免责声明：此答案是自己整理的，不保证正确，错了别怪我。（不过你也不知道我们是谁是吧 hh）

1.1 Windows 平台应用程序有哪些种类？使用 VS12 如何创建这些应用程序？

答：

（1）控制台应用程序，特点：运行控制台应用程序时系统会创建控制台终端用于用户交互，交互方式限于字符形式，屏幕利用率极低。

（2）窗体应用程序，特点：使用真彩色图形化输出，支持鼠标定位，能及时响应用户操作，操作方式便利，输出效果绚丽。

（3）动态链接库，特点：不能独立运行，必须由其他运行中的程序调用才可运行。

（4）服务程序，特点：服务程序注册到用户的计算机中，随机器启动并自动执行，用户不能直接运行或终止服务程序；服务程序没有运行界面，也不与用户交互，用户只能通过特殊控制命令使其启动或终止运行。

（5）Web 应用程序，特点：基于 HTML 语言的网页程序，将程序控制语句嵌入在标准的 HTML 文本中。

如何创建略。

5.1 简述异常机制能否由判断语句替代。

（1）程序执行过程中会由于客观情况无法完成预定的任务，例如因为连接参数不正确及数据库软件没有启动导致数据库连接不上、有的程序引用了没有初始化的对象，普通的判断语句很难甚至不能解决这钟问题。但是异常机制可以解决程序无法正常执行而产生特殊跳转方式，提供用户反馈信息，减少程序崩溃带来的麻烦。异常处理虽然不能改变程序无法正常运行的事实，但是给出具有参考价值的出错信息，并且能够大大增强系统的健壮性。

（2）举一个很简单的例子。如果在错误发生的地方虽然知道了发生的是具体哪一类错误，但却没有足够的信息去将错误很好地呈现和处理，这时程序需要将错误向上传递，直到有足够的信息或者说到了从逻辑上讲应该处理该错误的地方再去处理该错误。异常机制里可以将异常往上抛，而判断语句很难做到这一点，即使能做到，也会使得逻辑结构十分混乱，使代码可读性差。由此可以看出，异常机制大大简化了解决问题的方式，远比判断语句更适合处理此类问题。

5.2 简述具有继承关系的异常类的捕获顺序是什么样的。

多个 catch 块的捕获顺序是从顶部到底部，对于所引发的每个异常，都只执行一个 catch 块，如果一系列的 catch 块所设定的异常类存在继承关系，会按照 catch 出现的顺序找到匹配的第 1 个类，并执行其相应的代码，不再执行后续可匹配的异常类。异常是按照最先匹配处理，而不是最佳匹配，如果将捕获基类异常代码写在前面，则基类的代码先被调用而后面的代码则被忽略。

所以编写 try 语句组时异常类捕获次序应按照类派生的逆序出现，即异常派生类语句写在异常基类之前。

7.1 比较 MySQL 和其他数据库产品使用的 SQL 语句的不同。

与 SQL Server 相比：

(1) mysql 以;结束一条 SQL 语句，SQL server 以;或 go 或不写结束都可以

(2) 查询前 10 条记录：

mysql 语句

```
select * from student limit 10;
```

sql server 语句

```
select top 10 * from student ;
```

(3) 从数据库定位到某张表

mysql 写法：库名.表名

```
select password from Info.users where userName='boss'
```

Sqlserver 写法：库名.dbo.表名 ；或者：库名..表名 （注：中间使用两个点）

```
select password from Info.dbo.users where userName='boss'
```

或者

```
select password from Info..users where userName='boss'
```

(4) 强制不使用缓存查询

(5) 查询 temp 表

mysql 写法：

```
select SQL_NO_CACHE * from temp
```

Sqlserver 的没有，它只缓存 sql 的执行计划，不会缓存结果。

与 Oracle 相比：

(1) mysql 可以创建数据库，而 oracle 没有这个操作，oracle 只能创建实例；

(2) 创建表时：

mysql：

1、mysql 没有 number、varchar2()类型；

2、mysql 可以声明自增长：auto_increment；

3、mysql 有 double 类型；

oracle：

1、oracle 没有 double 类型、有 int 类型但多数会用 number 来代替 int；

2、oracle 不可以声明自增长：auto_increment，主键自带自增长；

3、oracle 小数只有 float 类型；

(3) 添加列时：

MySQL:

A. alter table 表名 add column 字段 数据类型;

B. alter table 表名 add column 字段 1 数据类型, add column 字段 2 数据类型;

注：其中关键字 column 可有可无。

Oracle:

A. alter table 表名 add 字段 数据类型;

B. alter table 表名 add (字段 数据类型);

C. alter table 表名 add (字段 1 数据类型, 字段 2 数据类型);

①主键

Mysql 一般使用自动增长类型，在创建表时只要指定表的主键为 auto increment,插入记录时，不需要再指定该记录的主键值，Mysql 将自动增长；Oracle 没有自动增长类型，主键一般使用的序列，插入记录时将序列号的下一个值付给该字段即可；只是 ORM 框架是只要是 native 主键生成策略即可。

②单引号的处理

MYSQL 里可以用双引号包起字符串，ORACLE 里只可以用单引号包起字符串。在插入和修改字符串前必须做单引号的替换：把所有出现的一个单引号替换成两个单引号。

③翻页的 SQL 语句的处理

MYSQL 处理翻页的 SQL 语句比较简单，用 LIMIT 开始位置, 记录个数；ORACLE 处理翻页的 SQL 语句就比较繁琐了。每个结果集只有一个 ROWNUM 字段标明它的位置, 并且只能用 ROWNUM<100, 不能用 ROWNUM>80

10.1 动态链接库的执行方式上有什么特点。

（1）动态链接是相对于静态链接而言的。所谓静态链接是指把要调用的函数或者过程链接到可执行文件中，成为可执行文件的一部分。而动态链接所调用的函数代码并没有被拷贝到应用程序的可执行文件中去，而是仅仅在其中加入了所调用函数的描述信息。仅当应用程序被装入内存开始运行时，才在应用程序与相应的 DLL 之间建立链接关系。当要执行所调用 DLL 中的函数时，根据链接产生的重定位信息，Windows 才转去执行 DLL 中相应的函数代码。

（2）动态链接有两种方式，载入时动态链接和运行时动态链接。载入时动态链接需要在链接时将函数所在 DLL 的导入库链接到可执行文件中，调用模块可以像调用本模块中的函数一样直接使用导出函数名调用 DLL 中的函数。运行时动态链接避免了导入库文件，运行时可以通过 LoadLibrary 或 LoadLibraryEx 函数载入 DLL，之后模块可通过调用 GetProcAddress 获取 DLL 函数的入口地址去执行函数。

（3）使用 DLL 中的函数与程序自身的函数没有区别，DLL 有自己的数据段，没有自己的堆栈，使用与调用它的应用程序相同的对堆栈模式，它在运行时需要分配的内存是属于它的进程的，不同程序即使调用相同函数所分配的内存也不会相互影响，DLL 函数中的代码所创建的任何对象（包括变量）都归调用它的线程或进程所有。

（4）1 个 DLL 在内存中只有一个实例，系统为每个 DLL 维护一个线程级引用计数，每当线程载入该 DLL，引用计数加 1，而程序终止引用计数会变为 0（仅指运行时动态链接库），系统会释放 DLL 占用的虚拟空间。

10.2 反射机制在程序中能起到什么作用。

（1）反射可以通过 System.Reflection 命名空间中的类以及 System.Type，可以获取有关已加载的程序集和在其中定义的类型（如类、接口和值类型）的信息

（2）可以使用反射在运行时创建类型实例，调用和访问这些实例

（3）使用 Assembly 定义和加载程序集，加载在程序集清单中列出的模块，以及从此

程序集中查找类型并创建该类型的实例。

（4）使用 Module 获得包含模块的程序集以及模块中的类等。

（5）获取在模块上定义的所有全局方法或其他特定的非全局方法。

（6）使用 ConstructorInfo 获得构造函数的名称、参数、访问修饰符（如 public 或 private）和实现详细信息（如 abstract 或 virtual）等。

（7）使用 Type 的 GetConstructors 或 GetConstructor 方法来调用特定的构造函数。

（8）使用 MethodInfo 获得方法的名称、返回类型、参数、访问修饰符（如 public 或 private）和实现详细信息（如 abstract 或 virtual）等。

（9）使用 Type 的 GetMethods 或 GetMethod 方法来调用特定的方法。

（10）使用 FieldInfo 获得字段的名称、访问修饰符和实现详细信息（如 static）等；并获取或设置字段值。

（11）使用 EventInfo 获得事件的名称、事件处理程序数据类型、自定义属性、声明类型和反射类型等；并添加或移除事件处理程序。

（12）使用 PropertyInfo 获得属性的名称、数据类型、声明类型、反射类型和只读或可写状态等；并获取或设置属性值。

（13）使用 ParameterInfo 获得参数的名称、数据类型、参数是输入参数还是输出参数，以及参数在方法签名中的位置等。

10.3 什么数据类型称为 blittable type

数据在托管和非托管环境中的定义和表示有区别也有相同的地方，.NET 平台中 Byte、Int16、UInt16、Int32、UInt32、Int64、UInt64、IntPtr、UIntPtr 等以及包含这些类型的一维数组在两种环境中的内存表示是相同的，称为 blittable 类型，而 Boolen、Char、Object 和 String 等类型不属于 blittable 类型。使用 bittable 类型可以有效地在托管和非托管环境中来回复制，对于复杂的数据类型则需要使用 Marshal 类进行转换。

10.4 托管代码调用非托管代码要注意哪些地方。

托管代码中调用非托管 API 时参数必须是 blittable 类型，当使用非 blittable 类型的参数时，需要遵循一些其他的规则，以 StringBuffer 类型的参数举例，当托管代码在使用 StringBuffer 类型作为非托管代码中 string 类型的参数时，要满足：

（1）不采用引用方式

（2）保证非托管代码使用 Unicode，或者使用 LPWSTR

（3）事先设置 StringBuffer 合适的尺寸避免溢出

当托管代码调用的是非托管的动态链接库时。.NET 平台使用 Interop 服务支持对非托管链接库的调用，在使用 Interop 服务将 API 声明为函数外部方法要注意下面三点。

（1）extern 修饰符声明调用外部 API

（2）使用 DllImport 指定库文件和方法的参数格式

（3）使用 static 关键字声明方法为静态

10.5 什么是 DLL 地狱问题。

（1）重新编译生成的 DLL 可能造成意想不到的后果，用户程序使用新版本的 DLL 库不能正常工作称为 DLL 地狱问题。

（2）DLL 地狱（DLL Hell）是指因为系统文件被覆盖而让整个系统像是掉进了地狱。

（3）简单地讲， DLL Hell 是指当多个应用程序试图共享一个公用组件（如某个动态连接库（DLL）或某个组件对象模型（COM）类）时所引发的一系列问题。最典型的情况是，某个应用程序将要安装一个新版本的共享组件，而该组件与机器上的现有版本不向后兼容。虽然刚安装的应用程序运行正常，但原来依赖前一版本共享组件的应用程序也许已无法再工作。在某些情况下，问题的起因更加难以预料。比如，当用户浏览某些 Web 站点时会同时下载某个 Microsoft ActiveX® 控件。如果下载该控件，它将替换机器上原有的任何版本的控件。如果机器上的某个应用程序恰好使用该控件，则很可能也会停止工作。

（4）这些问题的原因是应用程序不同组件的版本信息没有由系统记录或加强。而且系统为某个应用程序所做的改变会影响机器上的所有应用程序—现在建立完全从变化中隔离出来的应用程序并不容易。

（5）尽量避免 DLL 地狱的方法包括：不直接生成类的实例，不直接访问成员变量，不使用虚函数。

11.1 程序的并发与并行区别是什么。

程序的并行是指两个或多个线程在微观上（事实上）同一时刻在 CPU 的两个或多个核上运行，只有多核或者多 CPU 的机器中才能实现。而程序的并发是指多个就绪的线程在 Windows 平台排列成环形队列，一次分配时间片后运行，此情况下多个线程不在同一时刻运行，而是分享时间片交替运行，程序并发在宏观上表现为并发运行的形式，而在微观上则是

串行。并行实际上是指计算机同时做多个任务，而并发则是计算机交替做多个任务。

11.2 HANDEL 值是否就是内核对象的内存指针地址。

（1）不是。HANDEL(句柄)在 WinNT.h 中的定义为 typedef PVOID HANDLE

（2）句柄是一个 4B（64 位程序中为 8B）长的数值，用于标识应用程序中的不同实例，应用程序通过句柄访问相应对象的信息。

（3）在 Windows 系统中由于经常进行对空闲对象的内存释放和重新提交，对象的物理地址是经常变化的。因此操作系统禁止程序用物理地址去访问对象内容。

句柄不是对象的指针，虽然句柄的值是确定的，但是必须使用指定的 API 才可操作句柄指向的对象。

（4）系统在进程初始化时分配一个当前进程所有内核的句柄表。句柄表中每项内容具有内核对象的指针、访问权限掩码和一些标志位。内核对象创建函数返回句柄值。

（5）除此之外，HANDLE 值是进程相关的，同一个 HANDLE 值在不同的进程中意义是不一样的，不代表相同的内核对象。这进一步说明了 HANDLE 值和内存指针地址不同。

13.1 请举例说明最顶窗体可以不是激活的窗体。

（1）最顶窗体是针对窗体的前后关系而言的。Windows 系统通过垂直于屏幕的 Z 坐标给每个对象设置一个 Z-Order 值，用来表示窗体重叠的前后次序，最顶的窗体覆盖所有其他窗体，最低的窗体则被所有窗体覆盖，最顶的窗体具有 WS_EX_TOPMOST 样式值。

（2）激活窗体是针对系统消息队列分派目标而言的。操作系统把用户的输入统一处理为消息并放入系统消息队列，派送到当前激活窗体，隐藏状态的窗体不接受用户输入。Windows 操作系统仅有一个激活窗体，用户通过鼠标、键盘产生的信息都发送到激活窗体。

（3）由此可见，最顶窗体和激活窗体是两个不同的概念，最顶窗体可以不是激活窗体，一个具体的例子就是软键盘窗体。

（4）在软键盘窗体程序中，软键盘需要始终展示在最前端覆盖其他窗体，因此软键盘是一个最顶窗体。但是当用户点击软件盘上的按钮时，软键盘窗体不会获得焦点，而且此时会向程序中真正激活的窗体（可能是一个写字版窗体）发送字符。另外，此时用户的鼠标、键盘的输入也都会被发送到真正激活窗体而不是软键盘窗体。所以软键盘窗体虽然是一个最顶窗体，却并不是一个激活窗体。

往年卷子

一、 简答题

1-5 在上文出现了

6.简述 Windows 应用程序中的消息机制。

windows 具有一个系统消息队列按序存储全部消息，并根据规则将详细投放到进程的消息队列中。系统为每一个窗体对象创建一个消息队列，消息静分配后由系统队列进入到窗体队列，每一个窗体对象配置一个窗体线程运行消息循环任务。消息循环反复检查消息队列中的消息，根据消息值匹配执行相应的分支代码。 消息可以由系统自动派送，也可以由程序主动向其他程序发送，发送方式有两种，一种方式是将消息发送到先进先出消息队列结构中，这些消息也叫队列化消息，另一种方式是将消息直接发送到窗体函数中，这些消息叫非队列化消息。

驱动程序将用户的键盘与鼠标输入转化为消息结构放入系统消息队列，消息被分配到当前激活的窗体线程，窗体消息处理函数对消息进行匹配。而非队列和的信息则直接发送到了窗体过程。(如果感觉必要请在答卷时加上 123 页图 13-4)。

二、综合题

1.结合个人上机实验，讨论 COM 组件的创建和调用过程，并通过相应的代码示例，展现其过程。

见老师发的源码

2.结合个人上机实验，讨论托管动态链接库 DLL 的创建和调用过程，并通过相应的代码示例，展现其流程。

书 P90

```
private void btn1_Click_1(object sender, RoutedEventArgs e)
{
    clearComments();
    string doc_file_name = Directory.GetCurrentDirectory() +
@"\Files\content.doc";

    MsWord.Application oWordApplic;//a reference to Wordapplication
    MsWord.Document oDoc;//a reference to thedocument
    try
    {
            if (File.Exists(doc_file_name))
        {
            File.Delete(doc_file_name);
        }
        oWordApplic = new MsWord.Application();
        object missing = System.Reflection.Missing.Value;

        MsWord.Range curRange;
        object curTxt;
        int curSectionNum = 1;
        oDoc = oWordApplic.Documents.Add(ref missing, ref missing, ref missing,
ref missing);
        oDoc.Activate();
        Console.WriteLine(" 正在生成文档小节");
        showComment("正在生成文档小节");

        object section_nextPage = MsWord.WdBreakType.wdSectionBreakNextPage;
        object page_break = MsWord.WdBreakType.wdPageBreak;
        //添加三个分节符，共四个小节
        for (int si = 0; si < 4; si++)
        {
            oDoc.Paragraphs[1].Range.InsertParagraphAfter();
            oDoc.Paragraphs[1].Range.InsertBreak(ref section_nextPage);
        }

        Console.WriteLine(" 正在插入摘要内容");
        showComment("正在插入摘要内容");
        #region 摘要部分
        curSectionNum = 1;
        curRange = oDoc.Sections[curSectionNum].Range.Paragraphs[1].Range;
        curRange.Select();
        string one_str, key_word;
        //摘要的文本来自 abstract.txt 文本文件
        StreamReader file_abstract = new
StreamReader(Directory.GetCurrentDirectory() + @"\Files\abstract.txt");
        oWordApplic.Options.Overtype = false;//overtype 改写模式
        MsWord.Selection currentSelection = oWordApplic.Selection;
        if (currentSelection.Type == MsWord.WdSelectionType.wdSelectionNormal)
        {
            one_str = file_abstract.ReadLine();//读入题目
            currentSelection.TypeText(one_str);
            currentSelection.TypeParagraph(); //添加段落标记
```

```
            currentSelection.TypeText(" 摘要");//写入" 摘要" 二字
            currentSelection.TypeParagraph(); //添加段落标记
            key_word = file_abstract.ReadLine();//读入题目
            one_str = file_abstract.ReadLine();//读入段落文本
            while (one_str != null)
            {
                currentSelection.TypeText(one_str);
                currentSelection.TypeParagraph(); //添加段落标记
                one_str = file_abstract.ReadLine();
            }
            currentSelection.TypeText(" 关键字：");
            currentSelection.TypeText(key_word);
            currentSelection.TypeParagraph(); //添加段落标记
        }
        file_abstract.Close();
        //摘要的标题
        curRange = oDoc.Sections[curSectionNum].Range.Paragraphs[1].Range;
        curTxt = curRange.Paragraphs[1].Range.Text;
        curRange.Font.Name = " 宋体";
        curRange.Font.Size = 22;
        curRange.Paragraphs[1].Alignment =
        MsWord.WdParagraphAlignment.wdAlignParagraphCenter;
        //" 摘要" 两个字
        curRange = oDoc.Sections[curSectionNum].Range.Paragraphs[2].Range;
        curRange.Select();
        curRange.Paragraphs[1].Alignment =
        MsWord.WdParagraphAlignment.wdAlignParagraphCenter;
        curRange.Font.Name = " 黑体";
        curRange.Font.Size = 16;
        //摘要正文
        oDoc.Sections[curSectionNum].Range.Paragraphs[1].Alignment =
        MsWord.WdParagraphAlignment.wdAlignParagraphCenter;
        for (int i = 3; i < oDoc.Sections[curSectionNum].Range.Paragraphs.Count;
i++)
        {
            curRange = oDoc.Sections[curSectionNum].Range.Paragraphs[i].Range;
            curTxt = curRange.Paragraphs[1].Range.Text;
            curRange.Select();
            curRange.Font.Name = " 宋体";
            curRange.Font.Size = 12;
            oDoc.Sections[curSectionNum].Range.Paragraphs[i].LineSpacingRule =
            //oDoc.Paragraphs[i].LineSpacingRule =
            MsWord.WdLineSpacing.wdLineSpaceMultiple;
            //多倍行距，1.25 倍，这里的浮点值是以 point 为单位的，不是行距倍数
            oDoc.Sections[curSectionNum].Range.Paragraphs[i].LineSpacing = 15f;

oDoc.Sections[curSectionNum].Range.Paragraphs[i].IndentFirstLineCharWidth(2);
        }
        //设置" 关键字：" 为黑体
        curRange = curRange.Paragraphs[curRange.Paragraphs.Count].Range;
        curTxt = curRange.Paragraphs[1].Range.Text;
        object range_start, range_end;
        range_start = curRange.Start;
        range_end = curRange.Start + 4;
```

```
        curRange = oDoc.Range(ref range_start, ref range_end);
        curTxt = curRange.Text;
        //curRange = curRange.Range(ref range_start, ref range_end);
        curRange.Select();
        curRange.Font.Bold = 1;
        #endregion 摘要部分



        Console.WriteLine(" 正在插入目录");
        showComment("正在插入目录");
        #region 目录
        curSectionNum = 2;
        curRange = oDoc.Sections[curSectionNum].Range.Paragraphs[1].Range;
        curRange.Select();
        //插入目录时指定的参数
        object useheading_styles = true;//使用内置的目录标题样式
        object upperheading_level = 1;//最高的标题级别
        object lowerheading_level = 3;//最低标题级别
        object usefields = 1;//true 表示创建的是目录
        object tableid = 1;
        object RightAlignPageNumbers = true;//右边距对齐的页码
        object IncludePageNumbers = true;//目录中包含页码
        currentSelection = oWordApplic.Selection;
        currentSelection.TypeText(" 目录");
        currentSelection.TypeParagraph();
        currentSelection.Select();
        curRange = oDoc.Sections[curSectionNum].Range.Paragraphs[2].Range;
        //插入的表格会代替当前 range
        //range 为非折叠时，TablesOfContents 会代替 range，引起小节数减少。
        curRange.Collapse();
        oDoc.TablesOfContents.Add(curRange, ref useheading_styles, ref
upperheading_level,
        ref lowerheading_level, ref usefields, ref tableid, ref
RightAlignPageNumbers,
        ref IncludePageNumbers, ref missing, ref missing, ref missing, ref
missing);
        oDoc.Sections[curSectionNum].Range.Paragraphs[1].Alignment =
        MsWord.WdParagraphAlignment.wdAlignParagraphCenter;
        oDoc.Sections[curSectionNum].Range.Paragraphs[1].Range.Font.Bold = 1;
        oDoc.Sections[curSectionNum].Range.Paragraphs[1].Range.Font.Name = " 黑
体";
        oDoc.Sections[curSectionNum].Range.Paragraphs[1].Range.Font.Size = 16;
        #endregion 目录



        #region 第一章
        showComment("正在插入第一章内容");
        curSectionNum = 3;
        oDoc.Sections[curSectionNum].Range.Paragraphs[1].Range.Select();
        curRange = oDoc.Sections[curSectionNum].Range.Paragraphs[1].Range;
        Console.WriteLine(" 正在设置标题样式");
        showComment("正在设置标题样式");
```

```
        object wdFontSizeIndex;
        wdFontSizeIndex = 14;//此序号在 word 中的编号是格式 > 显示格式 > 样式和格式 > 显示所有样式的序号
        //14 即是标题一一级标题：三号黑体。
        oWordApplic.ActiveDocument.Styles.get_Item(ref wdFontSizeIndex).ParagraphFormat.Alignment = MsWord.WdParagraphAlignment.wdAlignParagraphCenter;
        oWordApplic.ActiveDocument.Styles.get_Item(ref wdFontSizeIndex).Font.Name = " 黑体";
        oWordApplic.ActiveDocument.Styles.get_Item(ref wdFontSizeIndex).Font.Size = 16;//三号
        wdFontSizeIndex = 15;//15 即是标题二二级标题：小三号黑体。
        oWordApplic.ActiveDocument.Styles.get_Item(ref wdFontSizeIndex).Font.Name = " 黑体";
        oWordApplic.ActiveDocument.Styles.get_Item(ref wdFontSizeIndex).Font.Size = 15;//小三
        //用指定的标题来设定文本格式
        object Style1 = MsWord.WdBuiltinStyle.wdStyleHeading1;//一级标题：三号黑体。
        object Style2 = MsWord.WdBuiltinStyle.wdStyleHeading2;//二级标题：小三号黑体。
        oDoc.Sections[curSectionNum].Range.Select();
        currentSelection = oWordApplic.Selection;
        //读入第一章文本信息
        StreamReader file_content = new StreamReader(Directory.GetCurrentDirectory() + @"\Files\content.txt");
        one_str = file_content.ReadLine();//一级标题
        currentSelection.TypeText(one_str);
        currentSelection.TypeParagraph(); //添加段落标记
        one_str = file_content.ReadLine();//二级标题
        currentSelection.TypeText(one_str);
        currentSelection.TypeParagraph(); //添加段落标记
        one_str = file_content.ReadLine();//正文
        while (one_str != null)
        {
            currentSelection.TypeText(one_str);
            currentSelection.TypeParagraph(); //添加段落标记
            one_str = file_content.ReadLine();//正文
        }
        file_content.Close();
        //段落的对齐方式
        curRange = oDoc.Sections[curSectionNum].Range.Paragraphs[1].Range;
        curRange.set_Style(ref Style1);
        oDoc.Sections[curSectionNum].Range.Paragraphs[1].Alignment = MsWord.WdParagraphAlignment.wdAlignParagraphCenter;
        curRange = oDoc.Sections[curSectionNum].Range.Paragraphs[2].Range;
        curRange.set_Style(ref Style2);
        //第一章正文文本格式
        for (int i = 3; i < oDoc.Sections[curSectionNum].Range.Paragraphs.Count; i++)
        {
            curRange = oDoc.Sections[curSectionNum].Range.Paragraphs[i].Range;
            curRange.Select();
```

```
            curRange.Font.Name = " 宋体";
            curRange.Font.Size = 12;
            oDoc.Sections[curSectionNum].Range.Paragraphs[i].LineSpacingRule =
            MsWord.WdLineSpacing.wdLineSpaceMultiple;
            //多倍行距，1.25 倍，这里的浮点值是以 point 为单位的，不是行距的倍数
            oDoc.Sections[curSectionNum].Range.Paragraphs[i].LineSpacing = 15f;

oDoc.Sections[curSectionNum].Range.Paragraphs[i].IndentFirstLineCharWidth(2);
        }
        #endregion 第一章

        Console.WriteLine(" 正在插入第二章内容");
        showComment("正在插入第二章内容");
            #region 第二章表格
        curSectionNum = 4;
        oDoc.Sections[curSectionNum].Range.Select();
        curRange = oDoc.Sections[curSectionNum].Range.Paragraphs[1].Range;
        currentSelection = oWordApplic.Selection;
        currentSelection.TypeText("2 表格");
        currentSelection.TypeParagraph();
        currentSelection.TypeText(" 表格示例");
        currentSelection.TypeParagraph();
        currentSelection.TypeParagraph();
        curRange = oDoc.Sections[curSectionNum].Range.Paragraphs[3].Range;

        oDoc.Sections[curSectionNum].Range.Paragraphs[3].Range.Select();
        currentSelection = oWordApplic.Selection;
        MsWord.Table oTable;
        oTable = curRange.Tables.Add(curRange, 5, 3, ref missing, ref missing);
        oTable.Range.ParagraphFormat.Alignment =
        MsWord.WdParagraphAlignment.wdAlignParagraphCenter;
        oTable.Range.Font.Name = " 宋体";
        oTable.Range.Font.Size = 16;
        oTable.Range.Cells.VerticalAlignment =
        MsWord.WdCellVerticalAlignment.wdCellAlignVerticalCenter;
        oTable.Range.Rows.Alignment = MsWord.WdRowAlignment.wdAlignRowCenter;
        oTable.Columns[1].Width = 80;
        oTable.Columns[2].Width = 180;
        oTable.Columns[3].Width = 80;
        oTable.Cell(1, 1).Range.Text = " 字段";
        oTable.Cell(1, 2).Range.Text = " 描述";
        oTable.Cell(1, 3).Range.Text = " 数据类型";
        oTable.Cell(2, 1).Range.Text = "ProductID";
        oTable.Cell(2, 2).Range.Text = " 产品标识";
        oTable.Cell(2, 3).Range.Text = " 字符串";
        oTable.Borders.InsideLineStyle = MsWord.WdLineStyle.wdLineStyleSingle;
        oTable.Borders.OutsideLineStyle = MsWord.WdLineStyle.wdLineStyleSingle;
        curRange = oDoc.Sections[curSectionNum].Range.Paragraphs[1].Range;
        curRange.set_Style(ref Style1);
        curRange.ParagraphFormat.Alignment =
        MsWord.WdParagraphAlignment.wdAlignParagraphCenter;

        showComment("插入表格，并设置边框格式");
```

```
        #endregion 第二章

        Console.WriteLine(" 正在插入第三章内容");
        showComment("正在插入第三章内容");
        #region 第三章图片
        curSectionNum = 5;
        oDoc.Sections[curSectionNum].Range.Paragraphs[1].Range.Select();
        curRange = oDoc.Sections[curSectionNum].Range.Paragraphs[1].Range;
        currentSelection = oWordApplic.Selection;
        currentSelection.TypeText("3 图片");
        currentSelection.TypeParagraph();
        currentSelection.TypeText(" 图片示例");
        currentSelection.TypeParagraph();

        currentSelection.InlineShapes.AddPicture(Directory.GetCurrentDirectory() +
@"\Files\whu.png",
        ref missing, ref missing, ref missing);
        curRange = oDoc.Sections[curSectionNum].Range.Paragraphs[1].Range;
        curRange.set_Style(ref Style1);
        curRange.ParagraphFormat.Alignment =
        MsWord.WdParagraphAlignment.wdAlignParagraphCenter;

        showComment("完成插入图片，并设置格式");
        #endregion 第三章


        Console.WriteLine(" 正在设置第一节摘要页眉内容");
        showComment("正在设置第一节摘要页眉内容");
        //设置页脚 section 1 摘要
        curSectionNum = 1;
        oDoc.Sections[curSectionNum].Range.Select();
        //进入页脚视图
        oWordApplic.ActiveWindow.ActivePane.View.SeekView =
        MsWord.WdSeekView.wdSeekCurrentPageFooter;
        oDoc.Sections[curSectionNum].
        Headers[MsWord.WdHeaderFooterIndex.wdHeaderFooterPrimary].
        Range.Borders[MsWord.WdBorderType.wdBorderBottom].LineStyle =
        MsWord.WdLineStyle.wdLineStyleNone;
        oWordApplic.Selection.HeaderFooter.PageNumbers.RestartNumberingAtSection =
true;
        oWordApplic.Selection.HeaderFooter.PageNumbers.NumberStyle
        = MsWord.WdPageNumberStyle.wdPageNumberStyleUppercaseRoman;
        oWordApplic.Selection.HeaderFooter.PageNumbers.StartingNumber = 1;
        //切换到文档
        oWordApplic.ActiveWindow.ActivePane.View.SeekView =
        MsWord.WdSeekView.wdSeekMainDocument;
        Console.WriteLine(" 正在设置第二节目录页眉内容");
        showComment("正在设置第二节目录页眉内容");
        //设置页脚 section 2 目录
        curSectionNum = 2;
        oDoc.Sections[curSectionNum].Range.Select();
        //进入页脚视图
        oWordApplic.ActiveWindow.ActivePane.View.SeekView =
        MsWord.WdSeekView.wdSeekCurrentPageFooter;
```

```
        oDoc.Sections[curSectionNum].
        Headers[MsWord.WdHeaderFooterIndex.wdHeaderFooterPrimary].
        Range.Borders[MsWord.WdBorderType.wdBorderBottom].LineStyle =
        MsWord.WdLineStyle.wdLineStyleNone;
        oWordApplic.Selection.HeaderFooter.PageNumbers.RestartNumberingAtSection =
false;
        oWordApplic.Selection.HeaderFooter.PageNumbers.NumberStyle
        = MsWord.WdPageNumberStyle.wdPageNumberStyleUppercaseRoman;
        //oWordApplic.Selection.HeaderFooter.PageNumbers.StartingNumber = 1;
        //切换到文档
        oWordApplic.ActiveWindow.ActivePane.View.SeekView =
        MsWord.WdSeekView.wdSeekMainDocument;
        //第一章页眉页码设置
        curSectionNum = 3;
        oDoc.Sections[curSectionNum].Range.Select();
        //切换入页脚视图
        oWordApplic.ActiveWindow.ActivePane.View.SeekView =
        MsWord.WdSeekView.wdSeekCurrentPageFooter;
        currentSelection = oWordApplic.Selection;
        curRange = currentSelection.Range;
        //本节页码不续上节
        oWordApplic.Selection.HeaderFooter.PageNumbers.RestartNumberingAtSection =
true;
        //页码格式为阿拉伯
        oWordApplic.Selection.HeaderFooter.PageNumbers.NumberStyle
        = MsWord.WdPageNumberStyle.wdPageNumberStyleArabic;
        //起如页码为 1
        oWordApplic.Selection.HeaderFooter.PageNumbers.StartingNumber = 1;
        //添加页码域
        object fieldpage = MsWord.WdFieldType.wdFieldPage;
        oWordApplic.Selection.Fields.Add(oWordApplic.Selection.Range,
        ref fieldpage, ref missing, ref missing);
        //居中对齐
        oWordApplic.Selection.ParagraphFormat.Alignment =
        MsWord.WdParagraphAlignment.wdAlignParagraphCenter;
        //本小节不链接到上一节
        oDoc.Sections[curSectionNum].Headers[Microsoft.Office.Interop.
        Word.WdHeaderFooterIndex.wdHeaderFooterPrimary].LinkToPrevious = false;
        //切换入正文视图

        oWordApplic.ActiveWindow.ActivePane.View.SeekView =
MsWord.WdSeekView.wdSeekMainDocument;
        //在输入全部内容后由于后面章节内容有变动，在此要更新目录，使页码正确
        Console.WriteLine(" 正在更新目录");
        showComment("正在更新目录");
        oDoc.Fields[1].Update();
        #region 保存文档
        //保存文档
        Console.WriteLine(" 正在保存 Word 文档");
        showComment("正在保存word文档");
        object fileName;
        fileName = doc_file_name;
        oDoc.SaveAs2(ref fileName);
        oDoc.Close();
```

```
        //Word 文档任务完成后，需要释放 Document 对象和 Application 对象。
        Console.WriteLine("正在释放 COM 资源");
        showComment("正在释放COM资源");

        //oWordApplic.Documents.Open(doc_file_name);

        //释放 COM 资源
        System.Runtime.InteropServices.Marshal.ReleaseComObject(oDoc);
        oDoc = null;
        oWordApplic.Quit();
        System.Runtime.InteropServices.Marshal.ReleaseComObject(oWordApplic);
        oWordApplic = null;

        System.GC.Collect();
        #endregion 保存文档
    }//try
    catch (Exception e2)
    {
        MessageBox.Show(e2.Message);
    }
    finally
    {

        Console.WriteLine(" 正在结束 Word 进程");
        showComment("正在结束Word进程");
        //关闭 word 进程
        Process[] AllProces = Process.GetProcesses();
        for (int j = 0; j < AllProces.Length; j++)
        {
            string theProcName = AllProces[j].ProcessName;
            if (String.Compare(theProcName, "WINWORD") == 0)
            {
                if (AllProces[j].Responding && !AllProces[j].HasExited)
                {
                    AllProces[j].Kill();
                }
            }
        }
        //Close word Process.

        OpenWord(doc_file_name);
    }
}

public static void OpenWord(string WordPath)
{

    //string tempPath = System.Environment.GetEnvironmentVariable("TEMP");
    //var filepath = Path.Combine(tempPath, WordPath);
    string winwordPath = "";
    Process[] wordProcesses = Process.GetProcessesByName("WINWORD");
    foreach (Process process in wordProcesses)
    {
        // Debug.WriteLine(process.MainWindowTitle);
```

```
            // 如果有的话获得 Winword.exe 的完全限定名称。
            winwordPath = process.MainModule.FileName;
            break;
        }

        Process wordProcess = new Process();

        if (winwordPath.Length > 0)    // 如果有 Word 实例在运行，使用 /w 参数来强制启动新实例，并将文件名作为参数传递。
        {
            wordProcess.StartInfo.FileName = winwordPath;
            wordProcess.StartInfo.UseShellExecute = false;
            wordProcess.StartInfo.Arguments = WordPath + " /w";
            wordProcess.StartInfo.RedirectStandardOutput = true;
        }
        else
        { // 如果没有 Word 实例在运行，还是
            wordProcess.StartInfo.FileName = WordPath;
            wordProcess.StartInfo.UseShellExecute = true;
        }

        wordProcess.Start();
        // 当前进程一直在等待，直到该 Word 实例退出。
        wordProcess.WaitForExit();
        wordProcess.Close();

    }
```

COM

自定义COM的生成与使用

1. 声明 COM 组件, 即定义接口

```
namespace DeviceInterfaces
{
    [Guid("9EDA6EA7-BB80-4B78-AE68-0C01C966F72D")]
    [ComVisible(true)]
    public interface ITransaction
    {
        void Connect(string connectString);
        void Disconnect();
        string GetVersion();
```

```
        string add(int a, int b);
        string multi(int a, int b);
    }
}
```

2. 实现 COM 组件, 即实现接口函数

```
namespace DeviceInterfaces {
    [Guid("944CA448-AE53-47C8-84ED-80DBD799E3BD")]
    [ComVisible(true)]
    [ClassInterface(ClassInterfaceType.None)]
    [Description("模拟事务记录")]
    public class SimTransaction : ITransaction
    {
        public void Connect(string connectString)
        {
        }
        public void Disconnect()
        {
        }
        public string GetVersion()
        {
            return "1.0";
        }
        public string add(int a, int b)
        {
            return string.Concat(a, "+", b, "=", a + b);
        }
        public string multi(int a, int b)
        {
            return string.Concat(a, "*", b, "=", a * b);
        }
    }
}
```

3. 注册 COM 组件, 可调用 .net 环境中的 Regasm 程序将 COM 组件注册到系统中
4. 调用/使用 COM 组件

```
namespace WPFTest {
    // 接口类 用guid定位
    class ComTest {
        public static ITransaction CreateTransaction(string _guid,string
connectionStr)
        {
            Console.Write("begin creating transaction");
            ITransaction iTransaction = null;
            try
            {
                // 调用注册表的CLSID, 并实例化
```

```
                Guid guid = new Guid(_guid);
                Type transactionType = Type.GetTypeFromCLSID(guid);
                object transaction =
Activator.CreateInstance(transactionType);
                Console.WriteLine(transaction);
                iTransaction = transaction as ITransaction;
                iTransaction.Connect(connectionStr);
            }
            catch (Exception ex) {
                Console.Write(ex.ToString());
            }
            Console.Write("end creating transaction");
            return iTransaction;
        }
        public static string add(string guid,string connectionStr,int a, int
b)
        {
            // COM 组件的创建与调用过程:
            //  4. 调用/使用 COM 组件
            ITransaction transaction = CreateTransaction(guid,
connectionStr);
            return transaction.add(a, b);
        }
        public static string multi(string guid, string connectionStr, int a,
int b)
        {
            // COM 组件的创建与调用过程:
            //  4. 调用/使用 COM 组件
            ITransaction transaction = CreateTransaction(guid,
connectionStr);
            return transaction.multi(a, b);
        }
    }
}
        // 调用接口类，传入guid后获取接口并使用
        private void btn1_Click_1(object sender, RoutedEventArgs e)
        {
            string strText1 = textBox1.Text.Trim();
            string strText2 = textBox2.Text.Trim();
            string ret = ComTest.add("944CA448-AE53-47C8-84ED-80DBD799E3BD",
"Simulation Transaction", int.Parse(strText1), int.Parse(strText2));
            textBox3.Text = String.Concat(ret);
        }
```

OfficeCOM的使用

见书P69之后或附件OfficeCOM

DLL

托管动态链接库DLL的创建与生成步骤

1. 采用csharp创建项目，项目类型为类库，名称为dll_csharp
2. 编写DLL函数，并将函数声明为静态类型

```
namespace dll_csharp {
    public class DLLCsharp
    {
        public static int AddF(int a, int b)
        {
            return a + b;
        }
    }
}
```

3. 编译工程项目生成dll

托管动态链接库DLL的调用过程

1. 创建csharp工程项目，项目类型为应用程序
2. 打开引用管理器，引用托管动态链接库的DLL文件
3. 在程序中调用dll中的函数

```
// 直接使用
private void btn1_Click_1(object sender, RoutedEventArgs e) {
    string strText = textBox1.Text.Trim();
    long ret = DLLCsharp.FactorialF(long.Parse(strText));
    textBox2.Text = String.Concat(ret);
}
// 反射使用
private void btn3_Click_1(object sender, RoutedEventArgs e)
{
    listBox1.Items.Clear();
    // DLL所在的绝对路径
    Assembly assembly=
Assembly.LoadFrom(AppDomain.CurrentDomain.BaseDirectory + "dll_csharp.dll");
    //注意写法：程序集.类名
    Type type = assembly.GetType("dll_csharp.DLLCsharp");
    System.Reflection.MethodInfo[] ms = type.GetMethods();
    foreach (System.Reflection.MethodInfo methodInfo in ms) {
        string method = "";
        method += methodInfo.ReturnType.Name;
        method += " " + methodInfo.Name;
        System.Reflection.ParameterInfo[] ps = methodInfo.GetParameters();
        int count = 0;
        method += "(";
        foreach (System.Reflection.ParameterInfo parameter in ps)
        {
            if (count == 0)
                method += parameter.ParameterType.Name + " " +
parameter.Name;
            else
```

```
                method += "," + parameter.ParameterType.Name + " " +
parameter.Name;
            count += 1;
        }
        method += ")";
        listBox1.Items.Add(method);
        //获取类中的公共方法GetResule
        //MethodInfo methed = type.GetMethod("GetResule");
        //创建对象的实例
        //object instance = System.Activator.CreateInstance(type);
        //执行方法  new object[]为方法中的参数
        //object result = methodInfo.Invoke(instance, new object[] { });
    }
```

非托管动态链接库DLL的创建步骤

1. 采用C++创建项目，项目类型为类库，名称为dll_cpp
2. 在头文件中声明函数原型

```
 extern "C" __declspec( dllexport ) int __stdcall testAdd ( int a, int b );
 extern "C" __declspec( dllexport ) int __stdcall testMulti ( int a, int b
);
```

3. 在cpp源代码文件中引用头文件，并实现函数

```
#include "stdafx.h"
#include "dll_cpp.h"
int __stdcall testAdd(int a, int b) {
    return a + b;
}
int __stdcall testMulti(int a, int b) {
    return a * b;
}
```

非托管动态链接库DLL的调用过程

1. 采用DllImport动态加载动态链接库文件中的函数
2. 重新声明

```
 class DllTest
 {
     [DllImport(@"dll_cpp.dll", EntryPoint ="testAdd", SetLastError = true,
CharSet = CharSet.Ansi, ExactSpelling = false, CallingConvention =
CallingConvention.StdCall)]
     public static extern int testAdd(int a, int b);
 }
```

3. 在程序中调用重新声明的函数

```
    private void btn1_Click_1(object sender, RoutedEventArgs e)
    {
        string strText1 = textBox1.Text.Trim();
        string strText2 = textBox2.Text.Trim();
        // 非托管动态链接库DLL的调用过程:
        // 在程序中调用重新声明的函数
        int ret = DllTest.testAdd(int.Parse(strText1), int.Parse(strText2));
        textBox3.Text = String.Concat(ret);
    }
```

进程

同步ping

```
private void runCMD(object sender, RoutedEventArgs e)
{
    clearComments();
    string strCmd = "ping www.sohu.com -n 20";
    if (!MyStringUtil.isEmpty(textBox1.Text))
        strCmd = "ping " + textBox1.Text.Trim() + " -n 20";
    cmdOutput = new StringBuilder("");
    cmdP = new Process();
    cmdP.StartInfo.FileName = "cmd.exe";
    cmdP.StartInfo.CreateNoWindow = true;
    cmdP.StartInfo.UseShellExecute = false;
    cmdP.StartInfo.RedirectStandardOutput = true;
    cmdP.StartInfo.RedirectStandardInput = true;
    //同步调用
    cmdP.Start();
    cmdP.StandardInput.WriteLine(strCmd);
    cmdP.StandardInput.WriteLine("exit");
    textBox2.Text = cmdP.StandardOutput.ReadToEnd();
    cmdP.WaitForExit();
    cmdP.Close();
}
```

异步ping

```
private void runCMD(object sender, RoutedEventArgs e)
{
    clearComments();
    string strCmd = "ping www.sohu.com -n 20";
    if (!MyStringUtil.isEmpty(textBox1.Text))
```

```
        strCmd = "ping " + textBox1.Text.Trim() + " -n 20";
    cmdOutput = new StringBuilder("");
    cmdP = new Process();
    cmdP.StartInfo.FileName = "cmd.exe";
    cmdP.StartInfo.CreateNoWindow = true;
    cmdP.StartInfo.UseShellExecute = false;
    cmdP.StartInfo.RedirectStandardOutput = true;
    cmdP.StartInfo.RedirectStandardInput = true;
    //异步调用
    cmdP.OutputDataReceived += new DataReceivedEventHandler(strOutputHandler);
    cmdP.Start();
    cmdStreamInput = cmdP.StandardInput;
    cmdStreamInput.WriteLine(strCmd);
    cmdStreamInput.WriteLine("exit");
    cmdP.BeginOutputReadLine();
    cmdP.WaitForExit();
    cmdP.Close();
}
```

消息

消息机制实现异步ping

```
public partial class C4_SY1 : ChildPage
{
    public static Process cmdP;
    public static StreamWriter cmdStreamInput;
    private static StringBuilder cmdOutput = null;
    public static IntPtr main_whandle;
    public static IntPtr text_whandle;
    #region 定义常量消息值
    public const int TRAN_FINISHED = 0x500;
    public const int WM_COPYDATA = 0x004A;
    #endregion
    #region 定义结构体
    public struct COPYDATASTRUCT
    {
        public IntPtr dwData;
        public int cbData;
        [MarshalAs(UnmanagedType.LPStr)]
        public string lpData;
    }
    #endregion
    //动态链接库引入
    [DllImport("User32.dll", EntryPoint = "SendMessage")]
    private static extern int SendMessage(
    IntPtr hWnd, // handle to destination window
    int Msg, // message
    int wParam, // first message parameter
    ref COPYDATASTRUCT lParam // second message parameter
```

```
    );
    [DllImport("User32.dll", EntryPoint = "FindWindow")]
    private static extern IntPtr FindWindow(string lpClassName, string
lpWindowName);
    [DllImport("user32.dll", EntryPoint = "FindWindowEx")]
    private static extern IntPtr FindWindowEx(IntPtr hwndParent, uint
hwndChildAfter, string lpszClass, string lpszWindow);
    //页面加载时，添加消息处理钩子函数
    private void ChildPage_Loaded(object sender, RoutedEventArgs e)
    {
        HwndSource hWndSource;
        WindowInteropHelper wih = new WindowInteropHelper(this.parentWindow);
        hWndSource = HwndSource.FromHwnd(wih.Handle);
        //添加处理程序
        hWndSource.AddHook(MainWindowProc);
    }
    //钩子函数，处理所收到的消息
    private IntPtr MainWindowProc(IntPtr hwnd, int msg, IntPtr wParam, IntPtr
lParam, ref bool handled)
    {
        switch (msg)
        {
            case WM_COPYDATA:
                try
                {
                    COPYDATASTRUCT mystr = new COPYDATASTRUCT();
                    Type mytype = mystr.GetType();
                    COPYDATASTRUCT MyKeyboardHookStruct =
(COPYDATASTRUCT)Marshal.PtrToStructure(lParam, typeof(COPYDATASTRUCT));
                    showComment(MyKeyboardHookStruct.lpData);
                    break;
                }
                catch (Exception e)
                {
                    Console.WriteLine(e.Message);
                    break;
                }
            case TRAN_FINISHED:
                {
                    showComment(cmdOutput.ToString());
                    break;
                }
            default:
                {
                    break;
                }
        }
        return hwnd;
    }
    private void clearComments()
    {
        listBox1.Items.Clear();
    }
    private void showComment(String comment)
```

```
    {
        if (MyStringUtil.isEmpty(comment)) {
            listBox1.Items.Add("");
            return;
        }
        listBox1.Items.Add(comment);
    }
    //定义回调
    private delegate void updateDelegate(object comment);
    public void update(object comment)
    {
        //showComment((string)comment);
        if (!listBox1.Dispatcher.CheckAccess())
        {
            //声明，并实例化回调
            updateDelegate d = update;
            //使用回调
            listBox1.Dispatcher.Invoke(d, comment);
        }
        else
        {
            showComment((string)comment);
        }
    }
    private void runCMD(object sender, RoutedEventArgs e)
    {
        clearComments();
        string strCmd = "ping www.whu.edu.cn -n 20";
        if (!MyStringUtil.isEmpty(textBox1.Text))
            strCmd = "ping " + textBox1.Text.Trim() + " -n 20";
        cmdOutput = new StringBuilder("");
        cmdP = new Process();
        cmdP.StartInfo.FileName = "cmd.exe";
        cmdP.StartInfo.CreateNoWindow = true;
        cmdP.StartInfo.UseShellExecute = false;
        cmdP.StartInfo.RedirectStandardOutput = true;
        cmdP.StartInfo.RedirectStandardInput = true;
        cmdP.OutputDataReceived+=new DataReceivedEventHandler(strOutputHandler);
        cmdP.Start();
        cmdStreamInput = cmdP.StandardInput;
        cmdStreamInput.WriteLine(strCmd);
        cmdStreamInput.WriteLine("exit");
        cmdP.BeginOutputReadLine();
    }
    //如果有输出，则重定向至输出对象，并向窗口对象发送特定的消息WM_COPYDATA和封装数据
COPYDATASTRUCT
    private  void strOutputHandler(object sendingProcess,
        DataReceivedEventArgs outLine)
    {
        //通过触发event，封装数据，并启动线程来异步更新控件
        cmdOutput.AppendLine(outLine.Data);
        //通过查找窗口，封装数据，发送消息的方式来异步更新控件
         //通过FindWindow API的方式找到目标进程句柄，然后发送消息
        IntPtr WINDOW_HANDLER = FindWindow(null, "demo");
```

```
        if (WINDOW_HANDLER != IntPtr.Zero)
        {
            //IntPtr hwndThree = FindWindowEx(WINDOW_HANDLER, 0, null, "");
            COPYDATASTRUCT mystr = new COPYDATASTRUCT();
            mystr.dwData = (IntPtr)0;
            if (MyStringUtil.isEmpty(outLine.Data))
            {
                mystr.cbData = 0;
                mystr.lpData = "";
            }
            else {
                byte[] sarr = System.Text.Encoding.Unicode.GetBytes(outLine.Data);
                mystr.cbData = sarr.Length + 1;
                mystr.lpData = outLine.Data;
            }
            SendMessage(WINDOW_HANDLER, WM_COPYDATA, 0, ref mystr);
        }
        /**或者通过枚举进程的方式找到目标进程句柄，然后发送消息
        Process[] procs = Process.GetProcesses();
        foreach (Process p in procs)
        {
            if (p.ProcessName.Equals("WPFTest"))
            {
                // 获取目标进程句柄
                IntPtr hWnd = p.MainWindowHandle;
                // 封装消息
                byte[] sarr = System.Text.Encoding.Unicode.GetBytes(outLine.Data);
                int len = sarr.Length;
                COPYDATASTRUCT cds2;
                cds2.dwData = (IntPtr)0;
                cds2.cbData = len + 1;
                cds2.lpData = outLine.Data;
                // 发送消息
                SendMessage(hWnd, WM_COPYDATA, 0, ref cds2);
            }
        }
        **/
    }
    private void fireEvent(string eventStr){
        FireNextEvent(eventStr);
    }
    private void closeCMD(object sender, RoutedEventArgs e)
    {
        cmdP.Close();
    }
}
```

消息机制实现异步getmac

```
public partial class C4_SY3 : ChildPage
{
```

```
    public static Process cmdP;
    public static StreamWriter cmdStreamInput;
    private static StringBuilder cmdOutput = null;
    #region 定义常量消息值
    public const int TRAN_FINISHED = 0x501;
    public const int WM_COPYDATA = 0x004B;
    #endregion
    #region 定义结构体
    public struct COPYDATASTRUCT
    {
        public IntPtr dwData;
        public int cbData;
        [MarshalAs(UnmanagedType.LPStr)]
        public string lpData;
    }
    #endregion
    //动态链接库引入
    [DllImport("User32.dll", EntryPoint = "SendMessage")]
    private static extern int SendMessage(
    IntPtr hWnd, // handle to destination window
    int Msg, // message
    int wParam, // first message parameter
    ref COPYDATASTRUCT lParam // second message parameter
    );
    [DllImport("User32.dll", EntryPoint = "FindWindow")]
    private static extern IntPtr FindWindow(string lpClassName, string
lpWindowName);
    [DllImport("user32.dll", EntryPoint = "FindWindowEx")]
    private static extern IntPtr FindWindowEx(IntPtr hwndParent, uint
hwndChildAfter, string lpszClass, string lpszWindow);
    //页面加载时，添加消息处理钩子函数
    private void ChildPage_Loaded(object sender, RoutedEventArgs e)
    {
        HwndSource hWndSource;
        WindowInteropHelper wih = new WindowInteropHelper(this.parentWindow);
        hWndSource = HwndSource.FromHwnd(wih.Handle);
        //添加处理程序
        hWndSource.AddHook(MainWindowProc);
    }
    //钩子函数，处理所收到的消息
    private IntPtr MainWindowProc(IntPtr hwnd, int msg, IntPtr wParam, IntPtr
lParam, ref bool handled)
    {
        switch (msg)
        {
            case WM_COPYDATA:
                try
                {
                    COPYDATASTRUCT mystr = new COPYDATASTRUCT();
                    Type mytype = mystr.GetType();
                    COPYDATASTRUCT MyKeyboardHookStruct =
(COPYDATASTRUCT)Marshal.PtrToStructure(lParam, typeof(COPYDATASTRUCT));
                    showComment(MyKeyboardHookStruct.lpData);
                    break;
```

```
                }
                catch (Exception e)
                {
                    Console.WriteLine(e.Message);
                    break;
                }
            case TRAN_FINISHED:
                {
                    showComment(cmdOutput.ToString());
                    break;
                }
            default:
                {
                    break;
                }
        }
        return hwnd;
    }
    private void clearComments()
    {
        listBox1.Items.Clear();
        //textBox2.Text = "";
    }
    private void showComment(String comment)
    {
        if (MyStringUtil.isEmpty(comment)) {
            listBox1.Items.Add("");
            return;
        }
        listBox1.Items.Add(comment);
        //textBox2.Text = comment;
    }
    private void runCMD(object sender, RoutedEventArgs e)
    {
        clearComments();
        string strCmd = "getmac";

        cmdOutput = new StringBuilder("");

        cmdP = new Process();
        cmdP.StartInfo.FileName = "cmd.exe";
        cmdP.StartInfo.CreateNoWindow = true;
        cmdP.StartInfo.UseShellExecute = false;
        cmdP.StartInfo.RedirectStandardOutput = true;
        cmdP.StartInfo.RedirectStandardInput = true;
        cmdP.OutputDataReceived+=new DataReceivedEventHandler(strOutputHandler);
        cmdP.Start();
        cmdStreamInput = cmdP.StandardInput;
        cmdStreamInput.WriteLine(strCmd);
        //cmdStreamInput.WriteLine("exit");
        cmdP.BeginOutputReadLine();
    }
    //如果有输出，则重定向至输出对象，并向窗口对象发送特定的消息WM_COPYDATA和封装数据
COPYDATASTRUCT
```

```
    private  void strOutputHandler(object sendingProcess,
        DataReceivedEventArgs outLine)
    {
        cmdOutput.AppendLine(outLine.Data);
        //通过查找窗口，封装数据，发送消息的方式来异步更新控件
        //通过FindWindow API的方式找到目标进程句柄，然后发送消息
        IntPtr WINDOW_HANDLER = FindWindow(null, "demo");
        if (WINDOW_HANDLER != IntPtr.Zero)
        {
            //IntPtr hwndThree = FindWindowEx(WINDOW_HANDLER, 0, null, "");
            COPYDATASTRUCT mystr = new COPYDATASTRUCT();
            mystr.dwData = (IntPtr)0;
            if (MyStringUtil.isEmpty(outLine.Data))
            {
                mystr.cbData = 0;
                mystr.lpData = "";
            }
            else {
                byte[] sarr = System.Text.Encoding.Unicode.GetBytes(outLine.Data);
                mystr.cbData = sarr.Length + 1;
                mystr.lpData = outLine.Data;
            }
            SendMessage(WINDOW_HANDLER, WM_COPYDATA, 0, ref mystr);
        }
        /*
        //或者通过枚举进程的方式找到目标进程句柄，然后发送消息
        Process[] procs = Process.GetProcesses();
        foreach (Process p in procs)
        {
            if (p.ProcessName.Equals("WPFTest"))
            {
                // 获取目标进程句柄
                IntPtr hWnd = p.MainWindowHandle;
                // 封装消息
                byte[] sarr = System.Text.Encoding.Unicode.GetBytes(outLine.Data);
                int len = sarr.Length;
                COPYDATASTRUCT cds2;
                cds2.dwData = (IntPtr)0;
                cds2.cbData = len + 1;
                cds2.lpData = outLine.Data;
                // 发送消息
                SendMessage(hWnd, WM_COPYDATA, 0, ref cds2);
            }
        }
        **/
    }
    private void closeCMD(object sender, RoutedEventArgs e)
    {
        cmdP.Close();
    }
}
```

消息机制实现抓屏

```
public partial class C4_SY5 : ChildPage
{
    private void ChildPage_Loaded(object sender, RoutedEventArgs e)
    {
        WINDOW_HANDLER = FindWindow(null, "demo");
        HwndSource hWndSource;
        WindowInteropHelper wih = new WindowInteropHelper(this.parentWindow);
        hWndSource = HwndSource.FromHwnd(wih.Handle);
        //添加处理程序
        hWndSource.AddHook(MainWindowProc);
    }
    //钩子函数，处理所收到的消息
    private IntPtr MainWindowProc(IntPtr hwnd, int msg, IntPtr wParam, IntPtr
lParam, ref bool handled)
    {
        switch (msg)
        {
            case WATCH_FILE:
                {
                    showComment(strfileinfo);
                    break;
                }
            default:
                {
                    break;
                }
        }
        return hwnd;
    }
    private void clearComments()
    {
        listBox1.Items.Clear();
        //textBox2.Text = "";
    }
    private void showComment(String comment)
    {
        if (MyStringUtil.isEmpty(comment)) {
            listBox1.Items.Add("");
            return;
        }
        listBox1.Items.Add(comment);
    }

    //定义回调
    private delegate void updateDelegate(string comment);
    public void update(string comment)
    {
        //showComment((string)comment);
        if (!listBox1.Dispatcher.CheckAccess())
        {
```

```
            //声明，并实例化回调
            updateDelegate d = update;
            //使用回调
            listBox1.Dispatcher.Invoke(d, comment);
        }
        else
        {
            showComment(comment);
        }
    }
    //动态链接库引入
    [DllImport("kernel32.dll")]
    static extern int GetTickCount();
    [DllImport("User32.dll", EntryPoint = "FindWindow")]
    private static extern IntPtr FindWindow(string lpClassName, string
lpWindowName);
    [DllImport("User32.dll", EntryPoint = "SendMessage")]
    private static extern int SendMessage(
    IntPtr hWnd, // handle to destination window
    int Msg, // message
    int wParam, // first message parameter
    int lParam // second message parameter
    );
    public static IntPtr WINDOW_HANDLER;
    public static string w_dir;
    public static ManualResetEvent e_wdirth_end;
    public const int WATCH_FILE = 0x600;
    static ManualResetEvent capture_terminate_e;     //结束抓屏线程事件
    static ManualResetEvent capture_this_one_e;      //抓屏事件
    public static ManualResetEvent[] me_cap = new ManualResetEvent[2];
    static void Capture_screen()
    {
        int s_wid = Screen.PrimaryScreen.Bounds.Width;
        int s_height = Screen.PrimaryScreen.Bounds.Height;
        Bitmap b_1 = new Bitmap(s_wid, s_height);
        Graphics g_ = Graphics.FromImage(b_1);
        String init_dir_fn =
Environment.GetFolderPath(Environment.SpecialFolder.Desktop);
        String dest_fn = null;
        //用事件的方法来抓获图片
        int index = WaitHandle.WaitAny(me_cap, 500);
        while (index != 0)
        {
            if (index == 1)
            {
                dest_fn = init_dir_fn;
                dest_fn += "\\bc\\";
                if (!Directory.Exists(dest_fn))
                {
                    Directory.CreateDirectory(dest_fn);
                }
                dest_fn += GetTickCount().ToString();
                dest_fn += "ab.bmp";
                g_.CopyFromScreen(0, 0, 0, 0, new System.Drawing.Size(s_wid,
```

```
s_height));
                b_1.Save(dest_fn, System.Drawing.Imaging.ImageFormat.Bmp);
                capture_this_one_e.Reset();
            }
            index = WaitHandle.WaitAny(me_cap, 500);
        }
        g_.Dispose();
        b_1.Dispose();
    }
    public static string strfileinfo;
    public static void WatchDir()
    {
        long now_t = DateTime.Now.ToFileTime();
        DirectoryInfo d_info = new DirectoryInfo(w_dir);
        string new_filename;
        FileInfo[] f_ins = d_info.GetFiles();
        //e_wdirth_end有信号，为true，则不用阻塞等待，
        //e_wdirth_end无信号，为false，则需要阻塞等待
        while (!e_wdirth_end.WaitOne(500))   //
        {
            for (int i = 0; i < f_ins.Length; i++)
            {
                now_t = DateTime.Now.ToFileTime();
                if (File.Exists(f_ins[i].FullName))
                {
                    strfileinfo = string.Format("监视到文件{0}\r\n",
f_ins[i].FullName);
                    if (WINDOW_HANDLER != IntPtr.Zero)
                            SendMessage(WINDOW_HANDLER, WATCH_FILE, 0, 0);
                    string newDir = w_dir + "\\ref\\";
                    if (!Directory.Exists(newDir))
                    {
                        Directory.CreateDirectory(newDir);
                    }
                    new_filename = newDir + now_t.ToString() + f_ins[i].Name;
                    if (!File.Exists(new_filename))
                    {
                        File.Move(f_ins[i].FullName, new_filename);
                    }
                }
            }
            f_ins = d_info.GetFiles();//重新获取新的目录信息
        }
    }
    //选择文件目录进行监视
    private void btn1_Click(object sender, RoutedEventArgs e)
    {
        e_wdirth_end = new ManualResetEvent(false);
        FolderBrowserDialog dialog = new FolderBrowserDialog();
        if (dialog.ShowDialog() == DialogResult.OK)
        {
            w_dir = dialog.SelectedPath;
        }
    }
```

```
    //启动文件目录监视线程
    private void btn2_Click(object sender, RoutedEventArgs e)
    {
        e_wdirth_end.Reset();
        Thread workThread = new Thread(new ThreadStart(WatchDir));
        workThread.IsBackground = true;
        workThread.Start();
    }
    //终止文件目录监视线程
    private void btn3_Click(object sender, RoutedEventArgs e)
    {
        e_wdirth_end.Set();
    }
    private void btn4_Click(object sender, RoutedEventArgs e)
    {
        //初始抓屏终止事件为未结束
        capture_terminate_e = new ManualResetEvent(false);
        //初始捕获终止状态为未结束
        capture_this_one_e = new ManualResetEvent(false);
        //启动捕捉线程
        me_cap[0] = capture_terminate_e;
        me_cap[1] = capture_this_one_e;
        ThreadStart workStart = new ThreadStart(Capture_screen);
        Thread workThread = new Thread(workStart);
        workThread.IsBackground = true;
        workThread.Start();
    }
    private void btn5_Click(object sender, RoutedEventArgs e)
    {
        capture_this_one_e.Set();
    }
    private void btn6_Click(object sender, RoutedEventArgs e)
    {
        capture_terminate_e.Set();
    }
}
```

线程

创建线程

```
private void btn2_Click(object sender, RoutedEventArgs e)
{
    clearComments();
    //实例化回调
    setCallBack = new setTextValueCallBack(showComment);
    //创建无参的线程
    Thread thread1 = new Thread(new ThreadStart(Thread1));
    //调用Start方法执行线程
    thread1.Start();
```

```
    //创建ThreadTest类的一个实例
    ThreadTest test = new ThreadTest(this);
    //调用test实例的MyThread方法
    Thread thread2 = new Thread(new ThreadStart(test.Thread2));
    //启动线程
    thread2.Start();
    //通过匿名委托创建
    Thread thread3 = new Thread(delegate () { update("我是通过匿名委托创建的线程");
});
    thread3.Start();
    //通过Lambda表达式创建
    Thread thread4 = new Thread(() => update("我是通过Lambda表达式创建的委托"));
    thread4.Start();
    //通过ParameterizedThreadStart创建线程
    Thread thread5 = new Thread(new ParameterizedThreadStart(Thread5));
    //给方法传值
    thread5.Start("这是一个有参数的委托");
    //Console.ReadKey();
}
```

前台线程与后台线程

```
//前台线程先结束，会自动结束后台线程（没有执行完的后台线程，不继续执行）
private void btn3_Click(object sender, RoutedEventArgs e)
{
    clearComments();
    //演示前台、后台线程
    BackGroundTest background = new BackGroundTest(this,10);
    //创建前台线程
    Thread fThread = new Thread(new ThreadStart(background.RunLoop));
    //给线程命名
    fThread.Name = "前台线程";
    BackGroundTest background1 = new BackGroundTest(this,20);
    //创建后台线程
    Thread bThread = new Thread(new ThreadStart(background1.RunLoop));
    bThread.Name = "后台线程";
    //设置为后台线程
    bThread.IsBackground = true;
    //启动线程
    fThread.Start();
    bThread.Start();
}
```

线程的sleep

```
private void threadMethod6(Object obj)
{
    Thread.Sleep(Int32.Parse(obj.ToString()));
    Console.WriteLine(obj + "毫秒任务结束");
```

```
    update(obj + "毫秒任务结束");
}
```

线程的Join

```
private void joinAllThread6(object obj)
{
    Thread[] threads = obj as Thread[];
    foreach (Thread t in threads)
        t.Join();
    Console.WriteLine("所有的线程结束");
    update("所有的线程结束");
}
```

线程的Abort

```
bThread.Abort();
```

线程同步调用

```
for (int i = 0; i < 5; i++)
{
 string name = string.Format($"btnSync_Click_{i}");
 this.DoSomethingLong(name);
 }
```

线程异步方法

```
Action<string> action = this.DoSomethingLong;
for (int i = 0; i < 5; i++)
{
    string name = string.Format($"btnSync_Click_{i}");
    // 调用委托(同步调用)
    //action.Invoke("btnAsync_Click_1");
    // 异步调用委托
    action.BeginInvoke(name, null, null);
}
```

线程异步的回调方法实现同步

```
private void btnAsyncAdvanced_Click(object sender, RoutedEventArgs e)
{
    Action<string> action = this.DoSomethingLong;
    IAsyncResult asyncResult = null;
    // 定义一个回调，其中lamda表达式中的p为参数
    AsyncCallback callback = p =>
    {
        Console.WriteLine($"到这里计算已经完成了。
{Thread.CurrentThread.ManagedThreadId.ToString("00")}。");
        update($"到这里计算已经完成了。" +
Thread.CurrentThread.ManagedThreadId.ToString("00") + "。");
    };
    for (int i = 0; i < 5; i++)
    {
        string name = string.Format($"btnSync_Click_{i}");
        asyncResult = action.BeginInvoke(name, callback, null);
    }
}
```

AutoResetEvent(与Manual的差别见PPT)

```
AutoResetEvent event1 = new AutoResetEvent(false);
AutoResetEvent event2 = new AutoResetEvent(false);
AutoResetEvent event3 = new AutoResetEvent(false);
public void Method1()
{
    update("thread1 begin");
    Thread.Sleep(1000);
    update("thread1 end");
    event1.Set();//置为true
}
public void Method2()
{
    update("thread2 begin");
    Thread.Sleep(2000);
    update("thread2 end");
    event2.Set();
}
public void Method3()
{
    update("thread3 begin");
    Thread.Sleep(3000);
    update("thread3 end");
    event3.Set();
}
```

WaitAny

```
private void BtbAutoResetEvent(object sender, RoutedEventArgs e)
{
    clearComments();
    Thread vThread1 = new Thread(new ThreadStart(Method1));
    Thread vThread2 = new Thread(new ThreadStart(Method2));
    Thread vThread3 = new Thread(new ThreadStart(Method3));
    AutoResetEvent[] vEventInProgress = new AutoResetEvent[3]
    {
        event1,
        event2,
        event3
    };
    vThread1.Start();
    vThread2.Start();
    vThread3.Start();
    WaitHandle.WaitAny(vEventInProgress, 8000);
    update("current thread end;");
    if (vThread1 != null)
        vThread1.Abort();
    if (vThread2 != null)
        vThread2.Abort();
    if (vThread3 != null)
        vThread3.Abort();
}
```

ManualResetEvent

```
ManualResetEvent event4 = new ManualResetEvent(false);
ManualResetEvent event5 = new ManualResetEvent(false);
ManualResetEvent event6 = new ManualResetEvent(false);
public void Method4()
{
    //event4.Reset();
    update("thread4 begin");
    //Thread.Sleep(1000);
    update("thread4 end");
    event4.Set();
}
public void Method5()
{
    //event5.Reset();
    update("thread5 begin");
    //Thread.Sleep(2000);
    update("thread5 end");
    event5.Set();
}
public void Method6()
{
    //event6.Reset();
    update("thread6 begin");
```

```
    //Thread.Sleep(3000);
    update("thread6 end");
    event6.Set();
}
```

WaitOne

```
private void waitOneTest()
{
    ManualResetEvent mansig;
    mansig = new ManualResetEvent(true);
    bool state = mansig.WaitOne(9000, true);
    Console.WriteLine("ManualResetEvent After WaitOne" + state);
    if (!state)
        update("ManualResetEvent After WaitOne " + state);
    mansig.Reset();
    if (!mansig.WaitOne(9000, true)) {
        Console.WriteLine("ManualResetEvent After WaitOne" + state);
        update("ManualResetEvent After WaitOne " + state);
    }
    mansig.Set();
    state = mansig.WaitOne(9000, true);
    Console.WriteLine("ManualResetEvent After WaitOne" + state);
    if (!state)
        update("ManualResetEvent After WaitOne " + state);
}
```

生产者消费者问题（Mutex版本，无Semaphore）

```
public class ProducerConsumer
{
    public static Mutex mut;
    public static Thread[] threadVec;
    public static int SharedBuffer;
    public void Consumer()
    {
        while (true)
        {
            int result;
            mut.WaitOne();
            if (SharedBuffer == 0)
            {
                Console.WriteLine("Consumed {0}: end of data\r\n", SharedBuffer);
                parent.update("Consumed " + SharedBuffer + " : end of data\r\n");
                mut.ReleaseMutex();
                break;
            }
            if (SharedBuffer > 0)
            { // ignore negative values
```

```
                result = SharedBuffer;
                Console.WriteLine("Consumed: {0}", result);
                parent.update("Consumed: " + result);
                mut.ReleaseMutex();
            }
        }
    }
    public void Producer()
    {
        int i;
        for (i = 20; i >= 0; i--)
        {
            mut.WaitOne();
            Console.WriteLine("Produce: {0}", i);
            parent.update("Produce: "  + i);
            SharedBuffer = i;
            mut.ReleaseMutex();
            Thread.Sleep(1000);
        }
    }
}
```

生产者消费者（Mutex+Semaphore版本，非老师版）

```
static readonly int BUFFER_SIZE = 5;
static Mutex bufferMutex = new Mutex();
static Semaphore fillCount = new Semaphore(0, BUFFER_SIZE);
static Semaphore emptyCount = new Semaphore(BUFFER_SIZE, BUFFER_SIZE);
private void button1_Click(object sender, EventArgs e)
{
    Random random = new Random();
    int CustomerNum;
    int ProducerNum;
    if (int.TryParse(textBox1.Text, out int result1))
    {
        ProducerNum = result1;
    }
    else
    {
        ProducerNum = 1;
    }
    if (int.TryParse(textBox2.Text, out int result2))
    {
        CustomerNum = result2;
    }
    else
    {
        CustomerNum = 1;
    }
    ThreadTextBox("生产者数量:" + CustomerNum + "\r\n");
    ThreadTextBox("消费者数量:" + ProducerNum + "\r\n");
```

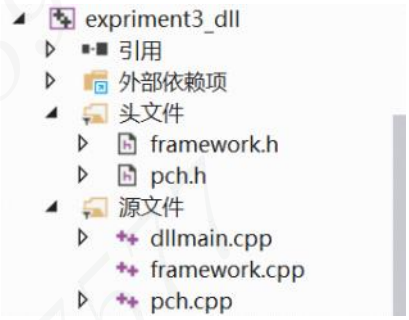
```
    Queue<int> products = new Queue<int>();
    Thread[] producerThreads = new Thread[ProducerNum];
    Thread[] customerThreads = new Thread[CustomerNum];
    for (int i = 0; i < ProducerNum; i++)
    {
        producerThreads[i] = new Thread((n) =>
        {
            while (true)
            {
                emptyCount.WaitOne();
                bufferMutex.WaitOne();
                int product = random.Next() % 100 + 100;
                ThreadTextBox("生产者:" + n + "生产:" + product + "\r\n");
                products.Enqueue(product);
                Thread.Sleep(1000);
                bufferMutex.ReleaseMutex();
                fillCount.Release();
            }
        });
    }
    for (int i = 0; i < CustomerNum; i++)
    {
        customerThreads[i] = new Thread((n) =>
        {
            while(true)
            {
                fillCount.WaitOne();
                bufferMutex.WaitOne();
                int product = products.Dequeue();
                ThreadTextBox("消费者:" + n + " 消费:" + product + "\r\n");
                Thread.Sleep(1000);
                bufferMutex.ReleaseMutex();
                emptyCount.Release();
            }
        });
    }
    for (int i = 0; i < ProducerNum; i++) producerThreads[i].Start(i);
    for (int i = 0; i < CustomerNum; i++) customerThreads[i].Start(i);
}
```

使用自定义非托管DLL（C#调用C++）

1.在VS中创建C++类库，使用预编译头，可以自动生成下面三个文件
2.在pch.cpp中写我们定义的DLL函数
3.将解决方案切换到Release模式，点击生成生成dll
4.将生成的dll放到我们需要调用它的项目的Debug或者Release目录下
5.在调用项目中使用DllImport引入函数，需要将其定义为public static extern
6.在调用项目中用调用C#方法调用引入的DLL方法

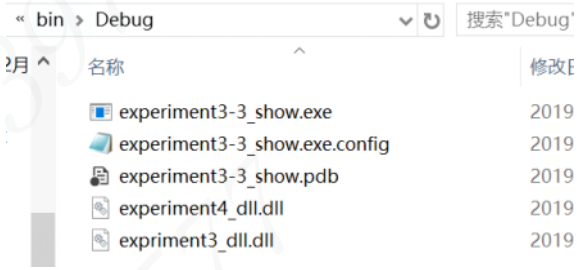
```
double Add(double a, double b)
{
    return a+b;
}

double Mul(double a, double b)
{
    return a*b;
}
```




```
[DllImport("expriment3_dll.dll")]
1 个引用
public static extern double Add(double a, double b);

[DllImport("expriment3_dll.dll")]
1 个引用
public static extern double Mul(double a, double b);
```



同步调用进程Ping，整体按照书上12.1.1节，需补充一行代码

process.StartInfo.FileName = "cmd.exe";
补充在重定向输出流那句之后即可

异步调用进程Ping

整体按照书上12.1.2节即可，需补充一行代码
1.在104页代码之后需加上
cmdStreamInput.WriteLine("ping " + textBox1.Text.Trim() + " -n 200");
意味对cmd进程写入一条命令

杀死Ping进程（关闭进程）

ping进程是cmd进程的子进程，杀死要用特殊方式，将cmd进程和它的所有子进程递归的杀死。
1.创建ManagementObjectSearcher对象和ManagementObjection对象获取传入方法的pid所有的子进程对象。（类似数据库查询）
2.对于每一个子进程，递归调用KillProcessAndChildrens方法杀死进程和子进程
3.处理完所有子进程后，通过Kill函数杀死传入方法的pid对应的进程

```
private static void KillProcessAndChildrens(int pid)
{
    ManagementObjectSearcher processSearcher = new ManagementObjectSearcher
      ("Select * From Win32_Process Where ParentProcessID=" + pid);
    ManagementObjectCollection processCollection = processSearcher.Get();

    // We must kill child processes first!
    if (processCollection != null)
    {
        foreach (ManagementObject mo in processCollection)
        {
            KillProcessAndChildrens(Convert.ToInt32(mo["ProcessID"])); //kill child proces
        }
    }

    // Then kill parents.
    try
    {
        Process proc = Process.GetProcessById(pid);
        if (!proc.HasExited) proc.Kill();
    }
    catch (ArgumentException)
    {
        // Process already exited.
    }
}
```

进程调用getmac

只需要将同步或异步调用进程Ping中输入cmd的命令改为"getmac"即可
如果是同步，需修改103页代码中strCMD
如果是异步，需要将我们添加的那条代码写入的指令换成"getmac"

打印当前线程Id

textBox3.Text = "Thread " + Thread.CurrentThread.ManagedThreadId.ToString() + " started in " + Thread.GetDomain().FriendlyName + " with AppDomainID = " + Thread.GetDomainID().ToString() + "." ;

输出样式为Thread 1 started in experiment7.exe with AppDomainID = 1.

无参方法创建线程

1.在类中声明一个无参的方法

2.创建线程

Thread thread1 = new Thread(new ThreadStart(method))

3.开启线程thread1.Start();

```
1 个引用
void method()
{
    //CreateThreadeOutput createThreade = new CreateThreadeOutput(this.CreateThreadAppendTextBox);
    //createThreade(0);
    SendMessage(main_whandle, CREATE_THREAD_NO_PARAMATER_METHOD, 0, 0);
}
```

实例方法创建线程

1.新建一个类，将线程将要实现的功能写成这个类内部的方法

2.通过以下语句创建线程

ThreadTest thread = new ThreadTest();

Thread thread2 = new Thread(thread.MyThread);

3.开启线程thread2.Start()

```
class ThreadTest
{
  public void MyThread()
  {
    SendMessage(main_whandle, CREATE_THREAD_OBJECT_METHOD,0, 0);
  }
}
```

匿名委托函数创建线程

将委托作为Thread构造函数的参数传入

```
Thread thread3 = new Thread(delegate () { SendMessage(main_whandle,CREATE_THREAD_ANONYMOUS_DELEGATE, 0, 0); });
thread3.Start();
```

Lambda表达式创建线程

将Lambda表达式作为Thread构造函数的参数传入

```
Thread thread4 = new Thread(() => { SendMessage(main_whandle, CREATE_THREAD_LAMBDA, 0, 0); });
thread4.Start();
```

通过有参委托创建线程

1.创建有参的方法，方法的参数必须是object类型

2.将Thread5作为参数创建ParameterizedThreadStart对象

3.将创建的ParameterizedThreadStart对象作为参数创建Thread对象

```
Thread thread5 = new Thread(new
ParameterizedThreadStart(Thread5));
//Thread5是我们定义的方法
```

4.启动线程

Thread5.Start();

```
static void Thread5(object obj)
{
  if (obj.Equals("这是一个有参数的委托"))
  {
    SendMessage(main_whandle, CREATE_THREAD_PARAMATER_METHOD, 0, 0);
  }
}
```

将线程更改为前台/后台进程

设置Thread对象的IsBackground属性即可。我们假设thread是一个Thread类的对象

thread.IsBackground = true;	//将线程设置为后台线程
thread.IsBackground = false;	//将线程设置为前台线程

使用Join实现线程阻塞

1.创建一个线程对象thread1,他的工作是休眠5秒

2.创建一个线程对象thread2,他的工作是先调用thread1的Join方法，然后休眠3秒

3.启动thread1和thread2。注意两者的启动顺序不能颠倒!因为thread2的执行任务中有thread1.Join()，这条指令要求thread1必须已经开始执行。

```
thread1.Start();
thread2.Start();
```

```
Thread thread1 = new Thread(() => {
  Thread.Sleep(5000);
  SendMessage(main_whandle, JOIN_5000, 0, 0);
});
Thread thread2 = new Thread(() => {
  thread1.Join();
  Thread.Sleep(3000);
  SendMessage(main_whandle, JOIN_3000, 0, 0);
});
```

结果解释，因为thread1.Start();和thread2.Start();这两条语句的开始执行时间几乎相同，所以两个进程几乎同时开始执行任务，但是thread2的第一条指令会让其阻塞直到thread1执行完后再继续执行。所以程序运行结果为，5s后thread1执行完成，8s后thread2执行完成。
如果将thread1.Join()这条指令注释掉，则运行结果为3s后thread2执行完成，5s后thread1执行完成。

同步、异步、异步回调调用线程（博客园上的一个简单例子）

首先，通过代码定义一个委托和下面三个示例将要调用的方法：

```
public delegate int AddHandler(int a,int b);
public class 加法类
{
    public static int Add(int a, int b)
    {
        Console.WriteLine("开始计算：" + a + "+" + b);
        Thread.Sleep(3000); //模拟该方法运行三秒
        Console.WriteLine("计算完成！");
        return a + b;
    }
}
```

异步调用

```
public class 异步调用
{
    static void Main()
    {
        Console.WriteLine("===== 异步调
用 AsyncInvokeTest =====");
        AddHandler handler = new AddHandler(加法类.Add);

        //IAsyncResult: 异步操作接口(interface)
        //BeginInvoke: 委托(delegate)的一个异步方法的开始
        IAsyncResult result = handler.BeginInvoke(1, 2, null,
null);

        Console.WriteLine("继续做别的事情。。。");

        //异步操作返回
        Console.WriteLine(handler.EndInvoke(result));
        Console.ReadKey();
    }

}
```

可以看到，主线程并没有等待，而是直接向下运行了。

但是问题依然存在，当主线程运行到EndInvoke时，如果这时调用没有结束（这种情况很可能出现），这时为了等待调用结果，线程依旧会被阻塞。

同步调用

委托的Invoke方法用来进行同步调用。同步调用也可以叫阻塞调用，它将阻塞当前线程，然后执行调用，调用完毕后再继续向下进行。

```
public class 同步调用
{
    static void Main()
    {
        Console.WriteLine("===== 同步调用 SyncInvokeTest =====");
        AddHandler handler = new AddHandler(加法类.Add);
        int result = handler.Invoke(1, 2);

        Console.WriteLine("继续做别的事情。。。");

        Console.WriteLine(result);
        Console.ReadKey();
    }
}
```

异步回调

用回调函数，当调用结束时会自动调用回调函数，解决了为等待调用结果，而让线程依旧被阻塞的局面。

```
public class 异步回调
{
    static void Main()
    {
        Console.WriteLine("===== 异步回
调 AsyncInvokeTest =====");
        AddHandler handler = new AddHandler(加法类.Add);

        //异步操作接口(注意BeginInvoke方法的不同！)
        IAsyncResult result = handler.BeginInvoke(1,2,new AsyncCallb
ack(回调函数),"AsycState:OK");

        Console.WriteLine("继续做别的事情。。。");
        Console.ReadKey();
    }

    static void 回调函数(IAsyncResult result)
    {   //result 是“加法类.Add()方法”的返回值

        //AsyncResult 是IAsyncResult接口的一个实现类，空间:
System.Runtime.Remoting.Messaging
        //AsyncDelegate 属性可以强制转换为用户定义的委托的实际类。
        AddHandler handler = (AddHandler)((AsyncResult)
result).AsyncDelegate;

        Console.WriteLine(handler.EndInvoke(result));
        Console.WriteLine(result.AsyncState);
    }

}
```

我定义的委托的类型为AddHandler，则为了访问 AddHandler.EndInvoke，必须将异步委托强制转换为 AddHandler。可以在异步回调函数（类型为

AsyncCallback）中调用 MAddHandler.EndInvoke，以获取最初提交的 AddHandler.BeginInvoke 的结果。

注意：作者用了中文的类名和方法名！不要被搞晕！！

实验上这部分有点太多了，因为要想实现老师要求的展现方式需要很多和线程调用无关的核心代码，所以我先粘了这篇我当时参考的博客上的，你看到这里后如果觉得需要我把实验上的内容也加上去告诉我一下我去加。

对于异步回调方法中

```
AddHandler handler = (AddHandler)((AsyncResult)
result).AsyncDelegate;
```

这一句代码使用了过多的系统里取获得正确的委托对象handler

有一种更简单的方式是使用Lambda表达式去写回调函数而不是将其单独作为一个静态成员方法，这样就可以直接使用之前定义好的handler

```
static void Main()
    {
        Console.WriteLine("===== 异步回
调 AsyncInvokeTest =====");
        AddHandler handler = new AddHandler(加法类.Add);
        //异步操作接口(注意BeginInvoke方法的不同！)
        IAsyncResult result = handler.BeginInvoke(1,2,
            (IAsyncResult res) =>{
                Console.WriteLine(handler.EndInvoke(result));
                Console.WriteLine(result.AsyncState);
            }
            , "AsycState:OK");

        Console.WriteLine("继续做别的事情。。。");
        Console.ReadKey();
    }
```

AutoResetEvent线程通信

1.创建一个AutoRestEvent对象

```
AutoResetEvent autoResetEvent = new AutoResetEvent(false);
```

2.创建5个线程，让它们的任务都是等待autoResetEvent信号等待10s

```
for (int i = 0; i < count; i++){
        thread = new Thread(() => { autoResetEvent.WaitOne(10000); });
        ThreadList.Add(thread);  //ThreadList是一个List<Thread>类型的变量
}
```

3.创建一个线程，它的任务是在等待autoResetEvent信号2s后（显然这2s内不会有人给它发信号，所以它相当于阻塞了2s），发出一个autoResetEvent信号

```
thread = new Thread(() =>
        {
          autoResetEvent.WaitOne(2000);
          autoResetEvent.Set();
        });
```

4.同时运行上面6个线程

```
for (int i = 0; i < count; i++)
        {
          ThreadList[i].Start();
        }
thread.Start();
```

结果分析：前2s不会有任何反应，第2s时第六个线程结束等待，发出信号令前五个线程中的其中一个继续执行，而这两个线程都没有后续任务，所以都会结束。剩下的4个线程等待信号一直到第10s，然后结束。

ManualResetEvent线程通信

1.创建一个ManualResetEvent对象

```
ManualResetEvent manualResetEvent= new ManualResetEvent(false);
```

2.创建5个线程，让它们的任务都是等待autoResetEvent信号等待10s

```
for (int i = 0; i < count; i++){
        thread = new Thread(() => { manualResetEvent.WaitOne(10000); });
```

```
        ThreadList.Add(thread);  //ThreadList是一个List<Thread>类型的变量
}
```

3.创建一个线程，它的任务是在等待manualResetEvent信号2s后（显然这2s内不会有人给它发信号，所以它相当于阻塞了2s），发出一个manualResetEvent信号

```
thread = new Thread(() =>
        {
           manualResetEvent.WaitOne(2000);
           manualResetEvent.Set();
        });
```

4.同时运行上面6个线程

```
for (int i = 0; i < count; i++)
        {
           ThreadList[i].Start();
        }
thread.Start();
```

结果分析：前2s不会有任何反应，第2s时第六个线程结束等待，发出信号，前五个线程中同时收到该信号开始继续执行，而这六个线程都没有后续任务，所以都会结束。

Invoke函数

在Windows下，窗体（Form）是由单独的线程控制的，这意味着，我们如果在控件的触发逻辑代码中新建了线程，这个线程是没有办法修改我们现有的窗体控件属性的。

这时为了达到修改窗体控件属性的效果，我们需要调用控件的Invoke函数。Invoke方法首先检查发出调用的线程(即当前线程)是不是UI线程，如果是，直接执行委托指向的方法，如果不是，它将切换到UI线程，然后执行委托指向的方法。

1.创建一个委托

```
public delegate void AppendTextBoxDelegate(string msg);
```

2.创建一个方法作为委托的具体实现

```
public void AppendTextBox(string msg)
     {
        textBox3.AppendText(msg);
     }
```

3.在需要的时候调用Invoke方法，传入的2个参数分别为委托对象和委托的参数

```
private void button16_Click(object sender, EventArgs e) {
       textBox3.Invoke(new AppendTextBoxDelegate(AppendTextBox), "期末复习要注意休息呀\r\n");
}
```

生产者、消费者问题

教材配套代码如下

```
private static Mutex mut;
private static int SharedBuffer;
private static int BufferState;
private static Thread[] threadVec;
private static AutoResetEvent hNotFullEvent, hNotEmptyEvent;
public const int FULL = 1;
public const int EMPTY = 0;

public static int ProdecerIterater;
```

```
private static void Consumer()
     {
        //int result;
        while (true)
        {
           mut.WaitOne();
           if (BufferState == EMPTY)
           { // nothing to consume
              mut.ReleaseMutex();
              // wait until buffer is not empty
              hNotEmptyEvent.WaitOne();
              continue; // return to while loop to contend for Mutex
```

```
private static void Producer()
    {
        //int ProdecerIterater;

        for (ProdecerIterater = 20; ProdecerIterater >= 0;
ProdecerIterater--)
        {
            while (true)
            {
                mut.WaitOne();

                if (BufferState == FULL)
                {
                    mut.ReleaseMutex();
                    hNotFullEvent.WaitOne();
                    continue; // back to loop to test BufferState again
                }
                // got mutex and buffer is not FULL, break out of while loop
                break;
            }//end of while
             // got Mutex, buffer is not full, producing data
             //Console.WriteLine("Produce: {0}", ProdecerIterater);
            SendMessage(main_whandle, PRODUCERR_IN_LASTAIM, 0,
0);
            SharedBuffer = ProdecerIterater;
            BufferState = FULL;
            mut.ReleaseMutex();
            hNotEmptyEvent.Set();
            Thread.Sleep(1000);
        }//end of for
    }//end of Producer thread
```

again
        }

这里接上面的Consumer函数！！！

```
if (SharedBuffer == 0)
    { // test for end of data token
        //Console.WriteLine("Consumed  {0}: end of data\r\n", SharedBuffer);
        SendMessage(main_whandle, CONSUMER_ENDOFDATA_IN_LASTAIM, 0, 0);
        mut.ReleaseMutex();
        break;
    }
    else
    {
        //result = SharedBuffer;
        //Console.WriteLine("Consumed: {0}", result);
        SendMessage(main_whandle, CONSUMER_IN_LASTAIM, 0, 0);
        BufferState = EMPTY;
        mut.ReleaseMutex();
        hNotFullEvent.Set();
    }
  }//end of while
}//end of Consumer thread
```

```
private void button18_Click(object sender, EventArgs e)
{
  textBox3.Clear();
  SharedBuffer = 20;
  mut = new Mutex(false, "Tr");
  hNotFullEvent = new AutoResetEvent(false);
  hNotEmptyEvent = new AutoResetEvent(false);
  threadVec = new Thread[2];
  threadVec[0] = new Thread(new ThreadStart(Consumer));
  threadVec[1] = new Thread(new ThreadStart(Producer));
  threadVec[0].Start();
  threadVec[1].Start();
  threadVec[0].Join();
  threadVec[1].Join();
}
```

WinForm使用消息机制通信

1.定义结构体

```
//lpData是我们传输的字符串数据
public struct COPYDATASTRUCT
  {
    public IntPtr dwData;
    public int cbData;
    [MarshalAs(UnmanagedType.LPStr)]
    public string lpData;
  }
```

消息发送者

1.引用系统动态连接库中的SendMessage函数和FindWindow函数

```
[DllImport("User32.dll", EntryPoint = "SendMessage")]
private static extern int SendMessage(IntPtr hWnd, int Msg, int wParam, ref
COPYDATASTRUCT lParam);
[DllImport("User32.dll", EntryPoint = "FindWindow")]
private static extern int FindWindow(string lpClassName, string lpWindowName);
```

2.寻找消息接收者

```
int hWnd = FindWindow(null, @"消息接受者");
```

3.调用SendMessage函数发送信息

```
    public const int WM_COPYDATA = 0x004A;
    private void button3_Click(object sender, EventArgs e)
    {
      //byte[] sarr = System.Text.Encoding.Default.GetBytes(txtString.Text);
      byte[] sarr = Encoding.Default.GetBytes(messageSendTextBox.Text);
      int len = sarr.Length;
      COPYDATASTRUCT cds;
      cds.dwData = (IntPtr)Convert.ToInt16(0);//可以是任意值
      cds.cbData = len + 1;//指定lpData内存区域的字节数
      cds.lpData = messageSendTextBox.Text;//发送给目标窗口所在进程的数据
      SendMessage(hWnd, WM_COPYDATA, 0, ref cds);  //main_whandle是消息接
```

消息接受者

1.覆盖从父类中继承的消息处理函数DefWndProc

```
    protected override void DefWndProc(ref Message m)
    {
      switch (m.Msg)
      {
        case WM_COPYDATA:
          messageReceiveTextBox.Clear();
          COPYDATASTRUCT cds = new COPYDATASTRUCT();
          Type t = cds.GetType();
          cds = (COPYDATASTRUCT)m.GetLParam(t);
          string strResult = cds.lpData;   //得到发送的消息
          messageReceiveTextBox.AppendText(strResult);
          break;
        default:
          base.DefWndProc(ref m);
          break;
      }
    }
```

受窗体的句柄
    }

WinForm事件机制通信

1.创建委托

```
public delegate void MessageEventHandle(string msg);
```

消息发送者

触发消息接受者的事件

```
SendEvent(eventSendTextBox.Text);
```

注意：在实验中，消息接受者和消息触发者是同一个窗体，所以直接调用SendEvent即可，如果不是同一个窗体，则需要先取得消息接受者窗体对象，然后调用该对象的方法，即类似form1.SendEvent(eventSendTextBox.Text);的形式。

因为不确定到时具体怎么创建两个对象和保存状态，这里先没有写详细过程。PPT第5章第61页有基于继承的实现，我觉得考试时最好能在一个窗体创建时直接作为另一个窗体的成员变量存起来。

消息接受者

1.为委托注册事件

```
public event MessageEventHandle SendEvent;
```

2.实现事件的具体处理逻辑

```
SendEvent += (string msg) =>
        {
            eventReceiveTextBox.Clear();
            eventReceiveTextBox.AppendText(msg);
        };
```

WPF消息通信机制

1.定义结构体

```
public struct COPYDATASTRUCT
  {
    public IntPtr dwData;
    public int cbData;
    [MarshalAs(UnmanagedType.LPStr)]
    public string lpData;
  }
```

消息发送者（和WinForm的消息发送者完全相同）

1.引用系统动态连接库中的SendMessage函数和FindWindow函数

```
[DllImport("User32.dll", EntryPoint = "SendMessage")]
private static extern int SendMessage(IntPtr hWnd, int Msg, int wParam, ref
COPYDATASTRUCT lParam);
[DllImport("User32.dll", EntryPoint = "FindWindow")]
private static extern int FindWindow(string lpClassName, string lpWindowName);
```

2.寻找消息接收者

```
int hWnd = FindWindow(null, @"消息接受者");
```

3.调用SendMessage函数发送信息

```
    public const int WM_COPYDATA = 0x004A;
    private void button3_Click(object sender, EventArgs e)
    {
      //byte[] sarr = System.Text.Encoding.Default.GetBytes(txtString.Text);
      byte[] sarr = Encoding.Default.GetBytes(messageSendTextBox.Text);
      int len = sarr.Length;
      COPYDATASTRUCT cds;
      cds.dwData = (IntPtr)Convert.ToInt16(0);//可以是任意值
      cds.cbData = len + 1;//指定lpData内存区域的字节数
      cds.lpData = messageSendTextBox.Text;//发送给目标窗口所在进程的数
据
      SendMessage(hWnd, WM_COPYDATA, 0, ref cds);  //main_whandle是消
息接受窗体的句柄
    }
```

消息接收者

1.创建一个消息处理方法对收到的消息进行处理，返回值是IntPtr类型

```
IntPtr WndProc(IntPtr hwnd, int msg, IntPtr wParam, IntPtr
lParam, ref bool handled)
    {

      if (msg == WM_COPYDATA)
      {
        messageReceiveTextBox.Clear();
        COPYDATASTRUCT cds = (COPYDATASTRUCT)
System.Runtime.InteropServices.Marshal.PtrToStructure(lParam,
typeof(COPYDATASTRUCT));
        messageReceiveTextBox.AppendText(cds.lpData);
      }

      return hwnd;

    }
```

2.在主窗体加载时，为我们创建的消息处理方法注册钩子事件，使其能够接受消息

```
private void mainWindowLoad(object sender, RoutedEventArgs
e)
    {
      (PresentationSource.FromVisual(this) as
System.Windows.Interop.HwndSource).AddHook(new
System.Windows.Interop.HwndSourceHook(WndProc));
      define_Event();
    }
```

WPF事件机制通信

和WinForm事件机制通信完全一样，在上面有。

使用OLEDB对Excel数据进行管理

1.创建OleDbConnection 对象

```
String path = @"..\..\hotel.xls"
excelConnString = "Provider=Microsoft.ACE.OLEDB.12.0;Data Source=" + path + ";Extended Properties='Excel 12.0 Xml;HDR=YES';";
OleDbConnection excleConn = new OleDbConnection(excelConnString);
```

2.打开连接，进行查询，并将查询保存到一个DataTable对象

```
public DataTable ReadToTableExcelSheet_1()//excel存放的路径
    {
        try
        {
            excleConn.Open();
            DataTable sheetsName = excleConn.GetOleDbSchemaTable(OleDbSchemaGuid.Tables, new object[] { null, null, null, "Table" }); //得到所有sheet的名字
            string firstSheetName = sheetsName.Rows[0][2].ToString(); //得到第一个sheet的名字
            string sql = string.Format("SELECT * FROM [{0}]", firstSheetName); //查询字符串
            OleDbDataAdapter ada = new OleDbDataAdapter(sql, excelConnString);
            DataSet set = new DataSet();
            ada.Fill(set);
            excleConn.Close();
            return set.Tables[0];
        }
        catch (Exception)
        {
            excleConn.Close();
            return null;
        }

    }
```

3.将上一步中得到的DataTable对象给dataGridView作为数据源

```
DataTable dt = ReadToTableExcelSheet_1();
dataGridView1.DataSource = dt;
```

如果需要更改excel表，可以修改第一步中path的值

如果需要更改查询条件，可以修改第二步中的sql语句

使用DataAdapter操作SQLite数据库

1.创建SQLiteConnection 对象和MySqlDataAdapter对象

```
public SQLiteConnection createSqliteConn(string path)
    {

        sqliteConnString = "Data Source ="+path+"; Version = 3;";
        SQLiteConnection conn = new SQLiteConnection(sqliteConnString);
        return conn;
    }
MySqlConnection sqliteConn = createSqliteConn(@"..\..\demo.db");
MySqlDataAdapter sqliteAdapter = new SQLiteDataAdapter("SELECT FILE_NO as 文件编号,ID_KEY as 文件名称,SUBJECT as 主题,PUBLISH_DATE as 发布时间,IMPLEMENT_DATE as 施行时间,PUBLISH_ORG as 发布单位,FILE_NAME as 文件全名 FROM [tbl_fgwj]", sqliteConn);
```

2.创建DataTable对象并用sqliteAdapter 去填充DataTable对象

```
sqliteTable = new DataTable(); // Don't forget initialize!
sqliteAdapter.Fill(sqliteTable);
```

3.将sqliteTable设置为dataGridView3的数据源

```
dataGridView3.DataSource = sqliteTable;
```

4.如果我们修改了dataGridView3并点击了保存按钮，则可以通过SQLiteCommandBuilder对象和sqliteAdapter对数据库进行修改

```
 private void button6_Click(object sender, EventArgs e)
    {
        SQLiteCommandBuilder scb = new SQLiteCommandBuilder(sqliteAdapter);
        sqliteAdapter.Update(sqliteTable);
    }
```

5.若是需要从代码中增加数据库一条记录，可以参考如下代码

```
public void addSqliteRow(string file_name, string file_no, string publish_org, string publish_date, string implement_date, st ring subject, string full_name)
    {
        sqliteTable.Rows.Add(file_no, file_name, subject, publish_date, implement_date, publish_org, full_name);
        SQLiteCommandBuilder scb = new SQLiteCommandBuilder(sqliteAdapter);
        sqliteAdapter.Update(sqliteTable);
    }
```

6.若是需要从代码中删除数据库中的记录，可以参考如下代码

```
private void button7_Click(object sender, EventArgs e)
    {
        //从下往上删，避免沙漏效应
        for (int i = dataGridView3.SelectedRows.Count - 1; i >= 0; i--)
        {
            dataGridView3.Rows.RemoveAt(dataGridView3.SelectedRows[i].Index);
        }
```

```
        SQLiteCommandBuilder scb = new SQLiteCommandBuilder(sqliteAdapter);
        sqliteAdapter.Update(sqliteTable);
    }
```

7.若是需要dataGrid中的值或对其进行修改，可使用如下代码

```
dataGridView1.Rows[i].Cells["第1列列名"].Value
```

修改完后仍需要记得使用sqliteAdapter.Update(sqliteTable);保存修改

使用DataReader访问数据库

1.创建MySqlConnection对象

```
String mysqlConnectStr = "server=127.0.0.1;port=3306;user=" + userTextBox.Text + ";password=" + secretTextBox.Text +
";Database=windowsprogramdesignscheme;";
MySqlConnection mySqlConn= new MySqlConnection(connetStr);
```

2.打开连接并创建MySqlDataReader对象

```
mySqlConn.Open();
MySqlCommand mySqlCommand = new MySqlCommand("select * from book", mySqlConn);
MySqlDataReader mysqlReader = mySqlCommand.ExecuteReader();
```

3.使用mysqlReader读取列名并加入DateTable对象中

```
public DataTable mySqlTable = new DataTable();;
string[] attr = new string[mysqlReader.FieldCount];
for (int i = 0; i < mysqlReader.FieldCount; i++)
{
      attr[i] = mysqlReader.GetName(i);
}
mySqlTable = new DataTable();
for (int i = 0; i < mysqlReader.FieldCount; i++)
{
    mySqlTable.Columns.Add(attr[i]);
}
```

4.循环读取下一行并添加到中

```
object[] idx = new object[mysqlReader.FieldCount];
while (mysqlReader.Read())
{
       mysqlReader.GetValues(idx);
       mySqlTable.Rows.Add(idx);
}
```

5.将mySqlTable作为dataGridView的数据源

```
dataGridView4.DataSource = mySqlTable;
```