.net站点下载，在我们的环境中用的是1.5.4版。下载解压后，你将会看到创建的目录中包含了一个Lib和一个Jar目录。同时，你还必须在你的系统中安装JUnit，在JDeveloper环境中安装JUnit Extension。以上这些安装完成后，将会创建一个名为JUnit Runtime的运行库，我们假设你的HttpUnit安装目录为HTTPUNIT_HOME。

创建必要的库

为了引用HttpUnit引擎，你需要在JDeveloper环境中创建一些必要的库。

1. 在JDeveloper环境中，选择菜单Tools | Manage Libraries；
2. 选择创建一个新的User Library；
3. 把这个新创建的库命名为：HttpUnit Runtime，之后我们将以这个名字来引用它；
4. 添加以下的Jar文件：
   - HTTPUNIT_HOME/httpunit-1.5.4/lib/httpunit.jar
   - HTTPUNIT_HOME/httpunit-1.5.4/jars/Tidy.jar
   - HTTPUNIT_HOME/httpunit-1.5.4/jars/js.jar
5. 单击OK。

到这里已经搭建好了HttpUnit单元测试环境，下一步就可以开始编写你的HttpUnit单元测试用例了，祝你成功！

投稿信箱：tougao@csdn.net

Oracle JDeveloper 10g支持最新的J2EE 和 Web Services标准,提供了易用的可视化工具帮助开发JSP、Servlet、Enterprise JavaBean和 Web Services, 而且,JDeveloper10g现在也支持最新的J2EE设计模式,例如Data Transfer Object 和Session Facade模式。特别是ADF框架对J2EE应用开发的支持更是全方位、多层次的,在大幅度降低开发难度的同时也给与你极大的选择余地和灵活性。

用于内置J2EE应用服务器的新管理工具可以帮助管理数据源、JAZN以及其他的J2EE应用设置。开发者甚至可以单击一下就可以把J2EE应用发布到Oracle Application Server、BEA WebLogic、 JBoss和Tomcat等应用服务器上,的确很方便对吧?

JDeveloper10g目前也对Web Services提供了很大的支持。以前要开发Web Services必须深入了解一些关键的技术(如SOAP封装的结构、SOAP标题元素等),现在通过JDeveloper10g提供的可视化工具和向导,一个没有任何Web Services技术(如SOAP或WSDL)背景知识的开发人员也可以在短时间内快速创建Web Services应用。此外, JDeveloper10g现在已可以处理WS-I Basic Profile兼容测试、Web Services的UML建模和可视化、单击方式从Java类到Web Services的创建,还可以创建复杂的PL/SQL Web Services等等。

JDeveloper10g还集成了对OracleAS TopLink持久化映射框架的支持,TopLink开发者现在完全可以直接在JDeveloper10g IDE环境中定制Java对象、Enterprise JavaBean到关系数据库的映射。

IDE —— JBuilder的孪生姐妹

众所周知, JDeveloper是Borland公司授权的, 它的内核来自于JBuilder, 是JBuilder的孪生姐妹。但是在此基础上JDeveloper10g为开发者提供了许多非常有用的、新的IDE特性,其中包括HotSwap调试、增强的代码编辑器、XML 模式编辑器(XML Schema Editor)、增强的Window资源管理以及内置的帮助系统等。

新的可视化编辑器可以帮助开发者使用HTML、JSP和ADF UIX来交互的设计Web页面,"所见即所得"的可视化编辑器与属性inspector、代码编辑器和结构窗口紧密集成在一起并保持同步,共同提供了一个全面的、高级的可视化编辑环境。

针对Jakarta Struts框架的新的可视化页面流程编辑器(Page Flow Editor)可以帮助开发者创建新的page(页面)和action(动作),并且可视化的连接它们来设计Web用户界面的流程。

增强的代码编辑器包含了快速存取Javadoc、语法高亮显示以及import语句帮助等等有用的功能。

在JDeveloper10g中,特别值得注意的是一个全新的XML Schema 可视化编辑器,这个编辑器显示了XML Schema文件的结构、内容和语义。XML Schema 编辑器与结构窗口、component palette和属性inspector紧密集成,以简单的"拖拉"操作方式支持XML Schema文件的创建和编辑;同时,这个可视化的XML Schema 编辑器与XSD (XML Schema Definition)定义代码完全同步,可以同时可视化的编辑和查看代码。

Oracle JDeveloper10g中的审计(Audit)特性帮助开发者静态分析代码以发现代码中可能存在的影响代码质量和违反编码标准的"瑕疵",这个特性提供了一个可以定制的审计定义文件以便你选择适合你的审计规则,同时伴随这个特性而来的一个框架可以允许你添加定制的审计规则。

建模和数据库开发—— Oracle得天独厚的优势

Oracle作为数据库的顶级厂商打造的JDeveloper10g在数据库开发方面更是具有特色,不仅仅可以管理所有的数据库对象,数据建模和生成数据库模式,测试查询,创建和调试PL/SQL语句,可视化管理数据库中的数据,而且还内置了功能强大的建模器。

新的数据库模式建模器可以帮助开发者以可视化图表的方式来捕获、创建和修改数据库模式。数据库对象的定义可以在项目中离线(offline)编辑,JDeveloper10g可以根据可视化图表或者离线的数据库对象定义创建SQL CREATE或者ALTER语句来创建新的数据库对象或者改变已有的数据库对象。

JDeveloper10g现在支持平台独立的UML建模器,可以创建UML类图、UML用例图、UML活动图来捕获需求和进行分析;同时,也支持平台相关的UML profile建模器来保持与具体实现的双向同步,这些具体实现可以是Java程序、EJB、ADF业务组件、WebService、数据库和页面流程等等。此外,还支持UML类模型和Java或者ADF业务组件之间的MDA类型的转换。

JDeveloper10g引入了数据库模式、用例和页面流程建模器。同时, 业务组件建模器现在也支持应用模块

（Application Module）和视图对象（View Object）的建模。此外，建模器还增加了包括图表的快速自动布局、小图标预览、以Javadoc的方式发布图表等功能，还包括了到其他图表、外部文件和URL的链接功能。

JDeveloper10g在数据库存储过程的开发、调试方面也提供了强大的支持，不仅可以创建、调试、发布PL/SQL存储过程，而且还可以创建、调试、发布Java存储过程。JDeveloper10g提供了可视化的界面来满足开发者对数据库开发的需求。

JDeveloper10g 先睹为快

看了以上这些吸引人的功能，是不是有点跃跃欲试了？你可以到Oracle站点(otn.oracle.com/software/products/jdev/index.html)免费注册下载JDeveloper 10g preview版（版本 9.0.5.0）。安装JDeveloper 10g的步骤非常简单：只需将下载的压缩包按原路径直接解压，解压缩后到相应的安装路径去找Jdevw.exe运行即可。启动后的JDeveloper10g开发环境如图2所示，是不是有点像JBuilder？

图 2

结束语

JDeveloper 10g继承了Borland开发工具的优良品质，同时又融入了针对数据库开发所独有的功能。特别是伴随JDeveloper 10g而来的ADF框架，它的可插入式技术带给你极大的技术选择余地。如果你要开发基于Oracle数据库的Java应用，那么这杯来自Oracle的咖啡将正合你的口味。

从《魔兽争霸III》看DataChunk的应用

▶撰文/吴亚西

魔兽争霸III是暴雪公司历时多年打造的一款精品PC游戏,凝聚了暴雪公司创作人员的智慧,本人作为一名游戏开发者,在互联网的帮助下窥得了魔兽III开发技术的一斑,即本文的主题:关于DataChunk的应用。如今,DataChunk已经成为我开发过程中不可缺少的一种工具,其原理简单却威力强大,在此我极力推荐给各位读者,同时也把我开发的一个DataChunk相关工具分享给大家。

DataChunk的说明

什么是DataChunk?DataChunk就是一个数据块,由数据块头部和实际数据组成,一般的格式是:

数据块ID 4Bytes + 数据块长度 4Bytes + 实际数据

DataChunk的应用

如今众多应用软件,比如3DS Max、PhotoShop以及Flash等等在进行文件存储的时候都用到了DataChunk,

我们若打开3DS Max SDK的工程sample,参照其中的代码,就可以看到Max的文件存储是怎么去使用DataChunk。同样,放在游戏开发中,魔兽争霸3也充分利用了DataChunk,笔者通过对魔兽争霸3标准动画文件.mdx的分析,进一步掌握了DataChunk的用法,并深深体会到了结构化数据存储的威力。

DataChunk的优势

向下兼容

DataChunk的头部中有ID,这个ID就是用来标识数据块类型的,DataChunk对应的读取程序可以根据这个ID进行不同处理,因为软件存在版本升级的问题和用户自定义数据的问题,一套DataChunk读取程序无法识别全部的DataChunk,又不能因为有无法识别的数据就放弃继续读取那些可以识别的数据,许多知名的大型应用软件都是尽可能从文件中提取出可以识别的数据,就拿笔者开发的魔兽争霸3 Mdx动画文件播器来说,还不能识别Mdx文件中所有的数据,不过因为Mdx文件遵循DataChunk的规则,忽略那些不能识别的数据,影响其他数据识别的可能性就小了很多。

可扩展性

由于DataChunk的存储独立性较高,可以做到与在文件中存储的位置无关,这样,在已有的DataChunk中添加新的子DataChunk后,原有的DataChunk读取程序只需要做小范围的修改就可以继续正常工作。简言之,增加一个数据块,只要遵守规则,几乎不会影响到原来的数据,甚至只需要在loading的时候把自己的的那部分代码加上去就行了。不需要关心原来的代码是如何的,这就是DataChunk最具魅力的地方。在这个前提之下,我们对数据的升级分为两部分,一个是定义新的DataChunk,然后再save和load的地方添加对自己的支持。

动态数据添加

我们在游戏开发过程中会遇到一个这样的问题,游戏设计还未定稿,数据结构有部分不能够完全确定,但是相关开发工具、文件格式的程序开发工作要同步进行,为了保证数据结构在一定程度上的可扩展性,在数据结构中预留数据字段的做法比较常见,比如:

```
struct Something
{
        ……
        int nUnknown[2];
        ……
        DWORD dwReserved[3];
}
```

但是当数据非常复杂,数据结构变动较大,这种预留数据段的方法显得有点力不从心,而且日后在实际更新数据结构和升级代码的时候,有可能会产生某种意义上的倒退。

我们还常常遇到这样的情况,当我们制作好地图编

辑器后,并且只做了一部分的地图,这时候因为在游戏设计初期没有考虑到某些因素,导致地图数据需要升级,如果此时我们的地图数据结构不支持升级特性,那我们就需要采用修改地图数据结构,重新制作的方案,或者采用扩展数据文件的方法,就算我们不考虑代码升级的成本,美术和策划进行新资源制作消耗的成本已经是很可观的了。况且,我们还不能保证日后不再次发生这样的事情,因为游戏开发的变化因素很多,这时候就会产生这样的需求:即在保留原有已经制作好的资源的情况下,可以把新的数据增加到原有的文件中。比如我原本有一张地图,后来这张地图扩展了一个数据块,就是每一个地图的tile需要保留一个32bit的描述,基于DataChunk,我们就只需要升级地图编辑器,让地图编辑器仍旧可以载入原来的文件,而新增的数据加入用新的输出程序输出一次就可以了。

降低数据依赖

DataChunk的另一个特性是降低了数据的依赖性,一个数据块的解释和读取可依赖于之前的一个数据块,也可以独立存在。比如骨骼动画中每一个骨骼的运动轨迹需要之前整个骨架的父子关系,而一个材质的描述则不需要知道骨骼动画的信息。再举例来说,魔兽争霸3的Mdx文件中存放了很多的数据对象,比如纹理、材质、网格、骨骼动画、粒子发射器以及碰撞盒等等,同时每个数据对象都可以保留自己的子数据对象,比如网格对象中有顶点、法线、面索引、纹理坐标、顶点颜色、骨骼组等,这么多的数据就需要一个科学合理的方法来组织。

DataChunk实现技术

TAG

TAG是一个宏,它把一个帮助辨认的字符组编码成一个DWORD。为什么要编码呢,我想是因为我们开发者都喜欢用UltraEditor之类的软件以二进制方式打开数据文件直接察看吧:

```
#define DC_TAG(x)
(DWORD)
(
 (((DWORD)x&0x0000ff00)<<8)
+(((DWORD)x&0x000000ff)<<24)
+(((DWORD)x&0x00ff0000)>>8)
+(((DWORD)x&0xff000000)>>24)
)
```

当我们把一个TAG输出到文件中,比如DC_TAG('test'),我们在文件相应地方就可以看见'test'这个字符串,有了这个可以阅读的字符串,对程序员日后分析和调试就会带来很大的帮助。

DataChunk自动生成器是用来帮助用户生成DataChunk的工具,它的特点是可读性强、辅助纠错、自动计算数据块长度等,用户首先声明一个CDataChunkWriter类实例:CDataChunkWriter w(1024*1024),Data Chunk Writer的工作方式是让用户写入的数据不直接写文件,而是先写入内存缓冲,然后再写文件,所以我们需要先分配足够大的内存空间。接下来我们分别用基本的方法、手动DataChunk和DataChunk自动生成器来示范性输出一个网格体。

一个网格体由顶点、面索引、贴图坐标构成:

```
class CMesh
{
        union UPointer
        {
                void*     p;
                char*     c;
                int*      i;
                DWORD*    dw;
                float*    f;
                short*    st;
                D3DXVECTOR3* v;

                UPointer(void* in) :p(in)
                {
                }
        };

protected:
        int m_nVertCount;
        D3DXVECTOR3* m_pVerts;
        float* m_pUVs;
        int m_nFaceCount;
        short* m_pFaces;

        char* m_pMemoryBuffer;

public:
        CMesh():        m_nVertCount(0),
                        m_pVerts(NULL),
                        m_pUVs(NULL),
                        m_nFaceCount(0),
                        m_pFaces(NULL),
                        m_pMemoryBuffer(NULL){}
        void Destroy();
        BOOL InitSampleData();
        BOOL Save0( const char* pszFilename );
        BOOL Save1( const char* pszFilename );
        BOOL SaveUseDataChunk( const char* pszFilename );
        BOOL LoadUseDataChunk( const char* pszFilename );

        BOOL LoadFaces( UPointer inP, int nSize );
```

```
        BOOL LoadVerts( UPointer inP, int nSize );
        BOOL LoadUVs( UPointer inP, int nSize );
};
```

一：基本方法

```
BOOL CMesh::Save0( const char* pszFilename )
{
        FILE* fp = fopen( pszFilename, "wb" );
        if( !fp )return FALSE;
        fwrite( &m_nVertCount, sizeof( int ), 1, fp );
        fwrite( m_pVerts, sizeof( D3DXVECTOR3 )*m_nVertCount,
              1, fp );
        fwrite( m_pUVs, sizeof( float )*2*m_nVertCount, 1, fp );
        fwrite( &m_nFaceCount, sizeof( int ), 1, fp );
        fwrite( m_pFaces, sizeof( short )*3*m_nFaceCount, 1, fp );
        fclose( fp );
        return TRUE;
}
```

二：手动 DataChunk

```
BOOL CMesh::Save1( const char* pszFilename )
{
        FILE* fp = fopen( pszFilename, "wb" );
        if( !fp )return FALSE;

        DWORD dwTag;
        int nSize;
        dwTag = DC_TAG( 'vert' );
        nSize = sizeof( m_nVertCount )+
                m_nVertCount*sizeof( D3DXVECTOR3 );
        fwrite( &dwTag, sizeof( DWORD ), 1, fp );
        fwrite( &nSize, sizeof( int ), 1, fp );
        fwrite( &m_nVertCount, sizeof( int ), 1, fp );
        fwrite( m_pVerts, sizeof( D3DXVECTOR3 )*m_nVertCount,
              1, fp );

        dwTag = DC_TAG( 'uvuv' );
        nSize = m_nVertCount*sizeof( float )*2;
        fwrite( &dwTag, sizeof( DWORD ), 1, fp );
        fwrite( &nSize, sizeof( int ), 1, fp );
        fwrite( m_pUVs, sizeof( float )*2*m_nVertCount, 1, fp );

        dwTag = DC_TAG( 'face' );
        nSize = sizeof( m_nFaceCount )+m_nFaceCount*sizeof( short )*3;
        fwrite( &dwTag, sizeof( DWORD ), 1, fp );
        fwrite( &nSize, sizeof( int ), 1, fp );
        fwrite( &m_nFaceCount, sizeof( int ), 1, fp );
        fwrite( m_pFaces, m_nFaceCount*sizeof( short )*3, 1, fp );
        fclose( fp );
        return TRUE;
}
```

三：自动 DataChunk输出

```
BOOL CMesh::SaveUseDataChunk( const char* pszFilename )
{
        CDataChunkWriter w(1024*1024);
        w.StartChunk( DC_TAG( 'vert')  );
        {
                w.WriteInt( m_nVertCount );
                w.Write( m_pVerts, sizeof( D3DXVECTOR3 ),
                        m_nVertCount );
        }
        w.EndChunk( DC_TAG( 'vert' ) );
        w.StartChunk( DC_TAG( 'vtuv' ) );
        {
                w.Write( m_pUVs, sizeof( float )*2, m_nVertCount );
        }
        w.EndChunk( DC_TAG( 'vtuv' ) );
        w.StartChunk( DC_TAG( 'face' ) );
        {
                w.WriteInt( m_nFaceCount );
                w.Write( m_pFaces, sizeof(short)*3, m_nFaceCount);
        }
        w.EndChunk( DC_TAG( 'face' ) );
        w.SaveToFile( pszFilename );
        return TRUE;
}
```

我们在 StartChunk 和 EndChunk 的时候都声明了一个 TAG。EndChunk 的作用是结束一个DataChunk，同时TAG是用来匹配startchunk的，当用户因为编程出现笔误的时候，DataChunk生成器会报警，帮助程序员调试。同时可以看到我们用一对{}来包括一个数据块，这样做的目的是为了方便排版，而且数据块的层次关系一目了然，日后程序员在实践的时候就会发现DataChunk输出部分代码如同写诗一般，原本枯燥繁琐的代码就被有效的组织起来。

接下来，我们来读取这个网格。首先，我们把整个网格文件全部读入内存中，在这里，我们用到了一个技巧，就是一次性把整个文件读入内存，读取完毕后不释放分配的内存，而让网格的指针指向正确的位置，

这样做有助于减少内存碎片，加快读取速度。可以看到，我们用三个独立的函数来分别读取顶点、贴图坐标和法线，这三个函数都使用一样的参数：LoadSomething(UPointer inP, int nSize)，第一个参数用来传递一个指针，第二个是这个指针指向的DataChunk的大小，我们可以在读取函数中加入内存越界检查：

```
BOOL CMesh::LoadFaces( UPointer inP, int nSize )
{
        UPointer p(inP.p);
        m_nFaceCount = *p.i++;
        m_pFaces = p.st;
        p.st += m_nFaceCount*3;
        return TRUE;
}

BOOL CMesh::LoadVerts( UPointer inP, int nSize )
{
        UPointer p(inP.p);
        m_nVertCount = *p.i++;
        m_pVerts = p.v;
```

```
        p.v += m_nVertCount;
        return TRUE;
}

BOOL CMesh::LoadUVs( UPointer inP, int nSize )
{
        UPointer p(inP.p);
        m_pUVs = p.f;
        p.f += 2*m_nVertCount;
        return TRUE;
}

BOOL CMesh::LoadUseDataChunk( const char* pszFilename )
{
        FILE* fp = fopen( pszFilename, "rb" );
        if( !fp )return FALSE;
        fseek( fp, 0, SEEK_END );
        int nSize = ftell( fp );

        m_pMemoryBuffer = new char[nSize];
        rewind( fp );
        fread( m_pMemoryBuffer, sizeof( char )*nSize, 1, fp );
        fclose( fp );

        UPointer p( m_pMemoryBuffer );
        while( p.c < m_pMemoryBuffer+nSize )
        {
                switch( DC_TAG( *p.dw ) )
                {
                case 'vert':
                        {
                                p.dw++;
                                int size = *p.i++;
                                LoadVerts( p, size );
                                p.c += size;
                        }
                        break;
                case 'vtuv':
                        {
                                p.dw++;
                                int size = *p.i++;
                                LoadUVs( p, size );
                                p.c += size;
                        }
                        break;
                case 'face':
                        {
                                p.dw++;
                                int size = *p.i++;
                                LoadFaces( p, size );
                                p.c += size;
                        }
                        break;
                default:
                        {
                                p.dw++;
                                int size = *p.i++;
                                p.c += size;
                        }
                }
        };
        return TRUE;
}
```

现在,我们已经把网格完整的读取出来了,可以看到把一个网格分为不同Data Chunk进行结构化存储之后代码的确复杂了一些,不过接下来我们就可以体会到DataChunk带来的好处。

在本文的附带源代码中,有如下代码:

```
int _tmain(int argc, _TCHAR* argv[])
{
        CMesh mesh;
        mesh.InitSampleData();
        mesh.SaveUseDataChunk( "test0.mesh" );
        mesh.Destroy();
        mesh.LoadUseDataChunk( "test0.mesh" );
        mesh.SaveUseDataChunk( "test1.mesh" );
        return 0;
}
```

我们先用手动方法产生一个网格,用DataChunk方法保存此网格,再用DataChunk方法读取这个网格,最后用DataChunk方法另存一个文件,我们发现产生的两个文件内容是一样的,验证了我们的Save和Load是有效的。

现在,网格除了顶点、uv坐标、面索引之外又加入了顶点法线:

```
class CMesh
{
protected:
        int             m_nVertCount;
        D3DXVECTOR3*    m_pVerts;
        D3DXVECTOR3*    m_pNorms;
        float*          m_pUVs;
        int             m_nFaceCount;
        short*          m_pFaces;
};
```

我们在输出函数中增加如下代码,

```
w.StartChunk( DC_TAG('norm') );
{
        w.Write(m_pNormals, sizeof(D3DXVECTOR3),m_nVertCount );
}
w.EndChunk( DC_TAG('norm') );
```

然后在读取代码中加入一个读取法线的函数:

```
BOOL CMesh::LoadNorms( UPointer inP, int nSize )
{
        UPointer p(inP.p);
        m_pNormals = p.v;
        p.v += m_nVertCount;
        return TRUE;
}
```

继续在LoadUseDataChunk函数中加入一个DataChunk的识别:

```
while( p.c < m_pMemoryBuffer+nSize )
{
```

```
        switch( DC_TAG( *p.dw ) )
        {
        case 'vert':
                {
                        p.dw++;
                        int size = *p.i++;
                        LoadVerts( p, size );
                        p.c += size;
                }
                break;
        // 新增加的法线DataChunk识别
        case 'norm':
                {
                        p.dw++;
                        int size = *p.i++;
                        LoadNorms( p, size );
                        p.c += size;
                }
                break;
        case 'vtuv':
                {
                        p.dw++;
                        int size = *p.i++;
                        LoadUVs( p, size );
                        p.c += size;
                }
                break;
        case 'face':
                {
                        p.dw++;
                        int size = *p.i++;
                        LoadFaces( p, size );
                        p.c += size;
                }
                break;
        default:
                {
                        p.dw++;
                        int size = *p.i++;
                        p.c += size;
                }
        }
};
```

这样就完成了数据的升级,想象一下,我们已经在保持原有数据信息的情况下增加了法线信息,也并没有破坏原有的数据,在这个DataChunk规则基础上,我们理论上可以对网格的数据成员进行无限扩展,只要我们在存储和读取的时候对新增数据成员进行识别和校验就可以了。

DataChunk自动生成器的工作原理

DataChunk的应用是非常常见的,在我接触过的代码和实际工作中,发现很多效率低下的代码,有的代码为了生成一个DataChunk硬性计算偏移量和数据块长度,这种方法在数据块类型较少、数据块嵌套深度较小的情况下还可以凑合,但也付出了很大的代价,就是需要仔细的检查,同时每个DataChunk的载入都需要严格对应。DataChunk自动生成器的工作原理很简单,就是先写数据到内存中,保存好每个数据块的头部和起始地址,此时并不计算实际数据的长度,实际数据长度的计算推迟到如下程序显示的调用数据块结束的方法:

数据块头部

数据块头部的数据结构定义如下:

```
struct ChunkHdr
{
        DWORD dwName;
        DWORD dwSize;
};
```

DWORD dwName:数据块的名字
DWORD dwSize:数据块的长度

堆栈匹配

每个DataChunk都会有一个ID,当一个DataChunk准备写入的时候,都会把当前ID、偏移量等数据push到一个特殊的堆栈中,当数据块写入完毕就把堆栈顶端的内容pop出来,这个数据块的结构定义如下

```
struct ChunkDesc
{
        ChunkHdr header;
        DWORD dwOffset;
};
```

ChunkHdr header:数据块头部
DWORD dwOffset:数据块相对于整个内存缓冲池的偏移量

DataChunk自动生成器中使用了一个调试功能,就是在输出DataChunk的时候,程序员首先声明一个数据块的开始:StartChunk(DWORD dwName);当前数据块输出完毕,再调用一个EndChunk(DWORD dwName);如果此时EndChunk中的参数dwName和栈顶的dwName不匹配,生成器会报错,告诉程序员书写代码的时候出错了,这样一来,程序员可以很方便地调试代码,一定程度上保证了可以正确地输出DataChunk。

自动偏移量计算以及零数据长度数据块忽略:

这个功能可以自由开关,当程序员希望输出的数据都是有用的或者希望输出文件尺寸相对小一些的时候可以打开。

附:DataChunk自动生成器源代码及使用范例,可以在《CSDN开发高手》杂志频道下载。

深入 QuickReport (下)

▶撰文 / 董维春

第三部分:报表中的其他问题

通过前两部分的学习,我想你对QuickRep已经有了一定的了解了,在这部分我们对报表设计中的其他一些问题做一下简单介绍。

此部分内容都以例程的形式讲解,为了保持文章的完整性,例子的编号接上部分。

例五、TQRChart 组件的使用

TQRChart组件,是在报表使用的图表组件。可按以下步骤完成属性设置:

1)把TQRChat图表组件放置在一个适当的Band区段中。

2)双击TQRChart图表组件,此时将会出现Editing QRDBChart1图表编辑对话框,如图1。

图 1

3)Editing QRDBChart1图表编辑对话框默认的选项卡是Chart页,单击'Add…'按钮,接着出现TeeChart Gallery窗口,如图2。

4)选定图表类型后,单击OK将返回到Editing QRDBChart1图表对话框,对话框中出现我们刚才选取的图表类型,此时图表的名称默认为Seres1,如果我们不想用这个标题,另给报表一个新的标题,我们只要单击'Title…'按钮,就可以实现我们的想法,如图3。

5)切换到Series选项卡,在这个选项卡里还有多个

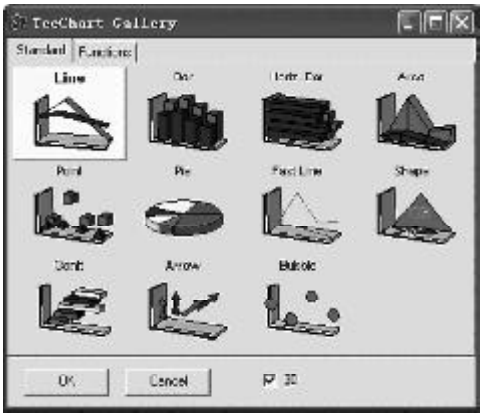


图 2

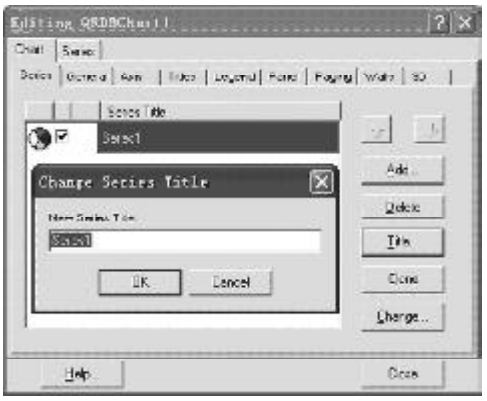


图 3

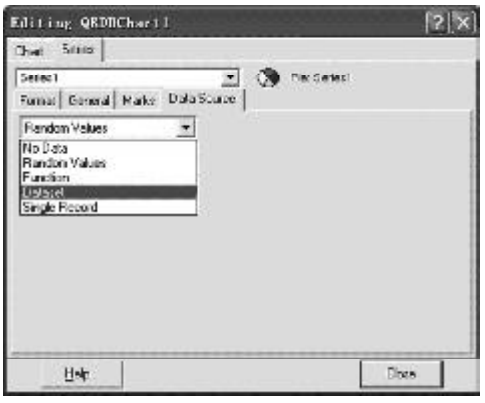


图 4

子选项卡,把子选项卡切换到Data Source页,接着选取数据源,因为我们要用数据库中的数据,所以将数据源选定为 DataSet,如图4。

投稿信箱:tougao@csdn.net

6）在Dataset字段中指定提供数据源的组件，这里取的是Table1组件，如图5。

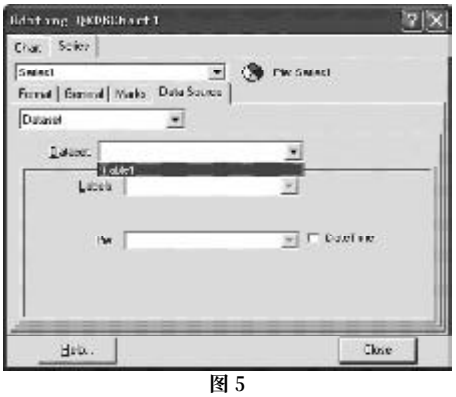


图 5

7）在Labels与Pie下拉框中，指定表中的段，然后单击Close按钮，完成图表的设计。

例六、报表的连接及保存

通过上面的例子，你也许会动手做了几个报表，有时你一定会想把其中的一些报表连接起来，组成一个综合报表，并作为整体来操作。在BCB中实现这一点并不难，我们这时要用到TQRCompositeReport组件。它提供了一个OnAddReprots事件，在创建报表时将触发这个事件，因此我们只要在这个事件中用Add方法将需要连接在一起的报表添加到该组件的事件中就可以了。下面给出一个示例程序段，这是把两个报表添加到综合报表中的，代码如下：

```
void __fastcall TForm1::QRCompositeReportAddReports(TObject *Sender)
{
        ((TQRCompositeReport*)Sender)->Reports->Add(Form2->
                QuickRep1);//添加第一个报表

        ((TQRCompositeReport*)Sender)->Reports->Add(Form3->
                QuickRep1);//添加第二个报表
}
```

做好的报表我们一定都想保存起来，保存的文件格式有：文本格式文件（TXT），组件TQRTextFilter；超文本格式文件(HTML/HTM)，组件TQRHTMLFilder；逗号分隔文件（CSV），组件TQRCSVFilter；以及报表文件。保存前三种格式文件需要调用ExportToFilter方法，而直接保存报表组件，则只需用Save。这个例子中我们放了一个TSaveDialog对话框和QuickReport组件页中的TQRTextFilter、TQRHTMLFilder、TQRCSVFilter三个组件。完整的代码如下：

```
void __fastcall TForm1:: SaveReportClick(TObject *Sender)
{
        AnsiString FileExt;
        // 打开保存文件对话框获得文件名
        if(SaveDialog1->Execute())
        {
                // 获得文件后缀
                FileExt = AnsiUpperCase(ExtractFileExt(SaveDialog1
                        ->FileName));

                // 输出Html超文本文件
                if((FileExt == ".HTML") || (FileExt == ".HTM"))
                 QuickRep1->ExportToFilter(new TQRHTMLDocumentFilter
                                        (SaveDialog1->FileName));

                // 输出txt文本文件
                else if(FileExt == ".TXT")
                 QuickRep1->ExportToFilter(new TQRAsciiExportFilter
                                        (SaveDialog1->FileName));

                // 输出CSV文件
                else if(FileExt == ".CSV")
                 QuickRep1->ExportToFilter(new TQRCommaSeparatedFilter
                                        (SaveDialog1->FileName));

                // 输出报表文件
                else
                {
                 QuickRep1->QRPrinter->Save(SaveDialog1->FileName);
                }
        }
}
```

例七、自定义报表预览窗口

QuickReport的报表预览功能总是不能达到令人满意的效果，因此，我们有必要自定义快速报表的预览窗口，以达到完美的设计要求。

1）设置预览窗口

新建工程，在Form1窗体上添加一个ToolBar控件，并在其上添加以下按钮：“调入报表”、“打印”、“打印设置”、“上一页”、“下一页”、“放大”、“缩小”和“关闭”。在Form1窗体上添加一个StatusBar，双击该组件，在编辑器中插入三项，在第三个项中显示页面信息。在Form1窗体上添加一个TQRPreview控件，对齐方式设为alClient，Form1窗体的外观如下页图6所示。

再新建一个窗体，设其Name为Form2，在该窗体上添加TQuickRep控件，设其Name为QuickRep1，其PrinterSetting中的Units属性设为mm（以毫米为计量单位），然后建立报表。

2）编程实现

（1）在Form2上选择QuickRep1，在其事件中选择

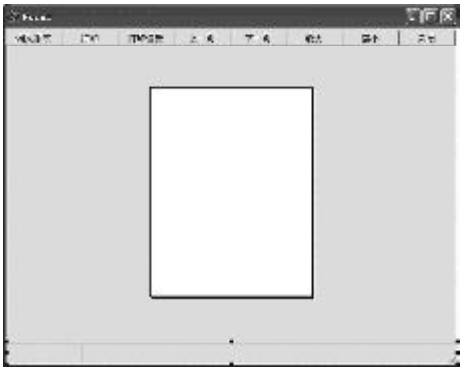


图 6

OnPreview，输入以下代码：

```
#include <vcl.h>
#pragma hdrstop

#include "Unit2.h"
#include "Unit1.h"//调用Form1的内容
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm2 *Form2;
__fastcall TForm2::TForm2(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------
void __fastcall TForm2::QuickRepPreview(TObject *Sender)
{
        //最为关键的一步
        Form1->QRPreview1->QRPrinter=Form2->QuickRep1->QRPrinter;
}
```

（2）为Form1中的各功能按钮的OnClick事件添加如下代码：

```
#include <vcl.h>
#pragma hdrstop
#include "Unit1.h"
#include "Unit2.h"//调用Form2的内容
#pragma package(smart_init)
#pragma resource "*.dfm"

TForm1 *Form1;

__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//调入报表
void __fastcall TForm1::QRLoadClick(TObject *Sender)
{
        //不加入该句,下一句的共几页将不能正确出现
        Form2->QuickRep1->Prepare();
        StatusBar1->Panels->Items[2]->Text="第"+
                IntToStr(Form1->QRPreview1->PageNumber)+
                "页"+"共" + IntToStr(Form2->QuickRep1->
                PageNumber)+"页";
        Form2->QuickRep1->Preview();//显示页面的信息
}
//打印报表
void __fastcall TForm1::PrintClilck(TObject *Sender)
{
        Form2->QuickRep1->Print();
}
//打印设置
void __fastcall TForm1::PrintSetupClick(TObject *Sender)
{
        Form2->QuickRep1->PrinterSetup();
        //不加这句可只能设置,不能打印
        Form2->QuickRep1->Print();
}
//上一页
void __fastcall TForm1::PageUpClick(TObject *Sender)
{
        if(QRPreview1->PageNumber>1)
        QRPreview1->PageNumber--;
        StatusBar1->Panels->Items[2]->Text="第"+
                IntToStr(Form1->QRPreview1->PageNumber)+"
                页"+"共" +IntToStr(Form2->QuickRep1->
                PageNumber)+"页";
}
//下一页
void __fastcall TForm1::PageDownClick(TObject *Sender)
{
        if(QRPreview1->PageNumber < Form2->QuickRep1->
          PageNumber)
        QRPreview1->PageNumber++;
        StatusBar1->Panels->Items[2]->Text="第"+
                IntToStr(Form1->QRPreview1->PageNumber)+"
                页"+"共" +IntToStr(Form2->QuickRep1->
                PageNumber)+"页";
}
//报表放大
void __fastcall TForm1::ZoomInClick(TObject *Sender)
{
        if(QRPreview1->Zoom<200)
        QRPreview1->Zoom+=5;
}
//报表缩小
void __fastcall TForm1::ZoomOutClick(TObject *Sender)
{
        if(QRPreview1->Zoom>5)
        QRPreview1->Zoom-=5;
}
//程序关闭
void __fastcall TForm1::CloseClick(TObject *Sender)
{
        Close();
}
```

例八、QR 组件的动态设置及探讨

BCB中提供了大量的VCL组件，有时难免要在程序中动态创建组件，VCL是用 Object Pascal写的，所以VCL类的对象我们只能在堆中创建。

如创建一个TQRLable对象，我们可以这样来创建：

```
TQRLable *MyQRLable= new TQRLable(From1);
```

即写成如下程式:

```
类名  *对象名=new 类名(…);
```

注意:()里面可以是你已创建的该类对象的父类名字、工程的名字、NULL或this。但最好是对象的父类名。

例:动态生成TQRLable组件

我们先在窗体(Form1)上,放上一个TQuickRep,并在其上放一个Band。在Form1窗体中按钮Button1的单击事件中写上如下代码:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
        TQRLable *MyQRLable= new TQRLable(From1);
        //关键的一句,否则你将看不到,但编译却是正确的
        MyQRLable->Parent=Band1;
        MyQRLable->Top=10;
        MyQRLable->Left=38;
        MyQRLable->Height=25;
        MyQRLable->Width=100;
        MyQRLable->Caption="I'm MyQRLable!";
}
```

通过这个例子我们应该清楚地看出动态创建QR组件的几个重要步骤:

1)要一个空间(内存);

```
TQRLable *MyQRLable= new TQRLable(From1);
```

2)指定其父组件,说直接了就是为我们要创建的这个对象指定一个容器;

```
MyQRLable->Parent=Band1;
```

3)指定组件要出现在父组件的哪个位置;

```
MyQRLable->Top=10; MyQRLable->Left=38;
MyQRLable->Height=25; MyQRLable->Width=100;
```

4)其它重要属性。

```
MyQRLable->Caption="I'm MyQRLable!";
```

注意:上面的步骤不能任意安排,否则你的程序会出笑话的。

在动态生成非BorLandVCL原有组件时要加上对应的头文件。我们要动态生成报表组件时一定要加入:

```
#include " Qrctrls.hpp "
```

若还有问题,你还要加入:

```
#include " QuickRpt.hpp "
```

另外由于BCB对内存管理或与系统、硬件的冲突,你的动态创建程序也许一点错误都没有,但就是编译不了;有时也许第一次通过了,第二次一样的程序却通过不了,出现这样那样的提示,最简单的办法就是注销一下系统,再试一下,多数就能解决了。

既然这样的动态产生组件会出现很多不可预见的问题,那我们还有没有更好的办法来实现类似的功能呢?答案是肯定的。

我们可以在TQuickRep中把所有组件都放置好,各区段组件,可视化组件都放在应该放的位置上,只是一些特定的属性我们先不给出,而通过程序给出,这样就可以仿制动态创建组件的方法来动态产生报表。说简单了就是事先把组件都准备好了,用的时候拿出来,不用时就不给定关键属性,即用属性废掉它,让它根本不起作用。

例:动态设置QRDBText的属性值

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
        QRDBText1->DataSet=Table1;
        QRDBText1->DataField="AREA";//改成双引号后,一切OKJ
        QuickRep1->Preview();
        Table1->Close();
}
//---------------------------------------------
```

或

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
        QRDBText1->DataSet=Table1;
        QRDBText1->DataField=Table1->Fields->Fields[3]->FieldName;
        QuickRep1->Preview();
}
```

单就一个QuickReport动态生成报表,我想不是三言两语就能说完的,单独成文也不为过,好在我也把动态生成报表的一些常见问题都说出来了,解决之道上面也提了,并且还给出了几个例子,通过这些介绍,你要是有精力的话一定会做出一个不出问题的动态生成报表系统。

以上程序均在XP系统、BCB6、EPSON C43UX下编译通过。

C++转向C#的疑惑：
C# 中类间通信方法初探

▶撰文/朱伟

C#中没有自由函数也没有自由变量，所有的一切都必须封装在类中。在C++中，通过全局变量来进行函数间通信和类间通信是常用的方法，然而这在C#中却变成不可能的任务，这的确让C++程序员苦恼。C#当然有它充分的理由不支持自由函数和自由变量，因为“自由”就暗示着冲突、不安全等等影响程序健壮性和可维护性的因素。当然有很多技术可以解决C#中的类间通信问题，本文就介绍其中的几种。

问题涉及的主要Class

在介绍之前先作一下说明。一般可以把类间通信需要的各相关数据元素封装成一个结构，我称之为“通信结构”，将其记为Information struct，这个结构除了公共的数据成员外，什么都没有。当类间需要交换消息时就可以生成一个“通信结构”实例，然后在类间传递。另外我把提供信息的类称为Provider Class（以下简称为Provider），接受信息的类称为Master Class（以下简称为Master）。当然，两个相互通信的类可以互为Provider和Master。请注意，后面将出现Provider object和Master object，它们对应于Provider和Master的实例。

Master从Provider获取所需信息主要有两种方式：正向获取和反向获取。每一种方式又有几种不同的实现方法。下面我将一一介绍。

正向获取信息

所谓“正向”获取信息，就是按照通常的直线思考方式来考虑，Master需要从Provider获取信息，就直接让Master访问Provider的某个成员。在这种方式下，Master处于主动地位，因为总是由它去主动拜访Provider，而Provider相对比较被动。

不出你所料，正向获取信息主要有以下几种方法。

1. Provider暴露一个成员变量或属性

Master可以通过访问这个成员变量或属性得到所需数据。这种方法对于得到一些简单的非保密性信息是比较方便的，通常这些信息可以被Provider自己直接提供，并且不易发生变化，比如获取某个控件（当然是一个Provider object）的size或color等等。此方法的不利之处是，如果Provider所提供的数据是保密性的，那么这个安全性就得不到保证。

2. Provider暴露一个方法

如果所需的信息比较复杂（比如需要通过计算才能得到），或经常处于变化之中，通过一个方法来提供这样的信息无疑是更好的选择。有两种实现方式：

◆通过返回值

这个我们经常用，在这种情况下，Master还可以通过参数来和Provider进行更多的交流。如：

```
public Information GetInformation( 参数 ) ;
```

◆通过ref或out参数

如果一个方法的返回值被用于其它目的，如判断方法是否执行成功等，此时可以用ref或out来将所需的信息从方法中“带出”。如：

```
public int GetInformation(out Information info) ;
```

或许你已经看出来，正向获取信息的方式的主要问题在于，Master不知道Provider会于何时准备好所需的数据，也就是说，Master是在一个合适的时间拜访Provider吗？如果时机不恰当，显然就得不到正确的数据。

为了解决这个问题，我们可以采取反向获取信息的方式。

投稿信箱：tougao@csdn.net

反向获取信息

所谓“反向”获取信息，就是Provider在准备好数据后，主动将数据提供给Master。与正向获取信息的方式相比较，可以看到，主——客关系发生了变化，现在是Provider主动去拜访Master。这就很自然地解决了正向获取信息方式的时机可能不成熟的问题。

在实际中，用的最多的就是反向获取信息的方式，普遍的情况是这样的：在Master需要数据时，创建一个Provider object，Provider object取得数据并作相关处理后将所需信息保存在一个Information object中，然后将该Information object提交给Master，自此该Provider object生命期结束，接着Master就可以处理得到的数据了。为了使读者有一个直观的认识，举个简单的例子：

图1 Master object

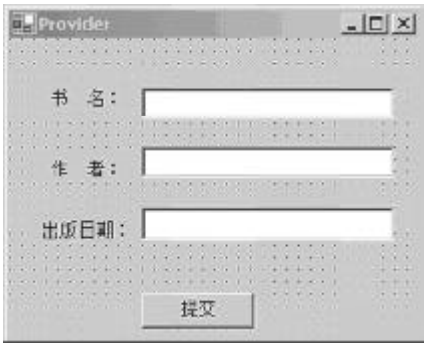


图2 Provider object

这是一个简单图书信息管理系统，图1是Master object界面，图2是Provider object界面，当点击图1的“添加”按钮时就会弹出图2所示界面，当用户填写完信息后点击“提交”按钮，即可将一个新的图书信息添加到管理系统中。

此时，Information结构可如下设计：

```
public struct Information
    {
        public string book_name ;
        public string author ;
        public string date ;
    }
```

1．通过类的静态成员变量进行数据通信

这是最简单的直接模拟C++中的利用全局变量进行通信的方法。我们把一个Information对象作为Master的静态成员变量，并将其修饰符设为public，则Provider object就可以对其进行写操作。Provider写操作完成之后，Master就可以处理静态的Information对象了。现在，问题出现了——Master怎么知道Provider完成了写操作呢？回调函数！！！——这是C++程序员的第一反应。在C#中可以吗？可以，只不过C#提供的是一种安全的函数指针，叫委托。我们前面已经指出Provider对象通常都是由Master对象创建的，那么Master对象在创建Provider对象时可以传给Provider对象一个委托实例。这样，当Provider完成写操作之后，就可以通过此委托实例完成回调。这个过程可如图3所示。

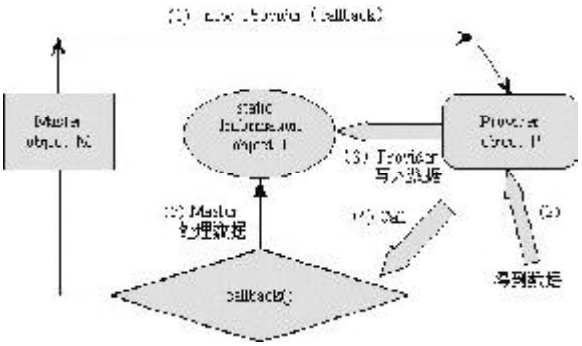


图3 通过类的静态成员变量进行类间通信

下面看看这种方法的主要C#伪码。

首先要定义委托：

```
public delegate void D_Callback() ;
```

再看Master类：

```
class Master
  {
        public static Information object_I ;
        public void GetInformation()
        {
          D_Callback callback=new D_Callback(ProcessInformation);
              Provider object_P=new Provider(callback) ;
              ...../ /如，显示object_P界面
```

```
                }
                private void ProcessInformation()
                {
                        ……//处理object_I
                }
        ……
    }
```

接下来看看Provider如何实现：

```
class Provider
{
        private D_Callback back ;
        //构造函数需以“回调函数”作为参数
        public Provider(D_Callback yourback)
        {
                this.back = yourback ;
        }
        //如当用户点击图2中的“提交”按钮时，将调
       // 用以下处理函数
        private void button1_Click()
        {
                //访问Master.object_I，并将信息写入
                ……
                back() ;//触发回调函数调用，即调用
                        //Master中的ProcessInformation函数
                this.close();// Provider对象完成任务，可以被GC
                                回收了。
        }
}
```

很简单，不是吗？是的，它只不过是C++中用全局变量进行通信的C#版本。然而这种方法好吗？不好！这种方法仅仅是完成了任务，但是干得并不漂亮，甚至，这种实现是非常丑陋的。原因至少有两点：

首先，使Information对象成为Master的public静态对象就是非常不合适的，因为如果这样，除了Provider外还有所有其它的类（包括含有恶意的类）都可以向这个静态区域写入信息，这就丧失了安全性。

其次，Master要将自己的成员函数（作为回调的函数）包装成委托传递给Provider作为构造参数，这就导致了Master和Provider非常紧密的耦合，如同联体婴儿一样。如果我们想仅仅复用其中的一个类而不牵带另一个类是完全不可能的，这违背了OOD的基本原则。

很糟糕，不是吗？让我们来看一个改良版本。

2．通过堆栈对象复制进行数据通信

在Provider object中将获取数据存放到自己的私有Information对象中，然后用该Information对象调用从构造函数传递进来的委托。这样一来，跳转到Master的回调函数时，因为Information是struct，而struct是值类型，所以将会在栈上复制这个Information对象，其副本就可以被回调函数访问了。如此，信息就从Povider传递到Master。

此过程可表示为图4所示的步骤。

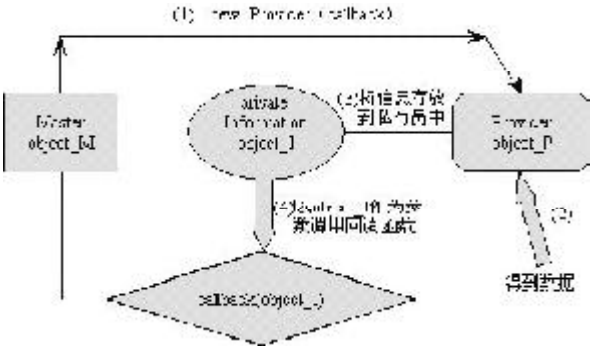


图4　通过类的静态成员变量进行类间通信

再来看看代码有何改变：

```
public delegate void D_Callback(Information object_I) ;
//此处要将Information对象作为回调函数的参数
class Master
{
        //不再拥有静态的Information成员
        public void GetInformation()
        {
                D_Callback callback=new D_Callback(ProcessInformation);
                Provider object_P = new Provider(callback);
                ……
        }
        private void ProcessInformation(Information object_I)
        {
                //处理object_I的副本
                ……
        }
          ……
}
```

接下来看看Provider的改变：

```
class Provider
{
        private D_Callback back ;

        //增加私有Information成员用以存放要传递的信息
        private Information object_I;

        public Provider(D_Callback yourback)
        {
                this.back = yourback ;
        }

        private void button1_Click()
        {
                ……  //将信息写入this.object_I
                //以object_I作为参数调用Master中的
                ProcessInformation函数
                back(this.object_I);
                this.close() ;
```

```
        }
}
```

通过堆栈对象复制避免暴露一块公共的内存区域(public static member),增强了安全性,但是Master和Provider之间的紧耦合性并没有减弱。难道没有完美的解决方案吗?有!那就是通过事件的发布——预定方式。采用这种方式,发布事件的类根本就不必知道预定事件的类的任何情况,从而将以前的Master和Provider之间的循环依赖转化为单向依赖(仅仅Master依赖于Provider),这就有效地降低了Master和Provider之间的耦合。

3. 通过事件机制进行通信

事件机制为类间通信提供了一种优雅、高效、灵活的解决方案。

我们只需在Provider类中声明并发布事件,并在适当的时候触发事件;然后在Master类中预定事件,并定义事件处理函数就可以了。此方式下,Provider类不仅能为Master类所用,而且也能为其它需要得到同样信息的类使用。如此看来,Master类只是Provider类的一个客户(client)而已。整个通信过程可描述为图5所示的步骤。

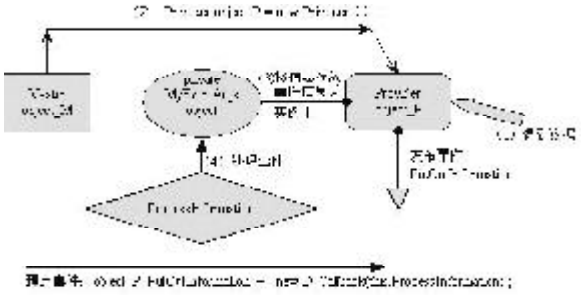


图5 通过事件机制进行通信

我们来看看是如何实现的。

首先一个委托的实例如果要成为事件,委托的原型声明必须是如下形式:

```
public delegate void D_Callback (object sender ,MyEventArgs e) ;
```

注意,MyEventArgs必须从EventArgs类继承,它将用来包装要传递的信息,这里我们可以简单的如下实现:

```
public class MyEventArgs:System.EventArgs
{
        public Information object_I;
}
```

接着看看Provider是如何发布事件并触发事件的:

```
class Provider
{
        public event D_Callback  PutOutInformation; //发布事件

        private void button1_Click()
        {
                MyEventArgs  args = new MyEventArgs() ;
                ......  //将信息写入args.object_I
                PutOutInformation(this, args);  //触发事件
                this.close() ;
        }
}
```

是不是简单、清晰、明了?!Provider根本就不需要知道Master的存在。

再来看看Master如何预定事件,以及事件发生时如何获得所需信息。

```
class Master
{
        public void GetInformation()
        {
                Privoder object_P = new Privoder(); //预定事件
                object_P.PutOutInformation
                        +=new D_Callback(this.ProcessInformation);
                .....
        }

        private void ProcessInformation(object sender,MyEventArgs e)

        {
                ...../处理e.object_I
        }
                .....
}
```

上面的代码简单明了,就不用我再罗嗦了。

总结:

C#是一种完全面向对象的语言,相对于C++,它剔除了全局变量和全局函数,并且引入了事件、属性等面向对象编程的常用特性。由于没有了全局变量,所以在C#中必须寻找另外的方案来解决类间通信的问题,而C#提供的事件机制恰好是解决这类问题的一剂良方,虽然在C#中也可以模拟C++通过静态存储区或栈复制来完成通信,但这些方式给人的感觉比较笨拙且不够优雅,所以,当你找不到更好的办法来实现类间通信时,就用事件机制吧。

BASIC四十年

▶撰文/王咏刚

你知道吗？BASIC语言今年四十岁了。和只有九岁的Java语言以及只有三岁的C#语言相比，BASIC可以算是程序设计语言中的长辈了。可奇怪的是，某些程序员居然坚持认为BASIC是所有语言中最"幼稚"的语言，这不是在颠倒黑白吗？现在，BASIC家族的支持者遍布全世界（在下也勉强算一个），未来的BASIC语言更没理由衰微——依我看，BASIC多半会变成程序设计领域里的"老顽童"，嘻嘻哈哈之余，没准儿还会练就一身绝世的武功；这其中的原因嘛，很简单：在过去的四十年里，BASIC几乎每天都充满了活力，这样一种健康活泼、激情无限的语言当然会在新世纪里大有作为！

1 名门正派

四十年前的1960年代是程序设计领域混沌初开的时代。在那个时代里，使用高级语言编写程序的人并不多见，可供选择的高级语言也寥寥无几。Fortran语言、Algol语言和COBOL语言异常繁琐，思想超前的LISP语言又在执行效率上表现不佳。用今天的眼光看，1960年代的程序员和学习程序设计的大学生们的确苦不堪言。

为了让程序设计和学习程序设计的过程变得更加轻松和愉快，1960年代早期，Dartmouth学院的J. Kemeny和T. Kurtz开始设计一种新的语言。他们为这种新语言定下的设计原则包括：

（1）简单，尽可能的简单

据说，J.Kemeny和T.Kurtz这种尽量追求简单的设计理念来自爱因斯坦的科学哲学观。在理论物理学领域，爱因斯坦毕生所持有的斯宾诺莎式的哲学信仰及其在理论研究中对简洁和优雅的追求历来都为科学史家们所称道。今天，我们已经很难判断，J.Kemeny和T.Kurtz在设计BASIC语言时是否具备了爱因斯坦那样深邃的哲学思辨精神；但毫无疑问的是，BASIC语言的两位创始人把爱因斯坦的思维方式发挥到了极致，他们在摒弃繁冗、追求简约方面的实践非常成功：早期的BASIC语言只有14条语句，但它却能解决当时已知的大部分计算问题。BASIC在语法上的简洁与Fortran、Algol和COBOL语言的繁琐形成了鲜明的对比。

（2）像英语，尽可能地像英语

从现代英语中借鉴关键词和语法规则并不是BASIC的创举，Fortran语言已经在这一点上为后来者做出了很好的表率。不过，BASIC语言的设计者对此显然更为注重，这甚至可以从"BASIC"这个语言名称上反映出来。人们通常认为，BASIC这个名字只是一系列单词的缩写，但少数研究者指出，BASIC这个名字似乎与C.K.Ogden的一项发明有关。1930年，为了让非英语国家的英语学习者更快地具备英语交流能力，C.K.Ogden别出心裁地发明了一种只有850个单词和几十条语法规则的"简化版英语"，并将其命名为"Basic English"。尽管这门奇怪的英语并没有像C.K.Ogden最初所设想的那样成为一种全球通行的"世界语"，但C.K.Ogden对英语进行简化以方便交流的思想在事实上促成了今天"商务英语"、"科技英语"等现代英语形态在全世界的流行。从这一点看，程序设计领域里的BASIC语言的确与Basic English有一定的血缘关系，这种血缘关系也在事实上为BASIC语言的流行铺平了道路——对于第一次接触程序设计的人来说，BASIC语言中的英语语法和英语单词看上去显然要比LISP语言中层层叠叠的括号亲切得多。

1964年，BASIC语言正式诞生。关于这种新语言的名称，J. Kemeny和T. Kurtz给出的官方注解是：BASIC是"初学者通用符号指令代码（Beginner's All Purpose Symbolic Instruction Code）"的缩写。第一个BASIC语言运行环境由J.Kemeny，T.Kurtz和几个学生在G.E系列计算机上实现，并成为了Dartmouth学院为G.E系列计算机开发的分时软件系统DTSS的一部分。

从语法特征上看，这种新的语言继承了Fortran和Algol语言的许多优点，其代码的外观与Fortran语言非常接近。但与以往大多数高级语言不同的是，BASIC语言是一种"交互式"的语言。也就是说，大多数BASIC语言程序以解释的方式运行，人们用不着预先编好全部代码并使用批处理的方式提交给计算机处理；使用BASIC语言时，我们可以在一种更加舒适、更加人性化的环境下，一边键入代码，一边观察代码的执行结果，并随时对代码进行修改和完善。BASIC语言的交互式特征被后来的许多解释型语言和脚本语言所借鉴。一个最典型的例子是，Smalltalk语言的许多实现（如

VisualWorks)不但继承了BASIC语言交互式编程的基本特征,而且还更进了一步,把语言的编辑和运行环境、类库、调试界面、功能插件都整合在了一个可以充分交互的集成环境里。

初生的BASIC语言首先在教学实践中取得了成功。从1964年开始,J.Kemeny和T.Kurtz在Dartmouth学院使用BASIC语言讲授程序设计课程。学生们对这种入门型语言的印象极为深刻,他们认为,掌握BASIC语言非常容易,在掌握BASIC语言后,学习Fortran等的高级语言也变得相对简单了。应当说,J.Kemeny和T.Kurtz设计BASIC时的设想大多被后来的教学实践所证实。很快,其他一些开设计算机课程的院校陆续将BASIC语言纳入了教学计划,研究人员也在随后的几年内将BASIC语言移植到了好几种不同的计算机系统上。

J.Kemeny和T.Kurtz在Dartmouth学院实现的BASIC语言被称作Dartmouth BASIC。在随后的几十年里,BASIC语言的两位创始人,以及他们的同事和学生们一直致力于对语言本身的语法特性进行优化和改进。1983年以前,Dartmouth BASIC已经陆续发展了七个版本。在J.Kemeny和T.Kurtz眼中,1964年的BASIC语言还远未成熟,那时的BASIC还只是一组编有行号的指令序列,其中还穿插着惹人生厌的GOTO和GOSUB语句。由E.W.Dijkstra发端,于七十年代兴起的结构化程序设计运动不但催生了Pascal等一大批结构化程序设计语言,也为BASIC语言的进一步发展指明了方向。1971年,Dartmouth BASIC新增了带参数、可调用的外部子程序功能,A.Luehrmann还为Dartmouth BASIC实现了一个交互式的图形库,这让BASIC语言在结构化的道路上前进了一大步。1976年,Dartmouth BASIC第一次拥有了完整的结构化语法规则。S.J.Garland在Dartmouth BASIC的基础上,实现了一种完全符合结构化编程思想的SBasic语言(SBasic是"Structured Basic"的缩写)。从这一年起,Dartmouth的学生们编写BASIC程序时,就已经不再需要行号和GOTO语句了。1979年,Dartmouth BASIC第一次引入了"包"(有时也被称为"组")的概念,这是一种类似于Ada语言的"包"结构的语法特征,它可以使BASIC语言拥有初步的数据封装能力。

1983年,Dartmouth学院的一些学生说服J.Kemeny和T.Kurtz将Dartmouth BASIC语言变成一种商业软件在市场上销售。为此,一家名为True BASIC的公司正式成立,公司推出的产品也叫True BASIC,它其实是由Dartmouth BASIC第7版演化而来的。最早的True BASIC可以在IBM PC和Macintosh计算机上运行,既能以解释方式执行BASIC代码,也能将其编译成机器语言,以提高运算速度。从问世的那一天起,True BASIC就是一种纯粹的结构化程序设计语言,它在后续的每一次版本变迁中都始终不渝地坚持走结构化的道路,并一直遵循着BASIC语言历来奉行的简约主义精神。今天,True BASIC已经发展出了DOS、MacOS、Windows、Unix和Linux等多种版本,并提供了图形、声音、窗口操作、数学运算等丰富的功能库。

和其他主流程序设计语言的发展规律相仿,ANSI和ISO等标准化组织也先后为BASIC语言制定了相关的标准。这包括ANSI于1978年,ISO于1984年通过的"最小化BASIC语言标准(Standard for Minimal BASIC)",以及ANSI于1987年,ISO于1991年通过的"完整的BASIC语言标准(Standard for Full BASIC)"。ANSI和ISO组织制定BASIC语言标准时的主要依据是Dartmouth BASIC和True BASIC。在今天我们可以见到的所有BASIC语言产品中,与ANSI和ISO的BASIC语言标准吻合得最好的自然也是True BASIC了——这句话的另一句潜台词是:很遗憾,除了True BASIC以外,目前市场上的其他BASIC语言几乎都与ANSI和ISO的标准相去甚远。

毋庸置疑,True BASIC以及ANSI、ISO的BASIC语言标准代表了BASIC语言的"名门正派"。但不幸的是,"名门正派"在市场上并没有取得应有的成功。正如本文后续部分将要提到的那样,从1970年代BASIC语言在个人电脑市场上崭露头角开始,J.Kemeny和T.Kurtz就几乎在普通程序员的心中消声匿迹了。当Microsoft把BASIC语言当作"摇钱树"卖给了几乎所有个人电脑用户的时候,Dartmouth学院的研究者们只会在实验室里指责市场上的BASIC都是些过时的垃圾;当QuickBASIC、Turbo Basic等非Dartmouth血统的结构化BASIC语言在市场上通行无阻的时候,True BASIC尽管身出名门,却无法抢到哪怕10%的市场份额;当Visual Basic、VBScript和Visual Basic.NET羽翼渐丰的时候,ANSI和ISO的标准也早已失去了应有的效力……很多人抱怨Microsoft不遵循ANSI和ISO的BASIC语言标准,以至于市场上的各种BASIC至今无法统一,但却很少有人问一问Dartmouth学院的研究者们,为什么不能把"名门正派"的产品尽快推向市场?为什

么总是让先进的技术和先进的标准在纸面上停滞不前？为什么在Microsoft已经成为了事实上的标准制定者之后，还要坚持所谓的“正统观念”？

当然，我没有指责J.Kemeny和T.Kurtz等人缺乏商业头脑的意思。我只是想告诉大家，BASIC语言的发展史中确实存在这样一个悖论：一方面，BASIC语言在计算机领域，特别是个人电脑中四处开花，迅猛发展；另一方面，BASIC语言的实现版本千差万别，程序员们莫衷一是。也许，BASIC语言天生对“简约”的追求与现实中软件系统的复杂性是一对永恒的矛盾。当Microsoft等实用主义者发现类似的矛盾难以调和的时候，他们会粗暴地修改BASIC语言的语法规则和设计理念，并用一种折中了“简单”与“实用”的新版本向“名门正派”发起挑战。到1980年代的时候，这种挑战已经发展到了让BASIC语言的创始人们坐立不安的地步。1984年，J.Kemeny和T.Kurtz出版了《回到BASIC——语言的历史、堕落和未来(Back to Basic:The History, Corruption,and Future of the Language)》一书，并在书中鲜明地指出，所有Dartmouth学院以外的BASIC都或多或少地背离了BASIC语言的设计初衷，它们都是“浪迹街头的BASIC语言（Street Basic）”，应当尽快“回家”。J.Kemeny和T.Kurtz所说的BASIC的“家”当然是指以True BASIC为代表的正统BASIC而言。此外，在1999年的一篇文章中，T.Kurtz还对Microsoft公司在Visual Basic中把原本全大写的BASIC名称变成了“Basic”这样大小写混合的做法表示了不满。T.Kurtz认为，全大写的“BASIC”一词才是“初学者通用符号指令代码”的缩写，才能体现出BASIC语言面向初学者的设计初衷和精神实质。

J.Kemeny和T.Kurtz维护正统BASIC尊严的做法无可厚非，这就像武林中名门正派的高手在旁门左道面前总会摆出满脸矜持以示高人一等一样。但市场规律并非哪个知名学者的一两句“呼吁”所能逆转：从BASIC祖师爷召唤BASIC回家至今已经整整二十年了，BASIC语言却在“浪迹街头”的道路上越走越远——这样残酷的现实除了让我们对“名门正派”无限同情以外，还能说明什么问题呢？

2 胜者为王

接下来，我们不妨看一看，那些被J.Kemeny和T.Kurtz视为“街头浪子”的BASIC语言是如何在Dartmouth学院以外蓬勃发展起来的。

在BASIC世界里，Microsoft是市场上的实际领导者，也是不折不扣的异教徒。就像几百年前新教在欧洲和北美大陆的迅速兴起一样，Microsoft在BASIC语言市场上的成功也几乎可以用“势如破竹”来概括。

1975年时，计算机历史上最大的变革——个人电脑的风暴已经蓄势待发了。就在这个时候，Microsoft公司的两位创始人，Paul Allen和Bill Gates为MITS公司生产的个人电脑雏形Altair 8800开发了一个BASIC语言解释器，这也是Microsoft公司的第一件产品。今天，当我们回过头来，再次检视那个英才辈出的年代时，我们不难发现：个人电脑和BASIC语言的结合实际上是一种无奈的选择，因为那个时候的个人电脑运算能力都非常有限，根本就无法运行任何复杂的应用，而语法简单、交互性强的BASIC语言正好可以满足个人电脑的需要；另一方面，正是个人电脑和BASIC语言的珠联璧合造就了Bill Gates和Microsoft的成功，这大概是因为，当时没有哪个大公司愿意纡尊降贵去编写那么“简单”的BASIC解释器，而大学还没有毕业的Bill Gates当然不会在意自己的产品是否有足够的技术含量。于是，一个能在Altair 8800计算机上运行的，不足4KB大小BASIC语言解释器揭开了Microsoft抢占软件市场的序幕。

1975到1979年间，Microsoft把自己的BASIC语言解释器先后移植到了TRS-80、Commodore、Atari等个人电脑上。1977年，Apple公司的S. Wozniak设计Apple电脑时，为了让Apple电脑也拥有BASIC语言运行环境，自己开发了一种只支持整型运算的Integer BASIC语言。Integer BASIC在功能上远不如Microsoft的同类产品强大。于是，1977年底，Microsoft和Apple又共同为Apple II开发了名为Applesoft的BASIC语言解释器。不过，1979年以前，Apple计算机内置的BASIC语言解释器仍然是Integer BASIC，Applesoft仅以磁盘方式发售。1979年，著名的Apple II Plus计算机中才第一次内置了Applesoft。随着Apple II Plus在商业上的成功，Microsoft几乎为当时市场上的所有个人电脑提供了BASIC语言解释器。

1981年，IBM PC的出现掀起了个人电脑发展的新高潮。众所周知，Microsoft为IBM PC提供了磁盘操作系统PC-DOS，同时，Microsoft也用MS-DOS的名称为其他PC兼容厂商提供一模一样的操作系统软件。PC-DOS内嵌了一个名叫BASICA的BASIC语言解释器。BASICA代表“高级的BASIC（Advanced BASIC）”，尽管它一点也不高级，还不具备结构化语言的特征。类似地，MS-DOS中也内嵌了一个同样的BASIC解释器，

只不过名字变成了GW-BASIC(有人说GW在这里是“Gee Whiz”的缩写，也有人说GW其实代表了Bill Gates的名字Gates William)。值得注意的是，BASICA和GW-BASIC都是捆绑在DOS系统中销售的免费软件——不用说，这也是Microsoft的一贯“伎俩”了。

BASICA和GWBASIC既非结构化语言，执行速度也不快，但却与DOS系统一道占领了PC市场。1982年，Microsoft首先解决了PC机上BASIC语言运行速度慢的问题，他们为IBM开发了一种BASIC语言编译器BASCOM 1.0。

1984年，Apple公司推出了第一种具备图形界面的个人电脑Macintosh，Microsoft相应地开发出了专用于Macintosh电脑的BASIC语言解释器MS-BASIC 1.0 for Mac，包含编译功能的MS-BASIC for Mac也在不久后上市。

1985年，True BASIC语言提醒下，Microsoft推出了一种完全结构化的BASIC语言——QuickBASIC 1.0。QuickBASIC有一个强大的集成开发环境，既包含BASIC代码的解释功能，也包含编译功能。与True BASIC追求简约的设计思路略有不同，QuickBASIC在设计上更注重语言功能的完整和执行效率的提高。正如其名称所暗示的那样，QuickBASIC几乎是那个时代里运行速度最快的BASIC语言，但其在语法上对“正统”BASIC语言的背叛也成为了Microsoft为T.Kurtz等人诟病的原因之一。QuickBASIC的最终版本是1988年发布的QuickBASIC 4.5。随后，1989年，Microsoft以新的名字推出了两个功能更丰富的BASIC开发环境，即BASIC PDS(PDS在这里代表“专业开发系统”)的7.0和7.1版本。1991年，Microsoft又在MS-DOS 5.0中捆绑了只包含解释功能的QuickBASIC的简化版本QBasic 1.0。

胜者为王。Microsoft公司在BASIC市场上取得了空前的成功，也自然成为了市场的领导者和BASIC语言标准的实际制定者。在Microsoft看来，Dartmouth学院里的人过于迂腐，而ANSI和ISO的BASIC语言标准又不能为我所用，此时，Microsoft所能做的就是不断用新的BASIC产品抢占市场，并不断按照自己的思路对BASIC语言进行改造。也许，从理论上很难讲清楚Microsoft的做法对BASIC语言的发展是利大于弊，还是弊大于利，但从市场角度看，胜利者的做法总是对的。无论如何，从1980年代起，BASIC语言已经沿着Microsoft的方向一去不返了。

除了Microsoft以外，在BASIC语言市场上获得过成功的公司屈指可数。Borland公司可以算一家。要不是Borland在推出了Turbo Basic之后，因为公司内部原因而放弃了BASIC市场的话，Microsoft后来的Visual Basic也许就不会有一览众山小的架势了。1987年，在开发工具领域毫不含糊的Borland公司成功地推出了Turbo Basic 1.0。Turbo Basic的原型是1980年前后由B.Zale开发的BASIC/Z。Turbo Basic和QuickBASIC非常类似，在某些功能上还更胜一筹。遗憾的是，1989年，Borland发布了Turbo Basic 1.1后便放弃了该产品线。Turbo Basic的最终命运是，B.Zale收回了自己的产品，并独立将其发展为今天的PowerBASIC。

3 脱胎换骨

如果说，Microsoft的GW-BASIC和QuickBASIC仅是在商业上取得了成功，而没为BASIC语言引入更多新思想的话，那么，从Visual Basic到Visual Basic .NET的历程应算是Microsoft为BASIC语言所做的最大贡献了。

Visual Basic是最早在商业上获得成功的一种快速应用开发(RAD)工具。它的出现背景是，1985年问世的Windows操作系统在1990年代迅速普及，程序员对于快速图形化应用开发的需求越来越强烈。1987年，A.Cooper和他的同事们在Microsoft编写了一个可视化开发工具的原型，它就是Visual Basic的前身。1991年，Visual Basic 1.0问世。Visual Basic集成了QuickBASIC的语法特性、编译功能和A.Cooper的可视化开发环境，允许程序员在一个所见即所得的图形界面中迅速完成开发任务。这对以往几十年里程序员们所熟悉的“编码－编译－连接－运行”的开发体验来说，的确是一个脱胎换骨的变革。

1992年，Microsoft又特意为当时仍占有相当市场份额的DOS操作系统发布了Visual Basic 1.0的MS-DOS版。与Windows版本类似，程序员可以在VB中通过鼠标点击和拖曳开发出基于事件驱动模型、拥有窗口和菜单机制的DOS程序。此后不久，DOS退出了历史舞台，这个版本的Visual Basic也成了Microsoft在DOS环境下发布的惟一的一款可视化开发工具。

1993年，Microsoft发布了Visual Basic 3.0。这一版本的Visual Basic支持ODBC、OLE等高级特性。1995年发布的Visual Basic 4.0不但支持Windows 95系统下32位应用程序开发，而且为Visual Basic引入了类(Class)等面向对象概念。1998年发布的Visual Basic

6.0是VB向VB.NET转变前的最后一个版本，也是传统Visual Basic中功能最全、应用最广的一个版本。

更震撼人心的变革发生在2001到2002年间。Microsoft以惊人的勇气进军.NET平台，同时也为程序员带来了一大堆让人应接不暇的新技术和新概念。伴随着.NET平台的横空出世，Visual Basic又经历了一次脱胎换骨的革命。为了实现.NET一统江湖的伟大构想，Microsoft甚至不惜让新版本的Visual Basic .NET放弃与传统VB程序的兼容。在Visual Basic .NET中，我们看到了完整的面向对象特性，看到了.NET风格的内存管理和异常处理机制，看到了对ASP.NET、Web Form和Web Services等新技术的支持，同时也遗憾地看到，已有的大量Visual Basic代码必须经过改动才能在.NET平台上顺利运行。

2002年正式发布的Visual Basic.NET让BASIC语言第一次拥有了和C++语言、Java语言一样强大的语法功能，使BASIC语言可以和业界最主流的运行环境.NET站在一起并肩战斗，也让四十年前J.Kemeny和T.Kurtz为BASIC语言确立的“简单至上”的精神实质丧失殆尽。今天，我们看到的Visual Basic.NET已经成了多种语言特征的混合体，在这个Microsoft一手塑造的巨人身体里，既有早期BASIC的单纯外貌，也有QuickBASIC的实用主义风格，既有.NET平台的无所不能，也有大多数面向对象语言都具备的复杂语法……我不知道Microsoft还会为Visual Basic.NET引入哪些风格特性，我只知道，Microsoft手中的BASIC语言已经越来越难以体现“初学者通用符号指令代码”的原始内涵了。

4 风雨江湖

Visual Basic.NET并不能代表BASIC语言的一切。在某种意义上，Microsoft为VBScript语言及其相关技术所做的努力更能反映出BASIC语言的发展趋势。

从1993年开始，Microsoft为Office软件提供了一种与Visual Basic类似的应用开发环境——Visual Basic for Applications。它是Visual Basic的一个子集。从应用角度看，它为Office用户提供了一种定制应用软件功能的可编程方法，我们可以把它看成是脚本化BASIC语言的前身。1995年，为了满足Internet和IE浏览器的应用需求，也为了和JavaScript语言展开竞争，脚本语言VBScript正式诞生。

VBScript语言对Visual Basic做了最大程度的简化，尤其是对数据类型进行了大刀阔斧地合并。这使得VBScript语言在某种程度上与J.Kemeny和T.Kurtz所提倡的正统观念靠得更近了些。最重要的是，VBScript为程序员们提供了一种远比JavaScript、csh、ksh、Perl、Python等其他脚本语言简单，也更容易为程序员们所掌握的脚本语言。在Windows平台上，配合Microsoft的ActiveX、ASP等技术体系，程序员们可以用VBScript语言迅速完成包括原型开发、文件管理、Web应用、数据库访问等在内的大多数日常任务。就像当年交互式开发环境的普及一样，脚本语言正逐渐成为最近几年里程序设计语言发展的又一个亮点。VBScript语言将脚本语言的学习难度降到了最低点，这也许表明，BASIC语言在脚本语言的世界里还会有更大的作为。

此外，如果走出Microsoft营造的BASIC王国，我们仍然能在许多场合找到各种充满活力的BASIC语言：首先，许多提供二次开发功能的商业软件（如群件系统、工作流管理系统和数据仓库系统）都将BASIC或VBScript语言作为首选的开发环境；其次，PowerBASIC、REALbasic、FutureBasic、OmniBasic等为数众多的商业软件仍然在Microsoft的王国外围坚守着自己的阵地；再次，随着开源运动的蓬勃发展，wxBasic、XBasic、Yabasic、SmallBASIC、Bywater BASIC等开放源码的BASIC语言解释器和编译器为程序员们发掘BASIC语言的潜力提供了最好的资源……BASIC的江湖依然是一派风云变换的热闹景象。有人说，这反映了BASIC语言的无秩序和不规范；但混乱未必就是坏事，谁又能断言，这种混乱的起因不会是BASIC语言天生就具有的那种旺盛生命力呢？

5 天下大势

《三国演义》说，“天下大势，分久必合，合久必分”。今天，BASIC的境况看上去比三国割据的形势更为微妙：一方面，Visual Basic.NET已经成为了Microsoft阵营中事实上的BASIC语言标准；另一方面，在Microsoft王国之外，层出不穷的BASIC变种也实在让人们难以取舍。

未来的BASIC语言需要统一吗？未来的BASIC语言将会以什么样的方式统一？BASIC语言的创始人能看到BASIC回归到简约、优雅的理想状态吗？也许，我们没必要认真寻找这些问题的答案；只要BASIC这个程序设计领域里的“老顽童”能不断为我们带来快乐，能让我们永远保有一颗童心，我们就非常满足了——为什么一定要让BASIC承担太多的责任和期望呢？

品牌安全气囊

▶撰文/龚敏敏

网络的冲击激发了汽车生产商的“创意”,于是气囊外包市场的竞争轰轰烈烈地展开了。

微软的系列气囊:

气囊98

偶尔会无故弹出,造成轻伤。也会在汽车受到强烈碰撞时出现蓝屏,如果用户侥幸还活着的话只要关上所有窗口,重新启动系统就可以了。

气囊XP

不会蓝屏。如果弹出失败,说明你的许可证已经过期,必须登陆微软网站重新激活系统。

气囊2003

绝对不会蓝屏,平时也很稳定,不会有无故弹出。缺点是过于小心,真正需要弹出时也会询问原因,可以选择“正碰”、“侧碰”……

气囊.NET

气囊.NET是Pop services平台。Pop services允许气囊通过 Internet 进行通讯和共享数据,而不管所采用的是哪种车型、设备或路面状况。气囊.NET平台提供创建Pop services并将这些服务集成在一起之所需。对驾驶员的好处是无缝的、吸引人的体验。

Visual气囊

带有可视化的环境,你随时都可以看得见那个气囊,还带有一个强大的叫做msdn的支持帮助系统。

气囊SQL Server

无论您是一名新手、老鸟,还是车库管理员,也无论您正在磨合期,还是已为飚车行动准备就绪,在气囊SQL Server的产品版本中,总会有一款适合您和您所在组织机构车辆的应用需求。但你必须忍受极大的系统消耗、巨量的空间占用、莫名其妙的Bug问题和蠕虫王的疯狂攻击。

SUN气囊

控制系统用JAVA完成,所以在气囊弹出时可能会出现“垃圾收集中,请稍候…”

IBM气囊

利用电子商务的强大优势,当出现危险时,会迅速地为你送去一个安全气囊。

金山囊霸

独创的闪电防护模式,0.4秒钟防护n种伤害。并可以随时利用汽车空闲时间,囊霸屏保启动后自动进行防护。但它会监视汽车的每一个动作,所以驾驶速度会明显下降。

RedHat气囊

从现在开始,我们只在企业版中集成安全气囊,个人版只有一条安全带。

Mandrake气囊

有超强的定制性,可以自定义几乎所有的参数——大小、弹出力度——幸运的话,你可以在事故发生前根据自身需要定制出适合自己的安全气囊。

TurboLinux气囊

该安全气囊实现了内核级的汉化。其汉化工作、美观、以及易用性都是很好的。

开源气囊

安全气囊是免费的,他们的时间和精力不是免费的。如果你不会安装而需要他们帮忙的话,你就需要支付技术支持费用。

Adobe气囊

PDF安全气囊的创建十分简便,使用的时候只需单击一下按钮即可。PDF安全气囊在保持你的外观和完整性的同时,还允许您通过电子方式与其他人分享,并且不受硬件和软件平台的限制。

Oracle气囊

Oracle的安全气囊最快最安全,但需要划分出半个驾驶舱的位置由气囊自己管理。

3721气囊

这种品牌的气囊很特别,你只需在方向盘上输入一个关键词,就能获得行车路线指南。坏处是该产品有强烈排他性,所以在副驾位安装百度气囊的可能性几乎为零。

SCO气囊

所有Linux技术的气囊都是抄袭我们的,法庭上见!

PHP模板引擎介绍

▶撰文/周献华

一、为什么使用模板

随着基于WEB Application的开发逐渐复杂,显示界面不再是单一的HTML,而可能是XML、XSLT、WML、IMAGE、PDF等等显示方式,如果这样还是使用传统的脚本代码(PHP)嵌入式的开发显然是低效的。PHP模板技术可以使你的开发,维护更轻松,开发效率比传统的HTML和PHP混合在一起的做法要高得多。它的核心思想是使得数据处理和数据显示互不影响,程序员可以专注于代码的编写,设计人员可以专注于界面的设计、美化等操作,也能使程序代码不改动或较少改动的情况下达到显示界面多样化的功能。看下面的代码:

程序代码

```
demo.php
<?php
require_once 'libs/Smarty.class.php';
$smarty = new Smarty;
$name = 'redhat';
$smarty->assign('name',$name);
$smarty->display('demo.tpl');
?>
```

模板代码

```
demo.tpl
<html>
<head>
<title>Demo Example</title>
</head>
<body>
hello,{$name},welcome to you!
</body>
</html>
```

在浏览器运行demo.php会输出“hello,redhat,welcome to you!”的字符串。

这里使用了PHP官方站点上的‘smarty’模板引擎做为例子,可以看出结构简单,逻辑清晰,PHP和HTML代码已经完全分离。下面就介绍PHP开发中几种主流的模板引擎。

二、模板资源介绍

1、FastTemplate

这是一个开发较早的模板类,来自同名的Per软件包,官方站点上看到的最新版是1.1.0。该模板主要是在PHP3的基础上开发而成的,当然也可使用在PHP4上。模板文件中模板标识简单,使用如“{TITLE}、{CONTENT}”之类的简单标识代替,模板文件中没有逻辑处理功能。在模板中输出多列数据时,必须在程序中循环使用方法“$tpl->assign()”和“$tpl->parse()”进行赋值、替换,所以在使用动态块的地方要用文件独立出来。

下载地址:http://www.thewebmasters.net/php/

2、TemplatePower

TemplatePower和FastTemplate比较相似,功能较FastTemplate相比有所加强(特别是动态区块的处理上),在其官方网站上称其速度比FastTemplate大约快6倍,目前最新版是3.0.2。

下载地址:http://templatepower.codocad.com

3、PHPLIB中的模板类

PHPLIB中包含了很多类库(数据库类,SESSION类,模板类……),是一个使用很广泛的PHP扩展类库,目前最新版是7.4。这里介绍的模板类就是其中的一个类,在压缩包中可以看到其模板类文件叫做“phplib/php/template.inc”。如果你使用过FastTemplate,那么你会发现PHPLIB的模板类很容易上手。在模板文件中的变量替换基本和FastTemplate是一样的,它允许动态的块嵌套。

如果你是使用PHPLIB进行你的项目开发的话,此模板类应该会是你的首选。

下载地址:http://phplib.sourceforge.net/

4、PEAR中的模板类

PEAR是PHP的一个有力扩展,其中的模板类主要有:HTML_Template_Flexy,HTML_Template_IT,HTML_Template_PHPLIB,HTML_Template_Sigma,HTML_Template_Xipe几个模板处理的类,可以在PEAR官方网站中的“HTML”包找到,里面有详细的

投稿信箱:tougao@csdn.net

文档说明。如果你是使用PEAR开发你的Web项目，这些模板类当然是首选。

下载地址：http://pear.php.net

5、Smarty

Smarty是PHP官方站点上推荐的模板引擎，目前最新版是2.6.1。它与我们上面所介绍的模板有一个很大的差别就是：Smarty支持“编译”和“缓存”，当第1次请求的时候把模板文件“解析编译”成可直接运行的PHP文件，以后请求的时候就直接运行已编译好的文件而不执行解析，速度较其它模板有了较大的提高，当然还可以使用它的缓存功能进一步提高程序运行速度。除此之外，Smarty还支持在模板中使用逻辑控制及字符串控制等功能。

Smarty支持模板扩充(插件)，正因为有了大量的插件从而使得Smarty的功能很强大，所以Smarty成为了在PHP世界中构建MVC框架的首选模板引擎。

下载地址：http://smarty.php.net

6、SmartTemplate

SmartTemplate是由作者“Philipp”个人开发的模板类，目前最新版是1.0.2。在它的官方网站上称其模板解析速度比Smarty快8倍，整体的框架构建上和Smarty大体相似，在模板标识的使用上和PHPLIB相似。除此之外，它还有其它几个类：目录读取，ini文件解析，缩略图像等等，不够的话还可以自己去扩充。

下载地址：http://www.smartphp.net

7、eval

eval是PHP自带的一个函数，能执行传入的字符串。可以用来做模板处理，示例：

模板代码

```
demo.tpl
<html>
<body>
$msg
</body>
</html>
```

PHP代码

```
demo.php
<?php
$fp = fopen("demo.tpl","rb");
while(true){
    $data = fgets($fp,4096);
    if($data == ''){
        break;
    }
    $buf .= $data;
}
fclose($fp);
$content = str_replace('"','\\"',$buf);
$msg = "Hello,World!";
eval("\$content = \"$content\";");
echo $content;
?>
```

执行后就可以得到如下结果：

```
<html>
<body>
Hello,World!
</body>
</html>
```

像VBB，PHPBB等论坛都是用eval来进行模板的处理，处理速度较快，使用灵活，也可以在此基础上进行一些扩展，构造一个适合自己项目开发的模板类。

8、Template class

在http://www.phpguru.org/template.html可以看到一个模板类，功能还算可以，使用较简单，支持模板嵌套，动态区块输出等等，该站上还有其它一些很不错的类和程序代码，值得学习！

其实模板的资源很多，我们只是介绍常用的几种，如果有需要的读者可以再做些其他的收集。

三、综述

大部分PHP模板类的构造思想和工作原理都大同小异，取得模板文件的内容后，使用正则表达式或其它一些字符处理函数对其进行解析后再输出，自己也可以根据需要对其源码进行相应的修改使之更符合你自己的习惯或要求。选择适合自己的模板引擎，灵活应用在各种各样的Web开发上，可以使你的项目不管是在开发还是在维护上都可以达到事半功倍的效果。选择模板类应该考虑其易用性、稳定性、扩展性、解析速度、发展前途等方面，当然更主要的还是和具体的项目有着很大的关系。有些模板类的作者可能会随时停止开发；有些模板引擎在小型项目表现出色但不适合大型项目，有些则相反；有些模板引擎的学习很简单，有些则需要花较多的心思在上面。在提高项目开发效率的同时多为以后的维护及扩展考虑。

模板只是一个工具，在MVC框架中充当表现层的角色，选择一个好的模板引擎可以使项目的开发更加规范、更加有效率，特别是多人进行一个较大项目开发时这种效果尤其明显。

解读软考新政策

▶撰文/CSAI

2003年12月底,在长沙举行的“中国系统分析与设计年会”上汇聚了一大批系统分析员、IT业内资深人士,共同交流与探讨了IT产业发展方向、系统分析与设计技术。期间,大家就许多热点问题进行了深入的交流,特别对近期出台的软考新政策(国家人事部[2003]39号文)给予了高度的关注。为了帮助大家在参加2004年软考时更有针对性,笔者特总结了此次会议探讨出的思想与看法,与大家分享。

改革的必需

开考14年来,软考在专业设置方面比较单一,仅聚焦于软件开发,已不能适应IT领域日新月异的变化和领域的细分。许多IT从业人员,都不能在软考中找到合适自己的级别考核,正如一个网友戏谑性的感叹:“我平时的工作从未涉及到程序开发,而认真看完考试大纲后,发现只有系统分析员可以报考,其它的都不行。”正是由于这样的局限性,软考的发展受到了很大的局限性,于是改革被提入议程。

改革的措施

早在1999年初,国家软考中心就提出了一个新的软考政策,其根据技术专业领域不同,将软考分成程序设计、软件工程、计算机网络、数据库、多媒体五个领域。当时就有很多人质疑,这样完全按技术领域的划分,能够满足需求吗?你想,这几方面,谁不是兼而顾之?此举后来无故“破产”,也许正是其较大的局限性、较差的可操作性所致吧。

而新发布的软考政策则很明显地按照应用领域进行了划分,这样将更加有利于不同领域的从业人员找到与自己相吻合的专业和级别。此细分办法更有针对性和可操作性。从下表的具体的设置来看,新政策借鉴了许多日本的经验,并根据国内实际情况进行了合理的调整,如表1所示。

在新政策中,对于中级以下的考试按应用领域细分为:计算机软件、计算机网络、计算机应用技术、信息系统、信息服务五类:

1) 计算机软件:主要针对软件开发的从业人员,在原来的基础上取消了初级程序员,保留了程序员和高级程序员级别,在考试内容和形式上基本保持不变,并且针对于软件公司重视软件质量,甚至设立专门的软件测试部门的趋势,在中级中加入“软件评测师”这一级别。

2) 计算机网络:从级别命名上看,应该在考试的内容方面较“网络程序员”和“网络设计师”会有一些改变:“网络工程师”应该会注重网络总体设计方面能力的考察;“网络管理员”应会注重网络日常管理与维护、简单网络架设的能力。也就是说,会从网络应用的开发转到网络的综合应用。

3) 计算机应用技术:多媒体、计算机辅助设计、嵌入式系统开发、电子商务都是现在计算机应用技术的热点,拥有大量的从业人员。以前针对这部分的软考内容与其知识结构是格格不入的,此次所进行的专业化级别设置,必将摆脱这一尴尬局面。

表1

	计算机软件	计算机网络	计算机应用技术	信息系统	信息服务
高级资格	信息系统项目管理师 系统分析师(原高级分析师) 系统架构设计师				
中级资格	软件评测师 软件设计师 (原高级程序员)	网络工程师	多媒体应用设计师 嵌入式系统设计师 计算机辅助设计师 电子商务设计师	信息系统监理师 数据库系统工程师 信息系统管理工程师	信息技术支持工程师
初级资格	程序员(原初级程序员和程序员)	网络管理员	多媒体应用制作技术员 电子商务技术员	信息系统运行管理员	信息处理技术员

“信息系统管理师”、“信息系统运行管理员”就是为他们量身定做的,当然也能够满足信息系统的

部署与实施人员的需要。“数据库系统工程师”则是为那些从事数据库系统设计、部署、维护的人士而设。另外，在新政策中还与国家信息工程监理管理办法配套，设置了“信息系统监理师”，其主要培养对象为信息系统质量把关的专业人员。

5）信息服务：当大家在讨论该专业的设置时，并没有具体的答案，不过比较一致地认为这主要是为应用信息技术的从业人员设置的。“信息技术支持工程师”就是能够指导别人正确地应用信息技术的专业人员。“信息处理技术员”则是能够正确地应用信息技术完成工作的技术人员。

值得注意的是，新政策对于高级资格部分，却并没有按照这样的五个应用领域进行细分，这就说明了，新的软考政策要求这一层次的人员能够有更宏观的理论基础和更广泛的经验与知识。为了能够适合在业务中不同的角色，新政策设置了相应的三种考试：

1）偏重项目管理的“信息系统项目管理师”，主要是考察承担项目的总体协调、管理、实施能力；

2）偏重系统分析、业务分析的“系统分析师”（即承担SA工作），主要是考察业务流程分析、项目可行性分析、系统分析等能力，可能会增加成本核算、经济学等方面的知识；

3）偏重系统设计的“系统架构设计师”，主要考察软件体系结构设计的能力。

改革后更具含金量的高级资格

在原来的“中国计算机技术与软件专业资格（水平）考试”中的最高级别只有一个，那就是系统分析员。根据原考试政策的规定，通过该级别就相当于高级工程师水平，也是参评高工的条件之一。但是“系统分析员”却一直未被受到应有的关注，甚至可以说是冷遇，在许多IT业的招聘会上，很多负责招聘的人力资源经理们都没听说过系统分析员，就更别谈什么优先录用。

分析其中原因，主要是长期以来，这项以考代职的考试，只对中级以下是一通过考试就获得相应职称，而系统分析员则往往并不能够直接获得高级职称。由于系统分析员考试的残酷，加上考试的难度甚至大于直接评聘高工，因此参加考试的人数并不多。自1989年开考以来，系统分析员考试的通过率一直在4-5%之间，总的通过人数也仅1000左右，因此网上也有许多人发出了“系统分析员比熊猫还珍贵”的感叹。

另外系统分析员这一名称，由于直接与技术员对应起来，所以给人带来很大的误解。我在报考的时候，还有很多人误以为其与网络程序员、网络设计师是同一个系列的，甚至是比网络程序员还低的一个级别。然而在这次的系统分析与设计年会上，与会的系统分析员们都工作在各个公司的重要岗位上，都为企业的发展做着具大的贡献。事实证明通过系统分析员考试的均是基础知识扎实，有丰富实践经验的高端人才。

我们看看软考发源地日本的另一番景象。首先在人数上，日本该级别的通过人数数十倍于我国；在重视程度方面，通过考试的人员将成为日本通产省的智囊团，直接加入政府信息化项目的专家组。

也许国家人事部、信息产业部认识到了这些问题，我们十分欣喜地看到在新的软考政策中，有几个针对性的措施：

1）通过高级资格考试，可聘任高级工程师职务：近年来一些省颁发了相应政策，系统分析员可以直接获得高级工程师职务，这让其它省市的系统分析员们羡慕不已；

2）将系统分析员更名为系统分析师，虽然是一字之差，却也足以避免原来的许多误解，使人们更容易理解；

3）增加了“信息系统项目管理师”、“系统架构设计师”两个高级资格考试，丰富了考试的内容，使其参考人员更有针对性。

而且听说自2004年开始，软考可能一年考两次，相信这些有利的举措都能够促使报名和通过人数的大幅提高。软考证书含金度的加强，将成为我国IT行业打造、考核人才的试金石。同时，我们也要提高警惕，不要重演1999年软考改革的历史。

综合感受

在网络、现实生活的讨论中，很多人都有“一场考试是否能够综合考核实力”的疑问。其实笔者和很多通过系统分析员考试的朋友，观点是一致的：这场考试是一把尺子，一把衡量自己水平的尺子；这场考试也是一条鞭子，一条激励自己向前走的鞭子。对于我们，面对日新月异的信息技术发展，唯一的选择就是不断学习。

所有参加考试的人都应该把考试当作对自己能力的一次检查，找到自己的不足；所有通过考试的人都应该把这当做一个新的起点，毕竟学无止境。学历、证书都是历史，只是一个缩影，关键还是自己真实的水平。

“通过参加考试，激励自己向前发展”才是正确的心态。

新书上架

▶编辑部点评

C++ 面向对象程序设计(第四版)

中国电力出版社

C++的书汗牛充栋,类似本书的“砖头式教程”也能数出一打。不过这本书还是有一点与众不同,那就是它辉煌的历史。这本书的前三版共销售了20万册,这个成绩在同类书中,只有Bjarne Stroustrup的TC++PL,Stan Lippman的C++ Primer和Deitel父子的C++How to Program可以与之比肩。销量大不一定意味着这本书在技术上有什么过人之处,但是一定意味着这本书思路清晰,行文流畅。

从优点上来讲,其内容丰富,阐述清晰,图文并茂,而且还有精心构思的练习题。第四版中令人称道的一点,就是从第一章开始,就以标准C++的风格讲述。因此,有一位14岁就开始学习这本书的少年赞叹说,这本书是C++入门的最佳引导。

不过经验丰富的程序员和评论者有不同的意见,ACCU的主席Francis Glassborow直言,他不能推荐这本书,因为他发现了书中一些技术性的错误,而且也认为这本书过早进入细节,并不那么适合初学者。

仁者见仁,智者见智,我认为,作为一个稍有基础的C++学习者,这本书不失为一本可供参考的佳作。书中对于一些数据结构的漂亮图示,给我留下深刻印象。这些图示和讲解对于那些基本了解C/C++语法,但是对“编程”这件事情没有多少理解的初学者来说,是非常有帮助的。不过没有必要给与过高的评价,那些已经熟悉C++的熟练程序员不会在这本书中发现什么令人兴奋的东西。

XML 程序的 UML 建模(影印版)

科学出版社

本书通过一个大型综合应用实例,讲解如何将XML和UML结合,创建动态的Web应用程序,实现最优的B2B应用集成。全书探讨了从XML词汇表生成DTD和Schema语言的过程,以及企业级集成和门户的设计方法。每章都附有一个“成功之路”,向读者提供了规划设计阶段的一些重要提示和值得注意的问题。作为Web系统和电子商务领域的系统分析师、事务分析师以及专业设计人员以及XML和UML的初学者,都是一个很好的参考书籍。

点击流数据仓库

电子工业出版社

这本书向你介绍了如何设计和建立一个数据仓库来对网站上的用户行为进行分析。例如:网站上的每个用户的每一次点击都会被记录在网页服务器的日志文件中,通过把这些点击流历史信息加载到一个设计良好的点击流数据仓库中,便可以转化为商业智能的珍藏品。不管进行纯粹的电子商务,还是从事传统的商业活动,只要是把网站作为众多商务渠道之一,那么这本书中所介绍的技术就可以使用。书中解释了构建点击流数据仓库所需要的Web技术和IT基础设施,并对设计、实现点击流数据仓库的整个过程提供全面的指导。

本书主要面向学习或在工作中运用点击流数据仓库技术的教师、学生或工程技术人员。特别适合对数据仓库技术有所了解,但希望进一步提高构建点击流数据仓库能力的应用开发人员。

Windows 游戏编程大师技巧(第二版)

中国电力出版社

本书的一大特色就是随处可见许多有趣但又有一定难度的源程序。作者循循善诱地从程序设计的角度介绍了在Windows环境下进行游戏开发所需的全部知识,包括Win32编程以及DirectX中所有主要组件(包括DirectDraw、DirectSound、DirectInput和DirectMusic)。

Java 数据结构和算法(第二版)

中国电力出版社

此书在第一版的基础上,除了新增章节外,添加了章末问题、试验等内容,这些更加有助于及时的测试自己的理解水平。此书通过使用JAVA语言说明重要的概念,避免了C/C++语言的复杂性,以便集中精力论述数据结构和算法。

投稿信箱:tougao@csdn.net

致作者、读者的一封信

名位亲爱的读者：

《开发高手》从创刊至今已半载，在这半年的时间里，感谢许多作者、读者朋友与我们携手前行！

为了使杂志的内容更加实用和适用，我们需要更多的热心读者、新作者加盟，如果您在开发过程中摸索出一种新的技巧、如果您希望与大家共享开发心得、如果您在开发中遇到了困难，需要朋友指点迷津，都请来信给杂志社。《开发高手》时刻期待着您的参与！

一、投稿范围

根据《开发高手》各个板块及栏目，有针对性的投稿，来信请以“**** 栏目投稿”为主题。如作者文章不适合杂志的栏目，可以文章所属领域为主题，如：“Web 开发投稿”。

二、投稿要求

- 文责自负。作者保证其拥有该作品的完全著作权（版权），该作品不得侵犯任何他人的著作权；
- 全权许可。杂志社有权以任何形式（包括但不限于纸媒体、网络、光盘等介质）使用、编辑、修改该作品，无须另行征得作者同意，无须另行支付稿酬；
- 独家使用。未经杂志社书面许可，稿件在发表后半年内作者不得同意任何单位和个人以任何形式使用（包括但不限于通过纸媒体、网络、光盘等介质转载、张贴、集结、出版）该作品，著作权法另有规定的除外。

三、稿酬标准

文章通过审核并刊登后，杂志社将在发表后一个月内支付稿酬，稿酬标准为：

- 原创稿件稿费为每千字 80-140 元（代码和图片按所占篇幅折字）；
- 翻译稿件稿费为每千字 60-80 元；

对于质量较高的稿件，稿酬将不受以上限制。

四、投稿方法

字数：不限，请参照杂志常用体例，文章不宜有整页代码，代码和图片请勿超过文章的 1/3；

格式：请使用 Word 文件格式，并插入文章配图及代码，以附件形式附在邮件中，如有源代码及可执行文件，请和原文一并压缩为一个文件；

来信时，请在邮件主题中标明投稿栏目或所属技术领域，并在邮件正文中注明作者投稿的文章标题、作者姓名、通信地址及邮政编码、身份证号码、联系电话和电子邮件地址，如需以笔名发表，请附笔名。

投递形式：E-mail 发送给 tougao@csdn.net，如有其他疑问，请来电：010-51661202 转 214。

五、注意事项

对于每个作者的来信，我们都将做到来信必复，并转交相关责任编辑。投稿人在一个月内收到责任编辑用稿通知，如未收到用稿通知，则杂志不予采用，恕不另行通知。无特殊情况，刊发稿件稿费将在发表后一个月内发放，如该月有多篇稿件被杂志采用刊登，则稿费累计发放；

稿件禁止一稿多投，且必须是未在其他媒体发表过的文章，一经发现有从其它媒体摘抄或一稿多投将取消稿费，并刊登谴责声明；

在提交稿件时，请在稿件末尾附上您的个人简介及技术专长领域，以便杂志社相关的活动邀请以及选题约稿。

《程序员》编辑部

我们深知，做出一本让每个开发爱好者都喜欢的杂志，是非常困难的。但我们希望尽量奉献出更好的文章、更实用的技术给读者。

让我们共同关注程序员成长，关注开发实践。
让我们和全国的程序员共同成长！

开发高手