程序设计实践指导书

(C++及 Java 版)

任课教师：郭勇

哈尔滨工业大学软件学院

2015-07-02

目 录

第1章 引　　言

1　本课程的目的及意义

程序设计实践课程以基于项目的教学方法进行设计。以一个项目为主线，重点锻炼学生软件项目开发能力，包括软件需求分析能力、计划管理、时间控制、团队协作、版本控制、软件测试方法以及主要文档的编写。通过本课程的学习和实践，使学生掌握软件项目开发过程中的主要环节，学会软件项目计划的制定、控制软件开发的时间、懂得团队协作的重要性、学会使用版本控制工具进行软件版本的管理，并学基本的软件调试及测试方法，进一步提高学生的编程能力并养成良好的编程习惯。

2　基本要求

以 2-3 人为一个小组，通过对各知识点的训练最后分工合作完成一个项目。各部分要有规范的文档，并且编码符合规范要求，能够按计划在规定时间内完成所开发的项目，所开发项目能正确运行，能够通过小组讨论相互协作解决项目开发过程中的问题，及验收测试的基本过程。

具体功能由学生自行分析，功有模块不少于 10 个。

3　考核基本要求

在指定时间完成规定的内容，整个项目组的代码风格一致、符合制定的规范，程序运行正确。

项目总分：

- 预期的全部功能完成情况、工作量及分工：30 分
- 程序是否正常运行、操作是否方便、数据是否正确：30 分
- 命名是否规范、代码结构是否清晰合理：15 分
- 程序设计实践报告：25

个人分数：项目总分&贡献排名-平时扣分

第2章 要完成的内容

1 项目计划的制定、项目需求分析及设计（6 学时）

1.1 项目计划的制定

项目成功的重要因素之一就是有一个良好的项目计划。制订一个清晰的项目计划需要花一定的时间，有的项目负责人可能会冒险跨过这一阶段直接进入执行阶段。但事先为路线做过准备的旅行者到达目的地总会更快更容易一些。同样花一些时间制订一个清晰的项目计划的负责人也会更快地实现目标。

制定项目计划时要明确角色和职责。项目负责人需要搞清楚和确定的是：谁负责什么事情，还有各利益相关方都要做的决定是什么。

编制范围表。 范围表（scopes tatement）是项目计划中最重要的文件。它不仅可以使利益相关方就这个项目计划达成共识，是利益相关方和赞助商同意补偿购入的基础，而且还可以降低沟通失败的可能性。这份文件通常会随着计划的实施而做相应的调整和改变。项目开发进度表如表 4-1 所示。

表 4-1 项目开发进度安排

日期	要完成的任务	任务执行人

1.2 需求分析及设计中功能结构图

功能结构图是对硬件、软件、解决方案等进行解剖，详细描述功能列表的结构，构成，剖面的从大到小，从粗到细，从上到下等而描绘或画出来的结构图。从概念上讲，上层功能包括 (或控制)下层功能，愈上层功能愈笼统，愈下层功能愈具体。功能分解的过程就是一个由抽象到具体、由复杂到简单的过程。图中每一个框称为一个功能模块。功能模块可以根据具体情况分得大一点或小一点。分解得最小的功能模块可以是一个程序中的每个处理过程，而较大的功能模块则可能是完成某一任务的一组程序。

矩形框:功能模块,功能通常用动词或动词短语描述。

连线：用于表示所属关系。

如图 2-1 及图 2-2 均为功能结构图。

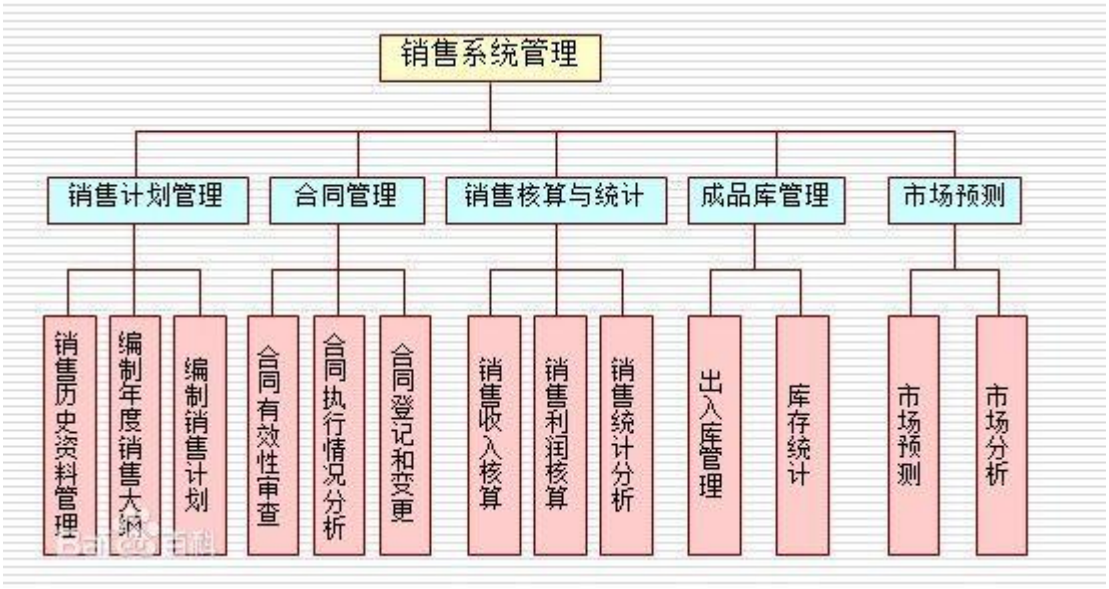


图 2-1 销售系统功能结构图

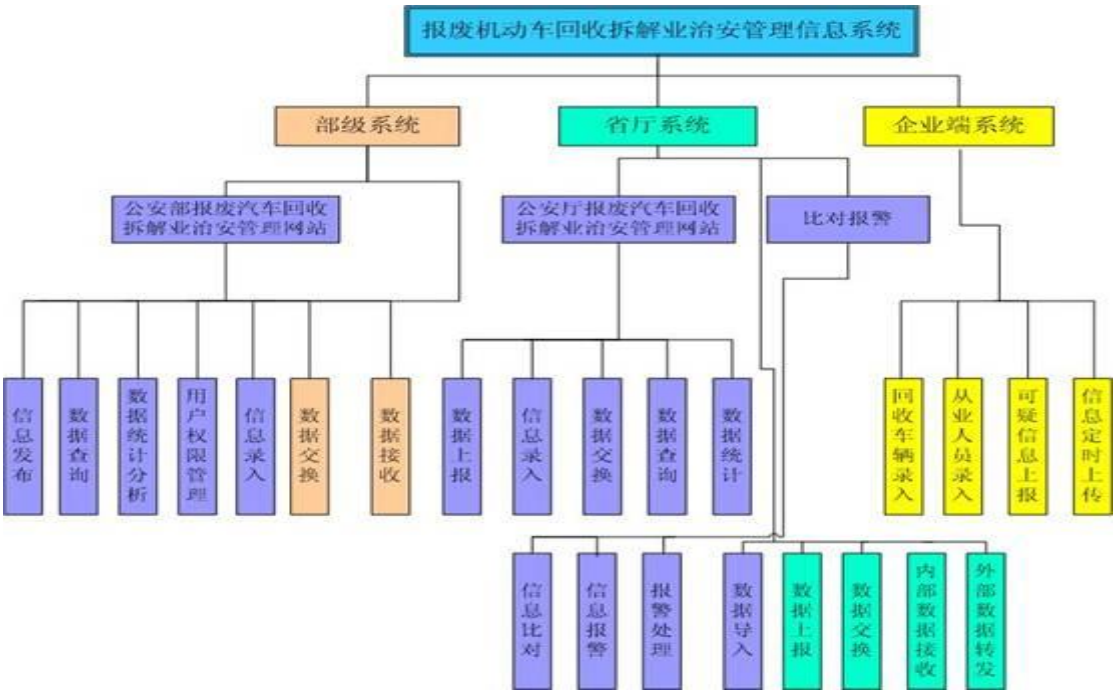


图 2-2 报废机动车回收拆解业治安管理信息系统功能结构图

2 软件编码实现（12 学时）

下面给出主要代码的实现思路，仅供参考，大家可在此基础上充分发挥自己的能力进一步增加系统的功能，力争将所开发的系统做到最好。

2.1 下面内容在 3 学时内完成。

- 根据需求分析得到系统功能，设计界面。

- 定义一个图书类：至少包括书名、书号、数量、入库日期等成员（同学自己分析所有的成员）；
- 定义一个学生类：至少包括姓名、性别、年龄、图书信息等成员；
- 定义一个学生信息录入函数(java中称方法)，用于对学生进行信息录入；采用动态分配的内存（new 和delete），对学生信息进行存储，要根据指定的输入数量进行内容分配；
- 定义二个比较函数(java中称方法)，函数名为comp,该函数即可以对姓名进行比较也可以对年龄进行比较。
- 定义二个排序函数(java中称方法)，所有学生的姓名排序，或按年龄排序
- 定义一个学生数组，将排序结果存入该数组中。

2.2 下面内容在 3 学时内完成。

- 定义管理员类、工作人员类，然后编写一个登录界面，在这个界面中可选择以管理员登录或选择以工作人员登录，登录后才能进行各项操作。
- 定义**学生管理类**，该类中主要包括排序函数、比较函数、信息录入函数（这三个函数可直接将实验一中的移植过来），再加入学生信息显示函数、查找函数、删除函数及修改函数。这三个新增加的函数要完成如下功能：
  - 学生信息显示函数：显示每个学生的所有信息
  - 查找函数：可分别按学号、姓名、班级查询学生，并显示信息
  - 删除函数：按学号或姓名删除某一学生。
  - 修改函数：对某一学生的信息进行修改。
- 工作人员可以对学生信息进行加入、查找、删除、显示、修改、排序
- 管理员除了可以做工作人员的所有工作外，还可以对工作人员的信息进行加入、查找、删除、显示、修改。

- 定义Person类，Person类作为学生类、管理员类以及工作人员类的父类，将管理员类、工作人员类和学生类中共有的属性及方法放入Person类中。

- 操作逻辑的设计：根据不同的登录用户显示相应的菜单。即如果是管理员登录，成功后显示管理员的主菜单,如果是工作人员登录,则成功后显示工作人员的主菜单。
  - 其中管理员的主操作菜单主要包括：学生资料管理、图书管理统计管理、学生查询管理等。
  - 工作人员主菜单主要包括：图书统计管理、学生查询管理等。其中图书统计管理主要包括如下功能：按日期统计查询、按书名统计查询等等（其它项的统计同学自己分析）。

2.3 下面内容在 3 学时内完成

- 在(1)及（2）的基础上，实现数据的文件（也可使用数据库）存贮，将所录入的信息存入文件（或数据库）。
- 并要求学生信息的显示、查找、删除及修改都必须对文件（或数据库）直接操作，不允许一次将所有数据读取到数组或动态分配的内存中再操作。
- 加入管理员及工作人员的登录用户名及密码的存储及修改功能,并将这些信息保存在一个文件中。
- 添加一个帮助菜单，点击如果可以显示程序的操作方法。
- 添加一个数据备份菜单，用于对同学信息进行数据备份。
- 添加一个数据恢复菜单，用于将备份数据恢复到正在使用的文件中。（比如学生信

息备份到某个文件后，正在使用的文件中某个学生的信息被删除了，可使用备份文件将其恢复。恢复时操作人员，要能够通过输入学号来指定恢复那个学生）

2.4 下面内容在 3 学时内完成

进一步完善报告各部分内容，详细分析需求（前期得到的需求可能不全面，因此同学自己要进一步分析图书馆管理系统的其它需求，可参考附录中的需求），完善系统，规范代码。

3 系统整体实现及测试（3 学时）

对系统进行全面系统的测试，进行界面美化，编写软件使用说明，准备系统验收。

4 系统验收（3 学时）

由教师进行验收。验收内容包括：功能完成情况、知识点覆盖情况、代码规范及文档完成情况。

5 提交内容

[1] 上交打印的文档(含成绩评定表)(合格后可进行下面步骤)

[2] 演示程序

[3] 老师提问并检查代码

[4] 提交文件:(文档及源代码) 由组长提交，其他人不要交。

- 压缩到一个 rar 文件中
- 命名方式：班号-组号-组长名.rar(例 11437103-1 组-张三.rar)
- 上面工作都做完，并得到老师确认后就可离开。

第3章 C++编码规范

1 目的、环境

1.1 目的

- 掌握基本的编码规范，并应用于实际程序的编写
- 依据编码规范改写和编写程序。

1.2 环境

- VisualStudio.net 或VC++6.0 或eclipse

2 文件结构

2.1 文件头注释

所有源文件均必须包含一个规范的文件头，文件头包含了该文件的名称、功能概述、作者、版权和版本历史信息等内容。

标准文件头的格式为：

```
/*! @file
/*****************************************************************************
**
模块名          :<文件所属的模块名称>
文件名          :<文件名>
相关文件        :<与此文件相关的其它文件>
文件实现功能    :<描述该文件实现的主要功能>
作者            :<作者部门和姓名>
版本            :<当前版本号>
-----------------------------------------------------------------------------
备注            :<其它说明>
-----------------------------------------------------------------------------
修改记录 :
日 期           版本       修改人                  修改内容
YYYY/MM/DD     X.Y        <作者或修改者名>      <修改内容>
*****************************************************************************
/
```

举例：

```
*****************************************************************************
*
模块名          : student 类定义文件
文件名          : student.cpp
相关文件        : stucent.h
文件实现功能    : student 类中函数及数据的定义
作者            : 比尔盖茨
版本            : 1.0
-----------------------------------------------------------------------------
备注        :
```

```
--------------------------------------------------------------------------------
修改记录 ：
日 期          版本          修改人          修改内容
2010/04/22      1.0          比尔盖茨          创建
*********************************************************************************
*
```

2.2 头文件

用 #include <filename.h> 格式来引用标准库和系统库的头文件(编译器将从标准库目录开始搜索)。

用 #include "filename.h" 格式来引用当前工程中的头文件（编译器将从该文件所在目录开始搜索)。

3 命名规则

- **同一性:**在编写一个子模块或派生类的时候，要保持命名风格在整个模块中的同一性。
- **标识符组成:**标识符采用英文单词或其组合，应当直观且可以拼读，可望文知意，用词应当准确。
- **最小化长度及最大化信息量原则：**在保持一个标识符意思明确的同时，应当尽量缩短其长度。
- **避免过于相似:**不要出现仅靠大小写区分的相似的标识符，例如“i”与“I”，“function”与“Function”等等。
- **避免在不同级别的作用域中重名:**程序中不要出现名字完全相同的局部变量和全局变量，尽管两者的作用域不同而不会发生语法错误，但容易使人误解。
- **正确命名具有互斥意义的标识符:** 用正确的反义词组命名具有互斥意义的标识符，如："nMinValue" 和 "nMaxValue"，"GetName()" 和 "SetName()" ..

3.1 类/结构体

C++类/结构的命名、类的名称都要以大写字母“C”开头，后跟一个或多个单词。为便于界定，每个单词的首字母要大写。

推荐的组成形式 类的命名推荐用"名词"或"形容词＋名词"的形式，例如：“CAnalyzer", "CFastVector"

传统 C 结构体的命名规则为：结构体的名称全部由大写字母组成，单词间使用下划线界定，例如："SERVICE_STATUS", "DRIVER_INFO" ....

3.2 函数

函数的命名 函数的名称由一个或多个单词组成。为便于界定，每个单词的首字母要大写。

函数的命名	函数的名称由一个或多个单词组成。为便于界定，每个单词的首字母要大写。
推荐的组成形式	函数名应当使用"动词"或者"动词＋名词"（动宾词组）的形式。例如："GetName()", "SetValue()", "Erase()", "Reserve()" ....
保护成员函数	保护成员函数的开头应当加上一个下划线“_”以示区别，例如："_SetState()" ....
私有成员函数	类似地，私有成员函数的开头应当加上两个下划线“__”，例如："__DestroyImp()" ....
虚函数	虚函数习惯以“Do”开头，如："DoRefresh()", "_DoEncryption()" ....
回调和事件处理函数	回调和事件处理函数习惯以单词“On”开头。例如："_OnTimer()", "OnExit()"

3.3 变量

变量应该是程序中使用最多的标识符了，变量的命名规范可能是一套 C++命名准则中最重要的部分：

<table>
<tr><td>变量的命名</td><td colspan="4">变量名由作用域前缀＋类型前缀＋一个或多个单词组成。为便于界定，每个单词的首字母要大写。 对于某些用途简单明了的局部变量，也可以使用简化的方式，如：i, j, k, x, y</td></tr>
<tr><td rowspan="9">作用域前缀</td><td colspan="4">作用域前缀标明一个变量的可见范围。作用域可以有如下几种：</td></tr>
<tr><td>前缀</td><td colspan="3">说明</td></tr>
<tr><td>无</td><td colspan="3">局部变量</td></tr>
<tr><td>m_</td><td colspan="3">类的成员变量（member）</td></tr>
<tr><td>sm_</td><td colspan="3">类的静态成员变量（static member）</td></tr>
<tr><td>s_</td><td colspan="3">静态变量（static）</td></tr>
<tr><td>g_</td><td colspan="3">外部全局变量（global）</td></tr>
<tr><td>sg_</td><td colspan="3">静态全局变量（static global）</td></tr>
<tr><td>gg_</td><td colspan="3">进程间共享的共享数据段全局变量（global global）</td></tr>
<tr><td rowspan="7">类型前缀</td><td>前缀</td><td>类型</td><td>内存规格描述</td><td>例子</td></tr>
<tr><td>ch</td><td>char</td><td>8-bit character</td><td>chGrade</td></tr>
<tr><td>ch</td><td>TCHAR</td><td>16-bit character if _UNICODE is defined</td><td>chName</td></tr>
<tr><td>b</td><td>BOOL</td><td>Boolean value</td><td>bEnabled</td></tr>
<tr><td>n</td><td>int</td><td>Integer (size dependent on operating system)</td><td>nLength</td></tr>
<tr><td>n</td><td>UINT</td><td>Unsigned value (size dependent on operating system)</td><td>nLength</td></tr>
<tr><td>w</td><td>WORD</td><td>16-bit unsigned value</td><td>wPos</td></tr>
</table>

	l	LONG	32-bit signed integer	lOffset
	dw	DWORD	32-bit unsigned integer	dwRange
	p	*	Ambient memory model pointer	pDoc
	lp	FAR*	Far pointer	lpDoc
	lpsz	LPSTR	32-bit pointer to character string	lpszName
	lpsz	LPCSTR	32-bit pointer to constant character string	lpszName
	lpsz	LPCTSTR	32-bit pointer to constant character string if _UNICODE is defined	lpszName
	h	handle	Handle to Windows object	hWnd
	lpfn	(*fn)()	callbackFar pointer to CALLBACK function	lpfnAbort
推荐的组成形式	变量的名字应当使用"名词"或者"形容词＋名词"。例如："nCode", "m_nState"，"nMaxWidth"			

3.4 常量

C++中引入了对常量的支持，常量的命名规则如下：

常量的命名	常量名由类型前缀＋全大写字母组成，单词间通过下划线来界定，如：cDELIMITER, nMAX_BUFFER .... 类型前缀的定义与变量命名规则中的相同。

3.5 枚举、联合、typedef

枚举、联合及 typedef 语句都是定义新类型的简单手段，它们的命名规则为：

枚举、联合、typedef 的命名	枚举、联合、typedef 语句生成的类型名由全大写字母组成，单词间通过下划线来界定，如：FAR_PROC, ERROR_TYPE ....

3.6 宏、枚举值

宏、枚举值的命名	宏和枚举值由全大写字母组成，单词间通过下划线来界定，如：ERROR_UNKNOWN, OP_STOP ....

4 代码风格与版式

4.1 总体原则

代码风格的重要性怎么强调都不过分。一段稍长一点的无格式代码基本上是不可读的。

空行的使用

空行起着分隔程序段落的作用。空行得体（不过多也不过少）将使程序的布局更加清晰。空行不会浪费内存，虽然打印含有空行的程序是会多消耗一些纸张，但是值得。所以不要舍不得用空行。

- 在每个类声明之后、每个函数定义结束之后都要加 2 行空行。
- 在一个函数体内，逻辑上密切相关的语句之间不加空行，其它地方应加空行分隔。

语句与代码行

- 一行代码只做一件事情，如只定义一个变量，或只写一条语句。这样的代码容易阅读，并且方便于写注释。
- "if"、"for"、"while"、"do"、"try"、"catch" 等语句自占一行，执行语句不得紧跟其后。不论执行语句有多少都要加 "{ }" 。这样可以防止书写失误。

最大长度

代码行最大长度宜控制在 70 至 80 个字符以内。代码行不要过长，否则眼睛看不过来，也不便于打印。

进和对齐

- 程序的分界符 "{" 和 "}" 应独占一行并且位于同一列，同时与引用它们的语句左对齐。
- "{ }" 之内的代码块在 "{" 右边一个制表符（4 个半空格符）处左对齐。如果出现嵌套的 "{ }"，则使用缩进对齐。
- 如果一条语句会对其后的多条语句产生影响的话，应该只对该语句做半缩进（2 个半角空格符），以突出该语句。

例如：

```
void Function(int x)
{
  CSessionLock iLock(*m_psemLock);

    for (初始化; 终止条件; 更新)
    {
      // ...
    }
    try
    {
      // ...
    }
    catch (const exception& err)
    {
      // ...
    }
    catch (...)
    {
      // ...
    }
}
```

注释	• 注释的位置应与被描述的代码相邻，可以放在代码的上方或右方，不可放在下方。 • 边写代码边注释，修改代码同时修改相应的注释，以保证注释与代码的一致性。不再有用的注释要删除。 • 注释应当准确、易懂，防止注释有二义性。错误的注释不但无益反而有害。 • 当代码比较长，特别是有多重嵌套时，应当在一些段落的结束处加注释，便于阅读。

与常量的比较	应当将常量写在运算符左边，变量写在运算符右边。这样可以避免因为偶然写错把比较运算变成了赋值运算的问题。例如：
	`if (NULL == p) // 如果把 "==" 错打成 "="，编译器就会报错`
	`{`
	`  // ...`
	`}`

4.2 类/结构

类是 C++中最重要也是使用频率最高的新特性之一。类的版式好坏将极大地影响代码品质。

注释头与类声明	与文件一样，每个类应当有一个注释头用来说明该类的各个方面。 类声明换行紧跟在注释头后面，"class" 关键字由行首开始书写，后跟类名称。界定符 "{" 和 "};" 应独占一行，并与 "class" 关键字左对其。
	
	`/*! @class`
	`*********************************************************************************`
	`*******`
	`类名称 : CXXX`
	`功能 : <简要说明该类所完成的功能>`
	`异常类 : <属于该类的异常类（如果有的话）>`
	`备注 : <使用该类时需要注意的问题（如果有的话）>`
	`典型用法 : <如果该类的使用方法较复杂或特殊，给出典型的代码例子>`
	`---------------------------------------------------------------------------`
	`作者 : <xxx>`
	`*********************************************************************************`
	`******/`
	`class CXXX`
	`{`
	`  // ...`
	`};`
	
	对于功能明显的简单类（接口小于 10 个），也可以使用简单的单行注释头：
	`//! <简要说明该类所完成的功能>`
	`class CXXX`
	`{`
	`  // ...`

<table>
<tr><td></td><td>

```
};
```

</td></tr>
<tr><td>继承</td><td>
基类直接跟在类名称之后，不换行，访问说明符（public, private, 或 protected）不可省略。如：

```
class CXXX : public CAAA, private CBBB
{
    // ...
};
```

</td></tr>
<tr><td>正确地使用const</td><td>
把不改变对象逻辑状态的成员都标记为const成员不仅有利于用户对成员的理解，更可以最大化对象使用方式的灵活性及合理性（比如通过 const 指针或 const 引用的形式传递一个对象）。
如果某个属性的改变并不影响该对象逻辑上的状态，而且这个属性需要在 const 方法中被改变，则该属性应该标记为 "mutable"。
例如：

```
class CString
{
public:
    //! 查找一个子串，find()不会改变字符串的值所以为 const 函数
    int find(IN const CString& str) const;
    // ...

private:
    // 最后一次错误值，改动这个值不会影响对象的逻辑状态，
    // 像 find()这样的 const 函数也可能修改这个值
    mutable int m_nLastError;
    // ...
};
```

</td></tr>
<tr><td>初始化列表</td><td>
应当尽可能通过构造函数的初始化列表来初始化成员和基类。
例如：

```
CXXX::CXXXX(IN int nA, IN bool bB) : m_nA(nA), m_bB(bB)
{
    // ...
};
```

初始化列表的书写顺序应当与对象的构造顺序一致，即：先按照声明顺序写基类初始化，再按照声明顺序写成员初始化。
如果一个成员 "a" 需要使用另一个成员 "b" 来初始化，则 "b" 必须在 "a" 之前声明，否则将会产生运行时错误（有些编译器会给出警告）。
例如：
</td></tr>
</table>

```
// ...
class CXXXX : public CAA, public CBB
{
    // ...
    CYY m_iA;
    CZZ m_iB;  // m_iA 必须在 m_iB 之前声明
};

CXXX::CXXXX(IN int nA, IN int nB, IN bool bC)
    : CAA(nA), CBB(nB), m_iA(bC), m_iB(m_iA) //  先基类，后成员，
                                //  分别按照声明顺序书写
{
    // ...
};
```

4.3 函数

函数原型	函数原型的格式为：
	[存储类] 返回值类型 [名空间或类]::函数名(参数列表) [const 说明符] [异常过滤器]
	例如：
	`static inline void Function1(void)`
	`int CSem::Function2(IN const char* pcName) const throw(Exp)`
	其中：
	• 以 "[ ]" 括住的为可选项目。
	• 除了构造/析构函数外，"返回值类型" 和 "参数列表" 项不可省略（可以为 "void" ）。
	• "const 说明符" 仅用于成员函数中
	• "存储类", "参数列表" 和 "异常过滤器" 的说明见下文

函数声明	函数声明的格式为： //! 函数功能简单说明（可选） 函数原型; 例如： //! 执行某某操作 static void Function(void); 函数声明和其它代码间要有空行分割。 声明成员函数时，为了紧凑，返回值类型和函数名之间不用换行，也可以适当减少声明间的空行。
函	函数定义使用如下格式：

数定义

```
/*! @function
*********************************************************************************
*********
<PRE>
函数名   :<函数名>
功能     :<函数实现功能>
参数     :<参数类表及说明（如果有的话），格式为：>
          [IN|OUT] 参数 1： 参数说明
          [IN|OUT] 参数 2： 参数说明
          ...
返回值   :<函数返回值的意义（如果有的话）>
抛出异常 :<可能抛出的异常及其说明（如果有的话），格式为：>
          类型 1： 说明
          类型 2： 说明
          ...
-------------------------------------------------------------------------
备注     :<其它注意事项（如果有的话）>
典型用法 :<如果该函数的使用方法较复杂或特殊，给出典型的代码例子>
-------------------------------------------------------------------------
作者     :<xxx>
</PRE>
*********************************************************************************
*********/
函数原型
{
   // ...
}
```

对于返回值、参数意义都很明确简单函数（代码不超过 20 行），也可以使用单行函数头：

```
//! 函数实现功能
函数原型
{
   // ...
}
```

函数定义和其它代码之间至少分开 2 行空行。

代码块

如果函数体中的代码较长，应该根据功能不同将其分段。代码段间以空行分离，并且每段代码都以代码段分割注释作为开始。 例如：

```
void CXXX::Function(IN void* pmodAddr)
{
if (NULL == pmodAddr)
   return;

{
    CSessionLock iLock(*sm_hSELock);

   //
```

```
//=============================================================================
    // = 判断指定模块是不是刚刚被装入，由于在 NT 系列平台中，“A”系列函数都是
    // = 由“W”系列函数实现的。所以可能会有一次 LoadLibrary 产生多次本函数调
    // = 用的情况。为了增加效率，特设此静态变量判断上次调用是否与本次相同。
    static PVOID pLastLoadedModule = NULL;
    if (pLastLoadedModule == pmodAddr)
    {
      return;  // 相同，忽略这次调用
    }
    pLastLoadedModule = pmodAddr;

    //
//=============================================================================
    // = 检查这个模块是否在旁路模块表中
    stringEx stModName;
    if (!BaiY_IMP::GetModuleNameByAddress(pmodAddr, stModName))
    {
      return;
    }

    if (CHookProc::sm_sstByPassModTbl.find(stModName)
       != CHookProc::sm_sstByPassModTbl.end())
    {
       return;
    }

   //
//=============================================================================
    // = 在这个模块中 HOOK 所有存在于 HOOK 函数表中的函数
    PROCTBL::iterator p;
    for (p=sm_iProcTbl.begin(); p!=sm_iProcTbl.end(); ++p)
    {
       p->HookOneModule(pmodAddr);
    }
  } // SessionLock
}
```

明显地，如果需要反复用到一段代码的话，这段代码就应当作为一个函数实现。

当一个函数过长时（超过 100 行），为了便于阅读和理解，也应当将其中的一些代码段实现为单独的函数。

何时使用宏	应当尽量减少宏的使用，在所有可能的地方都使用常量和内联函数来代替

	宏 。
边界效应	使用宏的时候应当注意边界效应，例如，以下代码将会得出错误的结果： #define PLUS(x,y) x+y cout << PLUS(1,1) * 2; 以上程序的执行结果将会是 "3"，而不是 "4"，因为 "PLUS(1,1) * 2" 表达式将会被展开为："1 + 1 * 2"。 在定义宏的时候，只要允许，就应该为它的替换内容括上 "( )" 或 "{ }"。例如： #define PLUS(x,y) (x+y) #define SAFEDELETE(x) {delete x; x=0}

5 异常

异常使 C++的错误处理更为结构化；错误传递和故障恢复更为安全简便；也使错误处理代码和其它代码间有效的分离开来。

何时使用异常?

异常机制只用在发生错误的时候，仅在发生错误时才应当抛出异常。这样做有助于错误处理和程序动作两者间的分离，增强程序的结构化，还保证了程序的执行效率。

确定某一状况是否算作错误有时会很困难。比如：未搜索到某个字符串、等待一个信号量超时等等状态，在某些情况下可能并不算作一个错误，而在另一些情况下可能就是一个致命错误。

有鉴于此，仅当某状况必为一个错误时（比如：分配存储失败、创建信号量失败等），才应该抛出一个异常。而对另外一些模棱两可的情况，就应当使用返回值等其它手段报告 。

第4章 VC++单步调试方法

1 目的、环境

1.1 目的

- 熟悉VC++开发环境。
- 学会使用单步跟踪方法对程序进行调试

1.2 环境

- VisualStudio.net 或VC++6.0

2 知识要点：

VC 调试技术程序出错的类型大致可以分为两种，**语法错误**和**逻辑错误**。语法错误可以通过编译器的出错信息得到纠正。然而逻辑错误则不能，所以各大 IDE（集成开发环境）中都提供了 debug 功能，用来分析和排除程序中的逻辑错误，排除逻辑错误的过程又称调试（或 debug）。

下面以 VC++6.0 的调试环境做介绍。

2.1 常用的调试命令主要有：

- **step into 命令** 快捷键：F11 单步执行每条语句，在遇到函数的时候，系统将进入函数，单步执行其中的语句。
- **step over 命令 快捷键**：F10 单步执行每条语句，但在遇到函数时候，自动执行其中的内容，而不进入函数内部单步执行。
- **run to cursor 命令 快捷键**：Ctrl+F10 系统将自动执行到用户光标所指的语句前。（这个功能很有用，可以将精力集中到有问题的地方，从而节省调试时间）
- **Go 命令 快捷键**：F5 系统将编译，连接，自动运行程序，但是会在程序设置了断点(breakpoint)处停下。
- BuildExcute 命令快捷键：Ctrl+F5 系统将编译，连接，运行编译好的程序代码，因此不会在断点处停留，但是在程序执行结束之后，系统会给一个 Pause，以方便用户观察输出结果。
- **Stop debug 命令 快捷键**：Shift+F5 本命令是用来终止动态调试过程的。

2.2 动态调试的主要方法

断点（breakpoint）是指在调试过程中，只要运行到断点处，系统就会自动停下（除非是使用 bulidexcute 命令，但那是在执行编译好的代码，在严格意义上说，这不能算是一个调试命令），通常和 go 命令和 step over 命令配合在一起使用。

设置断点的方法：在程序代码中，移动到需要设置断点的那一行上，按 F9 键，你可以看到代码行的左端出现了一个红色的圆点——那是 VC++中断点的标志，以后程序在调试过程中，每次执行到这里，都会停下，方便用户观察当前变量的状态。

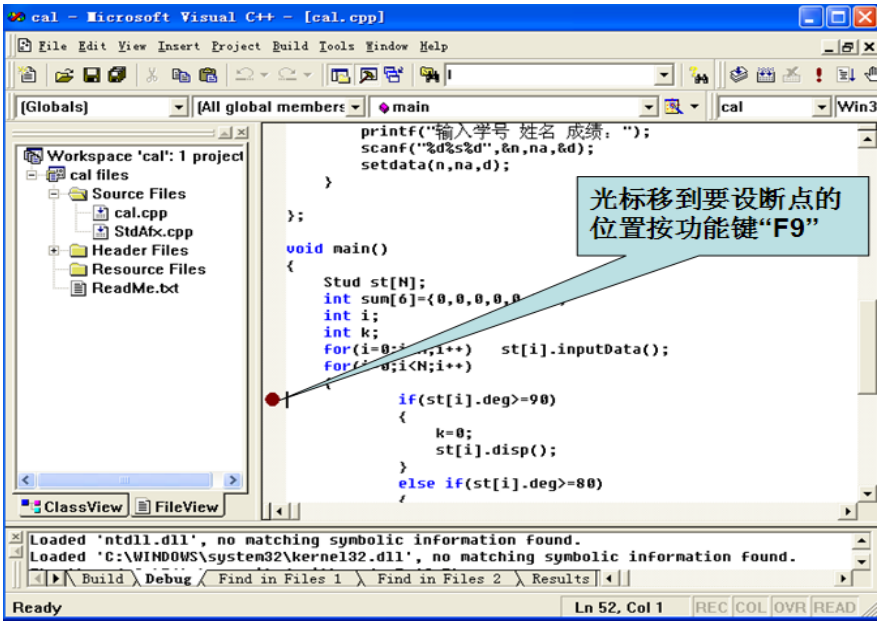


图 1 断点设置方法

去除断点:命令与设置断点的命令相同：在已设置断点的地方，再按一次 F9 键，左端的红色圆点就消失，断点被去除了。

有的时候，我们并不是不需要断点，而是"暂时"不需要它，这时可以在已设置断点的地方，按 Ctrl+F9 键，你可以看到原本实心的圆点变成了一个空心的圆圈——**断点暂时失效了**。恢复断点功能也是按 Ctrl+F9。这个功能在有多个断点的时候尤其有用。

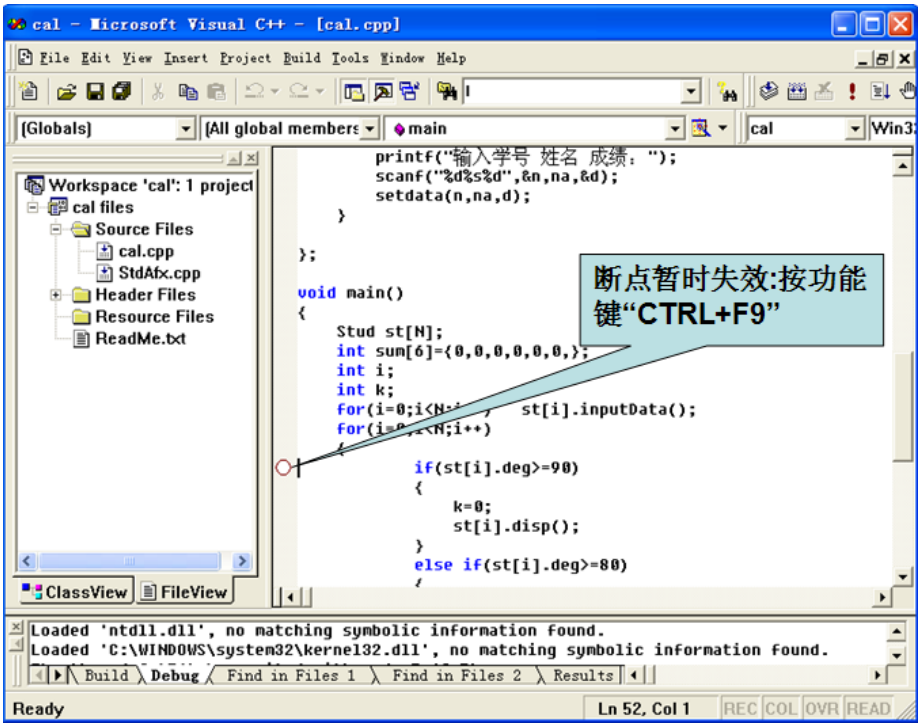
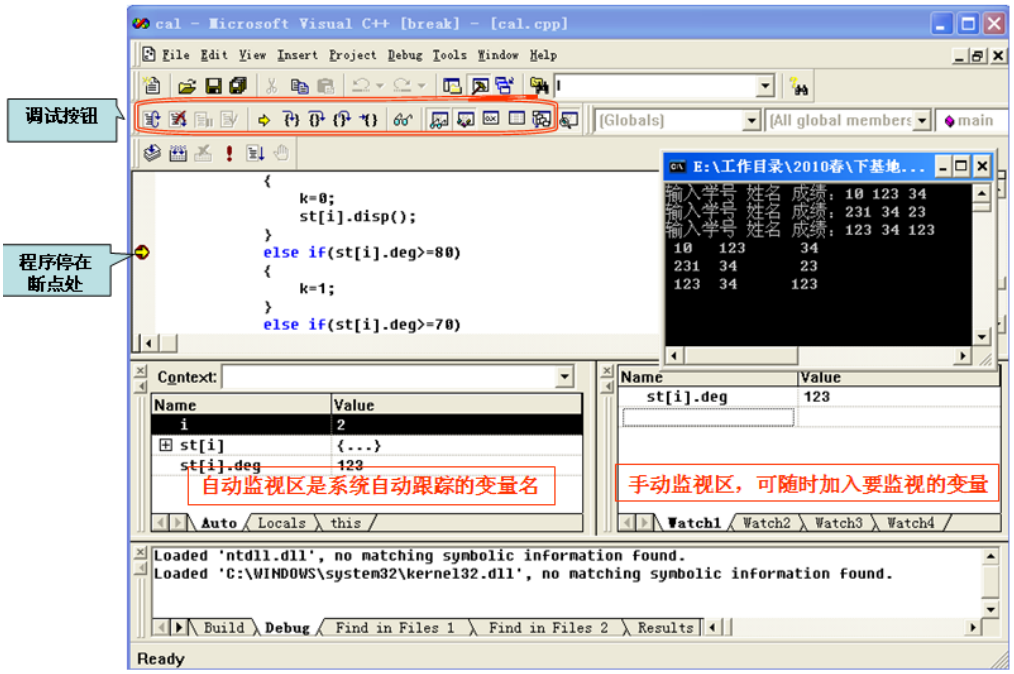


图 2 断点暂时失效

条件断点技术——其实就是在一些分支语句内部设置断点，这个技术很实用，尤其在程序的某个分支部分发生问题的时候。关于断点设置在哪里，则跟据实际需要，基本的原则就是，不要连续设置断点，在需要连续观察的地方，应当使用 step over 或者 step into 命令。

2.3 调试过程

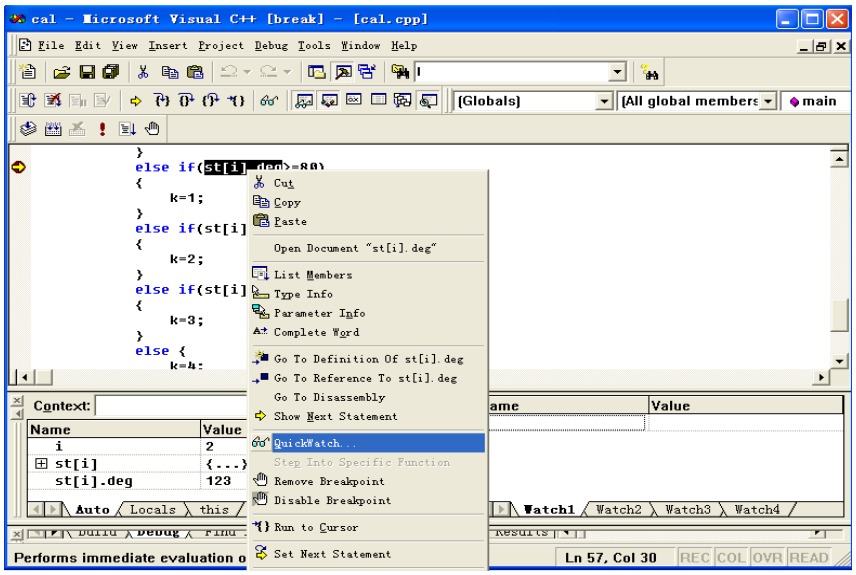
启动程序进入调试状态，设置好断点后，就可以启动程序接功能键“F5”进入调试状态，如图 3 为程序运行到断点处后时的界面。



如图 3 为程序运行到断点处后时的界面

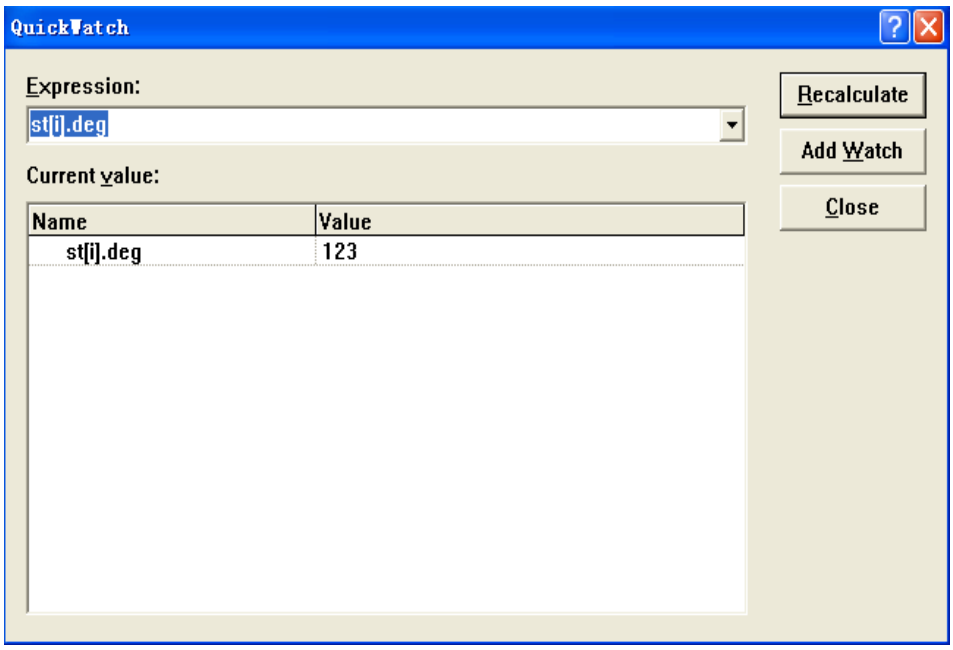
可通过界面中的调试按钮进行程序调试，也可以使用前面介绍的快捷键。

手动加入要监视的变量

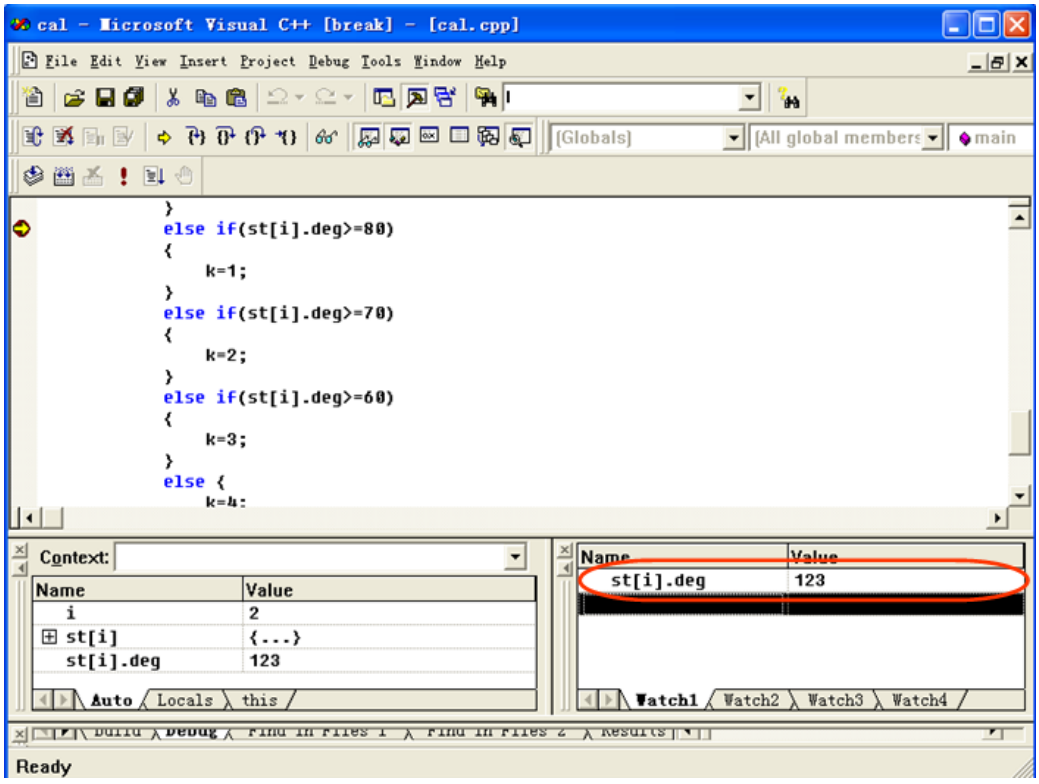


如图 4 手动加入要监视的变量

将光标移到要监视的变量处，点击鼠标右键，出现如图所示弹出菜单，选择“QuickWatch”项。出现如图 5 所示对话框，按“Add Watch”，便将选中的变量加入到了变量监视窗中。如图 6 所示，st[i].deg 已加入监视窗中，并显示出当前的取值。



如图 5 加入变量对话框



如图 5 加入变量后的监视框

watch（监视变量）:在程序编译通过以后，当使用了 step into,step over,run to cursor, go 命令使系统在程序执行的过程中停下之后，系统就会进入调试状态。

调试过程中，你的程序执行窗口会调到后台，而系统窗口中会显示你的程序，其中的黄色箭头指向的是系统下一步将要执行的语句。

而系统窗口下面的那个监视窗口就是我们将要介绍的重点．watch 窗口被左右分成了

两个部分，左面的那部分我们姑且称之为"自动监视区"（即 variable 窗口），而右面的我们称之为"手动监视区"。（即 watch 窗口）自动监视区是系统自动跟踪的变量名。

系统默认显示 auto 标签，那是显示在上一步执行过程中，程序中发生改变的变量。locals 标签跟踪的是某一个函数中的所有变量。 上面的 find sourse 组合框中指示的是当前在 locals 标签下在跟踪的变量是属于哪一个函数的。

说明：当 find sourse 组合框中的内容变成灰色时，说明系统正在运行程序，或者等待输入端的数据（通常会是这个情况），此时应当注意程序执行窗口中的内容。

通常仅仅只有自动监视区所监视的变量是不够的，有时我们需要自己定义一些需要跟踪的变量——这个时候我们就要在手动监视区中输入变量名（也可以是系统认为合法的表达式）来跟踪我们需要的值。注意：当用户定义了一个指向数组的 watch 之后，在变量的左边会出现一个小的'+'号，表示这个数组可以"展开"——显示其中每一个下标所指示的内容，这与其他高级语言的 IDE 有些不同。

值得一提的是 VC++的一个人性化设置：在用户定义的变量很多时，往往需要通过滚屏才能看到所有的变量——VC++在手动监视区中设定了 4 个标签以方便用户的使用，在这四个标签的功能是一样的。

第5章 VC++中使用 MFC 编程

1 目的、环境

1.1 目的

- 熟悉VC++中各种资源。
- 学会快速建立MFC应用程序
- 学会基本的单文档程序中菜单等的操作
- 学会桌面应用程序的基本编写方法
- 掌握单文档、多文档及对话框应用程序的基本结构
- 初步了解消息机制及事件的映射。

1.2 环境:

- VisualStudio.net 或VC++6.0

2 知识要点:

Windows 应用程序一般都包含有众多的图形元素，例如光标、位图、对话框等，在 Windows 环境下每一个这样的元素都是作为一种可装入应用程序的资源来存放。所谓资源，就是指用户可从中获得某种信息或进行某种操作的界面元素。

在Visual C++应用程序中，资源与源代码是分离的。一方面，多个应用程序可以引用同一资源定义；另一方面，可以在不影响源代码的情况下编辑资源，或同时开发资源与源代码，缩短应用程序的开发时间。

Windows 环境下的资源主要有以下几类：

- 加速键（Accelerator）：一系列按键组合，一般与菜单命令相连，作为选择菜单命令的快速方法，被应用程序用来引发一个动作。
- 工具条（Toolbar）：包含多个按钮的组合，也被用来作为快速选择菜单命令的方法。
- 光标（Cursor）：32*32 像素的位图，指示鼠标的当前位置。
- 对话框（Dialog）：包含多种控件的窗口，与用户完成交互功能。
- 图标（Icon）：代表最小化窗口的位图。
- 菜单（Menu）：以可视的方式提供了对应用程序功能的选择。
- 字符串列表（String Table）：包含一系列的格式化文本。
- 版本信息（Version）：定义应用程序的版本，包含一系列格式化文本。

这些所有类型的资源都可以由Visual C++提供的资源编辑器进行可视的编辑。相应于不同类型的资源，Visual C++提供了不同种类的编辑器，如对话框编辑器、菜单编辑器、工具条编辑器等等，这些编辑器的具体的使用方法将在介绍有关内容的同时加以介绍。

3 快速建立 MFC 应用

在VC6.0 的集成开发环境的“File”菜单下有“New”命令，用来建立各种新的内容。选择“File”菜单下的“New”命令，弹出“New”对话框，如图1-1 所示。

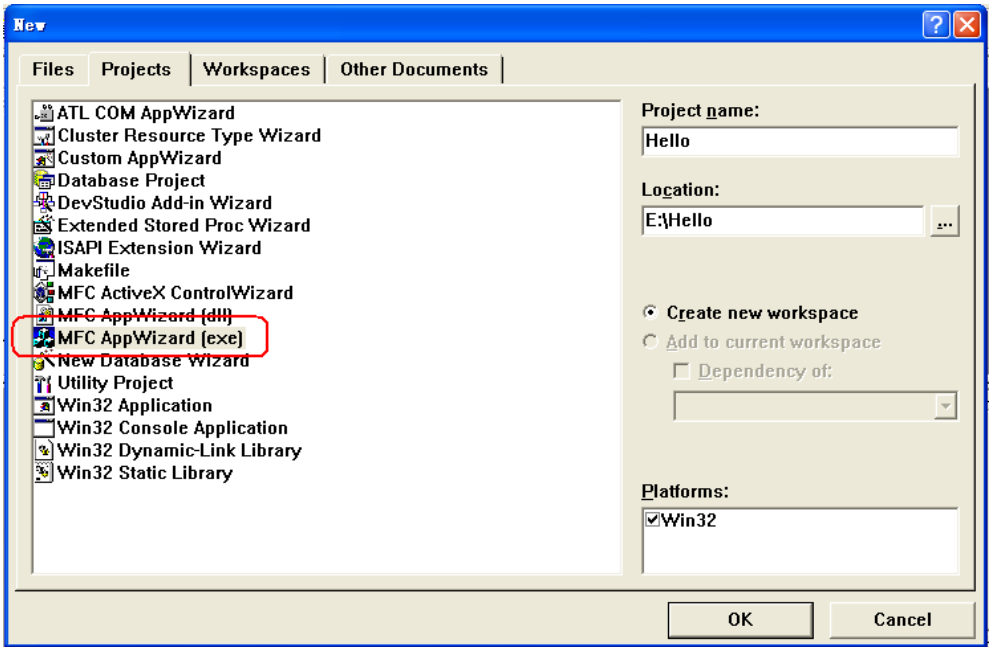


图 1-1 项目建立窗口

(1). 选择应用程序界面的类型

在“MFC AppWizard – Step 1”对话框中，主要是选择应用程序界面的类型和应用程序资源的语言种类，如图1-2 所示。

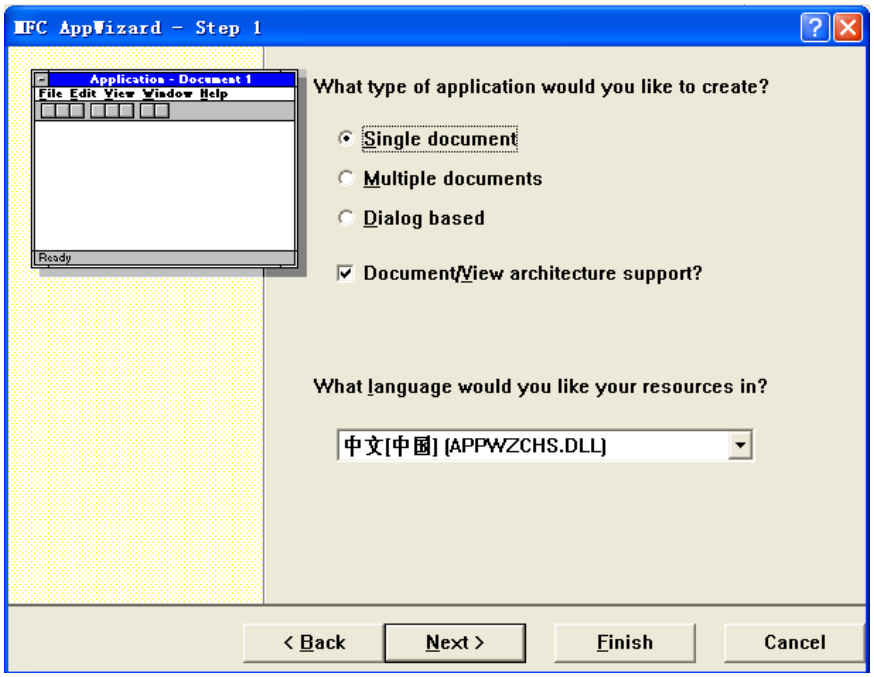


图 1-2 应用程序类型选择窗

在“Step 1”对话框中，有三种应用程序类型可供选择：

■ 单文档界面（SDI：Single Document Interface）应用程序：在程序的运行过程中，每次

只能打开一个文档。如果选择程序“File”菜单下的“New”命令或“Open”命令，程序首先将关闭当前正在使用的文档，然后才执行建立新文档或打开旧文档的操作。Windows 系统附件中的记事本就是一个典型的单文档界面应用程序。

■ 多文档界面（MDI：Multiple Document Interface）应用程序：在程序运行时，允许同时打开两个或两个以上的文档。程序一般拥有Window 菜单以调整各文档窗口之间的位置关系。Office 系列中的Word组件就是一个典型的多文档界面应用程序。

■ 对话框形式的应用程序：程序的主要界面表现为一个对话框，可以使用对话框编辑器方便地进行设计。作为一个简单的例子程序，“Hello，World！”程序只需要单文档界面。

在这里可选择“Single Document”选项，然后选“Finish”按钮即可，如想进一步细化你的应用程序类型，可选“Next”按钮，**※如想了解更多信息，请参考“VC++简明教程.PDF”文件中第19-24页。**

(2)．程序编辑管理界面

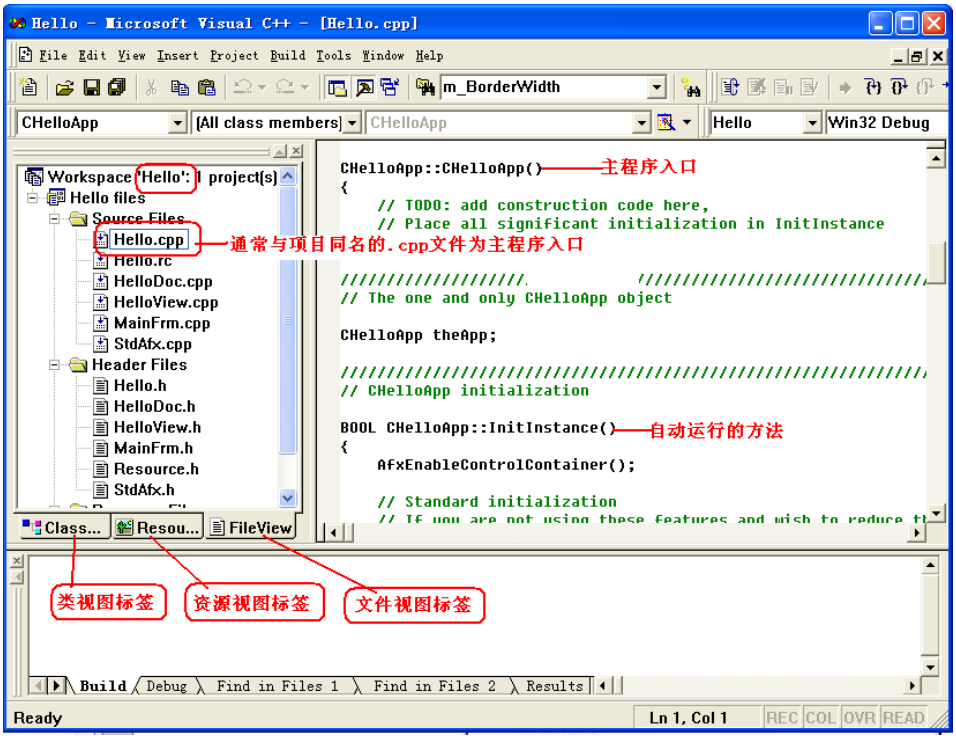


图 1-3 应用程序编辑窗口

程序说明，在MFC 应用程序的**文档/视图**结构

- **CHelloApp：应用程序类**
- **CMainFrame：主框架窗口类**
- **CHelloDoc：文档类，**文档类用来完成应用程序数据的存取工作。
- **CHelloView：视图类，**视图类完成显示数据和与使用者交互的工作。s
- **CAboutDlg：版本信息对话框类**

。

InitInstance()函数最后的工作就是显示和更新应用程序的窗口。如果InitInstance()函数中所有的初始化工作均正确地进行，函数将返回TRUE。在执行完初始化工作后，应用程序将进入消息循环，接收和处理系统发送来的各种消息，直到应用程序被关闭。

■ 详细信息请参考“VC++简明教程.PDF”文件中第29-39页。

下面是自动生成的程序运行界面。

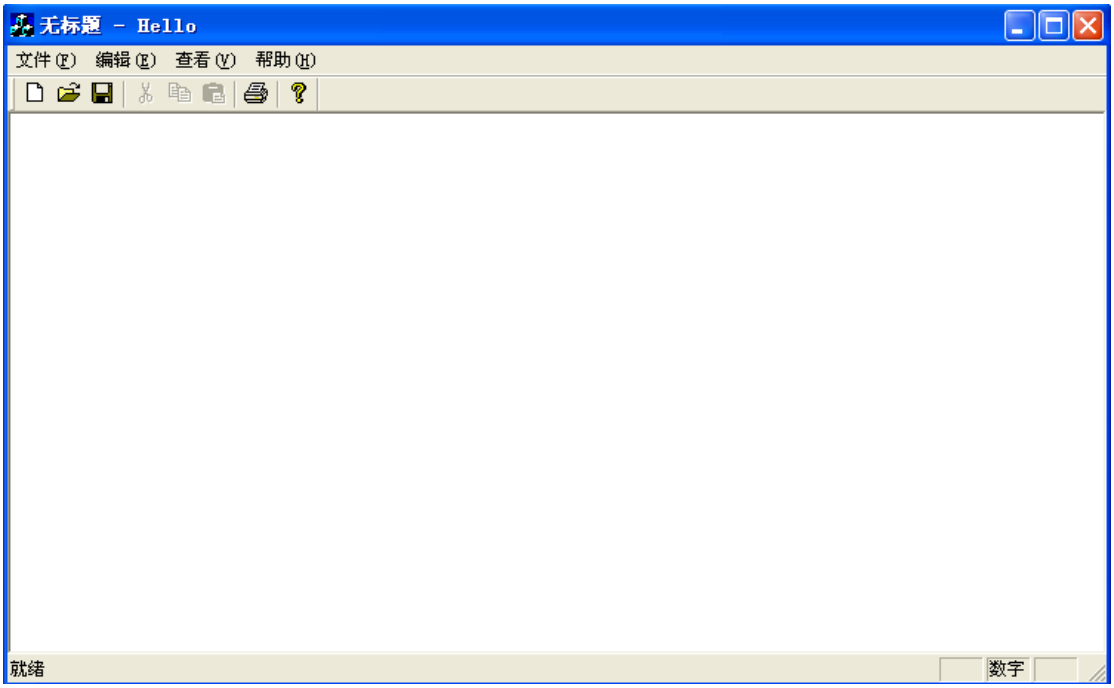


图 1-4 程序运行界面

(3)．给程序加入代码

- 修改菜单项加入新菜单

按图1-5所示的操作加入新菜单，也可直接删除有来不用的菜单。

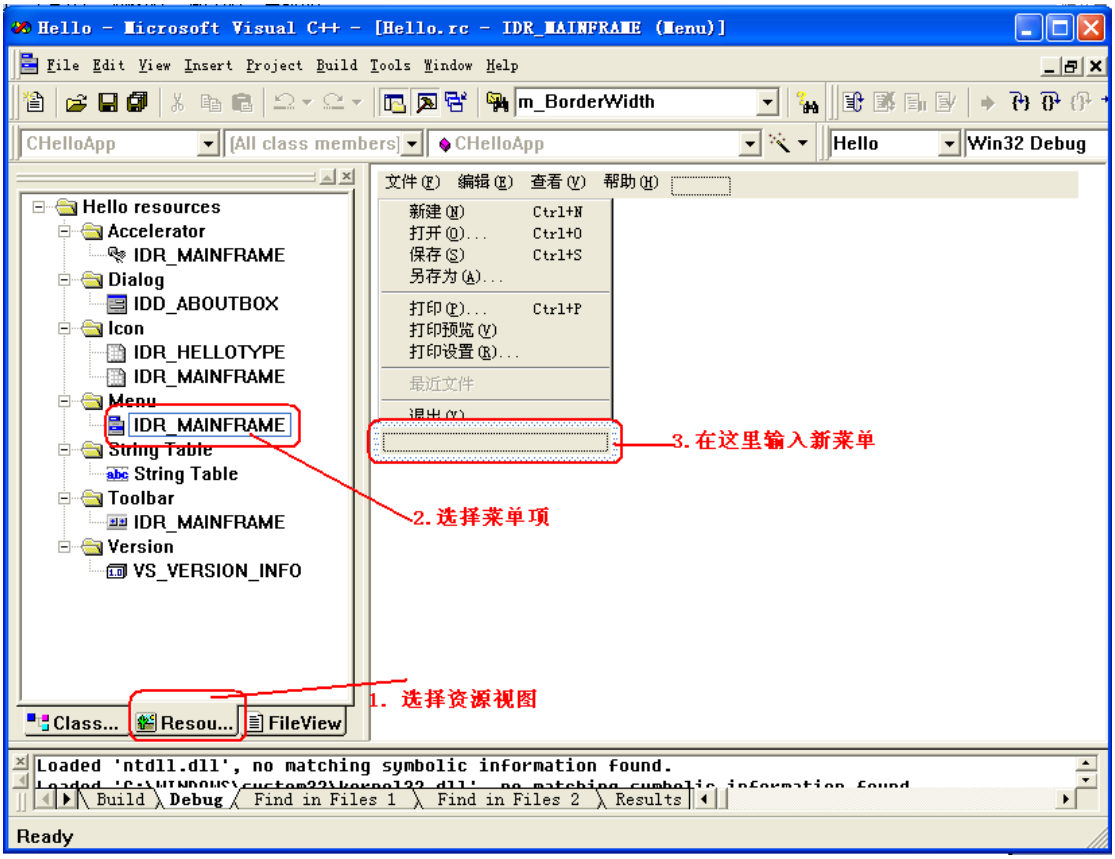


图 1-5 菜单编辑界面-1

当进行菜单名称输入时，会弹出如图1-6所示的菜单，按要求将名称和ID输入。菜单ID的命名一般是ID_MENUITEM_XXXX。其中“XXXX”是与菜单名称相关的单词。

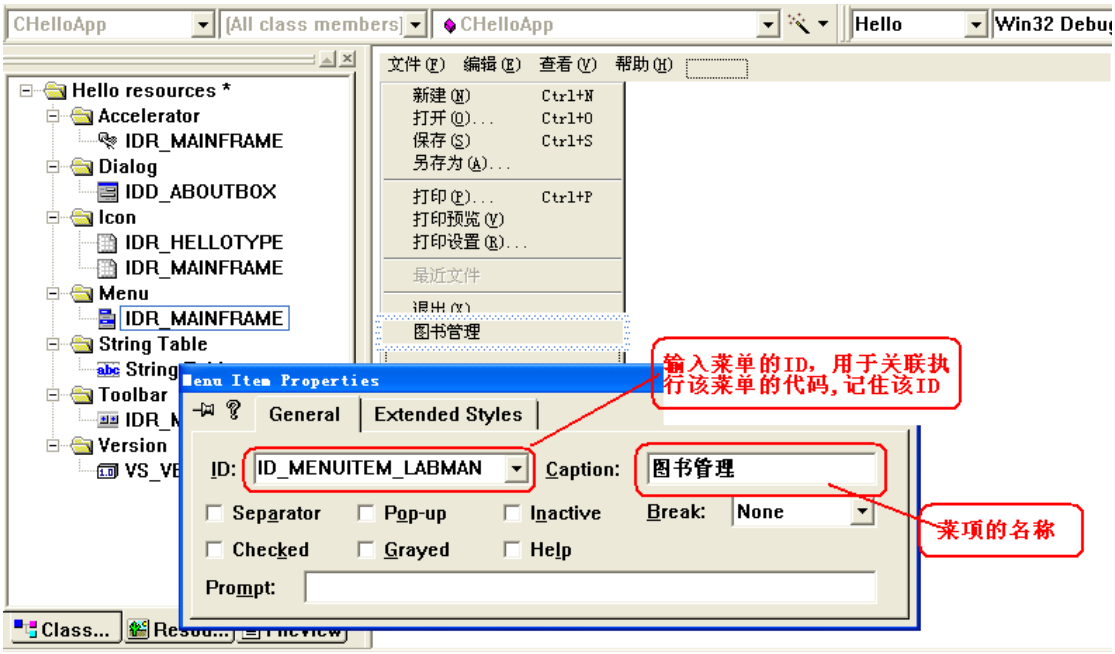


图 1-6 菜单编辑界面-2

- 给菜单项加入响应函数

进入代码编辑窗口，点击右键，如图 1-7 所示，选择类向导“ClassWizard”项，

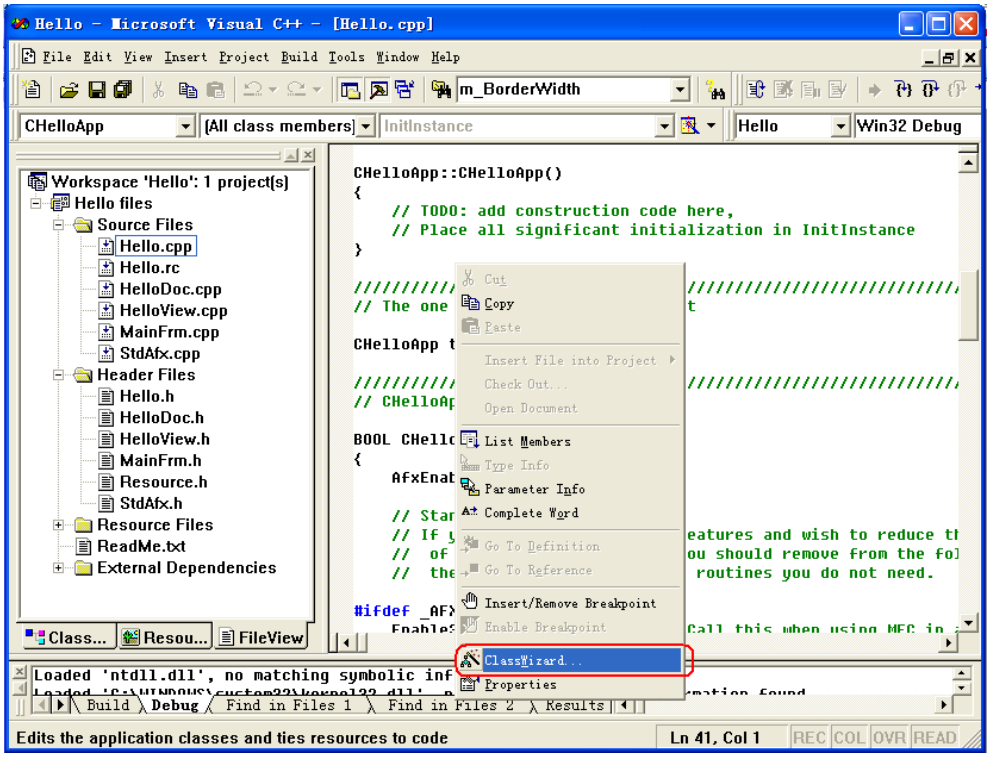


图 1-7 类生成向导

会弹出如下所示的 MFC 类向导对话框，然后按如图 1-8 所示步骤操作。

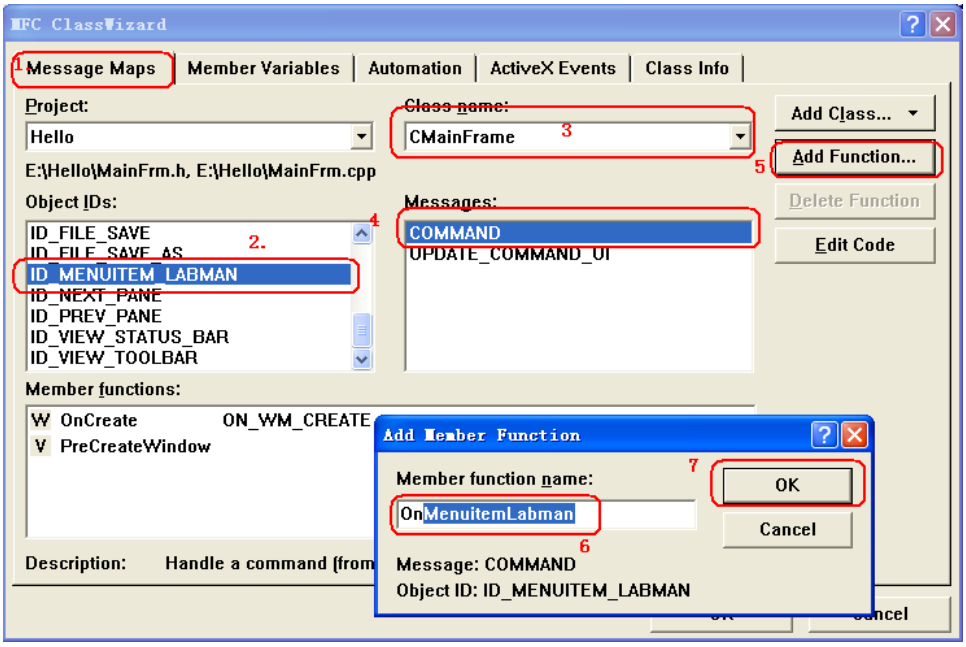


图 1-8 添加菜单函数

1. 确认选择的是 Message Maps 标签。
2. 在 Object IDs 窗口中找到你加入的菜单的 ID。
3. 在 Class Name 组合框中选择将菜单对应的代码加入的文件。最终代码会加入到你选择的这个文件中。
4. 在 Nessages 窗口中选择 COMMAND 项。
5. 点击“Add Function”按钮
6. 在“Member function name:”下面的编辑框中输入生成的函数的名称，也可以用默认的名字。
7. 点击 OK 按钮，函数加入完毕。显示 1-9 所示对话框。

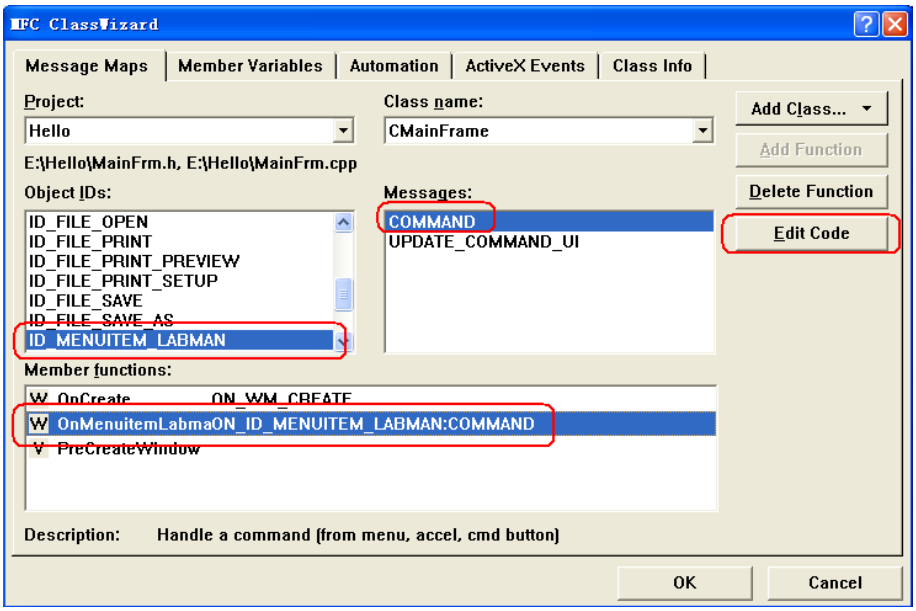


图 1-9 菜单函数添加完毕

此时“Edit Code”按钮为可操作状态，表明函数已加入完毕，可点击此按钮进入代码中，

加入必要的代码。如图 1-10 代码编辑窗口。

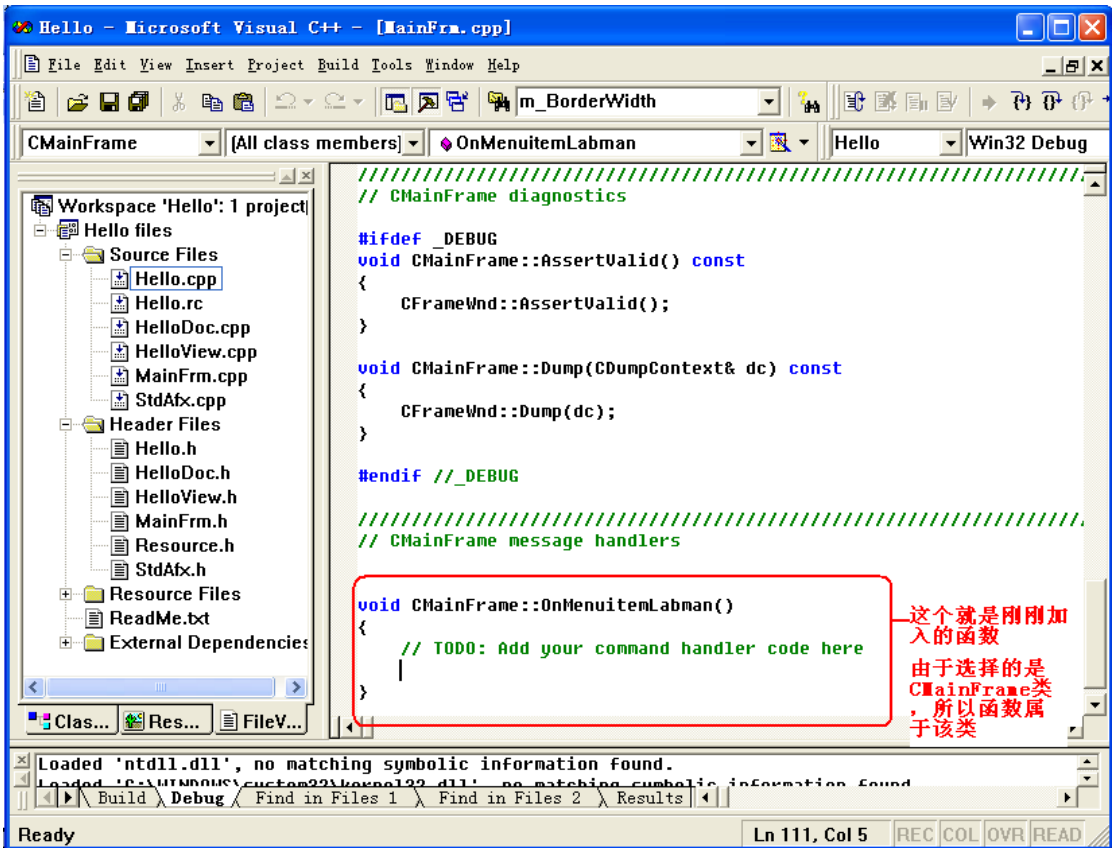


图 1-10 代码编辑窗口

加入代码：MessageBox("执行了图书管理菜单","你好",MB_OK);在图中所示位置，然后运行程序，可观察到菜单执行结果。

4 加入对话框资源

1. 下面介绍一下如何加入一个对话框资源。按如图所示操作，选择 Insert。

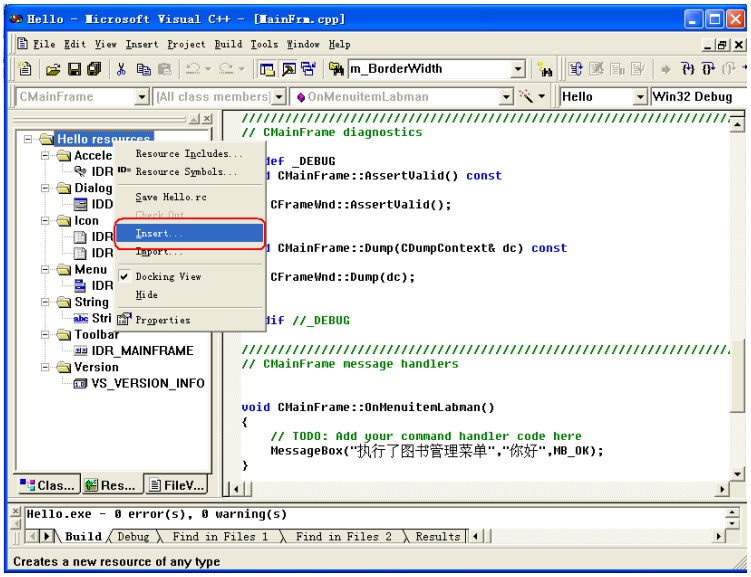


图 2-1 添加对话框

弹出如图 2-2 所示的对话框，选择 Dialog 然后按 New 按钮，就会出现如图 2-3 所示的界面，对话框 IDD_DIALOG1 已加入到资源中了。

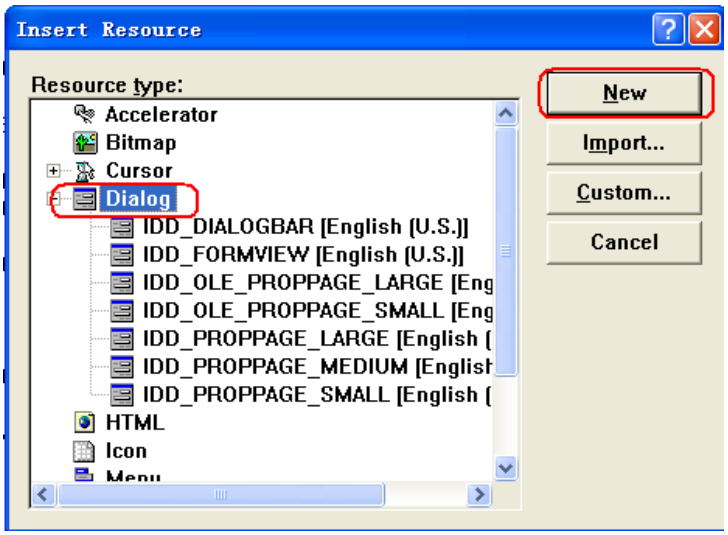


图 2-2 添加对话框操作窗口

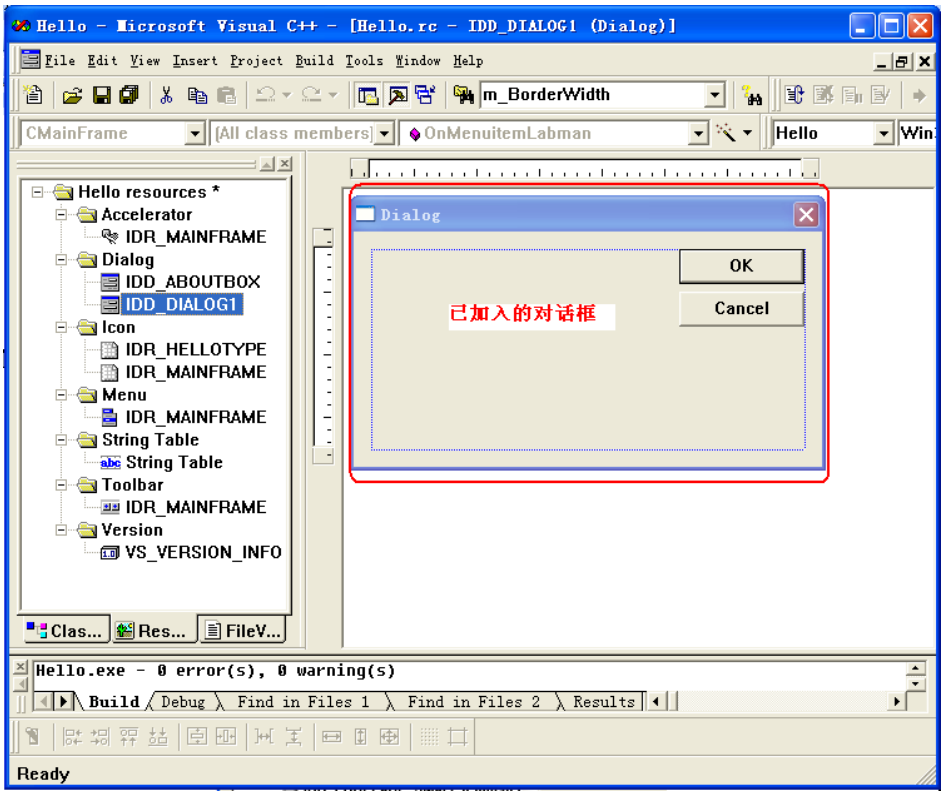
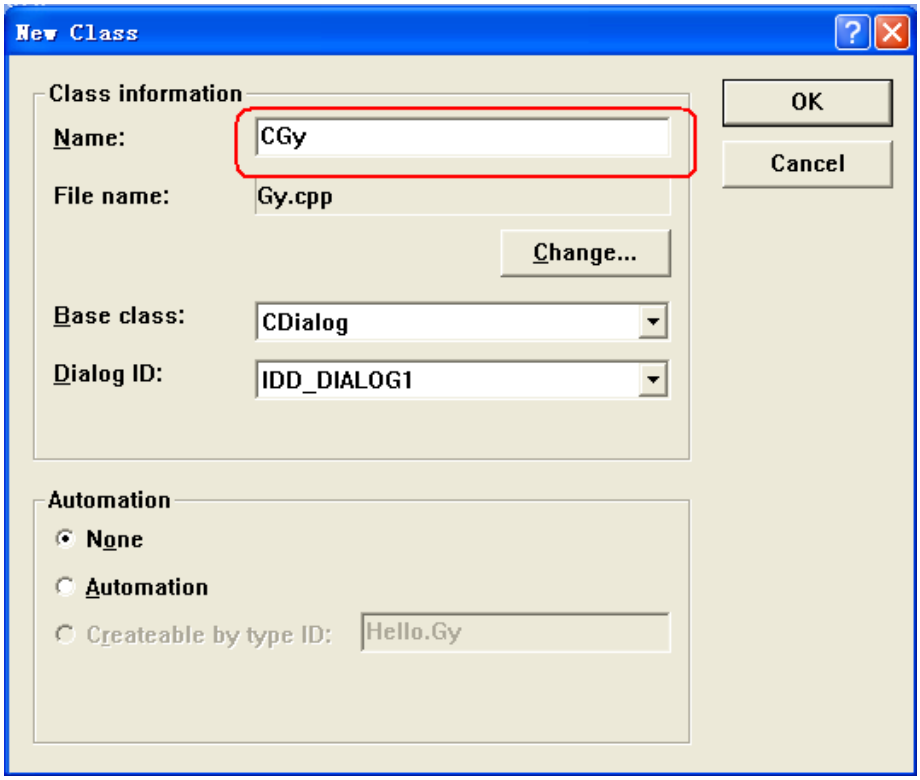


图 2-3 添加对话框操作窗口

现在该对话框还没有对应的类，所以必须加入必要的类才能对其进行各种操作。双击 OK 按钮，选择“Create a new class”，然后选择 OK 加入类，出现如图 2-4 所示界面。输入有一定含意的类的名称，如图 2-4 所示输入了 CGy(这个类名要根据实际情况自己定义),此时系统会自动指定源程序文件名为 Gy.cpp。



2-4 创建对话框对应的类

2.通过编程在菜单操作中打开此对话框,方法如下

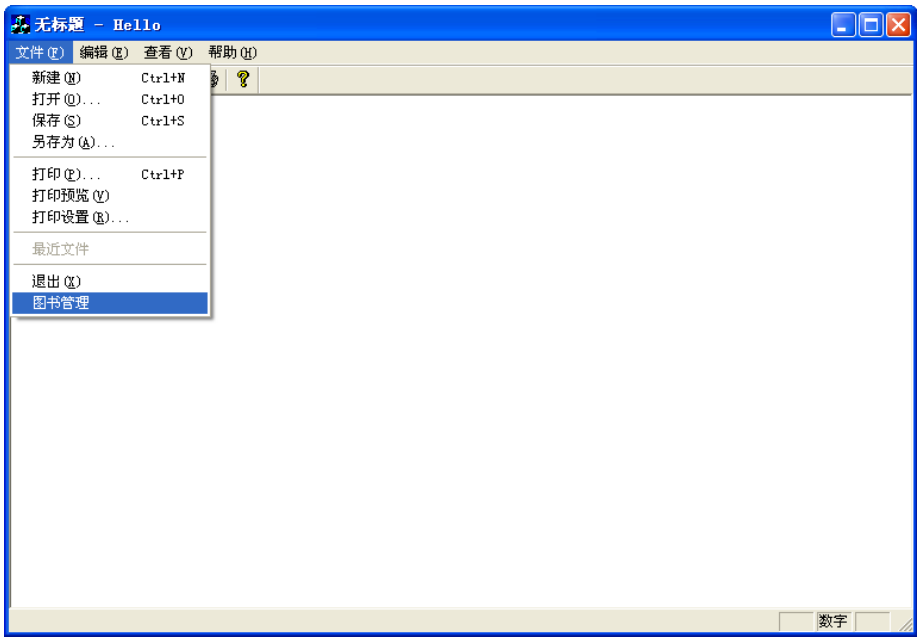
在图 1-10 所示的代码编辑窗口中的菜单函数中加入如下代码。

```
void CMainFrame::OnMenuitemLabman()
{
    // TODO: Add your command handler code here
    CGy cGy;
    cGy.DoModal();
}
```

并将“gy.h”文件加入 MainFrame.cpp 中。运行程序，执行菜单操作，就可看到打开的对话框了。

显示此对话框的具体操作:运行程序,显示如图 2-4 所示界面,然后执菜单“图书管理”,对话框就会显示出来。

详细信息请参考“VC++简明教程.PDF”文件中 62-72 页

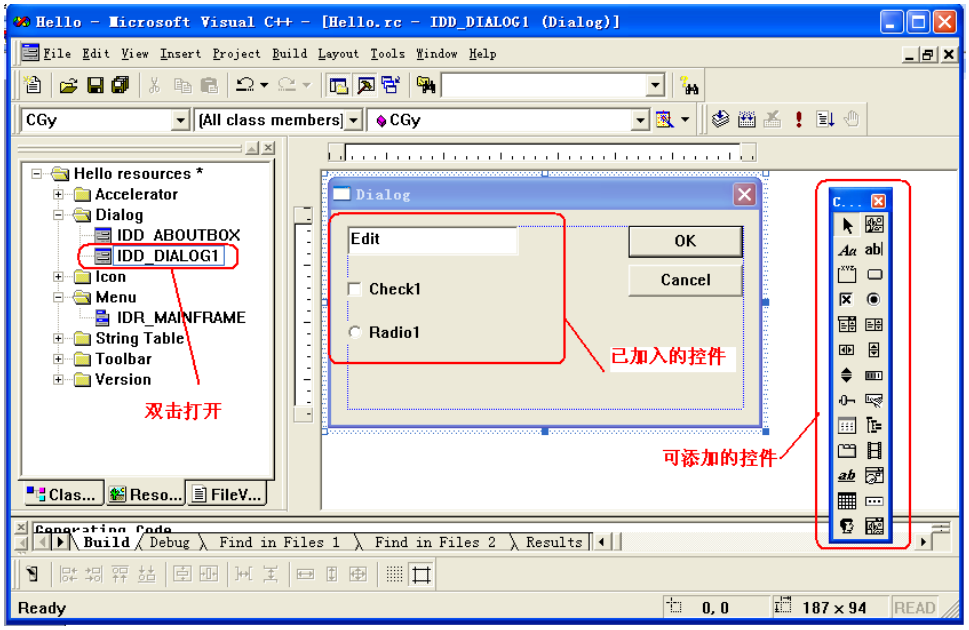


如图 2-4 程序运行界面

5 在对话框中加入其他资源

(1).加入编辑框

首先打开前面加入的对话框，会自动出现一个控件工具条，可选择里面的各种控件，加入到对话框中。如图 3-1 所示。**关于控件的具体内容请参考“VC++简明教程.PDF”文件中 58-62 页。**



如图 3-1 程序运行界面

(2).获取编辑框中输入的内容

有两种方式获取文本，一种是通过定义变量的方式，另一种是直接使用命令。下面简单

说一下直接使用命令的方式。

首先要知道已输入文本的编辑框的 ID，假设编辑框 ID 为 IDC_EDIT1，下面二条语句完成的是将编辑框中输入的文本内容，读到了变量 strBuf 中，其中 100 是指定的读取长度，如果输入框中文本小于此值则所有内容全部读到 strBuf 中，否则按指定的长度读取。

```
char strBuf[101];
GetDlgItemText(IDC_EDIT1,LPTSTR(strBuf),100);
```

(3).将文本读入编辑框中

通过下面两条语句就可将 strBuf 中的内容读到 IDC_EDIT1 这个编辑框中。

```
char strBuf[101]=”哈尔滨工业大学图书馆”;
SetDlgItemText(IDC_EDIT1,strBuf);
```

关于 MFC 类库的使用请参考 cms 网站上的：“MFC 类库详解.chm”

第6章 Java 编码规范

1 目的、环境

1.1 目的

在软件的生命周期中，维护的花费通常占很大的比例，且几乎所有的软件，在其整个生命周期中，开发人员和维护人员都不尽相同。编码规范可以改善软件的可读性，使程序员尽快而彻底地理解代码；同时，编码规范还可以提高程序代码的安全性和可维护性，提高软件开发的生产效率，所以，编码规范对于程序员而言至关重要。

为使开发项目中所有的 JAVA 程序代码的风格保持一致，增加代码的可读性，便于维护及内部交流，使 JAVA 程序开发人员养成良好的编码习惯，有必要对 JAVA 程序的代码编码风格做统一的规范约束。

1.2 环境

Eclipse 或 MyEclipse 或 netbeans

2 JAVA 源文件的组织结构

2.1 PACKAGE 的组织

Package 是组织相关类的一种比较方便的方法。Package 使我们能够更容易查找和使用类文件，并可以帮助我们在运行程序时更好的访问和控制类数据。

类文件可以很容易的组织到 Package 中，只要把相关的类文件存放到同一个目录下，给该目录取一个与这些类文件的作用相关的名称。如果需要声明程序包，那么每个 JAVA 文件（*.java）都需要在顶部进行 Package 的声明，以反映出包的名称。

例：

```
package com.fujitsu.fnst;
```

2.2 JAVA 源文件的内部结构

所有的 JAVA(*.java) 文件都应遵守如下的样式规则，如果 JAVA 源文件中出现以下相应的部分，应遵循如下的先后顺序。

编码时，即使某个类不是 public 类型的，也要在一个独立的 JAVA 文件（*.java）中声明，避免一个 JAVA 文件包含多个类声明。

版权信息

版权信息应在 JAVA 文件的开头，其他不需要出现在 JavaDoc 中的信息也可以包含在这里。

例：

```
/*
```

```
     * Copyright (C) Fujitsu Co., Ltd.
     * …
*/
```

Package/Import

Package 行要在 import 行之前，import 中标准的包名要在本地的包名之前，而且按照字母顺序排列。如果 import 行中包含了同一个包中的不同子目录，应 import 到某一个指定的类，避免*类型的 import。

例：

```
package com.fujitsu.fnst;

import java.io.InputStream;
import java.io.OutputStream;
```

Class

类的注释一般是用来解释类的，建议使用文档注释。

例：

```
/**
 * Class description goes here.
 * …
 */
```

接下来是类的定义，有可能包含了 extends 和 implements。

例：

```
public class CounterSet extends Observable implements Cloneable {
    …
}
```

Class Fields

public 的成员变量应使用 JavaDoc 注释，protected、private 和 package 定义的成员变量如果名字含义明确的话，可以没有注释。注释成员变量时，建议使用文档注释。

例：

```
/** classVar1 documentation comment */
public static int classVar1 = 0;


/**
 * classVar2 documentation comment that happens to be
 * more than one line long
 */
private static int classVar2 = 0;
```

存取方法

如果存取方法只进行简单的赋值或取值操作，可以写在一行上，否则不要写在一行上。

例：

```
/**
 * Get the counters
 */
public int getPackets() {
    packets++;

    return packets;
}

/**
 * Set the counters
 * …
 */
public void setPackets(int[] packets) { this.packets = packets; }
```

构造函数

构造函数应该按照参数数目的递增顺序进行书写。每个类都应具有至少一个构造函数。

例：

```
/**
 * constructor Blah documentation comment
 * …
 */
public Blah() {
    …
}

/**
 * constructor Blah documentation comment
 * …
 */
public Blah(int size) {
    …
}
```

克隆方法

如果这个类是可以被克隆的，那么下一步就是 clone 方法：

例：

```
/**
 * clone method for this class…
 */
public Object clone() {
    …
```

```
}
```

类方法

对于类的具体方法，应将功能相似的方法放置在一起。

例：

```
/**
 * Method doSomething documentation comment…
 */
public void doSomething() {
    …
}
```

toString 方法

每一个类都应该定义 toString 方法。

例：

```
/**
 *  toString method for this class…
 */
public String toString() {
    …
}
```

main 方法

每个文件的主类中应有一个 main 方法。main 方法应该提供简单的测试和 demo 功能，它应该写在类的底部。

例：

```
/**
 * The main method for simple test or demo use …
 */
public static void main(String[] agrs) {
    …
}
```

3 命名规则

3.1 JAVA 源文件的命名

JAVA 源文件名必须和源文件中所定义的类的类名相同。

3.2 Package 的命名

Package 名的第一部分应是小写 ASCII 字符，并且是顶级域名之一，通常是 com、edu、gov、mil、net、org 或由 ISO 标准 3166、1981 定义的国家唯一标志码。Package 名的后续部分由各组织内部命名规则决定，内部命名规则指定了各组件的目录名，所属部门名、项目名等。

例：

package com.fujistu.interstage.j2ee.deploytool.deployment;
package com.apple.quicktime.v2;

3.3 Class/Interface 的命名

Class 名应是首字母大写的名词。命名时应该使其简洁而又具有描述性。异常类的命名，应以 Exception 结尾。Interface 的命名规则与 Class 相同。

例：

```
public interface Set {
    …
};
public class CounterSet implement Set {
    …
};
public class InvalidException extends Exception {
    …
};
```

3.4 常量的命名

常量名的字母应全部大写，不同的单词之间通过下划线进行连接，并且名字组合应该赋予含义。

例：

```
static final int MIN_WIDTH = 4;
static final int MAX_WIDTH = 999;
```

3.5 普通变量

普通变量名的首字母小写，其它每个单词的首字母大写。命名时应该使其简短而又有特定含义，简洁明了的向使用者展示其使用意图。

例：

```
float    floatWidth    = 0.0;
double doubleWidth = 0.0;
```

3.6 约定变量

所谓约定变量，是指那些使用后即可抛弃（throwaway）的临时变量。通常 i、j、k、m 和 n 代表整型变量；c、d 和 e 代表字符型变量。

例：

```
int    i = 0;
char c = ‘a’;
```

3.7 方法的命名

方法名的第一个单词应是动词，并且首字母小写，其它每个单词首字母大写。

例：

```
void findPersonID(int nID);
void isStringEmpty(String str);
```

3.8 方法参数的命名

应该选择有意义的名称作为方法的参数名。如果可能的话，选择和需要赋值的字段一样

的名字。

例：

```
void setCounter(int size) {
        this.size = size;
}
```

4 样式结构

4.1 整体样式

4.1.1 缩进单位

一个缩进单位为四个空格，缩进排版时以缩进一个单位为最小缩进量。不要使用制表符（Tab 键），因为不同的系统对它的解释不尽相同。

4.1.2 缩进和对齐

缩进

当某行语句在逻辑上比下面的语句高一个层次时，该行下面的语句都要在该行的基础上缩进一个单位。

例：

```
public void someMethod(parameterA,parameterB) {
    int variantA=0;

    Sentence1;
    if (Conditions) {
        Sentence2;
    }
}
```

对齐

若干语句在逻辑上属于同一层次时，这些语句应对齐。

例：

```
public void someMethod(parameterA) {
    int variantA=0;

    Sentence1
    if (Conditions) {
        Sentence2;
        Sentence3;
    }
}
```

4.1.3 行宽

为了和 linux,unix 等字符界面的操作系统兼容，JAVA 代码行应限制在 80 个字符之内，多余部分应换行。

例：

```
variantA = someMethod(longExpression1, longExpression2, longExpression3);
```

应改为：

```
variantA = someMethod(longExpression1, longExpression2,
                                    longExpression3);
```

4.1.4 断行规则

当一句完整的语句大于 80 个字符时需要断行，断行时，应遵循下面规则。在逗号后换行

例：

```
variantA = someMethod(longExpression1, longExpression2,
                longExpression3, longExpression4);
```

在操作符前换行

例：

```
longName1 = longName2 * (longName3 + longName4 - longName5)
                        + 4 * longName6;
```

换行后，应和断行处的前一层次对齐

例：

```
longName1 = longName2 * (longName3 + longName4 - longName5)
+ 4 * longName6;
```

应改为：

```
longName1 = longName2 * (longName3 + longName4 - longName5)
                    + 4 * longName6;
```

换行时尽量选择高层次的地方进行换行

例：

```
longName1 = longName2 * (longName3 + longName4
                                    - longName5) + 4 * longName6;
```

应改为：

```
longName1 = longName2 * (longName3 + longName4 - longName5)
                        + 4 * longName6;
```

在使用上述的规则换行后对齐时，如果次行的长度大于 80 个字符，应改用两个单位的缩进来代替层次对齐

例：

```
private static synchronized horkingLongMethodName(int anArg,
                Object anotherArg, String yetAnotherArg,
                Object andStillAnother) {
        …
}
```

4.1.5 空白的使用

空格字符的使用

1) 关键字和括号()之间要用空格隔开

例：

```
while (condition) {
    Sentence1;
}
if (condition) {
    Sentence2;
}
```

2) 参数列表中逗号的后面应该使用空格

例：

```
public void methodA(parameterA, parameterB, parameterC) {
    Sentence1;
}
```

3) 所有的二元运算符，除了"."，应该使用空格将之与操作数分开

例：

```
longName1 = longName2 * (longName3 + longName4) + 4 * longName5;
```

4) 强制类型转换后应该跟一个空格

例：

```
methodA((byte) parameterA, (Object) parameterB);
```

5) 左括号右边和右括号左边不能有空格

例：

```
longName1 = longName2 * ( longName3 + longName4 );
```

应改为：

```
longName1 = longName2 * (longName3 + longName4);
```

6) 方法名与其参数列表的左括号之间不能有空格

例：

```
methodA (parameter1, parameter2);
```

应改为：

```
methodA(parameter1, parameter2);
```

7) 一元操作符和操作数之间不应该加空格，比如：负号("-")、自增("++")和自减("--")

例：

```
variantA += variantB --;
```

应改为：

```
variantA += variantB--;
```

空白行的使用

空白行将逻辑相关的代码段分隔开，以提高可读性，有如下几种情形：

1) 一个源文件的两个片段(section)之间用两个空白行

例：

用两个空白行将 JAVA 文件顶端的版权说明和下面的内容隔开

```
/*
 * Copyright (c) 1999-2003 Fujitsu, Co., LTD.
 * …
*/


package com.fujitsu.fnst;
```

2) 两个类声明或接口声明之间使用两个空白行

例：

```
public class A {
    …
}


private class B {
    …
}
```

3) 两个方法的声明之间使用一个空白行

例：

```
public class A {
    private void methodA() {
        …
    }

    priavte void methodB() {
        …
    }
}
```

4) 方法内的局部变量和方法的第一条语句之间使用一个空白行

例：

```
private void methodA() {
    int variantA = 0;
    int variantB = 0;
```

```
        variantA = variantB + 10;
        …
    }
```

5) 块注释或单行注释之前使用一个空白行

例：

```
private void methodA() {
    if (condition) {

      /*This sentence can run on all occasions,… */
variantA = variantB + 10;
        }
        …
}
```

6) 一个方法内的两个逻辑段之间应该用一个空白行

例：

```
private int methodA() {
        variantA = methodGet();
        variantB = methodGet();

        return variantA + variantB;
}
```

4.2 注释样式

JAVA 程序有两类注释，实现注释(implementation comments)和文档注释(document comments)。

实现注释，就是使用 /*...*/ 或 // 界定的注释。文档注释，又被称为"doc comments"或" JavaDoc 注释"，是 JAVA 独有的，由 /**...*/ 界定，并且文档注释可以通过 JavaDoc 工具转换成 HTML 文档。

在注释里，应该对设计决策中重要的或者不是显而易见的地方进行说明，但应避免对意思表达已经清晰的语句进行注释。

特别注意，频繁的注释有时反映出代码的低质量。当你觉得被迫要加注释的时候，考虑一下是否可以重写代码，并使其更清晰。

4.2.1 实现注释

块注释

块注释通常用于提供对文件，方法，数据结构和算法的描述。块注释被置于每个文件的开始处以及每个方法之前。它们也可以被用于其他地方，比如方法内部。在功能和方法内部的块注释应该和它们所描述的代码具有一样的缩进格式，并且，块注释之首应该有一个空白行，用于把块注释和代码分割开来。

例：

```
previousSentences;
```

```
/*
 * Here is a block comment.
 */
Sentences;
```

单行注释

单行注释显示在一行内，并与其后的代码具有一样的缩进层次。如果一个注释不能在一行内写完，应采用块注释，且单行注释之前应该有一个空白行。

例：

```
if (condition) {

/* Handle the condition. */
...
}
```

尾端注释

尾端注释与它们所要描述的代码位于同一行，但是应该有足够的空白来分开代码和注释。若有多个尾端注释出现于大段代码中，它们应该具有相同的缩进。

例：

```
if (condition) {
    return TRUE;            /* special case */
} else {
    return isPrime(a);      /* works only for odd a */
}
```

行末注释

行末注释的界定符是"//"，它可以注释掉整行或者一行中的一部分，一般不用于连续多行的注释文本。但是，它可以用来注释掉连续多行的代码段。

例：

```
if (condition) {
    // Do a double-flip.
    Sentence;.
}  else {
    return false;       // Explain why here.
}

//if (condition) {
//      //Do a triple-flip.
//      Sentence;
//}
//else {
//      return false;
//}
```

4.2.2 文档注释

置于/**...*/之中的注释称之为文档注释。

文档注释用来描述 Java 的类、接口、构造器、方法以及字段(field)，一个注释对应一个类、接口或成员，该注释应位于声明之前，与被声明的对象有着相同的缩进层次。

在类的声明中，各种类、接口、变量、常量、方法之前都应该有相应注释。

例：

```
/**
 * The Example class provides...
 */
public class Example {

    /**
     * this is variant A
     */
    int virantA = 0;

    /**
     * This is method A
     *@param parameterA The parameter A
     *@return return an Integer
     */
    public int methodA(parameterA){
        ...
    };
}
```

5 声明

5.1 变量的声明

一行只声明一个变量

例：

```
int variantA = 0, variantB = 0;
```

应改为：

```
int variantA = 0;
int variantB = 0;
```

声明变量时要对其进行初始化，如果是类中的变量，声明时不初始化，在构造函数中一定要初始化

例：

```
int variantA;
```

应改为：

```
int variantA = 0;
```

临时变量放在其作用域内声明

例：

```
int tempA = 0;

if (condition) {
    tempA = methodA();
    methodB(tempA);
}
```

应改为：

```
if (condition) {
    int tempA = 0;

    tempA = methodA();
    methodB(tempA);
}
```

声明应集中放在作用域的顶端

例：

```
if (condition) {
    int tempA = 0;

    tempA = methodA();
    int tempB = 0;

    tempB = methodB();
}
```

应改为：

```
if (condition) {
    int tempA = 0;
    int tempB = 0;

    tempA = methodA();
    tempB = methodB();
}
```

避免声明的局部变量覆盖上一级声明的变量

例：

```
int counter = 0;

if (condition) {
    int counter = 0;

    counter = methodA();
}
```

应改为：

```
int counter = 0;

if (condition) {
    int counterTemp = 0;

    counterTemp = methodA();
}
```

5.2 类和接口的声明

当编写类和接口时，应该遵守以下规则：

在方法名与其参数列表之前的左括号"("间不要有空格

左大括号"{"位于声明语句同行的末尾，并与末尾之间留有一个空格

右大括号"}"另起一行，与相应的声明语句对齐。如果是一个空语句，"}"应紧跟在"{"之后

方法与方法之间以空白行分隔

例：

```
public class Sample extends Object {
    private int ivar1;
    private int ivar2;

    public Sample(int i, int j) {
        ivar1 = i;
        ivar2 = j;
    }

    public int emptyMethod() {}
    …
}
```

6 语句

6.1 简单语句

每行至多包含一条完整语句

例：

```
variantA++; variantB++;
```

应改为：

```
variantA++;
variantB++;
```

在没有必要的情况下，不要在 return 语句中使用括号

例：

```
return (0);
```

应改成：

```
return 0;
```

6.2 复合语句

复合语句是包含在大括号中的语句序列，形如"{ 语句 }"，其编码应有如下基本规则：

被括其中的语句应该比复合语句缩进一个层次

左大括号"{"应位于复合语句起始行的行尾，并且空一个空格，右大括号"}"应另起一行并与复合语句首行对齐

复合语句即使只有一个语句，也要有大括号作为界定

每行至多包含一条完整语句

1. 判断语句

例：

if-else 语句应该具有如下格式：

```
if (condition) {
        Sentences;
}

if (condition) {
        Sentences;
} else if (condition) {
        Sentences;
} else {
        Sentences;
}
```

2. 选择语句

在选择语句中应添加 default 情况，防止不可预知的情况发生。

当一个 case 在没有 break 语句的情况下，它将顺着往下执行。应在 break 语句的位置添加注释。[下面就含注释/* falls through */]

例：

```
switch (condition) {
case ABC:
        Sentences;
        /* falls through */
case DEF:
        Sentences;
        break;

case XYZ:
        Sentences;
        break;

default:
        Sentences;
        break;
}
```

3. 循环语句

在 for 语句的初始化或更新子句中，如果存在多项，各项间应用逗号隔开。同时，应避免使用三个以上子句，从而导致复杂度提高；若确实需要，可以在 for 循环之前放置初始化子句或在 for 循环末尾放置更新子句。

例：

```
for (int i = 0, j = 10, k = 10, m = 50; i < j + k + m; i++, j--, k--, m--) {
    Sentences;
}
```

应改为：

```
int i = 0;
int j = 100;
int k = 1000;
int m = 500;

for (; i < j + k + m ;) {
    Sentences;
    i++;
    j--;
    k--;
    m--;
}
```

一个空的 for 语句和 while 语句只用一行

例：

```
for (initialization; condition; update) {
    Sentences;
}
for (initialization; condition; update);

while (condition) {
    Sentences;
}
while (condition);

do {
    Sentences;
} while (condition);
do {
    ;
} while (condition);
```

4. Try-Catch 结构语句

例：

```
try {
```

```
        Sentences;
    } catch (ExceptionClass e) {
        Sentences;
}

    try {
        Sentences;
    } catch (ExceptionClass e) {
        Sentences;
    } finally {
        Sentences;
}
```

7 典型示例

1) 方法的返回值表达应尽量简单。在 return 语句后一般不使用括号，但是如果返回中包含复杂表达式，则要使用括号。

例：

```
return ((x >= 0) ? x : y);
```

2) 比较对象用 equals() 方法代替操作符“==”。特别是不能用“==”去比较字符串类型的变量。

例：

```
String strA = ”abc”;
String strB = ”bcd”;
if (strA == strB) {
    …
}
```

应改为：

```
String strA = ”abc”;
String strB = ”bcd”;
if (strA.equals(strB)) {
    …
}
```

3) if、while 等语句中的条件判断部分，应将常量（如果有）写在“==”运算符的左边，而把变量写在右边。

例：

```
if (3 == var) {
    …
}
while (3 == var ) {
    …
}
```

4) 在每一个 package 中包含一个 package.html 文件用于对包的简要说明。
该文件结构请参考链接：
http://java.sun.com/j2se/javadoc/writingdoccomments/index.html#packagecomments

8 规范示例

8.1 完整示例

```
/*
 * @(#)Sample.java       1.00 2003/04/15
 *
 * Copyright (c) Fujitsu co, Ltd. …
 */


package com.fujitsu.fnst;

import java.io.File;
import java.util.Date;


/**
 * Sample class description goes here
 * @version          1.00 15 April 2003
 * @author           Your name
 */
public class Sample {

    /* * classVar1 documentation comment */
    public static int classVar1 = 0;

    /**
     * classVar2 documentation comment that happens to be
     * more than one line long
     */
    private static final String ERROR_INFO = “This is an error message.”

    /**
     * GET method for classVar1
     */
    public int getClassVar1() { return classVar1; }

    /**
     * SET method for classVar1
     */
```

```
    public void setClassVar1(int var) { classVar1 = var; }

    /**
     * Constructor Sample documentation comment
     */
    public Sample() {
        try {
            doSomething();
        } catch (Exception e) {
            …
        }
    }

    /**
     * clone method for this class
     */
    public Object clone() {
        …
    }

    /**
     * Method doSomething documentation comment
     */
    public void doSomething() {
        for (int i = 0; i < 10; i++) {
            classVar1 += i;
        }
    }

    /**
     * toString method for this class
     */
    public String toString() {
        …
    }

    /**
     * The main method for simple test or demo use
     */
    public static void main(String[] args) {
        …
    }
}
```

9 附 2 关于 JavaDoc

9.1 常用的 JavaDoc 标记

JavaDoc 标记是插入文档注释中的特殊标记。JavaDoc 标记由“@”、标记、专用注释引用组成。

JavaDoc 主要有以下标记：

* @author （用于类和接口，标明开发该类模块或接口的作者）
* @version （用于类和接口，标明该模块的版本）
* @param （用于方法和构造函数，说明方法中的某个参数）
* @return （用于方法，说明方法的返回值）
* @exception （用于方法，说明方法可能抛出的异常,同@throws）
* @see （对类、属性、方法的说明，参考转向，也就是相关主题）
* @since （说明引进该类、方法或属性的起始版本）
* @deprecated （说明某类或方法不被推荐使用，以及相应的替代类或替代方法）

关于 JavaDoc 标记使用的更详细信息请参考下面链接：
http://java.sun.com/j2se/1.4.1/docs/tooldocs/windows/javadoc.html#javadoctags

@author 和@version

@author 标记用于指明类或接口的作者。在缺省情况下 JavaDoc 工具将其忽略，但命令行开关-author 可以修改这项功能，使其包含的信息被输出。

句法：@author 作者名。

@author 可以多次使用，以指明多个作者，生成的文档中每个作者之间使用逗号“，”分隔。

@version 标记用于指明类或接口的版本。

句法：@version 版本号。

例：

```
/**
 * @author fancy
 * @author bird
 * @version Version 1.00
 */
```

@param、@return 和@exception

这三个标记都是用于方法说明。

@param 标记用于描述方法的参数。

句法：@param 参数名 参数的描述。

每一个@param 只能描述方法的一个参数，所以，如果方法需要多个参数，就需要多次使用@param 来描述。

@return 标记用于描述方法的返回值。

句法：@return 返回值描述。

一个方法中只能用一个@return。

@exception 标记用于说明方法可能抛出的异常，同@throws。

句法：@exception 异常类 抛出异常的原因。

一个方法可以有多个@exception 标记。

例：

```
/**
 * @param    param1    description of param1
 * @param    param2    description of param2
 * @return    true or false
 * @exception    java.lang.Exception    throw when switch is 1
 * @exception    NullPointerException    throw when parameter is null
 */
```

@see

该标记用于参考转向。

句法：

@see 类名

@see #方法名或属性名

@see 类名#方法名或属性名

@see HTML 链接字

关于类名，如果 JAVA 源文件中的 import 语句包含了该类，可以只写出类名；若没有包含，则需要写出类全名（如 java.lang.String）。对于方法或属性，如果没有指定类名，则默认为当前类。对于方法，还需要写出方法名及其参数类型，没有参数的，需要写一对括号。

例：

```
/**
 * @see String
 * @see java.lang.String The String Class
 * @see java.lang.StringBuffer#str
 * @see #str
 * @see #str()
 * @see #main(String[])
 * @see Object#toString()
 * @see <a href=”http://…/somepage.html”>some page</a>
 */
```

@since

该标记用于说明引进某个类、方法或属性的起始版本。

句法：@since 版本号

其中，对“版本号”没有特殊的格式要求。@since 标记可用于包、类、接口、方法、属性的文档注释里。表示它描述的修改或特性从“版本号”所描述的版本开始就存在了。

例：

```
/**
 * @since 1.2
```

```
 */
```

@deprecated

该标记用于指出某个类或方法不被推荐使用，以及相应的替代类或替代方法。

句法：@deprecated deprecate-Description

@deprecated 标记可用于包、类、接口、方法、属性的文档注释里。Deprecate-Description 里，首先要说明该类或者方法从什么时候（什么版本）起不再使用，其次要说明用哪个类或者方法可以替代它。可以使用@see 或@link 指明替代类或替代方法。

例：

```
/**
 * @deprecated    As of JDK 1.1, replaced by {@link setPrice(int)}
 */
```

又例：

```
/**
 * @deprecated    As of JDK 1.1, replaced by setBounds
 * @see #setBounds(int, int, int, int)
*/
```

9.2 JavaDoc 命令的使用

用法：

javadoc [options] [packagenames] [sourcefiles] [classnames] [@files]

主要选项：

-public	显示 public 类和成员及其注释
-protected	显示 protected/public 类和成员（缺省）及其注释
-package	显示 package/protected/public 类和成员及其注释
-private	显示所有类和成员及其注释
-d <directory>	输出文件的目标目录
-doclet <class>	指定生成输出文档使用的 doclet
-docletpath <path>	指定 doclet 类的搜索路径
-sourcepath <pathlist>	指定源文件的搜索路径
-classpath <pathlist>	指定用户类的搜索路径
-version	显示@version 段包含的内容
-author	显示@author 段包含的内容
-splitindex	将索引分为每个字母对应一个文件
-windowtitle <text>	文档的浏览器窗口标题
-verbose	是否显示 javadoc 运行的详细信息
-charset <charset>	设置生成 HTML 文档的字符集

关于 JavaDoc 工具如何使用的更多详细信息请参考下面链接：

http://java.sun.com/j2se/1.4.1/docs/tooldocs/windows/javadoc.html#options

JavaDoc 可以对单个文件或者一个 package 生成文档，有如下两个命令：

```
javadoc samples\sample.java
javadoc samplespackage
```

第一个命令对一个单独的 java 文件 sample.java 生成 HTML 文档；第二个命令对一

个 package 包 samplespackage 生成 HTML 文档。

1）-public、-protected、-package、-private

这四个选项，只需要任选其一即可。-public 在生成的文档中只包含 public 类型的类或者方法及其注释内容，-protected 选项生成 protected、public 的类和成员及其注释内容，-package 选项生成 package、protected、public 的类和成员及其注释内容，-private 选项生成所有的类和成员及其注释内容。

例：

javadoc -public sample.java

2）-d <targetpath>

选项允许你定义输出目录。如果不用 -d 定义输出目录，生成的文档文件会放在当前目录下。targetpath 可以是相对路径，也可以是绝对路径。

例：

javadoc -d doc\ samplespackage

3）-doclet <class>

指定一个类，该类用于启动生成 java 文档的 doclet，该 doclet 指定了输出的内容和格式。如果没有指定 doclet，将采用默认的 doclet。

例：

javadoc -doclet com.sun.tools.doclets.mif.MIFDoclet

4）-sourcepath <pathlist> 和 -classpath <pathlist>

-sourcepath 指定源文件的搜索路径，多个路径间用“；”分隔。注意，只有当向 javadoc 传入 package 名时，才需要-sourcepath。如果未指定-sourcepath，javadoc 会把-classpath 作为搜索路径，如果-classpath 也没有指定，则当前路径会被当作搜索路径。

例：

javadoc -sourcepath C:\user1\src;C:\user2\src com.mypackage

-classpath 指定 javadoc 搜索文档中引用类的类路径，多个路径间用“；”分隔。如果未指定-classpath，javadoc 把-sourcepath 指定的路径作为引用类的搜索路径。

例：

javadoc -classpath \user\lib -sourcepath \user\src com.mypackage

5）-version 和 -author

-version 和 -author 用于控制生成文档时是否包含@version 和@author 指定的内容。不加这两个参数的情况下，生成的文档中不包含版本和作者信息。

例：

javadoc -classpath \user\lib -version -author com.mypackage

6）-windowtitle 和-charset

-windowtitle 标记指定添加到 HTML 标记<title>中的文字。这些文字将显示在浏览器的标题栏。

例：

javadoc -windowtitle "help of myPackage" com.mypackage

-charset 指定生成的 HTML 页的字符集。比如，如果希望生成 HTML 页正确显示中文，就需要指定这个标记。

例：

javadoc -charset "gb2312 " mypackage

7）-verbose

指定是否显示 javadoc 运行时的详细信息。

例：

javadoc -classpath \user\lib -verbose com.mypackage

8） JavaDoc 工具组合示例

例：

JavaDoc -public -charset "Shift_JIS" -windowtitle "Fujitsu XML Enc Wrapper" -doctitle "Wrapper Interfaces OF Fujitsu XML Enc System [By FNST]" -d ./Docs/ -verbose–classpath G:/working/library;c:/systemlib -sourcepath G:/working com.fujitsu.xmldsig.enc.wrapper com.fujitsu.xmldsig.enc.wrapper.types

第7章 Eclipse 调试方法

1 目的

熟悉 Eclipse 开发环境。
学会使用单步跟踪方法对程序进行调试

2 环境

Eclipse + windowBuilder 插件

3 知识要点：

调试技术程序出错的类型大致可以分为两种，语法错误和逻辑错误。语法错误可以通过编译器的出错信息得到纠正。然而逻辑错误则不能，所以各大 IDE（集成开发环境）中都提供了 debug 功能，用来分析和排除程序中的逻辑错误，排除逻辑错误的过程又称调试（或 debug）。

下面以 Eclipse 的调试环境做介绍。

3.1 常用的调试命令主要有：

以 debug 方式运行 java 程序后，使用下面的快捷键
（F8）直接执行程序。遇到断点时暂停；
（F5）Step Into: 单步执行程序，遇到方法时进入；
（F6）Step Over: 单步执行程序，遇到方法时跳过；
（F7）Step Return:单步执行程序，从当前方法跳出。

3.2 最基本的操作是

首先在一个 java 文件中设断点，然后 debug as－－>open debug Dialog，然后在对话框中选类后－－> Run。当程序走到断点处就会转到 debug 视图下。

F5 键与 F6 键均为单步调试，F5 是 step into,也就是进入本行代码中执行，F6 是 step over，也就是执行本行代码，跳到下一行，F7 是跳出函数，F8 是执行到最后。

step Filter 逐步过滤 一直执行直到遇到未经过滤的位置或断点(设置 Filter：window-preferences-java-Debug-step Filtering)

resume 重新开始执行 debug,一直运行直到遇到 breakpoint

hit count 设置执行次数 适合程序中的 for 循环(设置 breakpoint view-右键 hit count)

inspect 检查 运算。执行一个表达式显示执行值

watch 实时地监视变量的变化

我们常说的断点(breakpoints)是指 line breakpoints,除了 line breakpoints,还有其他的断点类型：field(watchpoint)breakpoint,method breakpoint,exception breakpoint.

field breakpoint 也叫 watchpoint(监视点) 当成员变量被读取或修改时暂挂

添加 method breakpoint 进入/离开此方法时暂挂(Run-method breakpoint)

添加 Exception breakpoint 捕抓到 Execption 时暂挂

断点属性：

hit count 执行多少次数后暂挂 用于循环

enable condition 遇到符合你输入条件(为 ture\改变时)就暂挂

suspend thread 多线程时暂挂此线程

suspend VM 暂挂虚拟机

variables 视图里的变量可以改变变量值，在 variables 视图选择变量点击右键--change value.一次来进行快速调试。

debug 过程中修改了某些 code 后--〉save&build-->resume-->重新暂挂于断点

4 Eclipse 调试器和 Debug 视图

Eclipse SDK 是针对 Java™ 开发工具（Java™ Development Tools，JDT）的项目，它具有一个内置的 Java 调试器，可以提供所有标准的调试功能，包括分步执行、设置断点和值、检查变量和值、挂起和恢复线程的功能。除此之外，还可以调试远程机器上运行的应用程序。Eclipse 平台很健壮，因为其他编程语言可以将该平台提供的调试工具用于各自的语言运行时。正如下文所示，同一个 Eclipse Debug 视图也可以用于 C/C++ 编程语言。

Eclipse 平台工作台及其工具是围绕 JDT 组件构建的，该组件为 Eclipse 提供了下列特性：

项目管理工具

透视图和视图

构造器、编辑器、搜索和构建功能

调试器

Eclipse 调试器本身是 Eclipse 内的一个标准插件集。Eclipse 还有一个特殊的 Debug 视图，用于在工作台中管理程序的调试或运行。它可以显示每个调试目标中挂起线程的堆栈框架。程序中的每个线程都显示为树中的一个节点，Debug 视图显示了每个运行目标的进程。如果某个线程处于挂起状态，其堆栈框架显示为子元素。

在使用 Eclipse 调试器之前，假定您已经安装了合适的 Java SDK/JRE（我推荐使用 Java VM V1.4）和 Eclipse Platform SDK V3.3，而且两者的运行都没问题。一般来说，先用 Eclipse 示例测试一下调试选项比较好。如果想开发和调试 C/C++ 项目，还需要获得并安装 C/C++ 开发工具（C/C++ Development Tools，CDT）。关于 Java SDK/JRE、Eclipse 平台和示例以及 CDT，请参阅 参考资源。 图 1 显示了 Debug 透视图的一般视图。

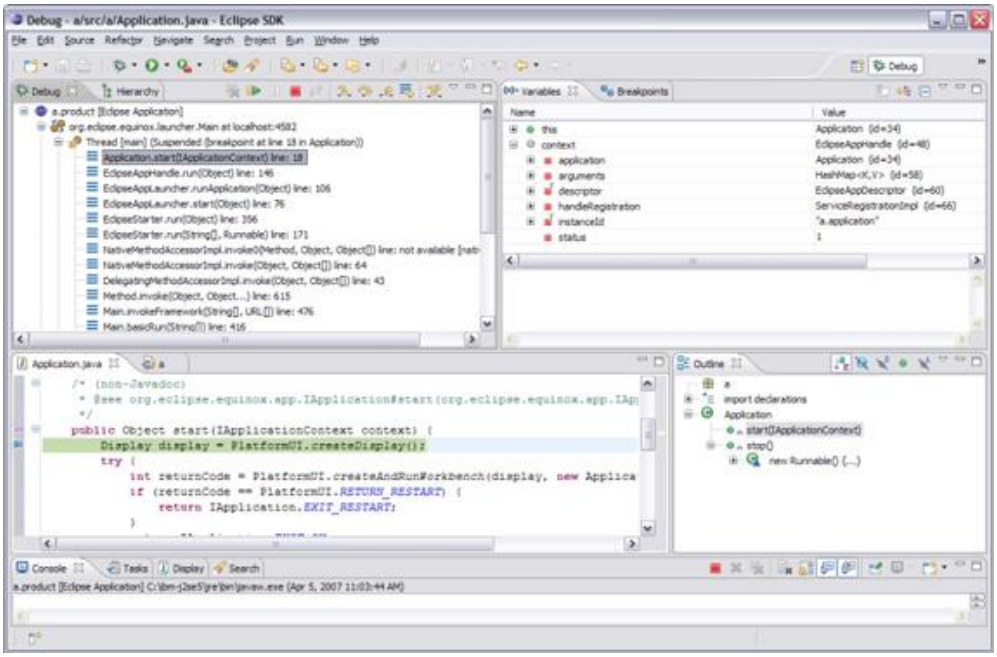


图 1. Eclipse Debug 透视图的一般视图

5 调试 Java 语言程序

在可以调试项目前，需要干净地编译和运行代码。首先，需要为应用程序创建一个运行配置，确保应用程序可以正确启动。然后，需要通过 **Run > Debug** 菜单以类似的方式设置调试配置。还需要选择一个类，将它作为调试的主 Java 类来使用（请参见图 2）。您可以按照自己的意愿为单个项目设置多个调试配置。当调试器启动时（从 **Run > Debug**），会在一个新的窗口中打开，这时就可以开始调试了。

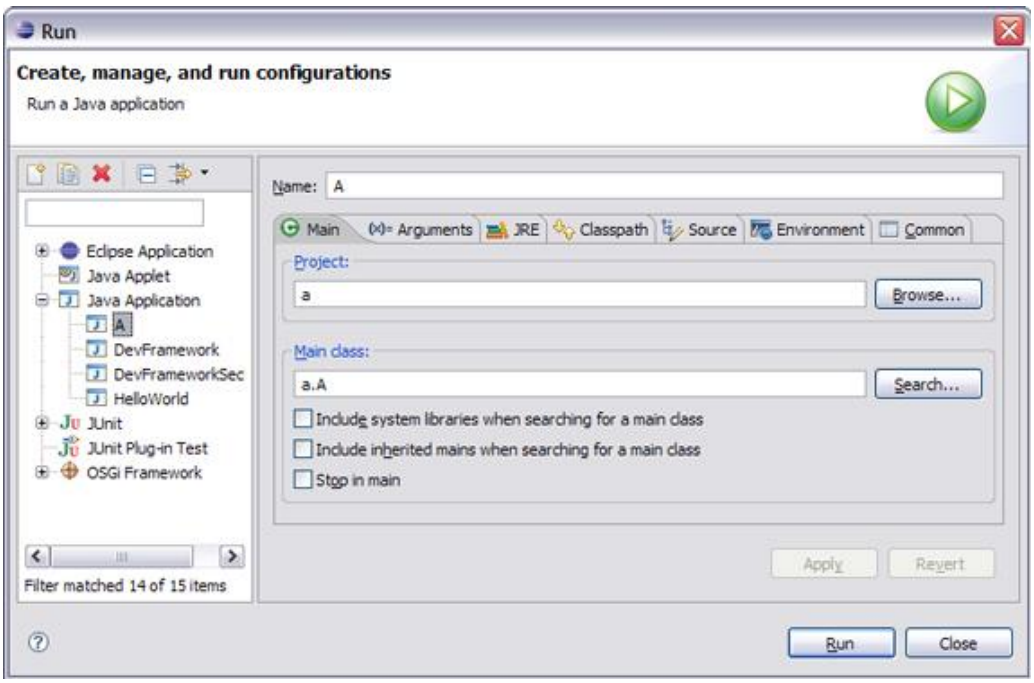


图 2. 在调试配置中设置项目的主 Java 类

接下来，将讨论 Eclipse 中的一些常用调试实践。

5.1 设置断点

在启动应用程序进行调试时，Eclipse 会自动切换到 Debug 透视图。毫无疑问，最常见的调试步骤是设置断点，这样可以检查条件语句或循环内的变量和值。要在 Java 透视图的 Package Explorer 视图中设置断点，双击选择的源代码文件，在一个编辑器中打开它。遍历代码，将鼠标放在可疑代码一行的标记栏（在编辑器区域的左侧）上，双击即可设置断点。

图 3. 编辑器左侧看到的两个断点

现在，从 **Run > Debug** 菜单启动调试会话。最好不要将多条语句放在一行上，因为会无法单步执行，也不能为同一行上的多条语句设置行断点。

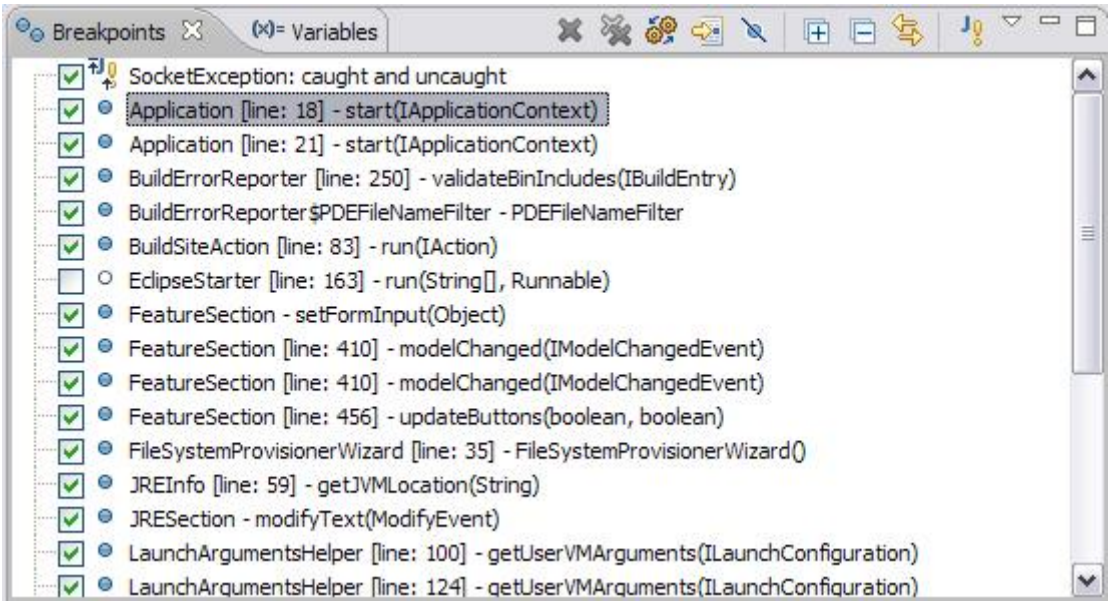


图 4. 视图中左侧空白处的箭头指示当前正在执行的行

还有一个方便的断点视图来管理所有的断点。

```
/* (non-Javadoc)
 * @see org.eclipse.equinox.app.IApplication#start(org.eclipse.equ
 */
public Object start(IApplicationContext context) {
    Display display = PlatformUI.createDisplay();
    try {
        int returnCode = PlatformUI.createAndRunWorkbench(display,
        if (returnCode == PlatformUI.RETURN_RESTART) {
            return IApplication.EXIT_RESTART;
        }
```

图 5. 断点视图

5.2 条件断点

一旦了解到错误发生的位置，您可能想要知道在程序崩溃之前，程序在做什么。一种方法就是单步执行程序的每行语句。一次执行一行，直到运行到可疑的那行代码。有时，最好只运行一段代码，在可疑代码处停止运行，在这个位置检查数据。还可以声明条件断点，它在表达式值发生变化时触发（请参见图 6）。除此之外，在输入条件表达式时，也可以使用代码帮助。

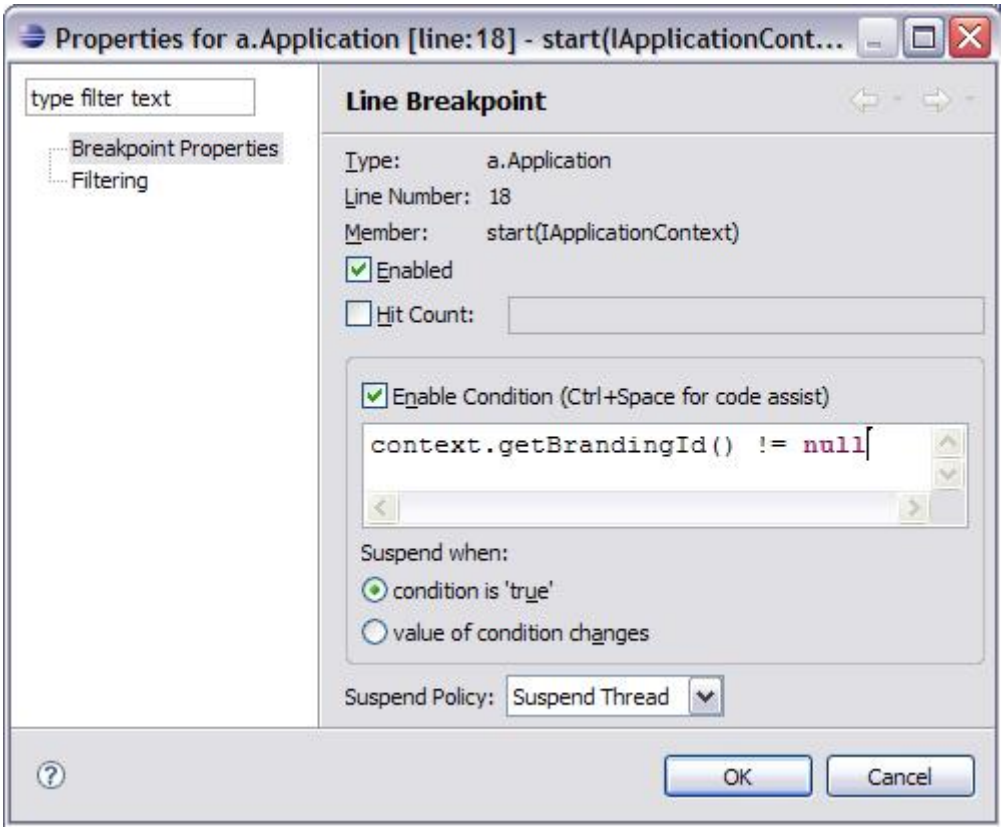


图 6 设置条件断点触发器

5.3 计算表达式的值

为了在 Debug 透视图的编辑器中计算表达式的值，选择设置了断点的那行代码，在上下文菜单中，通过 Ctrl+Shift+I 或右键单击您感兴趣的变量（参见图 7）选择 Inspect 选项。在当前堆栈框架的上下文中会计算表达式的值，在 Display 窗口的 Expressions 视图中会显示结果。

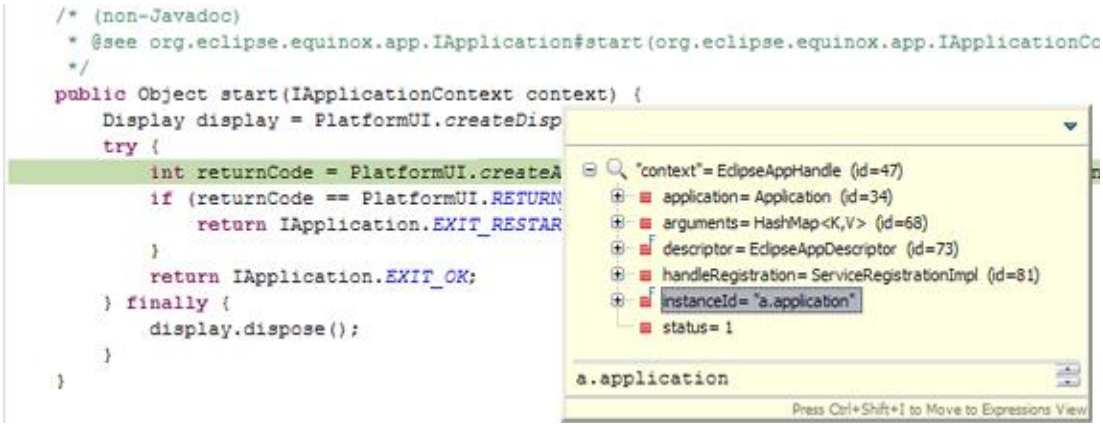


图 7 通过 Inspect 选项计算表达式的值

剪切活动代码

Display 视图允许您以剪切类型的方式处理活动代码（参见图 8）。要处理一个变量，在 Display 视图中输入变量名即可，视图会提示您一个熟悉的内容助手。

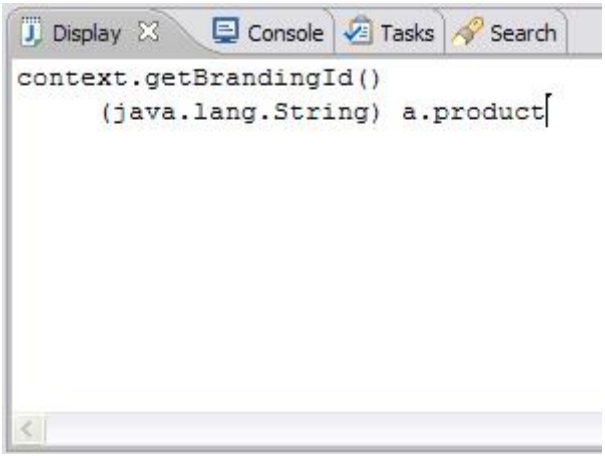


图 8. Display 视图

当调试器停止在一个断点处时，您可以从 Debug 视图工具栏（参见图 9）中选择 Step Over 选项，继续调试器会话。该操作会越过高亮显示的那行代码，继续执行同一方法中的下一行代码（或者继续执行调用当前方法的那个方法的下一行代码）。执行上一步后发生改变的变量会用某种颜色高亮显示（默认是黄色）。颜色可以在调试首选项页面中改变。

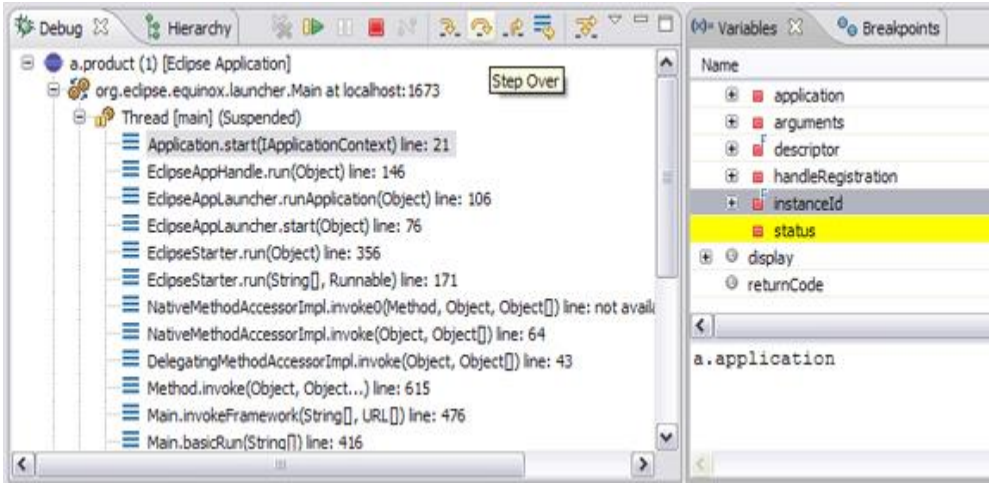


图 9. 改变颜色的变量

要在 Debug 视图中挂起执行线程，选择一个运行线程，单击 Debug 视图工具栏中的 Suspend。该线程的当前调用堆栈就会显示出来，当前执行的代码行就会在 Debug 透视图中的编辑器中高亮显示。挂起一个线程时，将鼠标放在 Java 编辑器中的变量上，该变量的值就会在一个小的悬停窗口中显示出来。此时，该线程的顶部堆栈框架也会自动选中，其中的可视变量也会在 Variables 视图中显示出来。您可以通过单击 Variables 视图中合适的变量名来检查变量。

热交换错误修正：动态代码修正

如果运行的是 Java 虚拟机（Java Virtual Machine，JVM）V1.4 或更高的版本，Eclipse 支持一个叫做热交换错误修正（Hotswap Bug Fixing）的功能，JVM V1.3 或更低的版本不支持该功能。该功能允许在调试器会话中改变源代码，这比退出应用程序、更改代码、重新编译、然后启动另一个调试会话更好。要利用该功能，在编辑器中更改代码后重新调试即可。由于 JVM V1.4 与 Java 平台调试器架构（Java Platform Debugger Architecture，JPDA）兼容，所以才有可能具备该功能。JPDA 实现了在运行的应用程序中替换经过修改的代码的功能。如果应用程序启动时间较长或执行到程序失败的地方时间很长，那么这一点特别有用。

如果在完成调试时，程序还没有全部执行一遍，在 Debug 视图的上下文菜单中选择 Terminate 选项。容易犯的一个错误是在调试器会话中使用 Debug 或 Run，而不是 Resume。这样做会启动另一个调试器会话，而不是继续当前会话。

远程调试

Eclipse 调试器提供了一个有趣的选项，可以调试远程应用程序。它可以连接到一个运行 Java 应用程序的远程 VM，将自己连接到该应用程序上去。使用远程调试会话与使用本地调试会话大致相同。但是，远程调试配置需要在 Run > Debug 窗口中配置一些不同的设置。需要在左侧视图中先选择 Remote Java Application 选项，然后单击 New。这样就创建了一个新的远程启动配置，会显示出三个选项卡：Connect、Source 和 Common。

在 Connect 选项卡的 Project 字段，选择在启动搜索源代码时要引用的项目。在 Connect 选项卡的 Host 字段，输入运行 Java 程序的远程主机的 IP 地址或域名。在 Connect 选项卡的 Port 字段，输入远程 VM 接收连接的端口。通常，该端口在启动远程 VM 时指定。如果想让调试器决定在远程会话中 Terminate 命令是否可用，可以选择 Allow termination of remote VM 选项。如果希望可以终止连接的 VM，则选择该选项。现在，在您选择 Debug 选项时，调试器会尝试连接到指定地址或端口的远程 VM，结果会在 Debug 视图中显示出来。

如果启动程序不能连接到指定地址的 VM，会出现一条错误信息。通常来说，是否可以使用远程调试功能完全取决于远程主机上运行的 Java VM。图 10 显示了一个远程调试会话的连接属性设置。

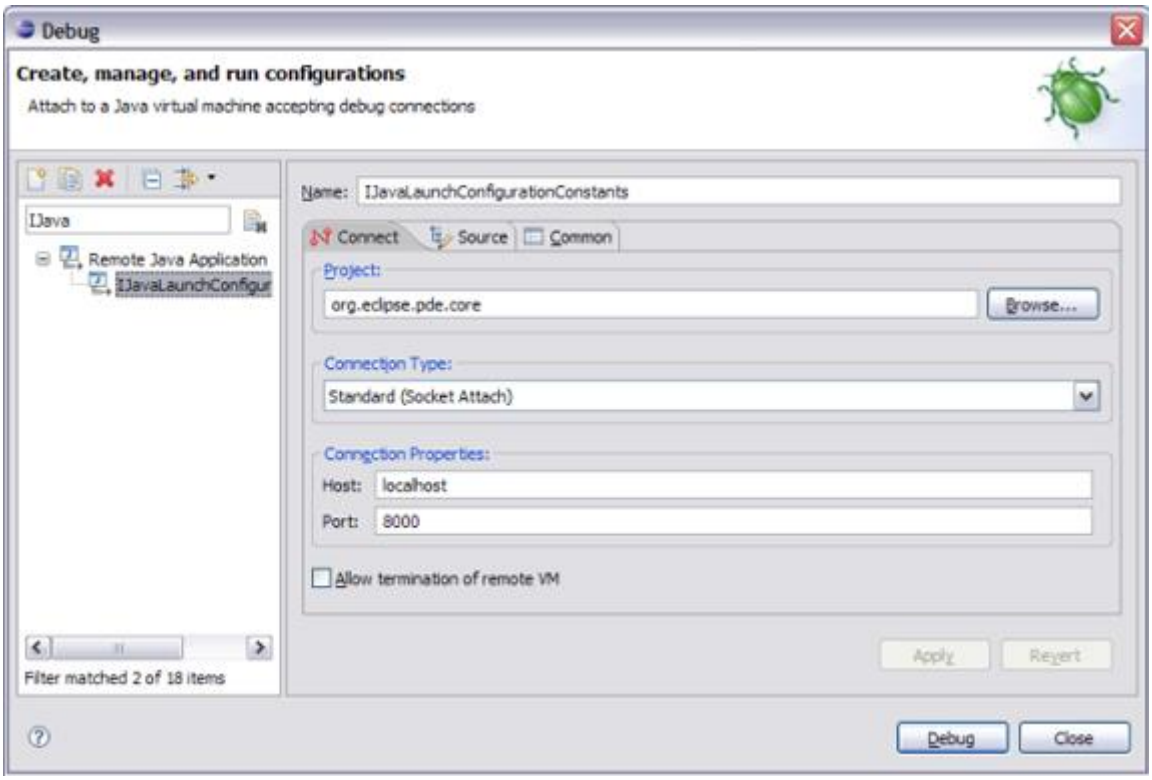


图 10. 设置一个远程调试会话的连接属性

调试其他语言

尽管 Java 语言是 Eclipse 使用的最广泛的语言，但是 Eclipse 是一个可扩展的平台，可以支持许多其他的语言。Eclipse 可以通过 C/C++ 开发工具（CDT）支持 C/C++ 。CDT 扩展了标准的 Eclipse Debug 视图，添加了调试 C/C++ 代码的功能，CDT Debug 视图可以在工作台中管理 C/C++ 项目的调试过程。CDT 中没有内部的调试器，但是它提供了一个 GNU GDB 调试器的前端，该调试器必须在本地可用。还有其他项目可以提供各自的调试器，例如 PHP 开发工具（PHP Development Tools ，PDT），请参见图 11。

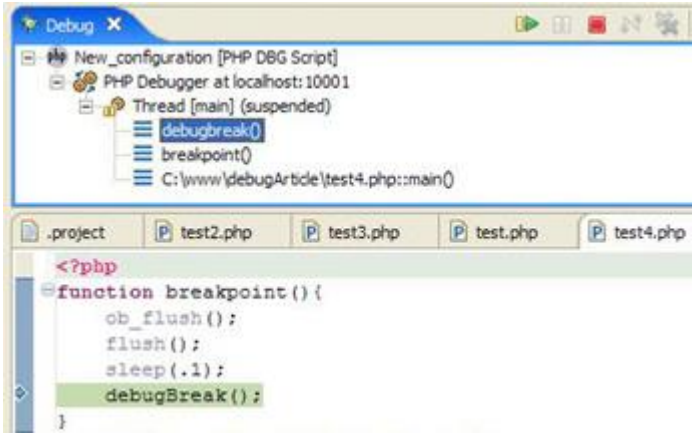


图 11 . PHP 调试器

6 结束语

Eclipse 平台提供了一个内置 Java 语言调试器，它带有标准的调试功能，包括分步执行、设置断点和值、检查变量和值以及挂起和恢复线程功能。它还可以用来调试运行在远程

机器上的应用程序。Eclipse 平台主要是一个 Java 开发环境，但是其中的 Eclipse Debug 视图还可以用于 C/C++、PHP 和许多其他的编程语言。

附：**Eclipse 快捷键调试大全**

Ctrl+M －－切换窗口的大小

Ctrl+Q －－跳到最后一次的编辑处

F2 ---重命名类名 工程名，－－当鼠标放在一个标记处出现 Tooltip 时候按 F2 则把鼠标移开时 Tooltip 还会显示即 Show Tooltip Description。

F3－－跳到声明或定义的地方。

F5－－单步调试进入函数内部。

F6 －－单步调试不进入函数内部，如果装了金山词霸 2006 则要把“取词开关”的快捷键改成其他的。

F7－－由函数内部返回到调用处。

F8－－一直执行到下一个断点。

Ctrl+Pg~ －－对于 XML 文件是切换代码和图示窗口

Ctrl+Alt+I －－看 Java 文件中变量的相关信息

Ctrl+PgUp －－对于代码窗口是打开“Show List”下拉框，在此下拉框里显示有最近曾打开的文件

Ctrl+/ －－在代码窗口中是这种//~注释。

Ctrl+Shift+/ －－在代码窗口中是这种/*~*/注释，在 JSP 文件窗口中是〈!--~--〉。

Alt+Shift+O(或点击工具栏中的 Toggle Mark Occurrences 按钮) 当点击某个标记时可使本页面中其他地方的此标记黄色凸显，并且窗口的右边框会出现白色的方块，点击此方块会跳到此标记处。

右击窗口的左边框即加断点的地方选 Show Line Numbers 可以加行号。

Ctrl+I 格式化激活的元素 Format Active Elements。

Ctrl+Shift+F 格式化文件 Format Document。

Ctrl+S 保存当前文件。

Ctrl+Shift+S 保存所有未保存的文件。

Ctrl+Shift+M(先把光标放在需导入包的类名上) 作用是加 Import 语句。

Ctrl+Shift+O 作用是缺少的 Import 语句被加入，多余的 Import 语句被删除。

Ctrl+Space 提示键入内容即 Content Assist，此时要将输入法中 Chinese(Simplified) IME-Ime / Nonlme Toggle 的快捷键（用于切换英文和其他文字）改成其他的。

Ctrl+Shift+Space 提示信息即 Context Information。

双击窗口的左边框可以加断点。

Ctrl+D 删除当前行。

第8章 Eclipse 中可视化编程

1 目的、环境

1.1 目的

熟悉 eclipse 中各种资源。

学会快速建立桌面应用程序

学会基本的单文档程序中菜单等的操作

学会桌面应用程序的基本编写方法

初步了解消息机制及事件的映射。

1.2 环境:

Eclipse + windowBuilder 插件

2 知识要点：

Windows 应用程序一般都包含有众多的图形元素，例如光标、位图、对话框等，在 Windows 环境下每一个这样的元素都是作为一种可装入应用程序的资源来存放。所谓资源，就是指用户可从中获得某种信息或进行某种操作的界面元素。

Windows 环境下的资源主要有以下几类：

- 加速键（Accelerator）：一系列按键组合，一般与菜单命令相连，作为选择菜单命令的快速方法，被应用程序用来引发一个动作。
- 工具条（Toolbar）：包含多个按钮的组合，也被用来作为快速选择菜单命令的方法。
- 光标（Cursor）：32*32 像素的位图，指示鼠标的当前位置。
- 对话框（Dialog）：包含多种控件的窗口，与用户完成交互功能。
- 图标（Icon）：代表最小化窗口的位图。
- 菜单（Menu）：以可视的方式提供了对应用程序功能的选择。
- 字符串列表（String Table）：包含一系列的格式化文本。
- 版本信息（Version）：定义应用程序的版本，包含一系列格式化文本。

这些所有类型的资源都可以由 Visual C++提供的资源编辑器进行可视的编辑。相应于不同类型的资源，Visual C++提供了不同种类的编辑器，如对话框编辑器、菜单编辑器、工具条编辑器等等，这些编辑器的具体的使用方法将在介绍有关内容的同时加以介绍。

3 快速建立程序

启动 eclipse 软件件，然后按如下步骤建立桌面程序。

创建一个空的 Java 工程

（1）第一步：选择“File->Java Project”，如图 4-1 所示。

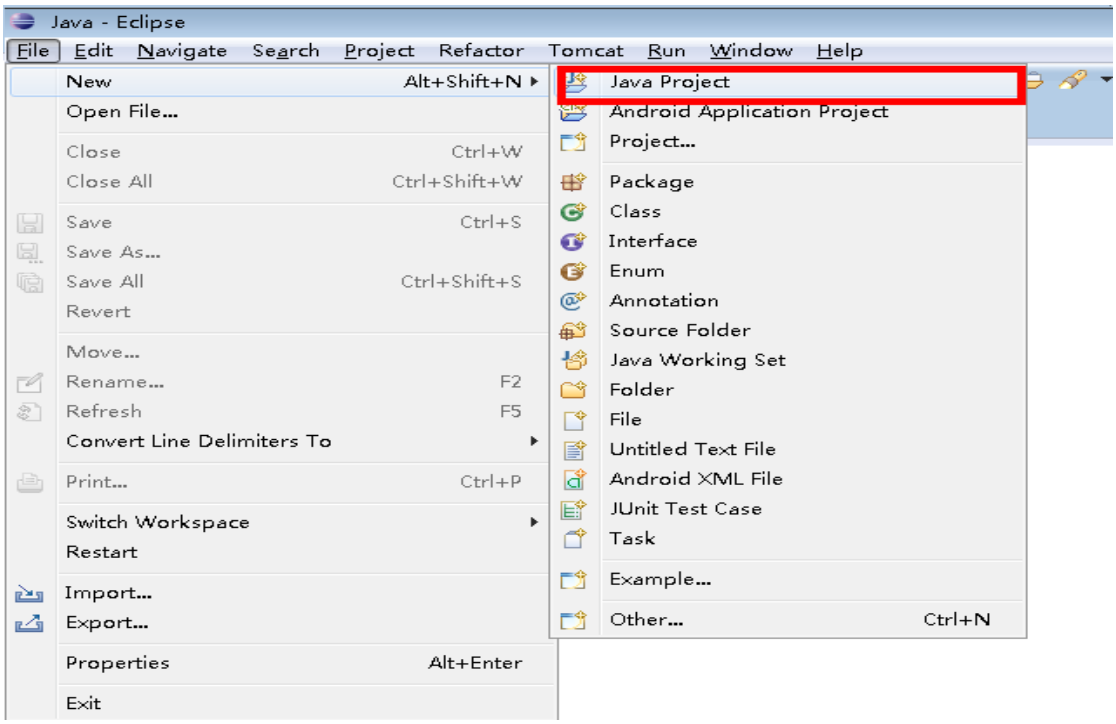


图 4-1 创建一个空的 Java 工程第 1 步

出现如图 4-2 所示的画面，输入工程名，然后按“Finish”按钮，空工程创建完毕。

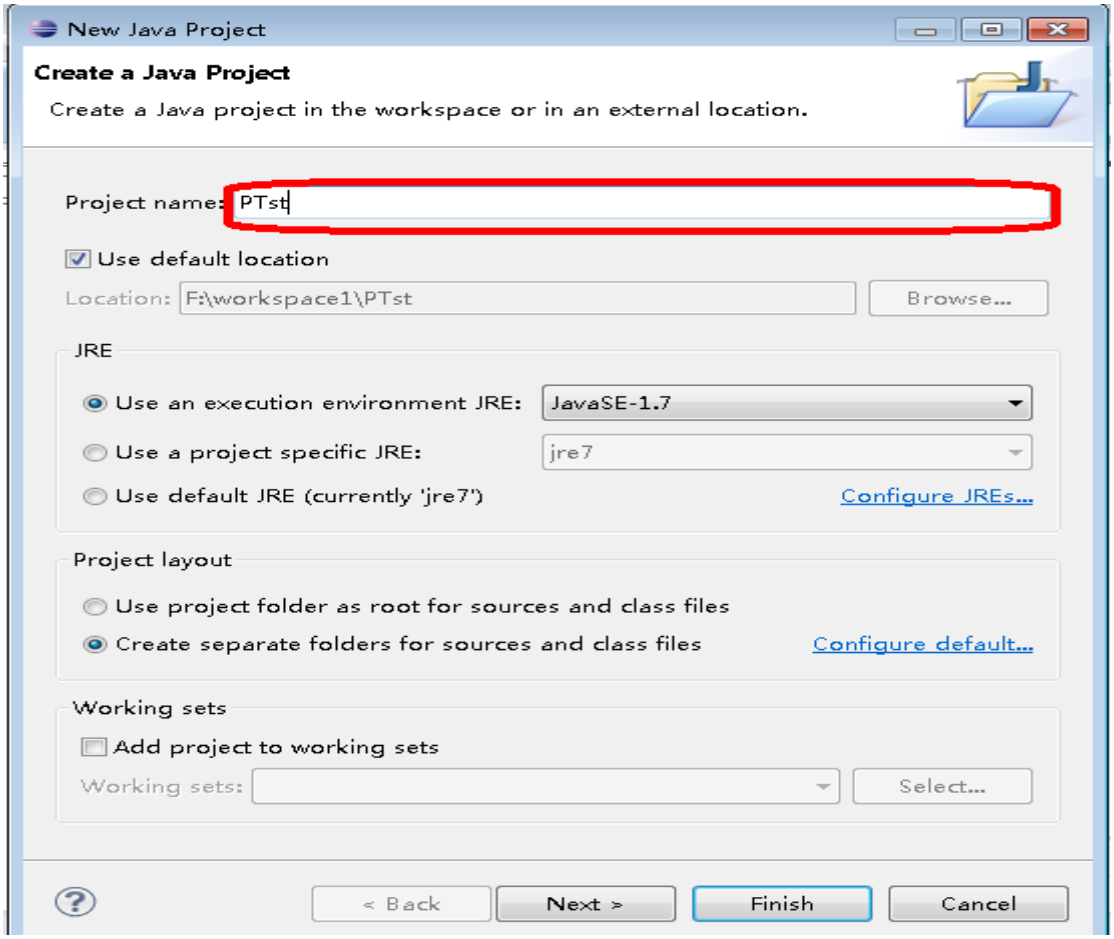


图 4-2 创建一个空的 Java 工程第 2 步

创建 swing 窗口界面

创建好空工程后，就可以在这个工程中创建程序窗口界面了，具体步骤如下。

依次选择 File→New→Other，如图 4-3 所示：

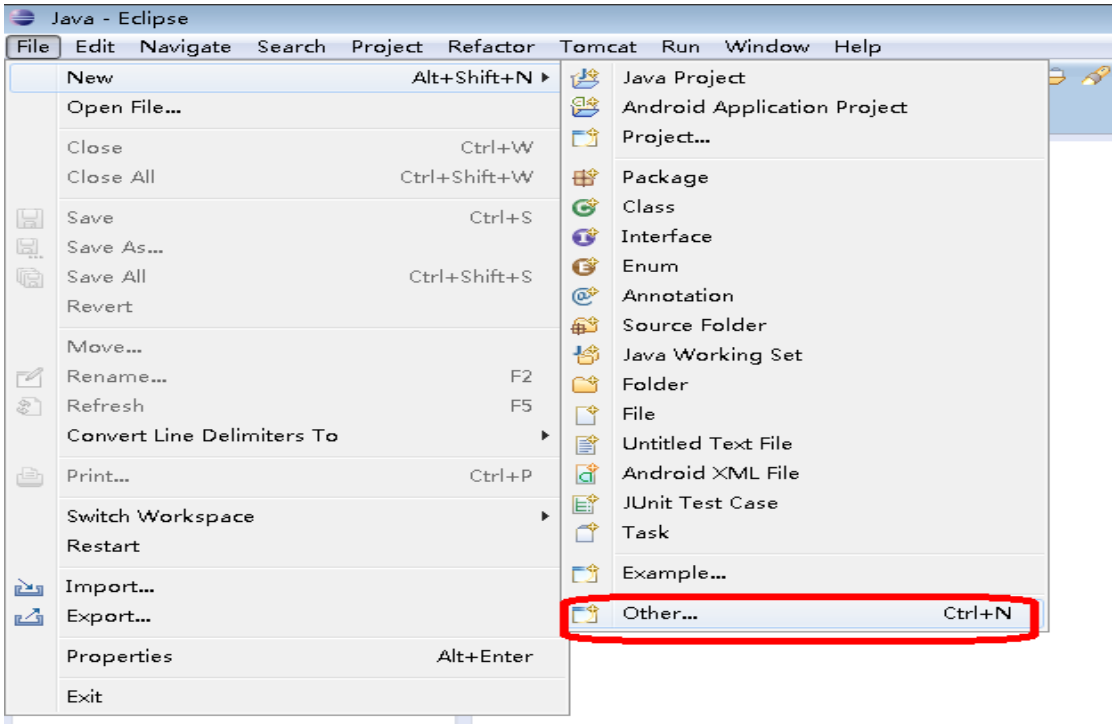


图 4-3 创建窗口界面第 1 步

弹出窗口如图 4-4 所示，移动滚动条找到 WindowBuilder->swing designer->window application。如果没有这个选项，则说明没有安装 WindowBuilder 插件。

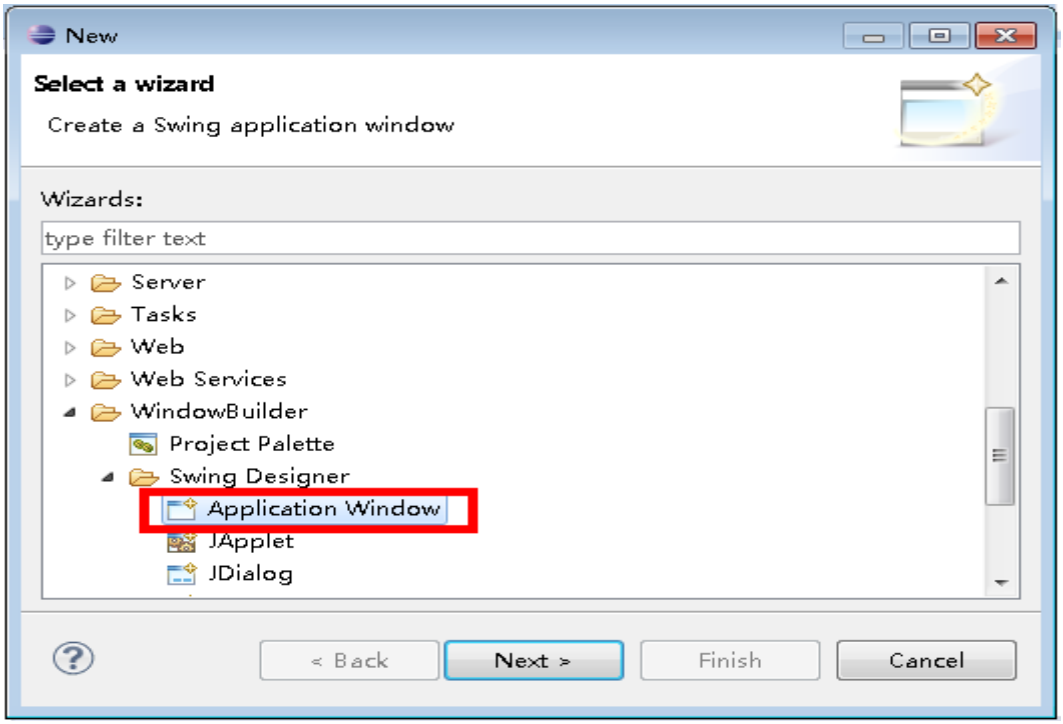


图 4-4 创建窗口界面第 2 步

操作后出现如下画面，按要求填上包名及程序名，然后按“finish”按钮。

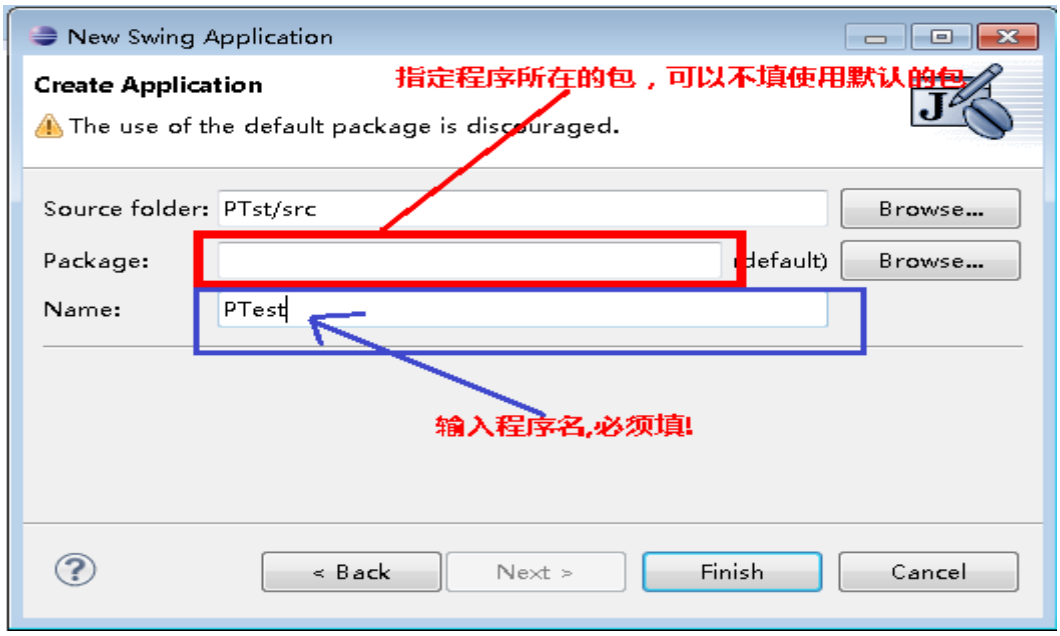


图 4-5 创建窗口界面第 3 步

按“finish”按钮之后就会自动生成一个程序窗口，如图所示是生成的窗口程序。如果想查看、编辑可视化的窗口，可按图中所示的“Design”标签。

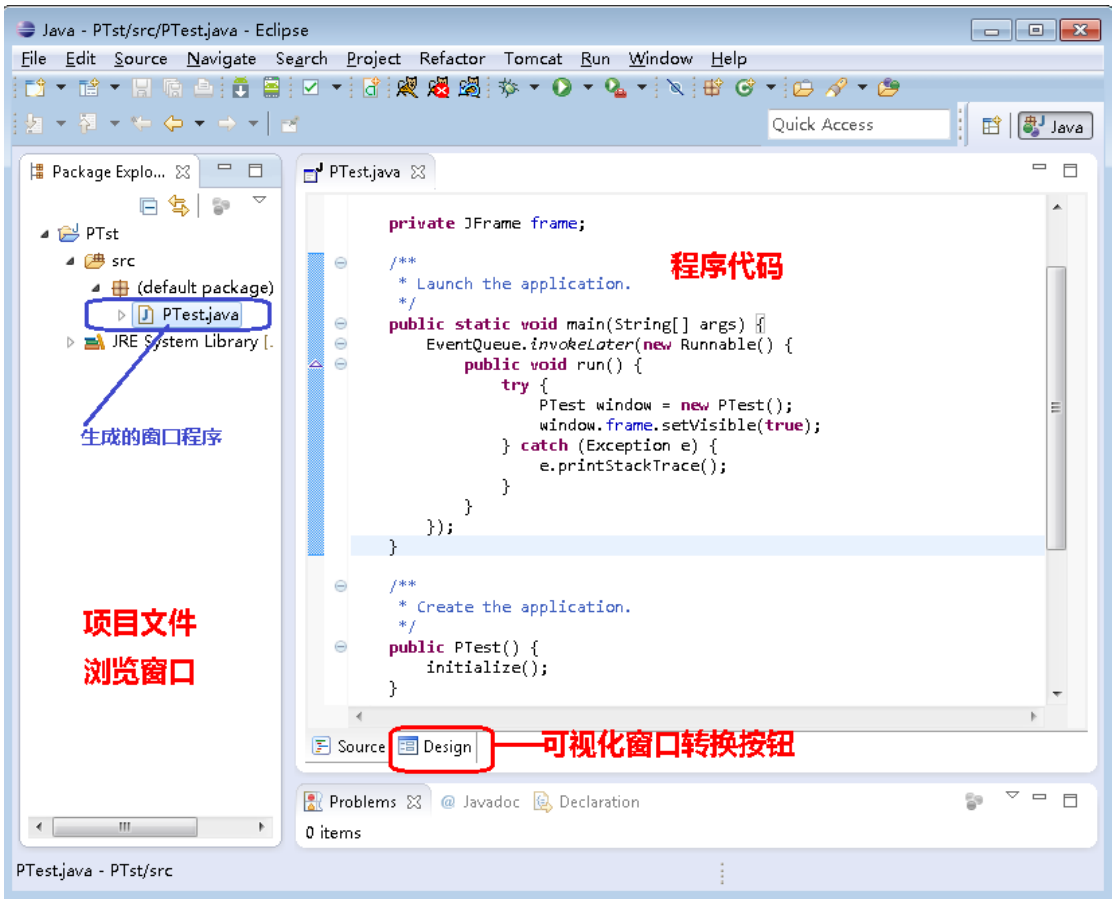


图 4-6 生成的窗口程序

鼠标点击“Design”后的 Eclipse 的界面如图 4-7 所示，即由源代码窗口转为了可视化设

计窗口，如图 4-7 所示。

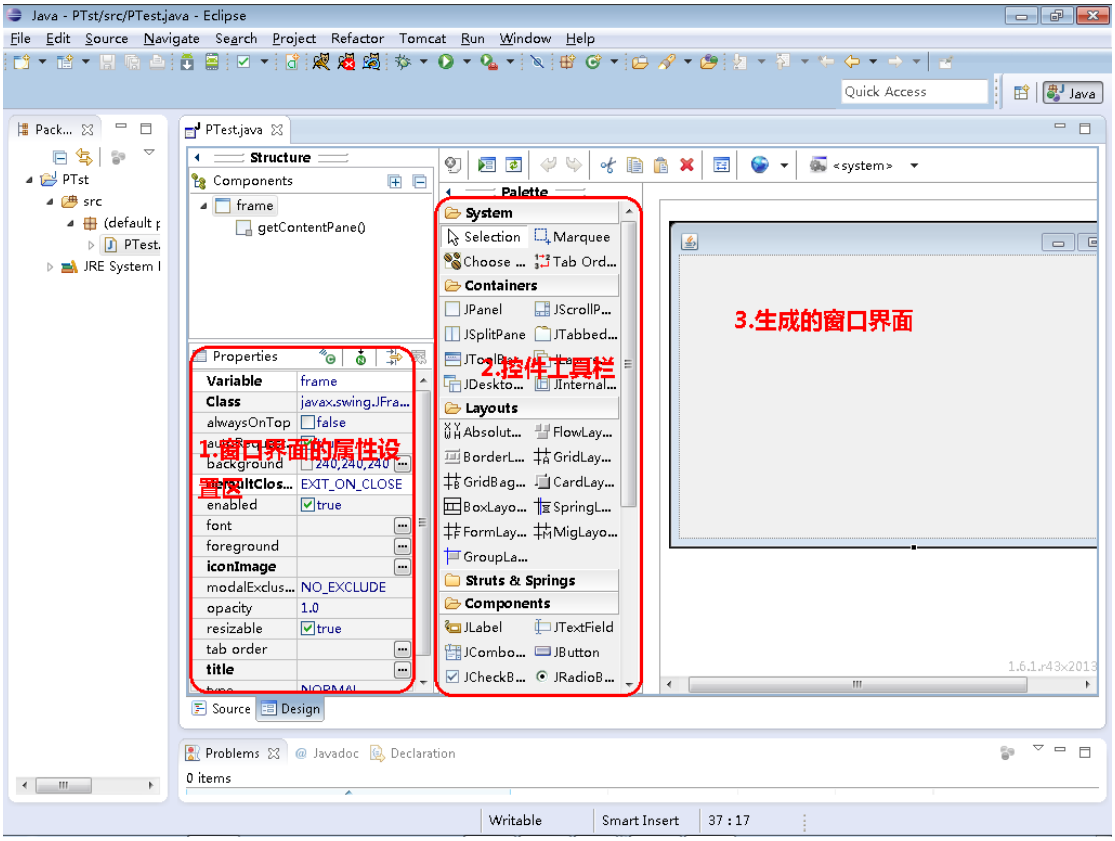


图 4-7 可视化界面设计窗口

图 4-7 中“3.生成的窗口界面”是与图 4-6 中的“程序代码”对应的，现在是可视化的状态，可以直接进行界面的设计，添加控件，比如菜单 menu、面板 Panel、设计各种布局，避免了手工填写代码。“2.控件工具栏”中是桌面程序可以用的各种控件及布局，“1.界面的属性设置”对添加到窗口中的各个控件进行性设置。

向窗口中添加菜单

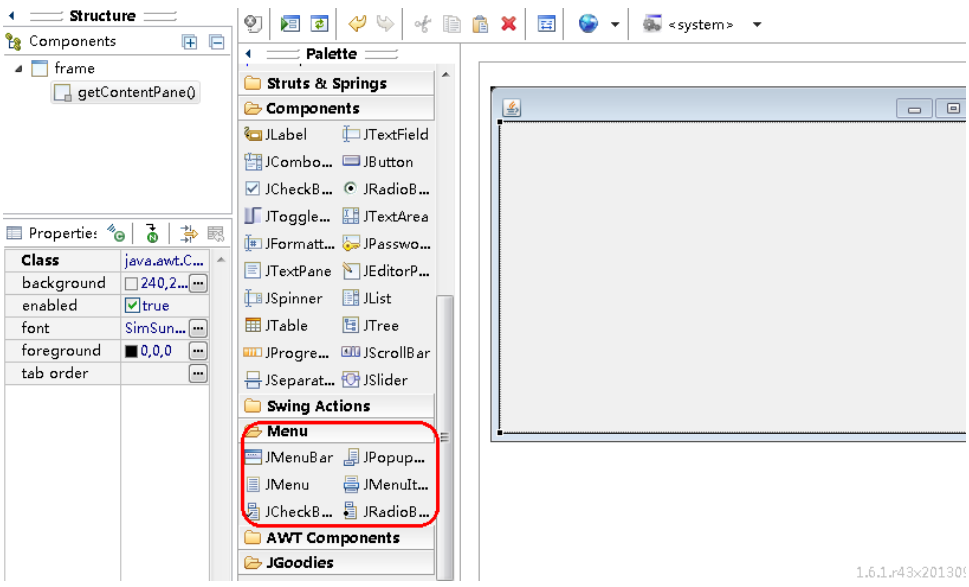


图 4-8 菜单控件的位置

菜单控件在如图 4-8 所示的位置。添加菜单的方法是

(1).首先用鼠标选中“JMenubar”，然后将鼠标移到窗口上部，再点按一下鼠标，“JMenubar”就会加到窗口中。如图所示为加入的“JMenubar”。

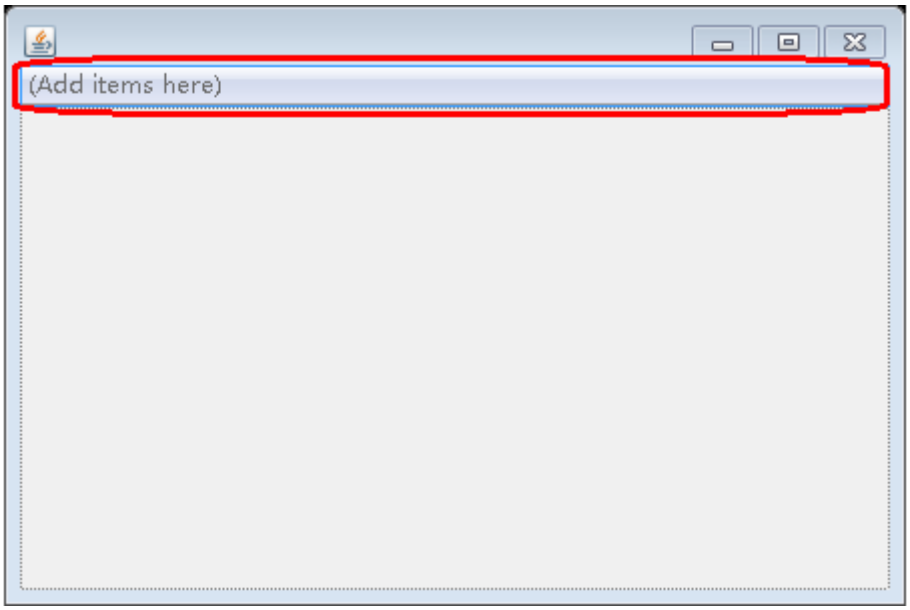


图 4-9 加入 JMenubar 后

(2).用鼠标选中“JMenu”，然后将鼠标移到窗口上部的“JMenubar”上，再点按一下鼠标，“New Menu”就会加到窗口中。加入后用鼠标选中刚加入的菜单，然后在属性窗口将菜单名“New Menu”改为正确的名称，如图所示将菜单名改为了“设置”。

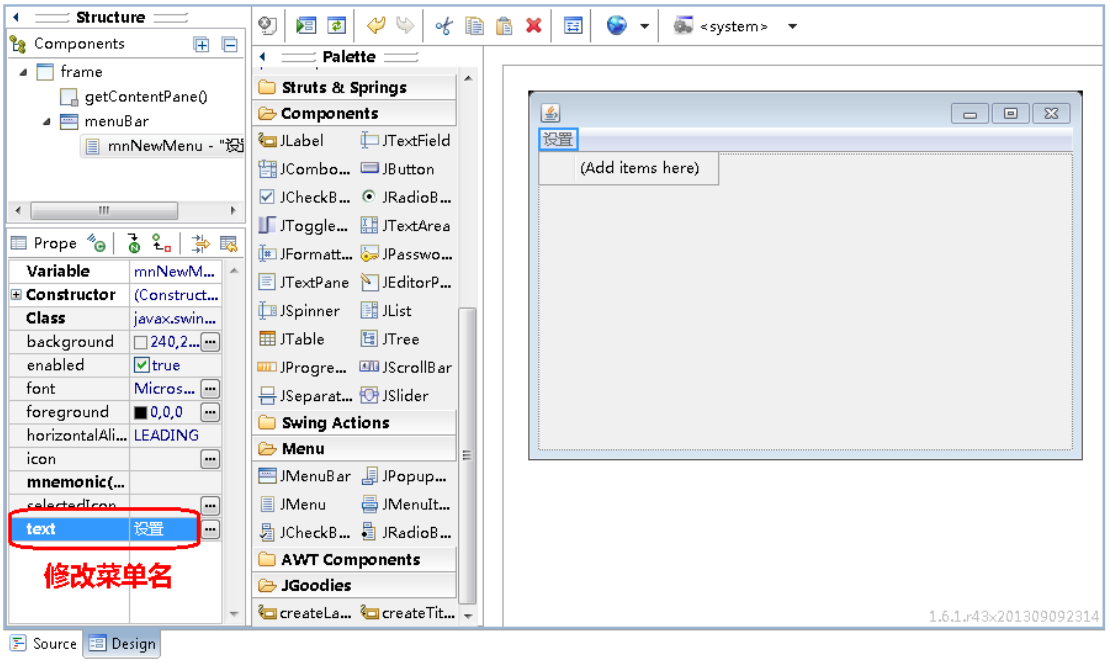


图 4-10 加入 JMenu 后

(2).加入菜单项“JMenuItem”，一个采单下面可以加多个菜单项。选中“JMenuItem”，然后移动鼠标，到菜单“设置”下面，当看到出现一条红线时点击鼠标，菜单项“New menu item”便被加入到菜单中，然后按上面的方法修改菜单项名称。如图所示改为了“端口设置”。

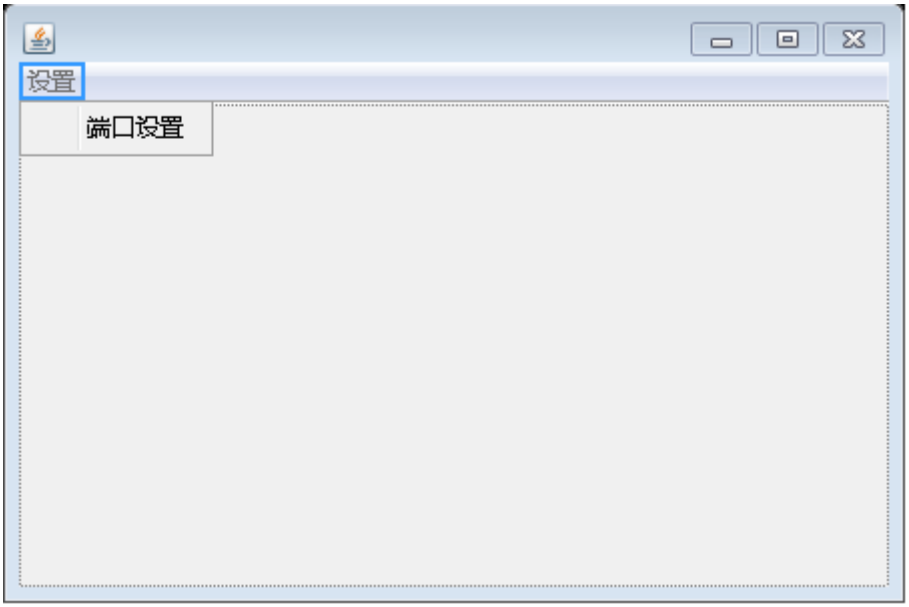


图 4-9 加入 JMenuItem 后

重复(2)-(3)两步将所需的菜单全部加入。

向菜单添加响应代码

加入菜单项后，需要加入响应事件，这样点击菜单便会执行指定的代码。加入代码的过程如下：

（1）选中菜单项

（2）然后鼠标选中“show event”图标，图中的不圆圈中的那个图标。

（3）双击“performed”

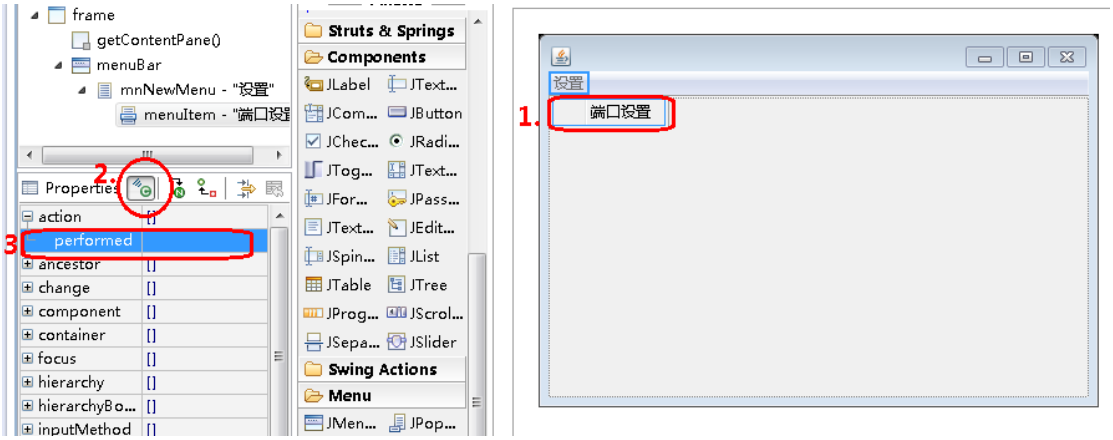


图 4-10 为“端口设置”菜单添加响应代码

双击“performed”后会自动切到源代码窗口，并且光标会定位在新加的代码位置，如图图 4-11 所示。此时可以手动添加该菜单操作的其它代码。

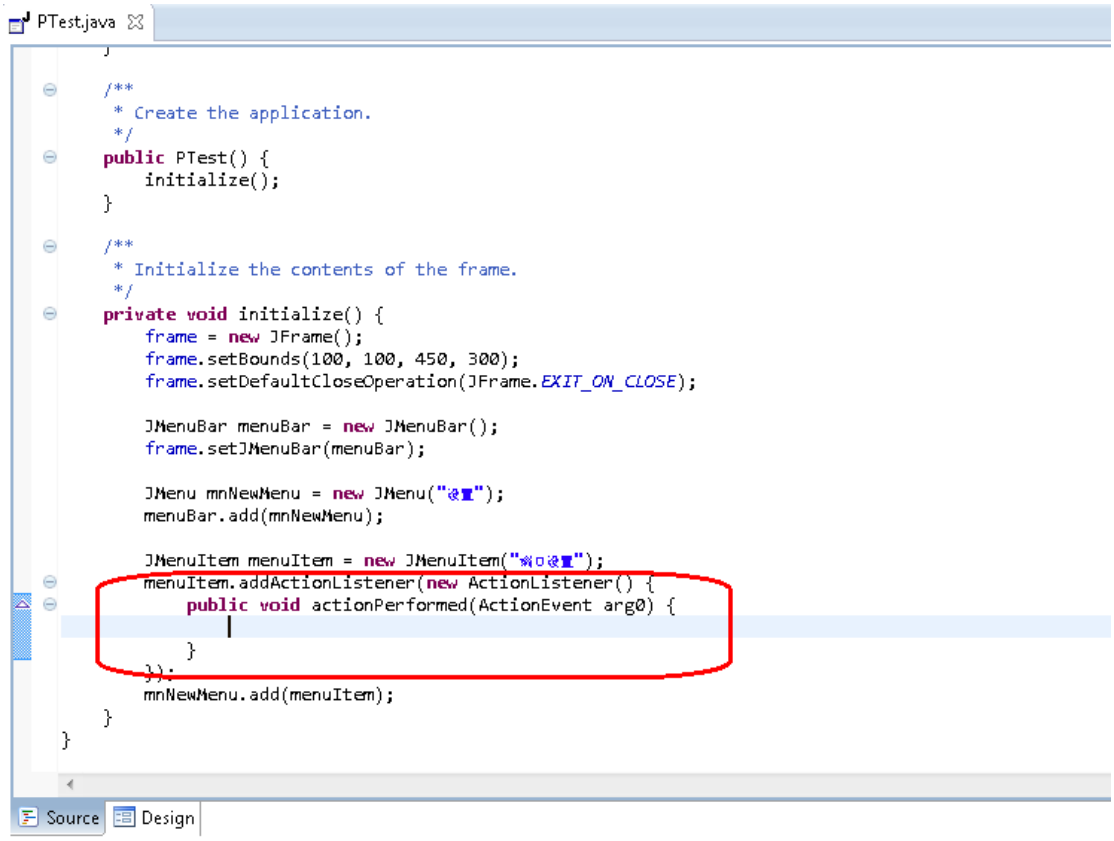
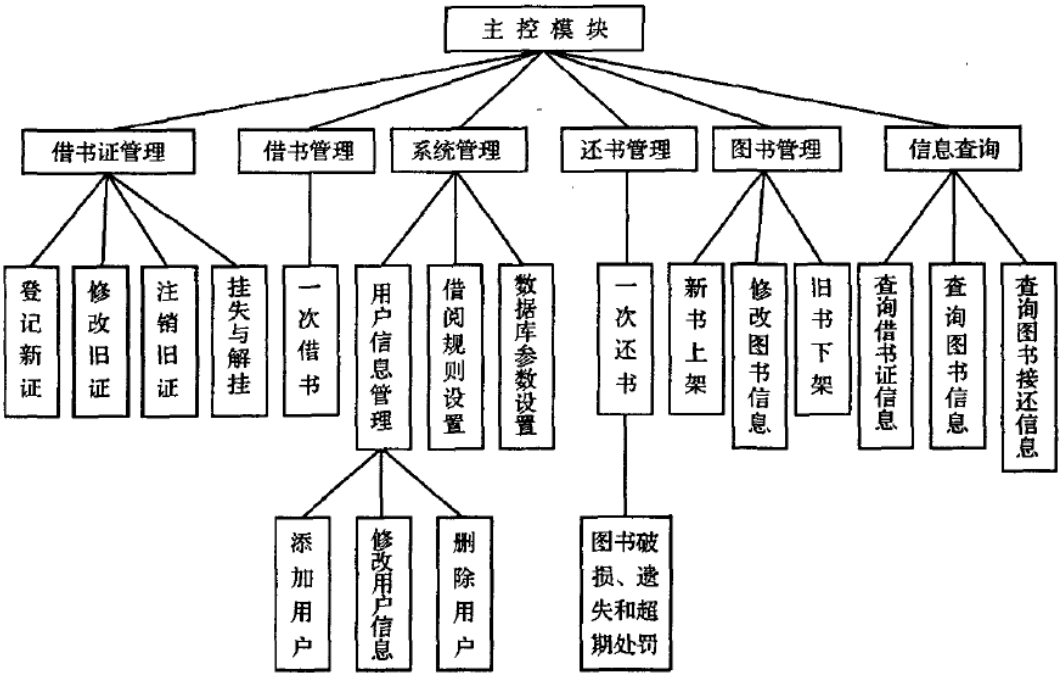


图 4-11 “端口设置”的响应代码

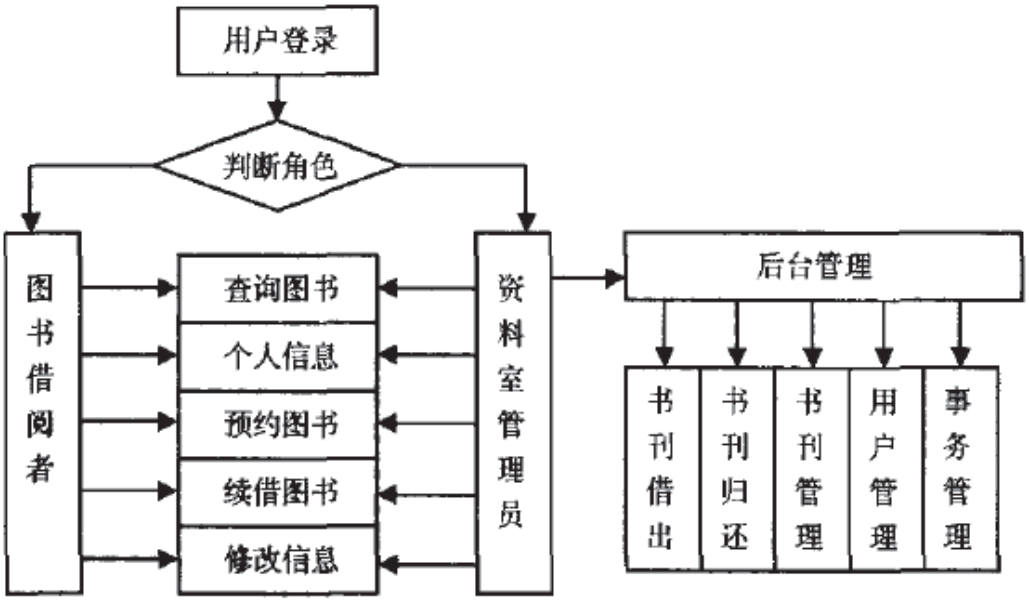
按钮及其它控件的控加方法比菜单添加的方法简单，可参考菜单添加方法，这果不再赘述。

附：

1.图书管理系统参考功能



2.图书管理系统参考功能



3.图书管理系统参考界面

2015-06-20