免责声明，此答案是自己整理的，不保证正确，错了别怪我

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？

（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。

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。

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：

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

查询前 10 条记录：

mysql 语句

```
select * from student limit 10;
sql server 语句
select top 10 * from student ;
```

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'
```

强制不使用缓存查询
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：
mysql 可以创建数据库，而 oracle 没有这个操作，oracle 只能创建实例；
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；

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。

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。

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。

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。

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。

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。

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。

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。

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。

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。

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

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。

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。

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。

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。

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。

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。

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。

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