字符串的包含

2016年5月2日　　18:05

问题描述： 给定一个长字符串a和一个短字符串b。请问，如何最快地判断出短字符串b中的所有字符都在长字符串a中？请编写函数bool StringContain(string &a, string &b)实现此功能。为简单起见，假设输入的字符串只包含大写英文字母。

下面举几个例子：

（1）如果字符串a是"ABCD"，字符串b是"BAD"，答案是true，因为字符串b中的字母都在字符串a中，或者说b是a的真子集。

（2）如果字符串a是"ABCD"，字符串b是"BCE"，答案是false，因为字符串b中的字母E不在字符串a中。

（3）如果字符串a是"ABCD"，字符串b是"AA"，答案是true，因为字符串b中的字母A包含在字符串a中。

解题思路：

1.最直接的办法，蛮力轮询。对于这种方法，如果长字符串a的长度为n，短字符串b的长度为m，那么此算法需要比较次数为O(n*m)。显然该方法时间开销很大。

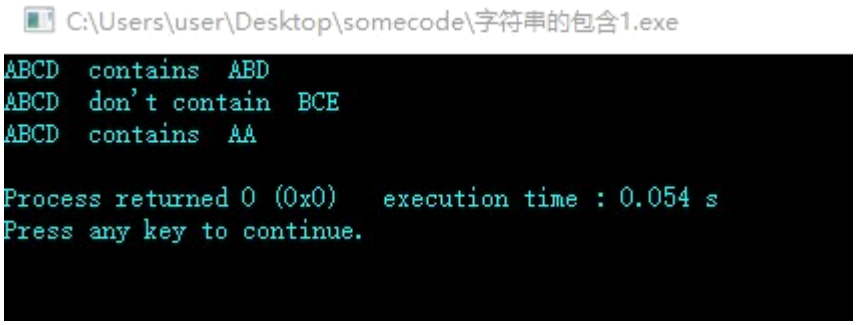
2.排序后轮询，根据题目只包括大写的英文字母，对两个字符串分别进行排序需要操作：O(mlogm)+O(nlogn),之后进行线性扫描需要O(n+m)

参考代码：

```
#include <bits/stdc++.h>
using namespace std;
bool StringContain( string &a , string &b )
{
    sort( a.begin() , a.end() );
    sort( b.begin() , b.end() );
    int lena = a.length();
    int lenb = b.length();
    for(int pa = 0 , pb = 0 ; pb < lenb ; )
    {
        while((pa < lena) && (a[pa] < b[pb]))
        {
            ++pa;
        }
        if((pa >= lena) || (a[pa] > b[pb]))
        {
            return false;
        }
        ++pb;
    }
    return true;
}
int main()
{
    string a  = "ABCD";
    string b[3];
    b[0] = "BAD";
    b[1] = "BCE";
    b[2] = "AA";
    for(int i=0;i<3;i++)
    {
        if(StringContain(a,b[i]))
            cout<<a<<"  contains  "<<b[i]<<endl;
        else
```

```
            cout<<a<<"  don't contain  "<<b[i]<<endl;
    }
    return 0;
}
```

GCC运行结果：



3.位运算法。

将长字符串a的所有字符都存放在一个散列表(hash table)中，然后轮询短字符串b，看b中的每一个字符是否都在散列表中，若果都在则返回true，否则返回false

可以用位运算（26位整数表示）给相应的长字符串a就算出一个相对应的"签名",然后将字符串b的每一个字符都放在a中进行查找。

参考代码：

```
#include <bits/stdc++.h>
using namespace std;
bool StringContain( string &a , string &b )
{
    int hash = 0;
    int lena = a.length();
    int lenb = b.length();
    for( int i = 0 ; i < lena ; i++)
    {
        hash |= (1 << (a[i] - 'A'));
    }
    for(int i = 0; i < lenb ; i++)
    {
        if( ( hash & ( 1 << ( b[i] - 'A' ) ) ) == 0)
            return false;
    }
    return true;
}
int main()
{
    string a  = "ABCD";
    string b[3];
    b[0] = "BAD";
    b[1] = "BCE";
    b[2] = "AA";
    for(int i=0;i<3;i++)
    {
        if(StringContain(a,b[i]))
            cout<<a<<"  contains  "<<b[i]<<endl;
        else
            cout<<a<<"  don't contain  "<<b[i]<<endl;
    }
    return 0;
}
```

GCC运行结果：

该算法的空间复杂度为O(1)，时间复杂度为O(n+m)。

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5334076>

单词翻转

2016年5月2日　　18:06

问题描述：输入一个英文句子，翻转句子中单词的顺序。要求单词内字符顺序不变，句子中单词由空格隔开。为简单起见，标点符号和普通字母一样处理。例如，若输入"I am a student."，则输出"student. a am I"

参考思路：首先将整个字符串进行翻转，然后根据空格来实现每个单词的翻转，关键在于判断结束为止以及下一个单词的开始位置。

参考代码：

```
#include <bits/stdc++.h>
using namespace std;
void swap_str(char* a, char* b)
{
    char t = *a;
    *a = *b;
    *b = t;
}
void str_reverse(char *start, char *over)
{
    while (start < over)
    {
        swap_str(start++, over--);
    }
}
void reverseString(char *s)
{
    char *start = s;
    char *over  = start;
for ( ; *over!= '\0'; over++ )
    {
    };
    over--;
str_reverse( start, over );
    start = s;
    over  = start;
    char* end_word = NULL;
while ( *(over-1) != '\0' )
    {
        if (*over == ' '|| *over == '\0')
        {
            end_word = over-1;
            str_reverse(start, end_word);
            start = over;
            while(*start == ' ')
            {
                start++;
            }
        }
        over++;
    }
}
int main()
{
    char str[] = "I am  a    student.";
    reverseString(str);
    printf("%s\n", str);
}
```

GCC运行结果：

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5330567>

字符串的旋转 （增改版）

2016年5月2日　　18:07

更多方法：http://blog.csdn.net/v_JULY_v/article/details/6322882

题目描述：给定一个字符串，要求将字符串前面的若干个字符移动到字符串的尾部。例如，将字符串"abcdef"的前3个字符'a'、'b'、'c'移动到字符串的尾部，那么原字符串将变成"defabc"。请写一个函数实现此功能。

方法一：蛮力移位

定义指向字符串的指针s,设字符串长度为n，首先实现LeftShiftOne(char *s ,int n)将一个字符移动到字符串的最后，然后调用m次实现将m个字符移动到字符串末尾

参考代码：

```
#include <bits/stdc++.h>
using namespace std;
void LeftShiftOne( char* s , int n )
{
    char t = s[0];
    for( int i = 1 ; i < n ; i++ )
    {
        s[i-1] = s[i];
    }
    s[n-1] = t;
}
void LeftRotateString( char* s , int n , int m )
{
    while( m-- )
    {
        LeftShiftOne( s , n );
    }
}
int main()
{
    char str[]="abcdef";
    cout<<str<<endl;
    LeftRotateString( str , 6 , 3 );
    cout<<str<<endl;
}
```

GCC运行结果：

方法二：三步翻转

拿上面的例子来说明：

（1）将原来的字符划分为两个部分A和B（划分依据为所移动m个字符）；

（2）将子字符串A和B进行翻转操作；分别为："abc"--->"cba" "def"--->"fed"

（3）最后将整体字符串进行翻转操作，从而实现题目要求的字符串翻转．过程为："cbafed"--->"defabc"

参考代码：

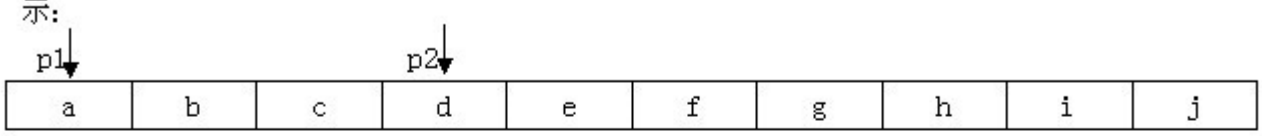
```
#include <bits/stdc++.h>
using namespace std;
void ReverseString( char *s , int from , int to )
{
    while( from < to )
    {
        char t = s[from];
        s[from++] = s[to];
        s[to--] = t;
    }
}
void LeftRotateString( char *s , int n , int m )
{
    m%=n;
    ReverseString(s,0,m-1);
    ReverseString(s,m,n-1);
    ReverseString(s,0,n-1);
}
int main()
{
    char str[]="abcdef";
    cout<<str<<endl;
    LeftRotateString(str,6,3);
    cout<<str<<endl;
}
```

GCC运行结果：

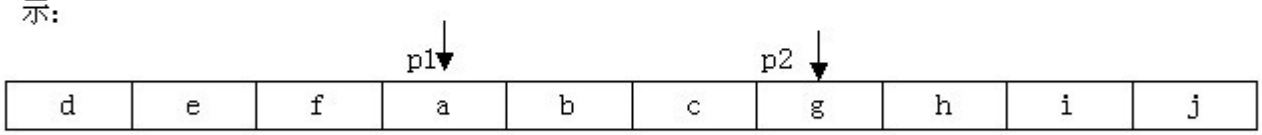
方法三：**指针翻转法 （感谢网友我非英雄的留言） 编程珠玑上还有另外一种解法，拿abcdef来说，保存a，将d移到a的位置，空出来的d的位置放a，a的位置空下来，放置e，依次遍历完abc**

下面，再针对上述过程，画个图清晰说明下，如下所示：

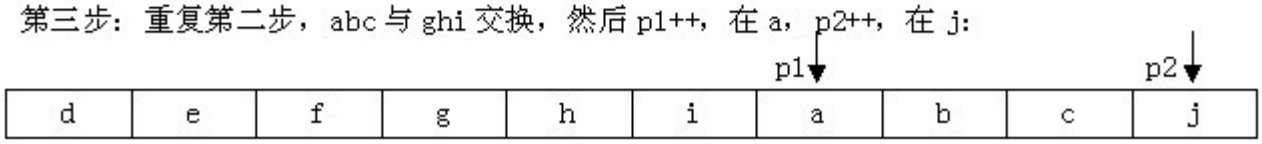
第一步：指针处于初始位置，其中 p1 指向首地址，p2 指向 p1+m（本例 m 为 3），如下图所示：



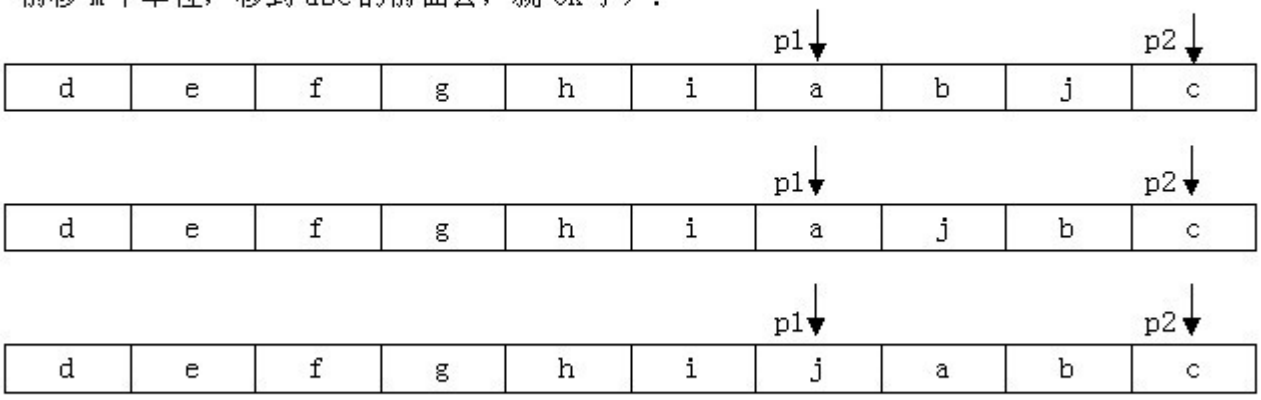
第二步：交换 p1 和 p2 所指元素，循环 m 次，abc 与 def 交换，然后 p1++,p2++，如下图所示：



第三步：重复第二步，abc 与 ghi 交换，然后 p1++，在 a，p2++，在 j：



第四步：如果 p2+m-1 不越界，说明 p2 到数组末尾之间所包含的元素为 m，即为上例（指 abcdefghi，九个元素）讨论的情况。否则，说明 p2 到数组末尾之间所包含的元素小于 m，保持 p1，p2 不变，将这些元素向左移动 m 个单元即可，如下图所示（下图的情况，就是 j 前移 m 个单位，移到 abc 的前面去，就 ok 了）：



至此，整个过程结束，得到最终结果。

解题思路：

1、首先让p1=ch[0]，p2=ch[m]，即让p1，p2相隔m的距离；

2、判断p2+m-1是否越界，如果没有越界转到3，否则转到4（abcdefgh这8个字母的字符串，以4左旋，那么初始时p2指向e，p2+4越界了，但事实上p2至p2+m-1是m个字符，可以再做一个交换）。

3、不断交换*p1与*p2，然后p1++，p2++，循环m次，然后转到2。

4、此时p2+m-1 已经越界，在此只需处理尾巴。过程如下：

4.1 通过n-p2得到p2与尾部之间元素个数r，即我们要前移的元素个数。

4.2 以下过程执行r次：

ch[p2]<->ch[p2-1]，ch[p2-1]<->ch[p2-2]，....，ch[p1+1]<->ch[p1]；p1++；

p2++；

参考代码：

```
#include<bits/stdc++.h>
using namespace std;
void RemoveString(char *s, int m)
{
    int len = strlen(s);
    if (len== 0 || m <= 0)
        return;
if (m % len <= 0)
        return;
int p1 = 0;
    int p2 = m;
    int k = (len - m) - len % m;
while (k --)
    {
        swap(s[p1], s[p2]);
        p1++;
        p2++;
    }
int r = len - p2;
    while (r--)
    {
        int i = p2;
        while (i > p1)
        {
            swap(s[i], s[i-1]);
            i--;
        }
        p2++;
        p1++;
    }
}
int main()
{
    char ch[]="abcdef";
    cout<<ch<<endl;
    RemoveString(ch,3);
    cout<<ch<<endl;
    return 0;
}
```

GCC运行结果：

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5330421>

面试问题5：const 与 define 宏定义之间的区别

2016年5月2日　　18:07

问题描述：const 与 define 宏定义之间的区别

(1) 编译器处理方式不同
　　define宏是在预处理阶段展开；
　　const常量是编译运行阶段使用；

(2) 类型和安全检查不同
　　define宏没有类型，不做任何类型检查，仅仅是展开；
　　const常量有具体的类型，在编译阶段会执行类型检查；

(3) 存储方式不同
　　define宏仅仅是展开，有多少地方使用，就展开多少次，不会分配内存；
　　const常量会在内存中分配(可以是堆中也可以是栈中)；

利用宏的优点：

1)让代码更简洁明了

一般来说，宏的名字更要注重有明确直观的意义，有时宁可让它长点。

2)方便代码维护

对宏的处理，在编译过程中称为“预处理”。也就是说在正式编译前，编译器必须先将代码出现的宏，用其相应的宏值替换，这个过程有点你我在文字处理软件中的查找替换。所以在代码中使用宏表达常数，归根结底还是使用了立即数，并没有明确指定这个量的类型。

利用const：

常量定义的格式为：

const 数据类型 常量名 = 常量值;

而const定义的常量具有数据类型，定义数据类型的常量便于编译器进行数据检查，使程序可能出现错误进行排查。常量必须一开始就指定一个值，然后，在以后的代码中，我们不允许改变此常量的值。

define定义表达式时要注意“边缘效应”，例如如下定义：

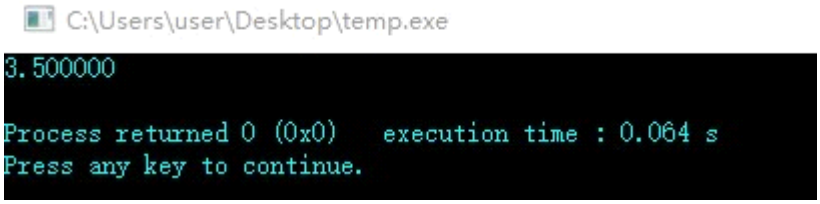
#define N 2+3 //我们预想的N值是5，我们这样使用N，int a = N/2; //我们预想的a的值是2.5，可实际上a的值是3.5原因在于在预处理阶段，编译器将 a = N/2处理成了 a = 2+3/2；这就是宏定义的字符串替换的“边缘效应”因此要如下定义：#define N (2+3)

举例：

```
#include <bits/stdc++.h>
#define N 2+3
using namespace std;

int main()
{
    double ans = N/2.0;
    printf("%lf\n" ,ans );
}
```

GCC运行结果：



来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5330283>

面试问题4：C语言预处理包括哪些

2016年5月2日　　18:07

问题描述：C语言 预处理包括哪些操作

C语言的三种预处理包括：宏定义（#define）、文件包含(#include)、条件编译(#if、#else、#endif)。

对于宏定义的介绍：

宏定义必须写在函数外，其作用域为宏定义起到源程序结束。C语言源程序中允许用一个标识符来表示一个字符串，称为“宏”。被定义为宏的标识符称为“宏名”。在编译预处理时，对程序中所有出现的宏名，都用宏定义中的字符串去代换，这称为宏替换或宏展开。

无参宏的宏名后不带参数。其定义的一般形式为： #define 标识符 字符串。 其中，“#”表示这是一条预处理命令(以#开头的均为预处理命令)。“define”为宏定义命令。“标识符”为符号常量，即宏名。“字符串”可以是常数、表达式、格式串等。

C语言允许宏带有参数。在宏定义中的参数称为形式参数，在宏调用中的参数称为实际参数。
对带参数的宏，在调用中，不仅要宏展开，而且要用实参去代换形参。 带参宏定义的一般形式为： #define 宏名(形参表) 字符串
举例说明：

```
#define INC(x) x+1  //宏定义
    y = INC(5);          //宏调用
```

在宏调用时，用实参5去代替形参x，经预处理宏展开后的语句为y=5+1。

带参宏注意事项：

（1）宏名和形参表的括号间不能有空格。
（2）宏替换只作替换，不做计算，不做表达式求解。
（3）函数调用在编译后程序运行时进行，并且分配内存。宏替换在编译前进行，不分配内存。
（4）宏展开不占用运行时间，只占编译时间，函数调用占运行时间(分配内存、保留现场、值传递、返回值)。
（5）为防止无限制递归展开，当宏调用自身时，不再继续展开。如：#define TEST(x) (x + TEST(x))被展开为1 + TEST(1)。

#include<a.h>和#include" a.h" 的区别：
使用尖括号表示在包含文件目录中去查找(包含目录是由用户在设置环境时设置的include目录)，而不在当前源文件目录去查找；
使用双引号则表示首先在当前源文件目录中查找，若未找到才到包含目录中去查找。用户编程时可根据自己文件所在的目录来选择某一种命令形式。

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5330238>

面试问题3：给一个单链表，怎么判断是否有环

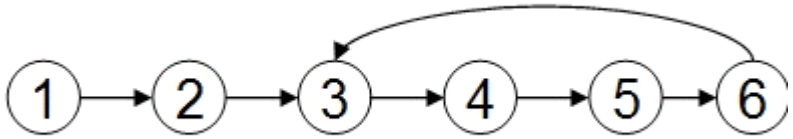
2016年5月2日　　18:07

问题描述：给一个单链表，怎么判断是否有环

前提条件：所给的指针指向的位置不确定（不一定指向头结点），单链表的长度未知，链表中存储的数据类型未知

个人方法：由于不知道链表的长度，以及所存储的数据类型，并且所给指针的指向任意的结点。那么不能按照从头依次遍历的方法。采用标记位的方法，初始化visited数组为0，将所给的指针依次移动，并且修改visited为1，每次移动判断该出的visited是否为0，若为0则表示未遍历到，当所得到的visited为1时，说明此单链表存在环。

其他方法之一：使用p、q两个指针，p总是向前走，但q每次都从头开始走，对于每个节点，看p走的步数是否和q一样。如图，当p从6走到3时，用了6步，此时若q从head出发，则只需两步就到3，因而步数不等，出现矛盾，存在环



其他方法之二：使用p、q两个指针，p每次向前走一步，q每次向前走两步，若在某个时候p == q，则存在环。

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5330203>

面试问题2：给一个5G的大文件，保存的数据为32位的整型，找到所有出现次数超过两次的数字

2016年5月2日 18:08

问题描述：给一个5G的大文件，保存的数据为32位的整型，找到所有出现次数超过两次的数字

大数据操作：

解决方法一：

依次遍历文件数据，

开始32二进制清0

每次读取一个数，先和二进制位与，如果为0 则没有，再把数字与二进制数位或。如果为1，则输出这个数

直到读取文件最后一个数字

举例说明：

0000 0000 0000 0000 0000 0000 0000 0000 开始的二进制位 倘若第一个数为2 那么

0000 0000 0000 0000 0000 0000 0000 0010 开始位与 结果不为2， 则二进制里面没有2 这个数字，然后用二进制与当前数字或运算结果为

0000 0000 0000 0000 0000 0000 0000 0010 再次读取一个数字 如果为 3 则开始先和新的二进制数位与

0000 0000 0000 0000 0000 0000 0000 0011 位与结果
为 0000 0000 0000 0000 0000 0000 0000 0010 结果不是3 则在和二进制位与那么现在的二进制数为：

0000 0000 0000 0000 0000 0000 0000 0011 在读取 一个数字 如果为8 和二进制数为与

0000 0000 0000 0000 0000 0000 0000 1000 结果为0，不等于，说明没有8，则再次为或

0000 0000 0000 0000 0000 0000 0000 1011 新的二进制数，在读取一个数字，如果为8

0000 0000 0000 0000 0000 0000 0000 1000 位与，等于.....1000 位与结果为8 那么8 就是出现2次的那个数字，

一下同理读取文件数据，直到文件结束。

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5330176>

面试问题1：给一个无序数组，找到其中位数。

2016年5月2日 18:08

问题描述：给一个无序数组，找到其中位数,并说出该算法的时间复杂度以及空间复杂度。

解题方法一：

中位数：当数组元素个数为奇数n时，中位数=a[n/2]。当元素个数为偶数时，中位数=(a[n / 2] + a[(n + 1) / 2])

按照这个中位数定义来求解。利用快速排序对数组进行排序，时间复杂度为O(nlogn) 空间复杂度为S(1); 之后的找到中位数所用时间复杂度为O(1) 空间复杂度为O(1)

解题方法二：

在了解方法二之前我们先来了解一下快速排序。对于快排算法，其中主元的选择对于数组的划分起到很大的影响，从而对于运行时间产生很大的影响。当主元不能很好地分割数组，即主元将数组分成一个子数组中有一个记录，而另一个子组组有 n -1 个记录时，下一次的子数组只比原来数组小 1，这是快速排序的最差的情况。如果这种情况发生在每次划分过程中，那么快速排序就退化成了冒泡排序，其时间复杂度为$O(n^2)$。老师曾经举过一个例子：甲、乙两同学去比较各自排序算法的好坏，他们可以互相阅读对方的原代码，然后互相给对方提供无序数组，最后比较那个运行比较快。乙同学看到甲同学的快排的代码，乙同学针对甲同学的代码专门写了所谓的无序数组（并非无序数组，只是甲同学的代码在解决这个无序数组的排序中会退化成冒泡排序）。然而甲同学看到乙同学的快排代码之后就老老实实的给了无序数组。为什么甲同学没有去找针对于乙同学的那个能够导致退化的无序数组呢？原因就是乙同学的快排主元的选择是随机化的。

随机化排序代码：

⊞

```
View Code
```

线性时间选择算法：

含义：是解决选择问题的分治算法

与快速排序不同的是：快排递归处理划分的两边，而线性选择只对其中

的一个部分进行处理

时间复杂度:O(n)

实现：需要用到随机划分函数

主要的过程好似：A[p...r]通过随机划分函数得到枢轴q

根据划分的两个区间A[p...q-1],A[q],A[q+1...r]

然后判断<=A[q]的元素数量k=q-p+1 与 需要寻找的第i小的关系

如果 k == i,直接返回A[q]

如果 k > i，说明第i小的元素只可能在区间A[p...q-1]中

如果 k < i ,说明第i小的元素在区间A[q+1...r]中 ， 并且问题转化为在A[q+1...r]中寻找第i-k小的元素

参考代码：

⊞

View Code

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5330155>

转载文章-----Rational Rose2007（v7.0）下载地址、安装及激活详解教程（图）

2016年5月2日　　18:08

转载地址：
http://www.cnblogs.com/leaven/p/3718361.html

最近需要画uml图，之前用的是Rose 2003版的，由于好久没进去了，结果发现原来的激活又失效了，omg又要激活一次？2003的激活真的很烦。。于是就去百度找新版的，由于很多链接都存在挂羊头卖狗肉的现象，以至于找了好久才找到资源。所以想出来分享一下。。

废话说得有点多了，进入正题，以下提供下载地址：

Rational Rose 2007（v 7.0）（含激活文件）：
http://pan.baidu.com/share/link?shareid=467021&uk=421719222

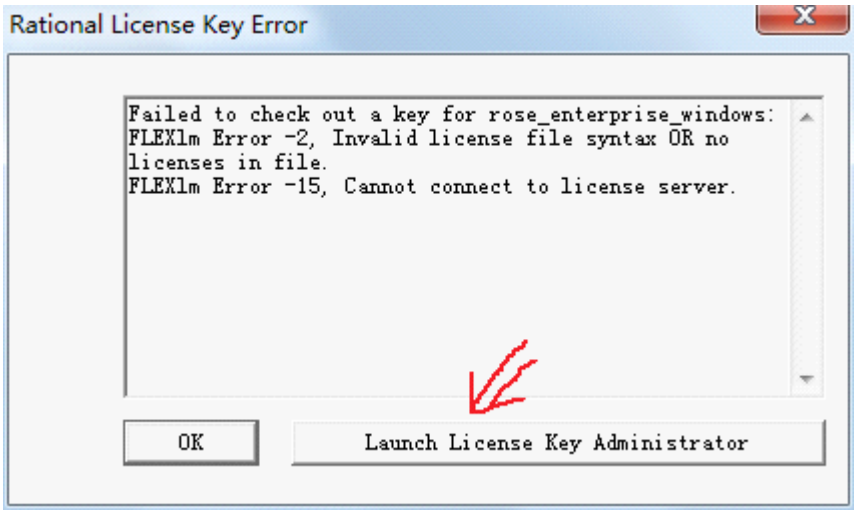
激活文件license.upd 下载地址：
http://download.csdn.net/detail/skl_tz/5377999

安装详解：

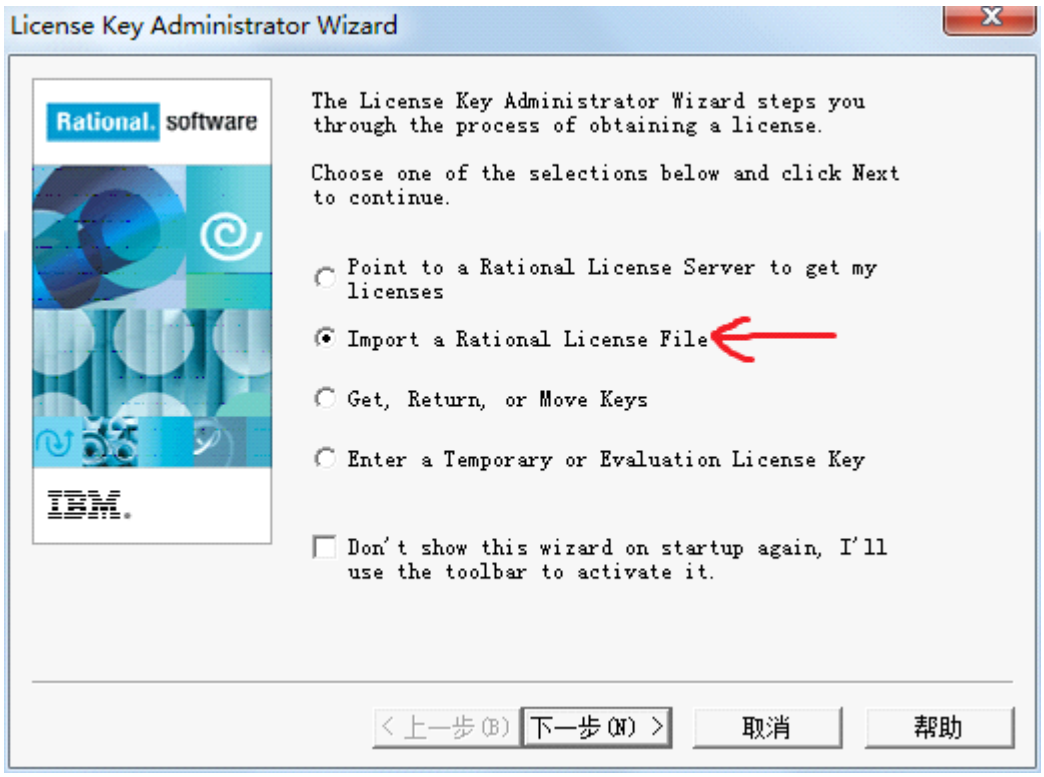
下载到的是一个压缩文件，里面包含文件：[IBM软件系列].IBM.Rational.Rose.Enterprise.v7.0-TFTISO.bin和license.upd，使用虚拟光驱打开bin文件，然后点击其中的setup.exe文件 -> 进入安装界面 -> 点击Install IBM Rational Rose Enterprise Edition-> Desktop installation from CD image-> 直至完成

激活详解：

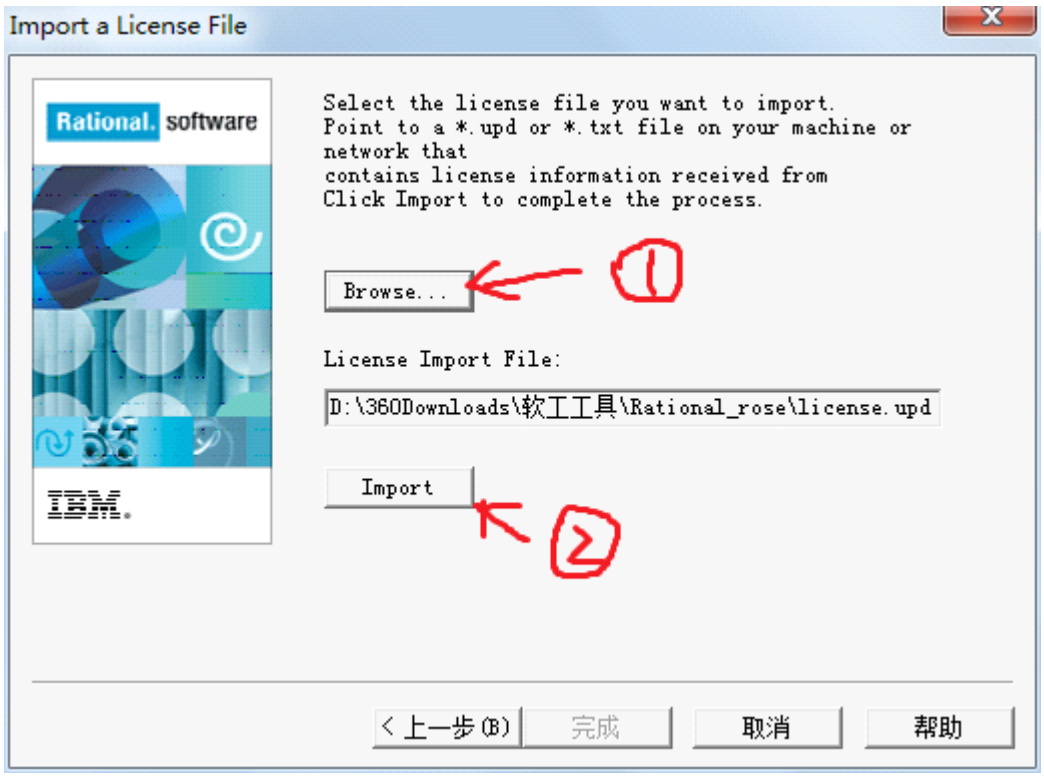
1. 安装完后，进入Rational所在目录(安装目录)下 -> Rose文件夹 -> 点击rose.exe ，由于尚未激活，会出现激活界面，如下：



2. 点击Launch License Key Administrator，出现如下界面：



3. 点击Import a Rational License File，出现如下界面：



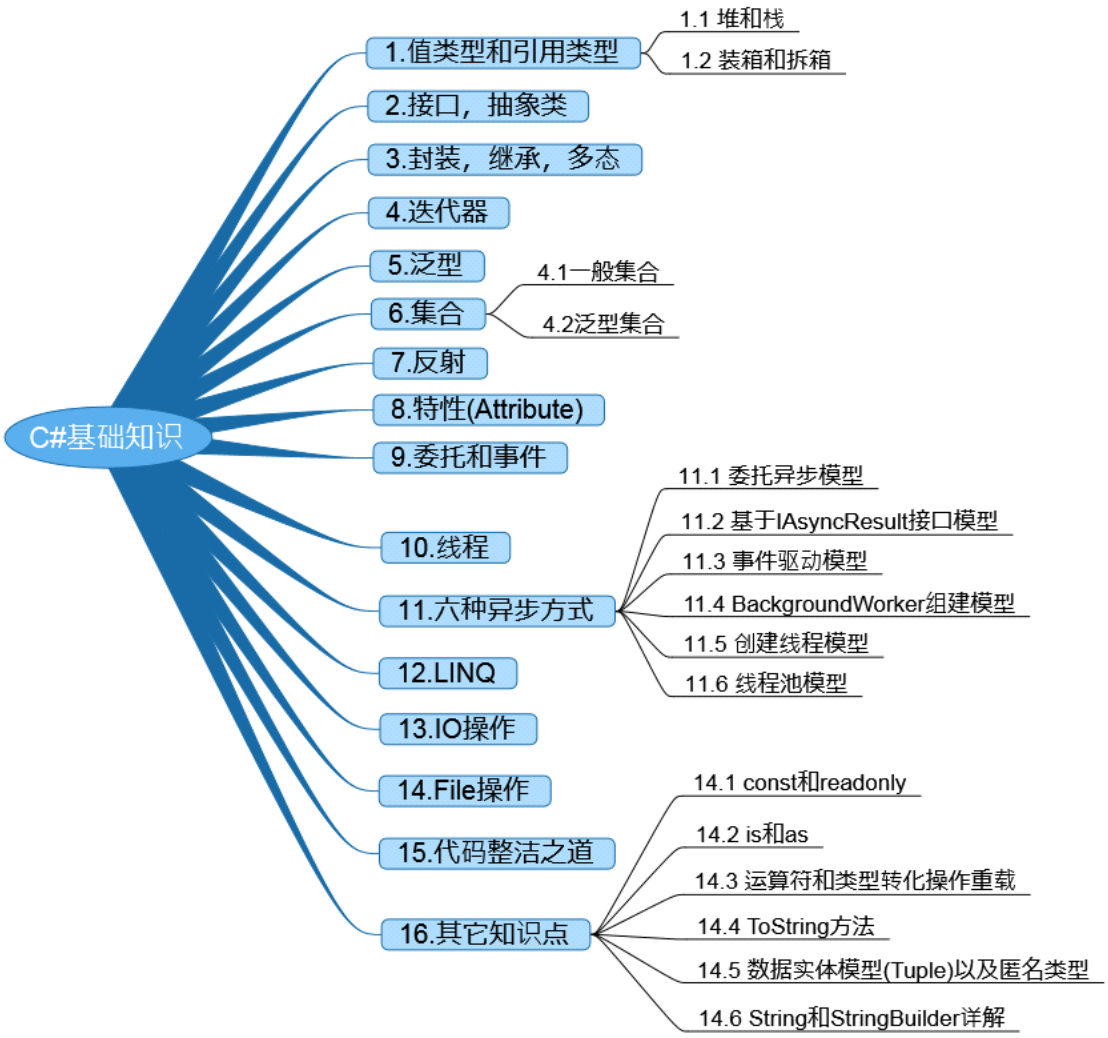
4. 点击Browse...，选择压缩文件附带的license.upd -> 点击Import ，待出现File imported successfully 提示时，表示激活文件导入成功，此时关闭激活窗口，再点击rose.exe，恭喜你，进入rose的世界了。。

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5277709>

转载文章----C#基础概念

2016年5月2日 18:08

转载地址：http://www.cnblogs.com/zhouzhou-aspnet/articles/2591596.html



1.值类型和引用类型

1.1堆和栈

简单的说值类型存放在堆栈上面，引用类型的数据存放在托管堆上面(它的引用地址却存放在堆栈上面)！

栈：它是一个内存数组，是一个先进后出的数据结构！

栈的特征：数据只能从栈顶进，从栈顶出！

堆：它是一个内存区域，可以分配大块区域存储某类型的数据，与栈不同的是它里面的数据可以任意排序和移除！

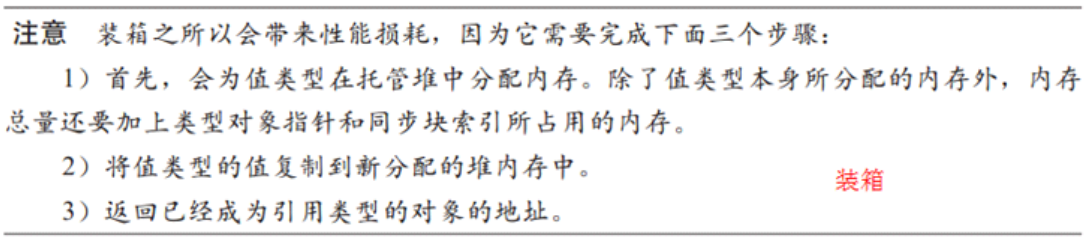
下面是园子的一张图，贴上来供大家参考啊！

问　　题	值　类　型	引 用 类 型
这个类型分配在哪里？	分配在栈上	分配在托管堆上

变量是怎么表示的？	值类型变量是局部复制	引用类型变量指向被分配得实例所占的内存
基类型是什么？	必须继承自System.ValueType	可以继承自除了System.ValueType以外的任何类型，只要那个类型不是sealed的
这个类型能作为其他类型的基类吗？	不能。值类型是密封的，不能被继承	是的。如果这个类型不是密封的，它可以作为其他类型的基类
默认的参数传递是什么？	变量是按值传递的（也就是，一个变量的副本被传入被调用的函数）	变量是按引用传递（例如，变量的地址传入被调用的函数）
这个类型能重写System.Object.Finalize()吗？	不能。值类型不好放在堆上，因此不需要被终结。	可以间接地重写
我可以为这个类型定义构造函数吗？	是的，但是默认的构造函数被保留（也就是自定义构造函数必须全部带有参数）	当然！
这个类型的变量什么时候消亡？	当它们越出定义的作用域时。	当托管堆被垃圾回收时。

1.2装箱和拆箱

关于装箱和拆箱是一个老生常谈的话题，也有很多文章来分析它，如：1. 6个重要的.NET概念:栈,堆,值类型,引用类型,装箱,拆箱 2. 值类型的装箱与拆箱浅析 3. 深入C#内存管理来分析值类型&引用类型，装箱&拆箱，堆栈几个概念组合之间的区别



这类的文章真的多了，再总结就没多大的意义了，看的时候多写写代码，多想想，就会明白的！

2.接口，抽象类，封装，继承，多态

接口和抽象类这两个概念还真不容易理解，有的时候理解一半，换一种方法考考你，你就会晕，到现在说实话我还没完全懂，一直没有把握它们的精髓，最近在看<<你必须知道的.NET>>，这是第二次看，收获很多...

大家还是有时间多看看**<<你必须知道的.NET>>**，这本书可以说是很详细的讲解了OO思想，还有看看设计模式的书，多想多练，可以时间会长一点，不过总有一点我们会开窍的...

这种东西不是通过总结一下就能熟练运用的，不过你起码要有一点面向对象的思想，要想有这种思想必须学习前辈留下的知识总结，这种才能理论结合实践，才能深入的了解OO思想

推荐文章：细细品味C#——抽象、接口、委托、反射（感谢虾皮老师啊...）

3.迭代器

主要是对foreach的深入理解，以及对两个接口的深入剖析(包括它们的泛型结构)：IEnumerable(可枚举类型)，IEnumertor(可枚举数)，文章入口：使用IEnumerable和IEnumerator接口，从yield关键字看IEnumerable和Collection的区别

4.泛型

泛型保证了类型安全，避免了装箱和拆箱的操作，提高了性能，可复用性也得到了很大的提高，下面就来说说基本的泛型语法吧！

项目中对于泛型和委托的结合运用也很多见，很多人不是为了语法而学习，而是泛型的扩展性让我们必须要知道它，把它实实在在的运用到项目中去，提高扩展性...

泛型语法不是很复杂，包括定义泛型类型，泛型方法，指定泛型约束，还有泛型约束包括只包括哪些类型等等，这些语法只要花些时间就能明白了，难的是一种思想，o(￣ヘ￣)o 我还很菜啊...

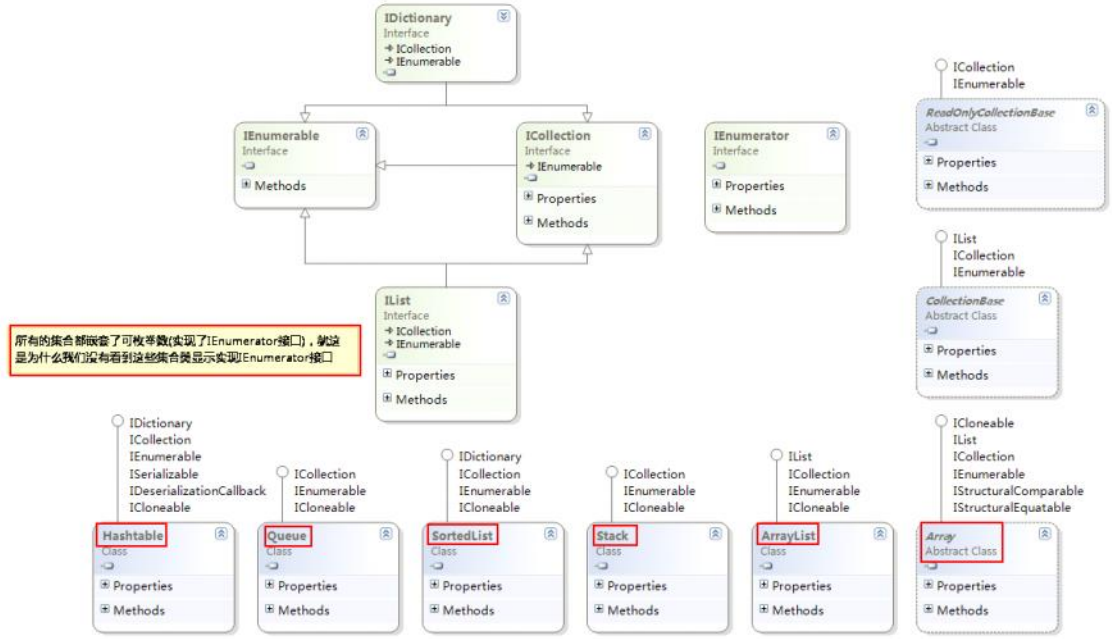
推荐文章：细细品味C#——泛型系列专题（虾皮帮我们已经整理关于泛型的精彩文章，看完之后会有很多的收获）

5.集合

5.1一般集合

.NET Frameword中关于集合的类存储在System.Collections命名空间下，其实一开始学习的时候感觉集合这个东西很神秘，能动态增加，删除，选择数据(比数据好用多了)，可是在学习之后，它的神秘感也随之消息，因为集合的底层代码跟数组有着密切联系的，请看：学习之路二：关于集合和数组内在联系的深入了解(里面也有个链接，可以点击学习)！

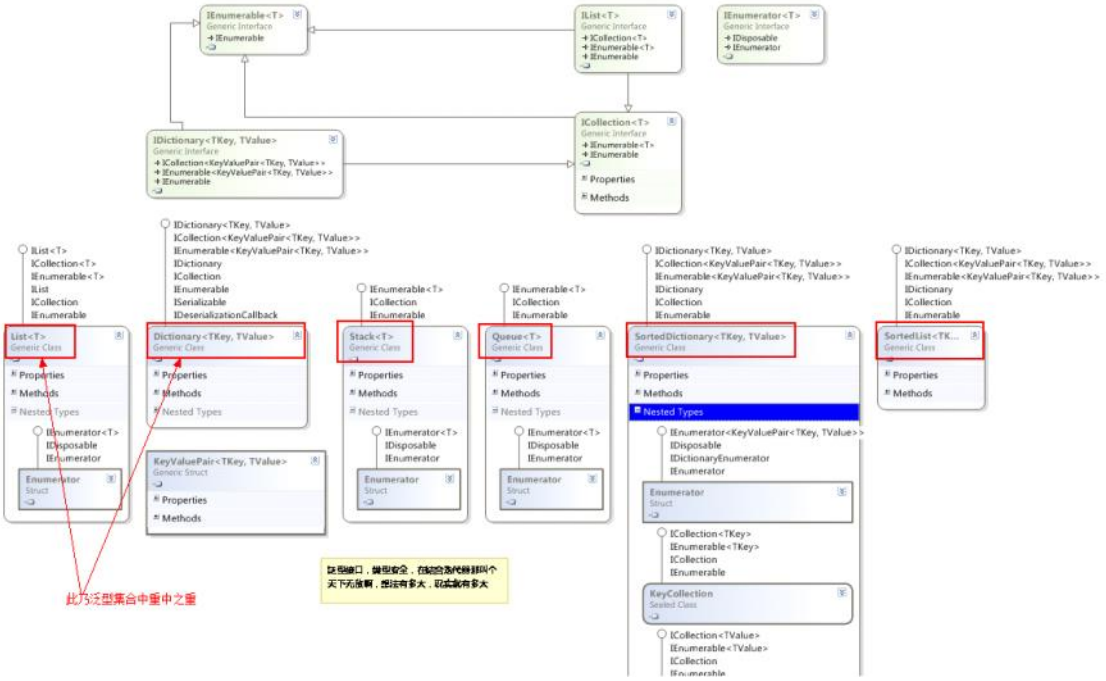
下面是非泛型集合类之间的关系图：



5.2泛型集合

自从.NET Framework引用泛型概念之后，它在C#编程方面掀起了一个泛型热潮，泛型实在太好用了，不仅是类型安全，可扩展性，重要的是在性能方面有了显著提高，这让我们苦逼的程序猿看到了曙光，哈哈...

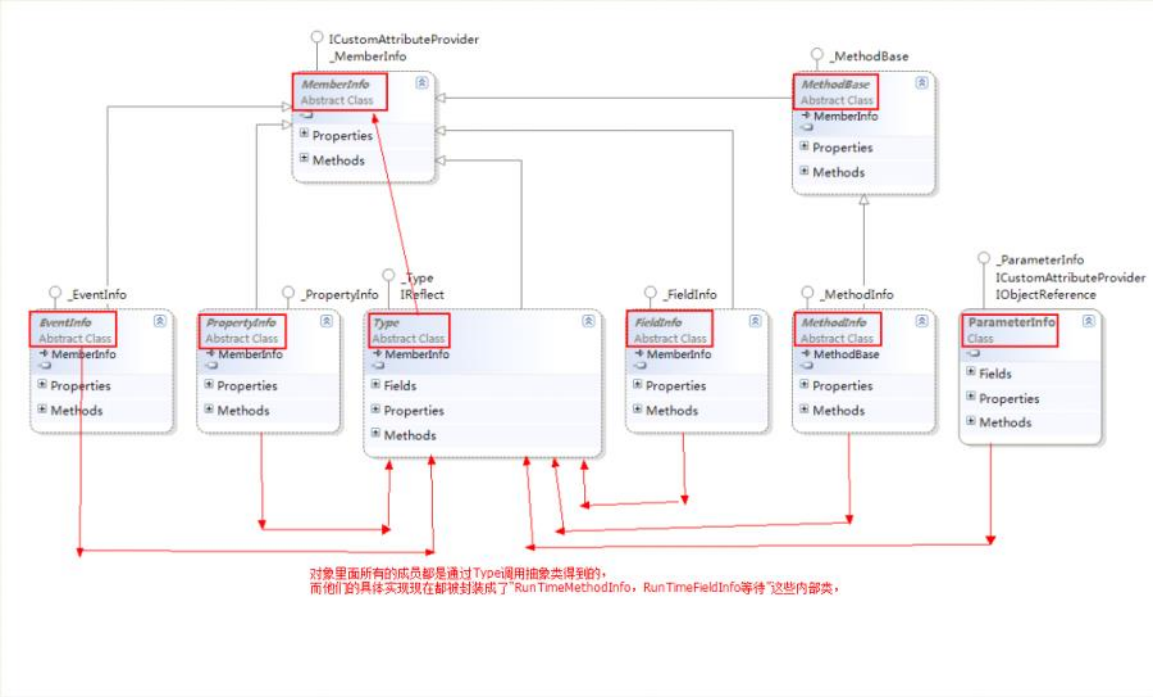
泛型集合类存储在System.Collections.Generic以及System.Collections.ObjectModel命名空间下，下面是集合类之间的关系图：



推荐文章：[细细品味C#——泛型系列专题](#)(有个pdf文件，下载下来回家慢慢看，同志们)

6.反射

反射这东西两面性很极端，很多人说它的坏，也有很它在某些方面有着重大的作用，下图是关于类型反射所需要用到的类之间的关系图：



除了类型反射之外，还有一种是程序集的反射，功能比较强大，可是我对它的研究比较少，我就推荐几篇好文章把(下面几篇文章我也正在学习中)...

推荐文章： 1..[.Net 中的反射(序章) - Part.1](#)

2..[.Net 中的反射(查看基本类型信息) - Part.2](#)

3..[.Net 中的反射(反射特性) - Part.3](#)

4..[.Net中的反射(动态创建类型实例) - Part.4](#)

7.特性(Attribute)

特性这个东西，在面向对象编程中有着非常重要的最用，在架构设计框架的时候，考虑使用特性的几率会非常的大！

特性结合反射技术就可以实现依赖注入，以前看到公司一个项目在写测试代码的时候，总是给每个方法加上[RollBack]的特性，当方法结束后，所有数据库的操作都将会回滚，我很费解，因为RollBack是自己定义的，怎么就一加上这个特性就自动完成回滚了！

下面就是完整的Rollback代码，可是我在使用它的时候遇到一个问题，就是它只可以用于单元测试，我尝试着把它用于一般的方法当中，可是一直没有实现回滚功能，我感到很费解，有兴趣的朋友可以帮我看看...

View Code

只能在单元测试里面进行调用：

```
 1     [TestClass()]
 2     public class ProgramTest : TestFixture  //继承这个类
 3     {
48         [TestMethod()]
49         [RollBack()]    //添加这个RollBack特性，就能实现回滚了
50         public void MyTestTest()
51         {
52             SqlConnectionStringBuilder connectionString = new
SqlConnectionStringBuilder
53             {
54                 DataSource = @"LBDZ-20120514VC\SQLEXPRESS",
55                 InitialCatalog = "My",
56             };
57             connectionString.IntegratedSecurity = true;
58
59             using (SqlConnection conn = new
SqlConnection(connectionString.ToString()))
60             {
61                 conn.Open();
62                 SqlCommand cmd = conn.CreateCommand();
63                 cmd.CommandText = "INSERT INTO dbo.MyTable (    id) VALUES
( 6666 )";
64                 cmd.ExecuteNonQuery();
65                 Console.WriteLine("OK");
66             }
68             Assert.IsTrue(true);
69         }
70     }
```

Question:这个RollBack我至今还没有弄懂它怎么来实现的，如果那个园友能看懂的话，可以私信给我或留言给我，我会打心里感谢你的，可能会涉及到AOP和IOC的知识，希望大家帮帮我把，纠结了很长时间啦...

推荐文章：1. 关于C# 中的Attribute 特性

2. C# 用Attribute实现AOP事务 [C# | AOP | Attribute | ContextAttribute | IContributeObjectSink | IMessageSink]

3. Attribute在.net编程中的应用（一）

8.委托和事件

其实把理解事件跟字段和属性联系起来，虽然这样说可能会不严谨点，但是从一些大的方面讲事件就是对委托的封装，类似于属性对字段的封装，这种说法还是行得通的！

想要定义一个完整的委托和事件，需要经历一下步骤(需要注意一些命名规范)：

① 定义事件 → 委托使用微软提供的EventHadler<TEventArgs>泛型委托，一般都会有两个参数：

A) “object sender”定义的事件依附的对象，也就是事件定义在那个类中，那么这个参数就为这个类的实例化对象，一般都会用“This”！

B) “EventArgs e”也就是用于传递一些参数信息的对象，也可以使用自己定制的参数了

② 创建参数类 → 如果有必要定制的数据参数类(这个类似于创建自己的实体类用来传递信息)，这个参数类应该继承于EventArgs这个类!

③ 执行事件 → 其实在执行事件的时候还是有一定的规范的，比如方法名必须为“On+事件名”，还有在执行事件要判断下时候为null，，然后在调用！

④ 注册事件 → 调用事件(在传递事件对象的时候最好用“this”关键字)

⑤ 依附事件的方法 → 最后定义依附在这个事件中的方法，也就是执行这个事件的方法体，深入了解，其实依附事件中的方法其实都最终依附在事件衣服的委托中，这个委托会生成一个委托实例，以及一个委托链！

委托和事件定义语法：

委托： 访问修饰符 + delegate + 返回值类型 + 委托名(参数列表);

事件： 访问修饰符 + event + 委托名 + 事件名;

委托和事件跟观察者模式联系比较密切，可是我还是没有理解它的精髓，可能是我还太菜了...

总结：灵活运用事件和委托将会给你的程序带来更好的扩展性，这需要丰富经验的积累，好了推荐几篇我曾经学习过的文章把！

推荐文章：1. 庖丁解牛——深入解析委托和事件

2. C# 中的委托和事件

3. C#中的委托和事件(续)

9.线程

对于线程学习过，可是一直没有做过多线程的项目，一直没有领悟到它的精髓，也只能停留在表面的高度！

我就想说下Thread中的后台线程和前台线程（默认为“前台线程”），在这里总结下(其实我也是学习前辈们的知识)。

前台线程：当所有的前台的线程都执行完毕以后才会退出程序！

后台线程：对于后台线程，程序是不管你是否是执行完成的，不过当你程序一旦强制退出，后台线程也会终止的！

```
1               Thread thread = new Thread(delegate()
2               {
3                   Console.WriteLine("线程开始工作");
4                   Thread.Sleep(2000);    //暂停两秒钟
5                   Console.WriteLine("线程结束");
6               });
7               thread.IsBackground = true;    //分别设置为true和false，看看控制台运行的情
况，我相信你能很快明白的
8               thread.Start();
```

```
9             Console.WriteLine("主线程结束");
```

总结：设置为后台线程相当于我们说的异步，而前台线程就相当于同步，执行好线程在执行主程序！

能够熟练使用多线程，还是要在项目中不断的实践，可是项目是可遇而不可求的东西，现在我的项目是肯定要不到了，只能自己看看文章，熟悉熟悉知识啊...

推荐文章：1. C# 温故而知新： 线程篇(一)

2. C# 温故而知新： 线程篇(二)

3. C#综合揭秘——细说多线程（上）

10.六种异步方式

10.1 委托异步模型

使用的是委托的BeginInvoke和EndInvoke异步执行模式！

必须要有两个条件：

① 必须要有个委托作为寄宿体

② 执行函数　ExecuteFunction

③ 回调函数　CallBackFunction ，所谓的回调函数就是获取执行函数的返回值！

有了上面三种条件之后，就可以直接调用Begin和End进行委托异步编程了，其中还有细节问题需要注意，下面我们就一一来看！

具体思路步骤：

① 选择一个适合的委托类型，如参数列表，返回值类型

② 创建一个执行函数，必须跟委托的参数列表和返回值类型对应起来

③ 创建一个回调函数，它只有一个参数没有返回值，参数类型为**IAsyncResult**类型，这是使用委托实现异步的规范写法，不可改变

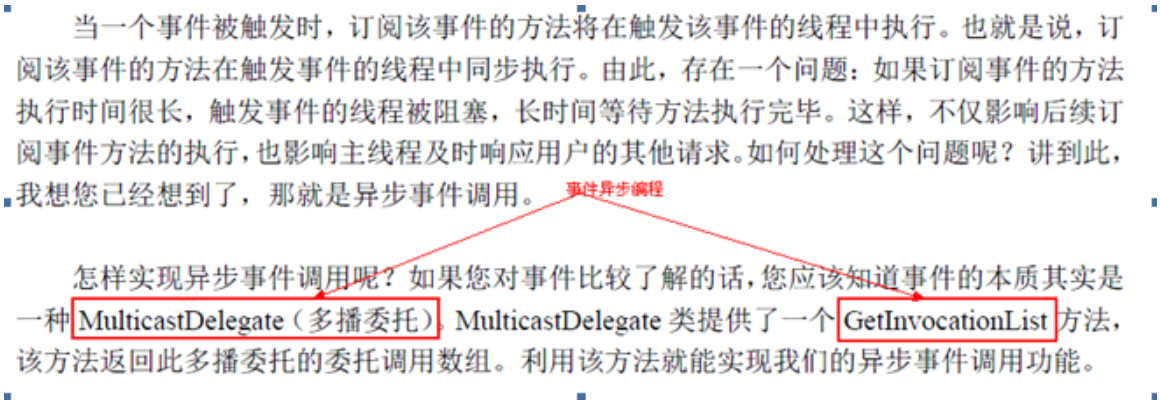
代码实现：

```
 1         //写一段简洁的代码
 2         private void button1_Click(object sender, EventArgs e)
 3         {
 4             //定义委托，并指定异步的执行方法
 5             Func<string, string> func = new Func<string, string>
(ExecuteFunction);
 6             //开始异步，并指定异步的回调函数
 7             func.BeginInvoke("实现了异步", new AsyncCallback(CallBackFunction),
"my");
 8         }
 9
10         private string ExecuteFunction(string str)          //执行函数
11         {
12             Thread.Sleep(2000);
13             // To Do
14             return str;
15         }
16
```

```
17          private void CallBackFunction(IAsyncResult ar)      //回调函数
18          {
19              //转化变量类型
20              //因为委托异步编程的类型为AsyncResult类，而这个类又是实现了IAsyncResult接口
的，可以说是它的基类！
21              AsyncResult async = ar as AsyncResult;
22              Func<string, string> func = async.AsyncDelegate as Func<string,
string>;
23              //获取异步执行函数的返回值
24              string str = func.EndInvoke(ar); MessageBox.Show(str);
25          }
26          //在最后进行类型转化的时候，尽量使用“as”进行转化！
```

10.2 事件驱动模型实现异步

这个模式的异步编程是所有异步方式中最为复杂的一个，我对它的理解也是很有限的，只限于使用它，不会自己构建它！



基于事件模型的异步不是一个通用型的，只有当需求要进行事件模型异步编程的时候才要进行事件模型异步的创建，主要有两种方法：

①通过获取事件中的委托列表，然后通过委托实现异步

首先这个异步思想是基于事件模型的，所以它会对你的事件定义有很大的要求，主要就是那两个参数的定义，可能会重新定义存储信息类！

实现思路：

a) 通过事件注册执行方法

b) 获取事件依附的委托

c) 最后就是通过委托调用BeginInvoke实现异步

代码如下：

```
View Code
```

②通过自定义事件模型实现异步（暂时不会，我会给出几篇文章参考）

10.3 使用IAsyncResult接口实现异步

使用IAsyncResult接口编程和使用委托异步编程的最大区别：

① 委托异步调用阻塞发生在线程池的工作线程

② IAsyncResult异步调用阻塞方式在线程池的I/O完成线程

Note：其实使用IAsyncRsult接口重要的是对于一些流操作或者一些文件操作都是固

定的Begin和End，所以只有当使用那些异步操作的时候才使用它

使用方法跟委托异步很相似，具体的实现代码就不贴了！

10.4基于BackgroundWorker组建实现异步

Note：其实这个组件最终也是基于事件异步模式进行创建的，所以说它就是封装了事件异步编程模型，从而使开发者使用更方便！

这个模型使用率比较高，因为微软都帮我们封装好了，只要学习一下就会使用它了！

```
1         private void button1_Click(object sender, EventArgs e)
2         {
3             BackgroundWorker worker = new BackgroundWorker();
4             //注册执行函数
5             worker.DoWork += ExecuteFunction;
6             //注册回调函数
7             worker.RunWorkerCompleted += CallBackFunction;
8             //执行事件，启动异步操作
9             worker.RunWorkerAsync("通过e.Argument来读取的");
10        }
11
12        //执行函数
13        public void ExecuteFunction(object sender, DoWorkEventArgs e)
14        {
15            Thread.Sleep(2000);
16            //因为result的类型为Object
17            //所以可以存入各种对象各种控件对象都可以存入
18            e.Result = "实现异步，这是我存入的值";
19            string str = e.Argument.ToString(); //这边的值是跟你启动异步方法中存入的参
数
20        }
21
22        //回调函数
23        public void CallBackFunction(object sender, RunWorkerCompletedEventArgs
e)
24        {
25            //读取存入的值，而这个类似于委托异步编程中获取执行函数的返回值
26            //相当于委托异步编程中 → 返回值的应用
27            MessageBox.Show(e.Result.ToString()); //读取
28        }
```

10.5创建后台线程实现异步

实现思路：

① 创建新的线程，并指定在线程运行的程序

② 设置线程为后台运行，推荐一篇文章：

③ 启动线程

代码实现：

```
1         private void button3_Click(object sender, EventArgs e)
2         {
3             //创建新的线程，并指定它的执行方法
4             Thread thread = new Thread(new
```

```
ParameterizedThreadStart(MyThreadMethod));
 5              thread.IsBackground = true;
 6              thread.Start("aa");
 7          }
 8          public void MyThreadMethod(object obj)
 9          {
10              //不可以把这个暂停的时间方法创建线程之前，因为你在线程之前暂停那个时候还没有创建
好，不能实现异步，所以这是不可以的
11              Thread.Sleep(2000);
12              MessageBox.Show("这是线程实现异步的");
13          }
```

10.6使用线程池实现异步

实现比较简单，**代码如下：**

```
 1          //实现代码：
 2          private void button4_Click(object sender, EventArgs e)
 3          {
 4              //这句代码很重要
 5              ThreadPool.QueueUserWorkItem(MyThreadMethod, "aaa");
 6          }
 7
 8          public void MyThreadMethod(object obj)
 9          {
10              Thread.Sleep(2000);
11              MessageBox.Show("这是线程实现异步的");
12          }
```

10.7六种异步模式的总结

其实这个总结是根据Fish文章进行总结的，o(∩_∩)o 哈哈...

1.异步委托调用：它的实现是将原来需要阻塞的操作交给线程池的**工作线程**来处理了，此时线程池的工作线程被阻塞了！但是此方法对于依赖【线程池的工作线程】来处理任务的编程模型来说是没有意义的，比如Asp.Net ， Windows Services以及Web Services这些服务类的编程模型！所以它比较适应一些单线程编程模型，比如Winform这种的单线程！

2.使用IAsyncResult接口：它的实现是将原来需要阻塞的操作交给线程池的**I/O完成线程**来处理了，所以它适合任何类型。但是有限制，因为不是所有的API都支持此类接口，不过许多的I/O操作是支持此接口的，还有实现起来会比较复杂！

3.基于事件的异步：这种方式可以认为是对其它异步方式的封装，其主要目的是简化异步调用模型，使用者可以直接调用事件就能实现异步，如果此模式是对第二种异步方式进行异步封装，那么它将具体第二种的所有优点！

4.创建新线程的异步：主要特点是在后台创建一个新的线程来执行异步方法，如果有很多用户同时执行异步操作，那么后台就会创建无数个线程，损害性能，尤其是对服务类的编程模型来说使用起来没有任何意义，所以在没有上诉情况下可以考虑使用这种异步方法！

5.使用线程的异步：基本上跟创建新线程类似，仅仅适用于一些桌面程序！

6.使用BackgroundWorker的方式：它的底层也是在使用线程池的工作线程，也是采用的基于事件的异步模式，只不过它使用起来真的很方便，它不是使用的IAsyncResult接口进行异步操作的，只是模式上类似于事件异步模式！

最后：在.NET中标准的异步模式都是使用的IAsyncResult接口，所以后三种并不算真正的异步，但它们却在某些场合有着很大的作用！

强烈推荐文章：1. C#客户端的异步操作

2. 我所知道的.NET异步

3. 详解.NET异步

上面的学不会也找我，:-)

11.LINQ

11.1 LINQ原理剖析

这里主要是关于LINQ家族中：Linq To Object，对它的研究还是比较多的！

LINQ是基于委托的一种高级语法，如果不能正确的理解委托的定义和使用，你就不会真正的理解**LINQ和委托**天衣无缝的结合，还有一点就是**LINQ和迭代器**的结合，所以理解委托和迭代器，LINQ你也就会用了！

11.2 Lambda表达式，匿名方法

LINQ也就是一些语法糖，会语法了，LINQ自然而然就会用了，下面就把常用的语法汇总一下，再复习一次：

```
 1   Func<string, string> myFunc = delegate(string str)  //匿名方法语法，括号内为参数列表
 2   {
 3       return "sss";
 4   };
 5   myFunc += strOne =>//这边的strOne是个方法，我没写，只要注意参数和返回值一致就可以了
 6   {
 7       return strOne;
 8   };
 9   myFunc += (string strTwo) =>//Lambda表达式，指定了参数类型
10   {
11       return strTwo;
12   };
13   myFunc += (strThree) =>//Lambda表达式，也可以不指定类型，系统会自动检测
14   {
15       return strThree;
16   };
17   myFunc += strFour => "ssssss";  //不需要加“return”，因为编译器会帮你自动加上去的！这样写法就搞急了，放眼一看说真的一开始还真看不懂！
18   Action myAction = () => Console.WriteLine("sssssssss");
19   myAction += () => Console.WriteLine("sssssss");    //如果没有参数，直接使用括号就可以了
20   myAction += delegate()
21   {
22       Console.WriteLine("sssssssssssss");
23   };
```

注意事项:①如果有参数的话，必须写明参数变量！

②可以不写参数类型，编译器会自动判断！

③如果没有参数必须要写一个空的括号，这样才说明没有参数！

④如果方法中只有一行代码，可以不要花括号，反之则要！

11.3 LINQ扩展方法汇总

Family	Query Operations
Filtering	OfType, Where
Projection	Select，SelectMany
Partitioning	Skip, SkipWhile, Take, TakeWhile
Join	GroupJoin, Join
Concatenation	Concat
Ordering	OrderBy, OrderByDescending, Reverse, ThenBy, ThenByDescending
Grouping	GroupBy, ToLookup
Set	Distinct, Except, Intersect,Union
Conversion	AsEnumerable, AsQueryable, Cast, ToArray, ToDictionary, ToList
Equality	SequenceEqual
Element	ElementAt, ElementAtOrDefault, First, FirstOrDefault, Last, LastOrDefault, Single, SingleOrDefault
Generation	DefaultIfEmpty, Empty, Range, Repeat
Quantifiers	All, Any, Contains
Aggregation	Aggregate, Average, Count, LongCount, Max, Min, Sum

11.4 LINQ详解方法使用细节

①写个我觉得比较难的“Group By”方法

分组语法最难理解的就是对“key”的理解，以及“into”关键字后面包含的是什么东西（***就是我们分组后要使用的数据源***），以及理解“by”关键后后面就是存放“key”的地方，代码示例：

```
 1   var queryTwo =   from book in SampleData.Books
 2                    group new { book.Title, book.PageCount } //分组的关键字
 3                    by new { Name = book.Publisher.Name, Subject =
book.Subject.Name } //给分组关键字设置别名
 4                    into newBook    //分组之后的数据源
 5                    select new { Books = newBook };
 6
 7   //使用嵌套“foreach”来循环遍历
 8   foreach (var item in queryTwo)
 9   {
10       Console.WriteLine(item.Books.Key);
11       foreach (var book in item.Books)
12       {
13           Console.WriteLine(book.Title + "  " + book.PageCount);
14       }
15   }
```

总结：理解 → group A by B into C (A：数据源中的关键字 B：分组之后给关键字去的别名 C：分组之后的数据源)！

因为最后的newBook是我的最终数据源，上面也提过最终数据源是包括所有分组的key，所以在调用key的时候应该拿最终的数据源集合调用，还有当大于1个分组关键字时进行读取的时候会以这样的方式出现{ Name=Book,Subject=aaa}，还有大部分情况遍历分组LINQ查询都会使用嵌套的Froeach语句来检索数据！

推荐一篇文章：[你能指出这个 ForEach 扩展方法中的错误吗？]，其实可以做的更好，我需要不断的学习啊...

11.5 自定义LINQ扩展方法

如果想写自定义LINQ方法，无非就是继承**IEnumerable**接口的扩展方法，写一个我们常用的foreach，这样可以节省我们好多的foreach代码，如下代码：

```
public static class Program
{
    static void Main(string[] args)
    {
        string[] strings = { "a", "b", "c" };
        strings.MyForeach(Console.WriteLine); //括号里面是个方法
    }

    public static void MyForeach<T>(this IEnumerable<T> source, Action<T> func)
    {
        foreach (T item in source) //先循环遍历我的数据源，然后把每个数据方法我的匿名方法中，从而输出值！
        {
            func(item); //item是参数
        }
    }
}
```

总结：写代码无时无刻，但是我们最关注是对代码的思考，对代码的感悟，思想的提升，不管怎么样，写完之后停下来想想，想什么全由个人来定，有人想性能，有人想重构，想优化，想简洁，只有努力过后思想强大了才能在编程道路上得心应手！

本人也研究了一些关于Linq To SQL的知识，可惜学艺不精啊！

强烈推荐文章:1. LINQ体验系列文章导航

2. （原创）一步一步学Linq to sql系列文章

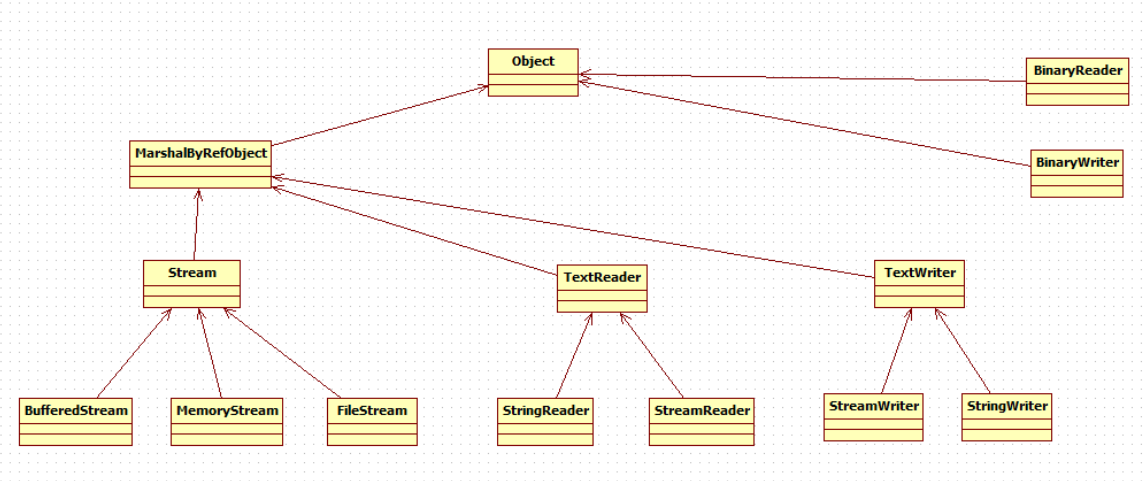
3. LINQ之路系列博客导航

看完上面三个系列，学不会找我，:-)

12.IO

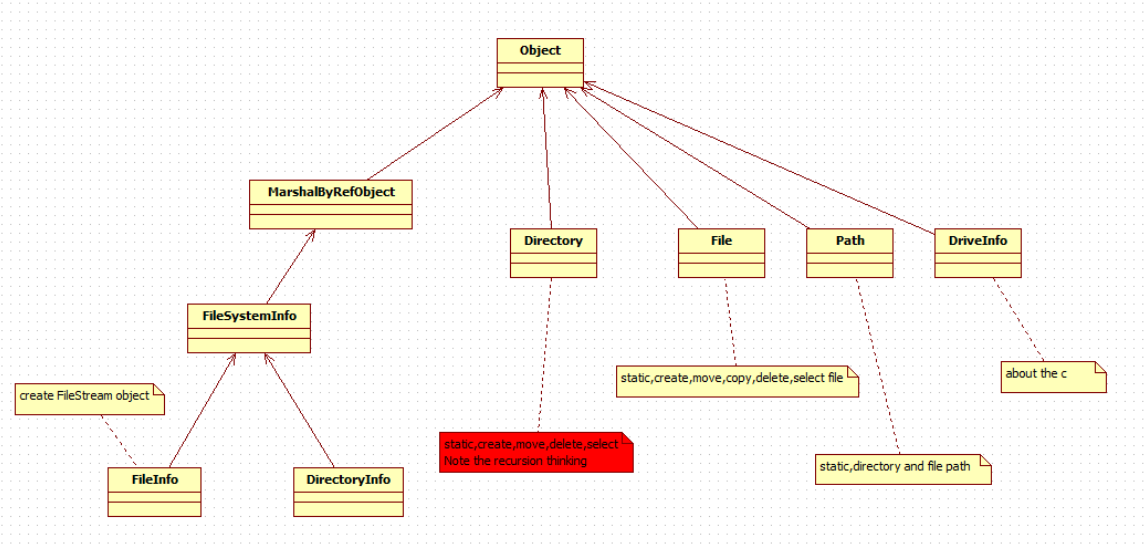
这个系列我没有系统的学习过，不过园子里面有个人写的很好，也正在学习中，推荐文章：**C# 温故而知新：Stream篇（一）**

下面是我整理的流操作系统类图：



13.File

熟悉文件系统的操作是学习.NET Frameword必不可少的一部分，我没有系统的学习过这些知识，但能运用一些常见方法进行项目开发，下面是文件系统主要的类图架构，理解它们之间的关系，相信学习起来也很方便啦！



推荐文章：[细细品味C#——文件操作]

14.代码整洁之道

最近看完了经典之作：Code Clean，收获很多，分享一下我的感悟，主要是代码规范，设计方面的知识！

整洁代码只做一件事，糟糕的代码逻辑混乱，想做很多事，导致了阅读，修改困难！

14.1 整洁代码的基本规则：

①**通**过所有的单元测试

②**没**有重复代码，如果同一段代码出现了两次以上，那就是提醒你该提取相同代码进行重构了，***时刻提醒自己不要重复***

③**体**现系统设计的理念

④**有**意义的命名

14.2 注重代码的读与写，它们的比例是“10：1”

> 这事概无例外。不读周边代码的话就没法写代码。编写代码的难度，取决于读周边代码的难度。要想干得快，要想早点做完，要想轻松写代码，先让代码易读吧。

14.3 抽离try/catch代码快，这个非常赞同这个原则， 上图：

14.4 关于注释(书中讲到的注释规则让我很有同感，因为现在项目中就有这样的现象)

请注意：注释也许真的不需要，学会使用代码就能完整表达设计和逻辑意图！

常见现象：代码在变动，在演化，彼此分离和重合，可是注释并不总是随着变动，上图：

> 我为什么要极力贬低注释？因为注释会撒谎。也不是说总是如此或有意如此，但出现得实在太频繁。注释存在的时间越久，就离其所描述的代码越远，越来越变得全然错误。原因很简单。程序员不能坚持维护注释。
>
> 这是一个很重要的原则

所以只有代码才能真正的告诉你它在干什么，它有什么作用！

> 真实只在一处地方有：代码。只有代码能忠实地告诉你它做的事。那是唯一真正准确的信息来源。所以，尽管有时也需要注释，我们也该多花心思尽量减少注释量。

Note：*与其花时间编写解释你那糟糕代码的注释，还不如花时间清洁那糟糕的代码！*

坏注释和多余注释的几点原则：

①有时候一段坏的注释不紧会影响代码的整洁，而且还会占用一定的时间，最终读注释的时间比阅读代码的时间还长，所以这种注释要删除它，影响我们阅读代码的时间

②日志式注释：这种注释最有感触，在class开头写上每次修改的记录，这种方式也有好处，但是这种情况应该在没有源代码控制的情况下进行记录(其实我听赞同在class头上写上每次修改的版本的)

③关于废话性和误导性的注释坚决不能存在

④能用函数和变量是就别用注释，所以变量和函数的命名真的很重要，可以让人一眼就能看出它的作用

请明白非常重要的一点：***注释的作用就是解释未能自行解释的代码，如果注释本身还需要解释，就太遗憾了！***

14.5 单元测试的重要性

一直觉得单元测试可有可无，那是因为我只是学习过从来没有真正的在项目中运用过，可是最近我下了狠心要在项目中构建一个单元测试框架，终于被我搞定了，我也感悟到单元测试对一个开发人员的重要性，想学习的话可以看看这篇文章：走进单元测试五：单元测试文章系列目录

建议大家看看<<代码整洁之道>>和<<.NET设计规范>>以及<<程序猿修炼之道之单元测试>>

15.其它知识点

15.1 const和readonly本质区别

理解两者是在“编译时”还是“运行时”常量，以及两者的作用域，那么它们将不会这么神秘！

编译时OR运行时：

const：编译时

readonly：运行时

作用域：

const:①本身就是静态变量

②只能定义基本类型，如int，string等等

③局部变量和全局变量都可以定义

④一旦定义就不能修改

readonly:①不是静态变量，如果需要需加上“static”关键字

②可以定义一切类型，可以是自己自定义的对象

③只能定义全局变量

④一旦定义可以在构造函数里面进行初始化变量

总结：园子里面还有很多对于它们性能方面的文章，有兴趣的可以搜搜看，推荐使用“readonly”吧！

15.2 is和as操作符

16.2.1 As和强制转化最本质的区别

As：进行转换的时候永远不是出现异常，当转换失败时会返回一个“null”值，所以你只需要进行一个null值的判断就知道转换失败还是成功了！

强制转化：会出现转换失败并抛出异常，所以我们都需要使用“try/catch”来捕获转换出错的异常，也可以使用“is”来判断是否是你要转换的类型！

16.2.2 一些常见注意点

① as不能用于值类型的转化

如：object number=100;int numberOne = number as int;

这是因为如果转换失败，那么就会返回一个“null”值，但是值类型是不允许为“null”的，所以在语法上是行不通的，就是你写成了“int?”也是不行的！

② 使用Is配合强制转换来进行类型转换

首先使用“Is”来判断是否是我需要转换的类型，然后在进行强制转换

⊞

View Code

③ 在没有泛型的foreach中，也是把“object”进行强制转化成所需要的类型，代码如下：

⊞

View Code

Note：或者使用GetType()方法来精确检测是否是你想要的转换类型！

15.3 运算符操作以及类型转化操作重载

这两个知识点还是比较容易学习的，一个是操作符的重载，一个是用于自定义强制转换的方法(你也可以使用“as”进行强制转换)，只要稍加注意一些语法就好了！

```
 1     public class MyPerson
 2     {
 3         public string Name { get; set; }
 4 
 5         /*
 6          *  ①必须为静态
 7          *  ②关键字operator
 8          *  ③需要定义重载的操作符
 9          *  ④定义返回值类型
10          */
11         public static MyPerson operator +(MyPerson personFather, MyPerson
personMother)
12         {
13             return new MyPerson() { Name = personFather.Name +
personMother.Name };
14         }
15 
16         /*
17             ①必须为静态
18          *  ②关键字“explict”和operator”
19          *  ③需要转化的类型：MyPerson
20          */
21         public static explicit operator MyPerson(MyPeople myPeople)
22         {
23             return new MyPerson(){ Name="YCG" };
24         }
25     }
26     public class MyPeople
27     { }
```

具体的用法如下：

```
1   MyPerson personOne = new MyPerson() { Name = "AAAAAA" };
2   MyPerson personTwo = new MyPerson() { Name = "BBBBB" };
3   MyPerson personThree = personOne + personTwo;  //操作符重载
4   Console.WriteLine(personThree.Name);
5 
6   MyPeople people = new MyPeople() { Name = "wang wei" };
7   MyPerson personFour = (MyPerson)people; //类型强制转换
8   Console.WriteLine(personFour.Name);
```

15.4 ToString方法

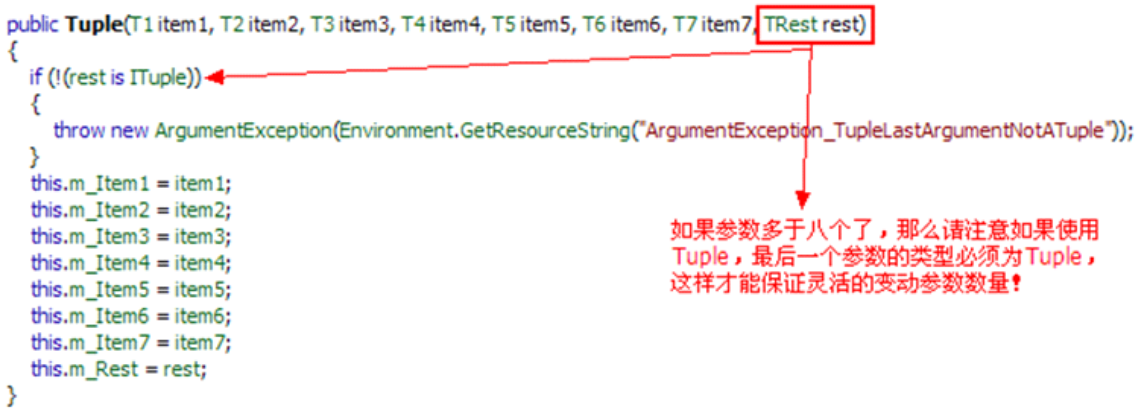
入口： [c# 扩展方法奇思妙用高级篇五：ToString(string format) 扩展]

15.5 数据实体模型(Tuple)以及匿名类型

16.5.1 Tuple实际上就是一个匿名的实体的模型，它的用处在于不要自己定义一个实实在在的Entity，使用它就能达到效果！

View Code

Note：在查看源代码的时候注意它的第八个参数：



16.5.2 匿名类型

```
1 var data = new { number = 11111, str = "ssssss" };  //不需要定义变量的类型，如果想知道匿名类型底层源码怎么写的，可以使用反编译查看源码，一目了然了！
2 Console.WriteLine(data.number + data.str);
```

15.6 String和StringBuilder详解

这个技术大妞们已经讨论的很多，我也没这个能力说的一清二楚，推荐几篇文章吧：1. 字符串的驻留（String Interning） 2. 深入理解string和如何高效地使用string 等等，实在很多啦...

17.Remoting

传送门：**.NET Remoting技术文章汇总** ，看过那些文章，对于Remoting的理解会有一个质的上升...

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5260111>

C#参考书的链接推荐

2016年5月2日 18:09

Visual C#.NET入门与提高
http://download.chinaitlab.com/soft/6330.htm

使用Visual C# 开发asp.NET入门
http://download.chinaitlab.com/soft/4114.htm

ASP.NET 基础教程-C#案例
版 http://download.chinaitlab.com/soft/9395.htm

Visual C#.NET入门与提高
http://download.chinaitlab.com/soft/6330.htm

使用Visual C# 开发asp.NET入门
http://download.chinaitlab.com/soft/4114.htm

ASP.NET 基础教程-C#案例版
http://download.chinaitlab.com/soft/9395.htm

ASP.NET C#教程http://download.chinaitlab.com/soft/6640.htm

C#程序设计案例教程
http://download.chinaitlab.com/soft/1776.htm

C#系列教程http://download.chinaitlab.com/soft/6099.htm

C#.NET编程培训教程
http://download.chinaitlab.com/soft/6331.htm

C#教程http://download.chinaitlab.com/soft/3070.htm

精通C#简体中文版http://download.chinaitlab.com/soft/6552.htm

C#语言规范（英文）
http://download.chinaitlab.com/soft/6375.htm

C#语言参考http://download.chinaitlab.com/soft/4814.htm

C# 程序员介绍(英文版)
http://download.chinaitlab.com/soft/6356.htm

C#入门经典http://download.chinaitlab.com/soft/6570.htm

ASP.Net 1.0入门经典C#篇
http://download.chinaitlab.com/soft/9396.htm

C#XML入门经典http://download.chinaitlab.com/soft/9393.htm

C#Web应用程序入门经典
http://download.chinaitlab.com/soft/9394.htm

C#编程从入门到精通
http://download.chinaitlab.com/soft/9392.htm

C#程序员参考手册http://download.chinaitlab.com/soft/6991.htm

C#类设计手册http://download.chinaitlab.com/soft/11825.htm

VB.NET and C#
Stephttp://download.chinaitlab.com/soft/7769.htm

C#文档中文版(微软)
http://download.chinaitlab.com/soft/1839.htm

C#.NET 开发者手册
http://download.chinaitlab.com/soft/7620.htm

C#进阶手册http://download.chinaitlab.com/soft/1596.htm

C#英文手册http://download.chinaitlab.com/soft/5929.htm

C# 完全手册http://download.chinaitlab.com/soft/6372.htm

C#语言参考手册http://download.chinaitlab.com/soft/5587.htm

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5260090>

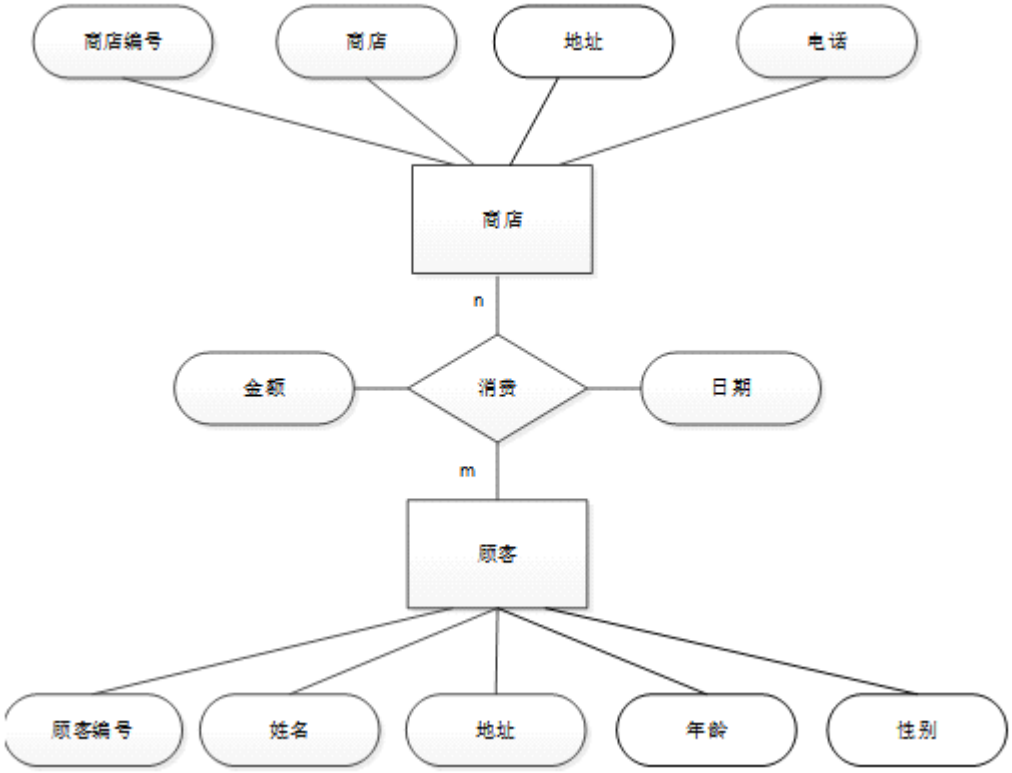
E-R图的基础练习

2016年5月2日　　18:09

第1题：

设有商店和顾客两个实体，“商店”有属性：商店编号、商店名、地址、电话，“顾客”有属性：顾客编号、姓名、地址、年龄、性别。假设一个商店有多个顾客购物，一个顾客可以到多个商店购物，顾客每次去商店购物有一个消费金额和日期，而且规定每个顾客在每个商店里每天最多消费一次。

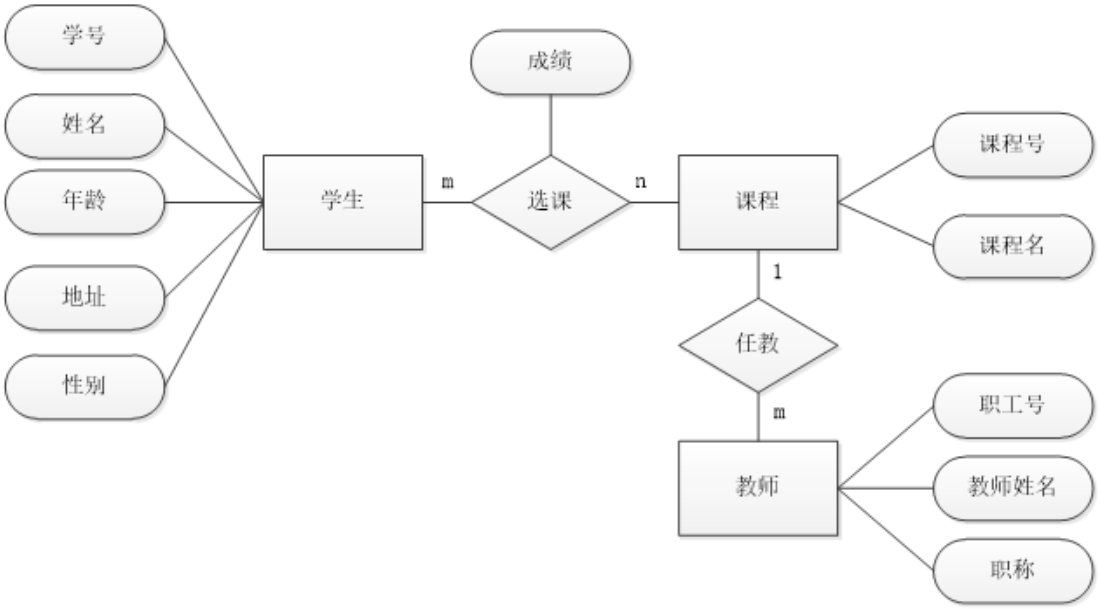
① 试画出ER图，并注明属性和联系类型。（5分）



第2题：

假设每个学生选修若干门课程，且每个学生每选一门课只有一个成绩，每个教师只担任一门课的教学,一门课由若干教师任教。“学生”有属性：学号、姓名、地址、年龄、性别。“教师”有属性：职工号、教师姓名、职称，“课程”有属性：课程号、课程名。

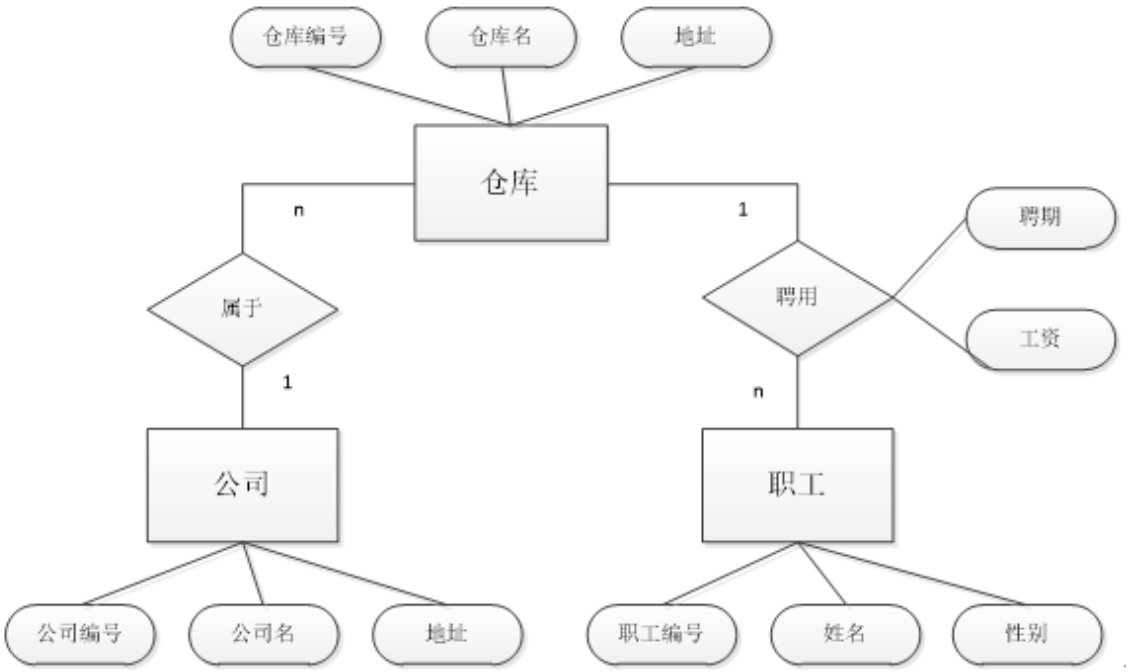
①试画出ER图，并注明属性和联系类型。（5分）



第3题：

设某商业公司数据库中有三个实体集，一是”公司”实体集，属性有公司编号、公司名、地址等；二是”仓库”实体集，属性有仓库编号、仓库名、地址等；三是”职工”实体集，属性有职工编号、姓名、性别等。每个公司有若干个仓库，每个仓库只能属于1个公司，每个仓库可聘用若干职工，每个职工只能在一个仓库工作，仓库聘用职工有聘期和工资。

①试画出E-R图 （5分）

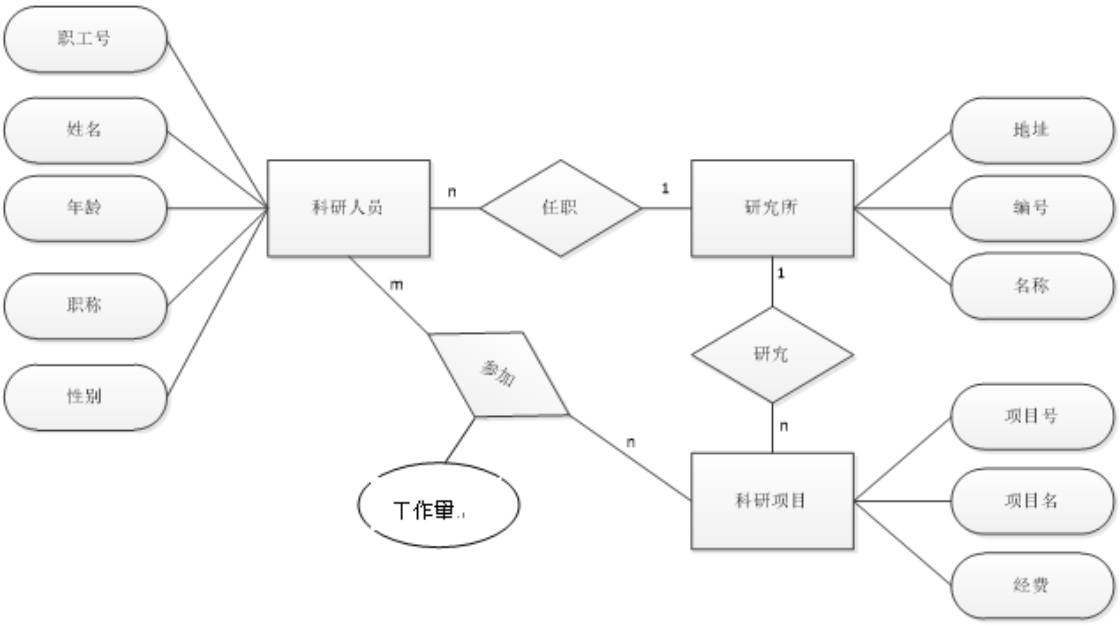


第4题：

某研究所有多名科研人员，每一个科研人员只属于一个研究所，研究所有多个科研项目，每个科研项目有多名科研人员参加，每个科研人员可以参加多个科研项目。科研人员参加项目要统计工作量。“研究所”有属性：编号，名称、地址，“科研人员”有属性：职工号、姓名、性别、年

龄，职称。“科研项目”有属性：项目号、项目名、经费。

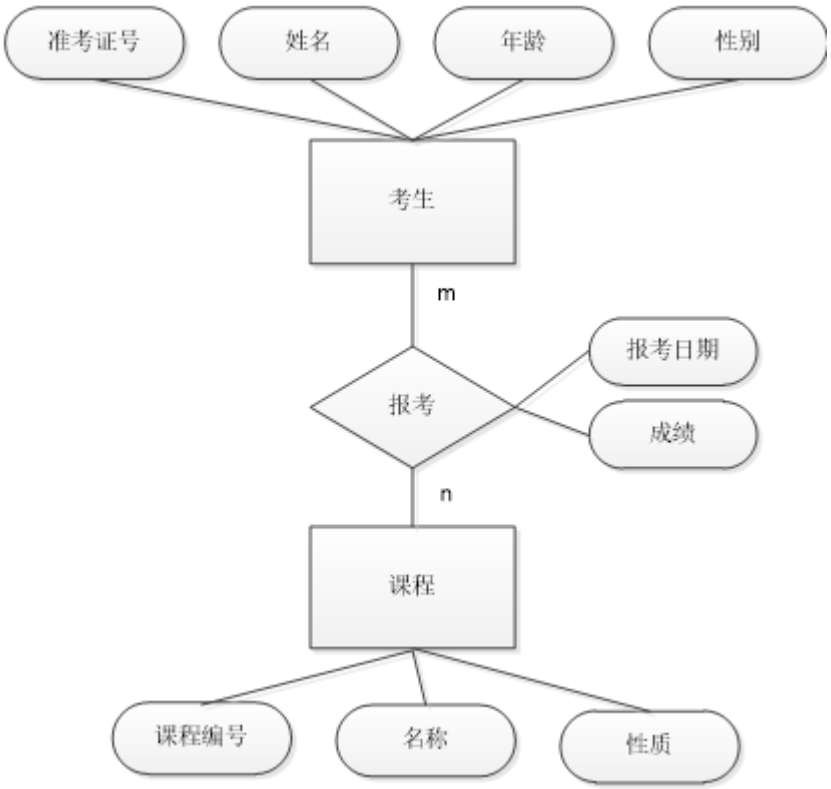
①试画出ER图，并注明属性和联系类型。（5分）



第5题：

现有学生报考系统，实体“考生”有属性：准考证号、姓名、年龄、性别，实体“课程”有属性：课程编号、名称、性质。一名考生可以报考多门课程，考生报考还有报考日期、成绩等信息。

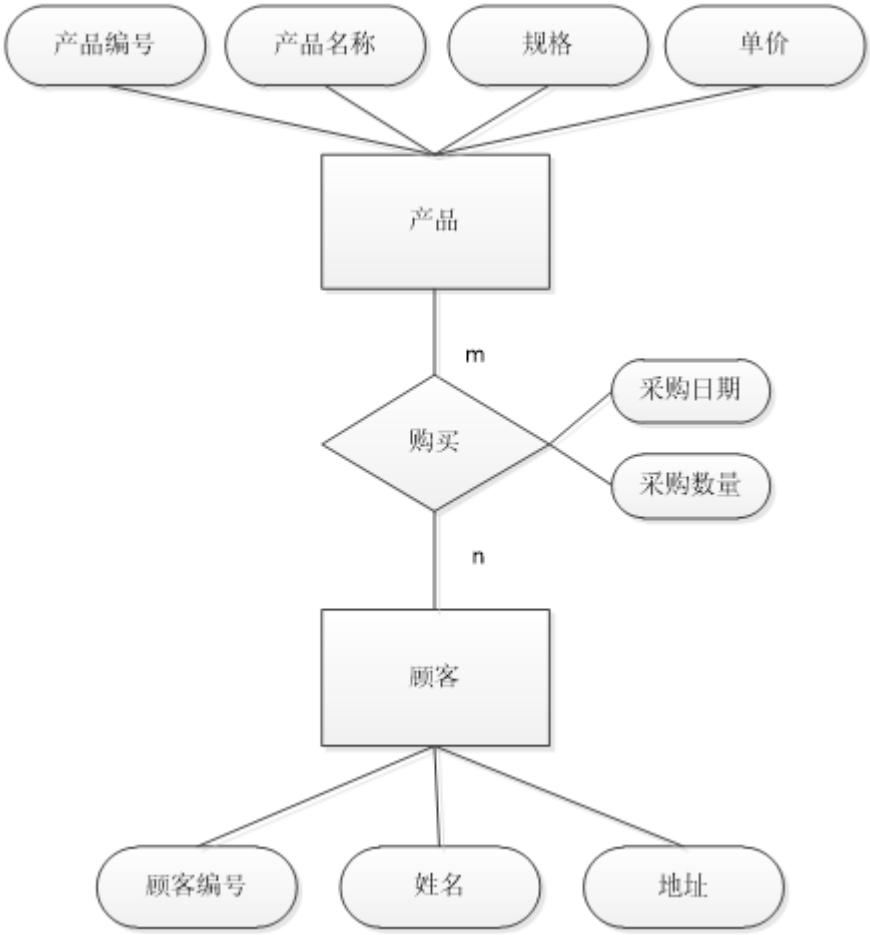
①画出ER图，并注明属性和联系类型。（5分）



第6题：

某厂销售管理系统，实体“产品”有属性：产品编号、产品名称、规格、单价，实体“顾客”有属性：顾客编号、姓名、地址。假设顾客每天最多采购一次，一次可以采购多种产品，顾客采购时还有采购日期、采购数量等信息。

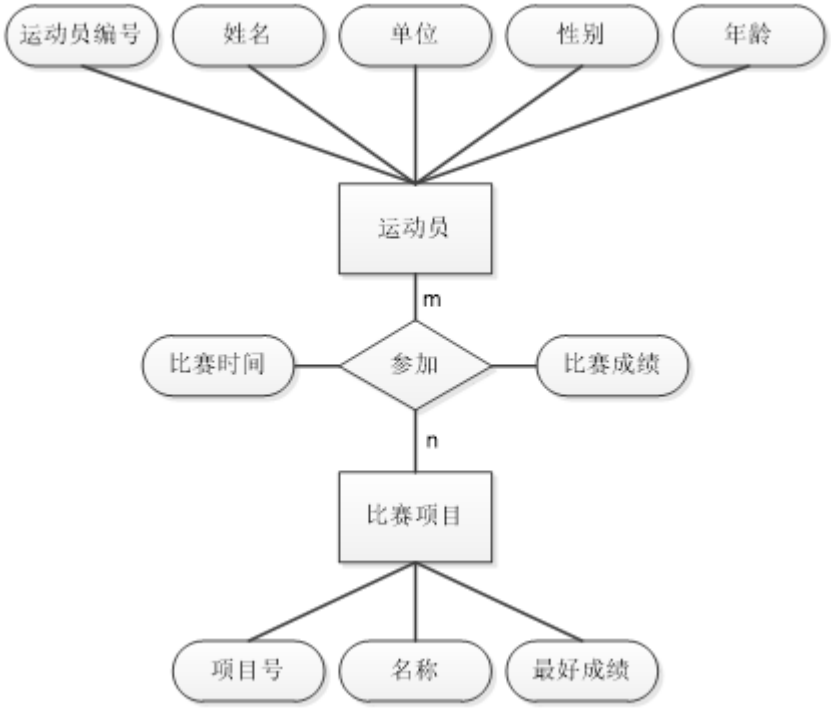
①试画出ER图，并注明属性和联系类型。（5分）



第7题：

设有运动员和比赛项目两个实体，“运动员”有属性：运动员编号、姓名、单位、性别、年龄，“比赛项目”有属性：项目号、名称、最好成绩。一个运动员可以参加多个项目，一个项目由多名运动员参加，运动员参赛还包括比赛时间、比赛成绩等信息。

①试画出ER图，并注明属性和联系类型。（5分）



第8题：

某工厂生产若干产品，每种产品由不同的零件组成，有的零件用在不同的产品上。这些零件由不同的原材料制成。不同的零件所用的材料可以相同。这些零件按所属的不同产品分别放在仓库中，原材料按类型放在若干仓库中。

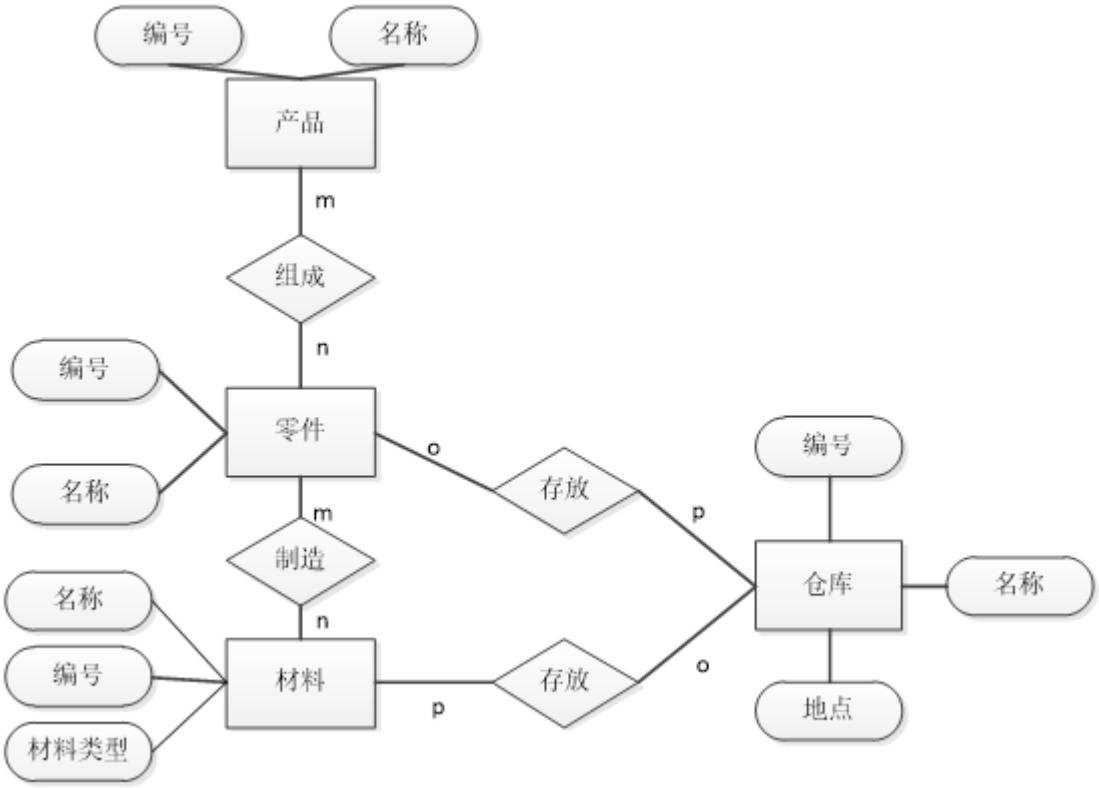
产品属性有：编号、名称

零件属性有：编号、名称

材料属性有：编号、名称、材料类型

仓库属性有：编号、名称、地点

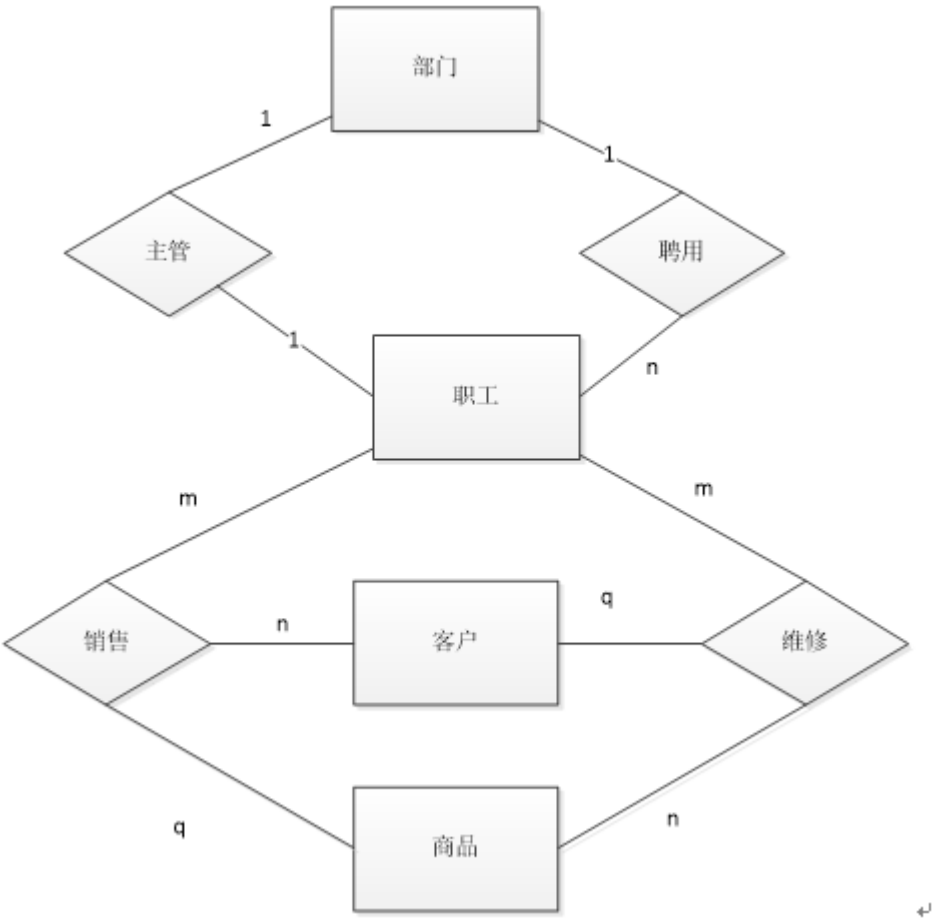
①请用E-R图画出工厂产品、零件、材料、仓库的概念模型，并注明属性和联系类型。（5分）



第9题：

某电脑公司是一家专门销售计算机整机、外围设备和零部件的公司。该公司有3个部门：市场部、技术部和财务部。市场部有18位业务员，负责采购和销售业务；技术部有14位工程师，负责售后服务、保修等技术性的工作；财务部有12位工作人员、1位会计和1位出纳，负责财务业务。公司需要将所有经营的计算机设备的客户、销售、维修（服务、保修）、职工等信息都存储在数据库中。

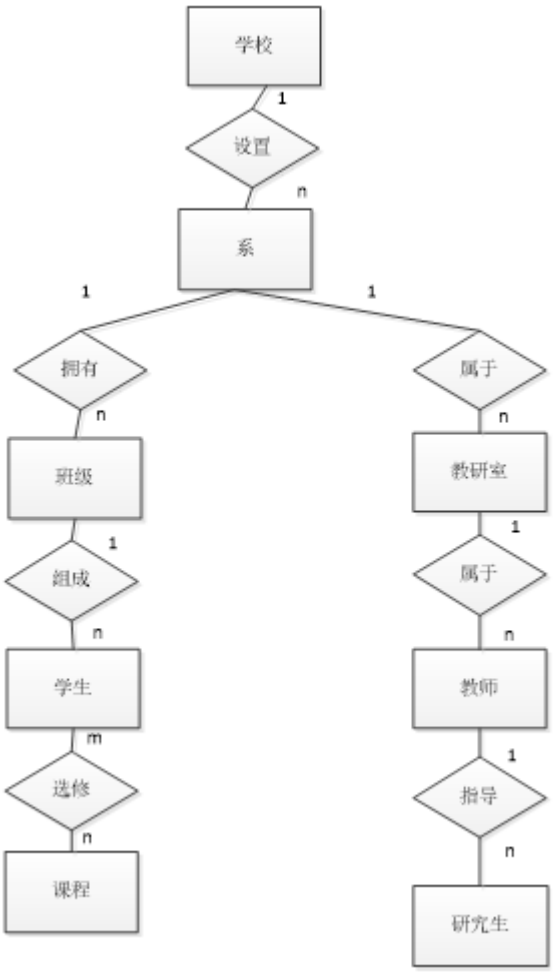
①根据公司的情况设计数据库的E-R图，并注联系类型。(5分)



第10题：

学校中有若干系，每个系有若干个班级和教研室，每个教研室有若干个教员，其中有的教授和副教授每人各带若干个研究生，每个班有若干学生，每个学生选修若干课程，每门课程可以有若干学生选修。

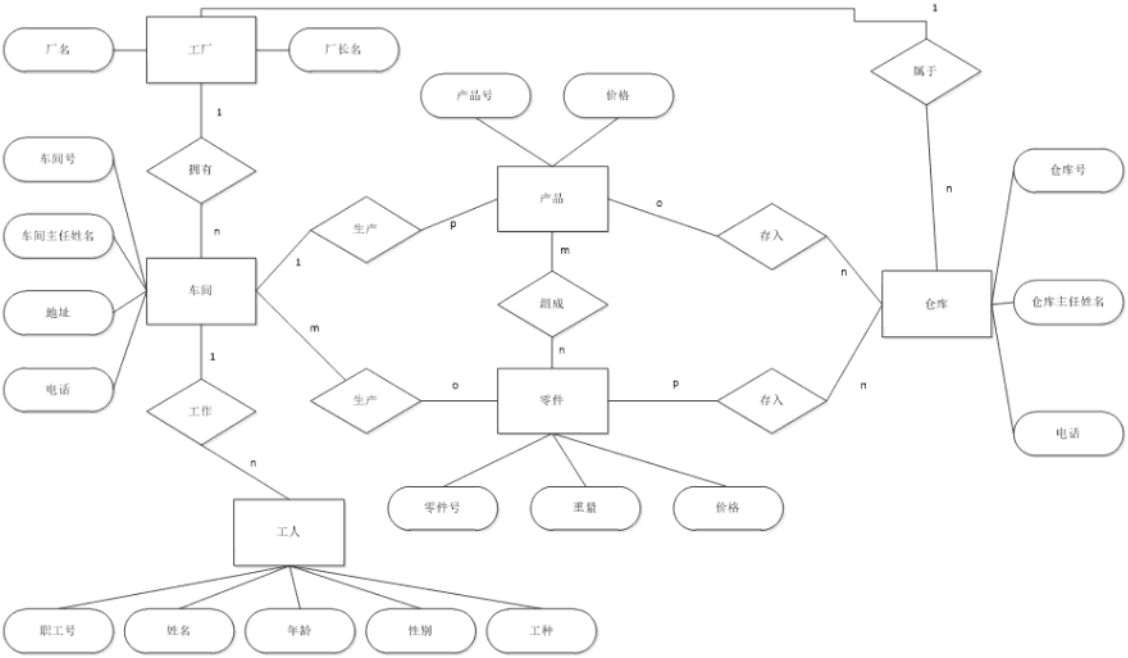
① 根据公司的情况设计数据库的E-R图，并注联系类型。(5分)



第11题：

工厂（包括厂名和厂长名）需要建立一个管理数据库存储以下信息：（1）一个厂内有多个车间，每个车间有车间号、车间主任姓名、地址和电话。（2）一个车间有多个工人，每个工人有职工号、姓名、年龄、性别和工种。3）一个车间生产多种产品，产品有产品号和价格。4）一个车间生产多种零件，一个零件也可能为多个车间制造。零件有零件号、重量和价格。（5）一个产品由多种零件组成，一种零件也可装配出多种产品。6）产品与零件均存入仓库中。7）厂内有多个仓库，仓库有仓库号、仓库主任姓名和电话。

（1）据工厂的情况，用E-R图画出概念模型，并注联系类型。(10分)



来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5256187>

转载文章----.NET 框架浅析

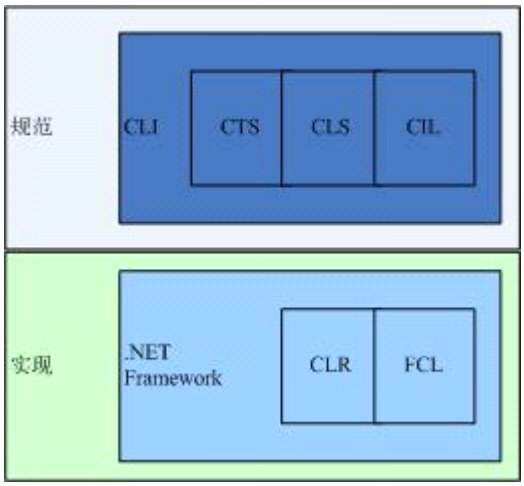
2016年5月2日　　18:09

转载地址：

http://www.cnblogs.com/yangmingming/archive/2010/01/27/1657850.html

.NET 框架概要：

.NET框架，即.NET FrameWork ,其本身作为.NET 技术两大方面之一。从狭义角度来讲，即通常所说的.NET框架。然而就.NET技术来讲，.NET技术可以以**规范**和**实现**两部分来划分：我们经常强调和提起的.NET FrameWork,主要包括**公共语言运行时**(Common Language RunTime,CLR) 和**.NET 框架类库**(FrameWork Class Library,FCL),它们本质上是.NET技术另一部分（规范）的**实现**部分；另一方面，即实现方面，我们称之为**公共语言架构**(Common Language Infrastructure,CLI),主要包括**通用类型系统**(Common Type System,CTS)、**公共语言规范**(Common Language Specification,CLS)以及**通用中间语言**(Common Intermediate Language,CIL)三部分。



上图即很形象了描述了.NET 技术的两大方面。

术语解释：

CLI:公共语言架构，.NET技术规范，已经得到ECMA批准；

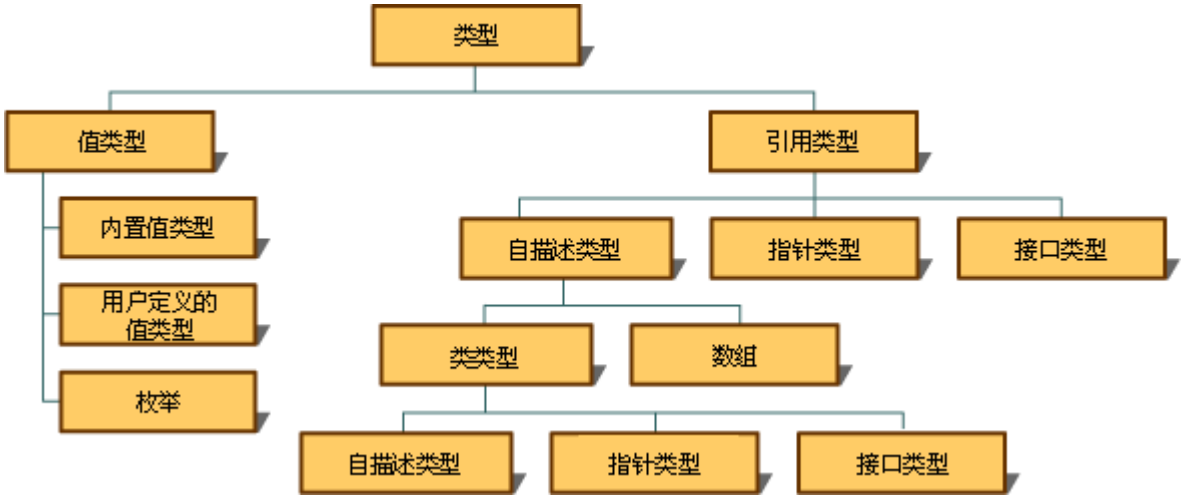
CLS:公共语言规范，CTS的子集，是进行.NET开发所使用语言的最小集合；

CIL:通用中间语言，可以认为是.NET编译后的中间代码；

CLR:.NET框架的核心，管理代码的执行，以及几乎一切的服务控制；

FCL:.NET框架类库,形成以system为根的树状组织结构。其中在各命名空间中包含数以万计的类可供使用；

CTS：通用类型语言，其包括两种基本类型：**值型**和**引用型**。每种类型又可细分为子类型，如下图可诠释：



综述之，.NET 框架基本构成如前所述。然而其各构成部分的相互关系，以及各部分的深化部分，没有实现（例如CTS类型的关系、FCL的深化理解）。这还需要在此基础上进一步巩固、加强。

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5251852>

转载文章----IL反编译利器——Ildasm.exe和Reflector.exe:

2016年5月2日　　18:10

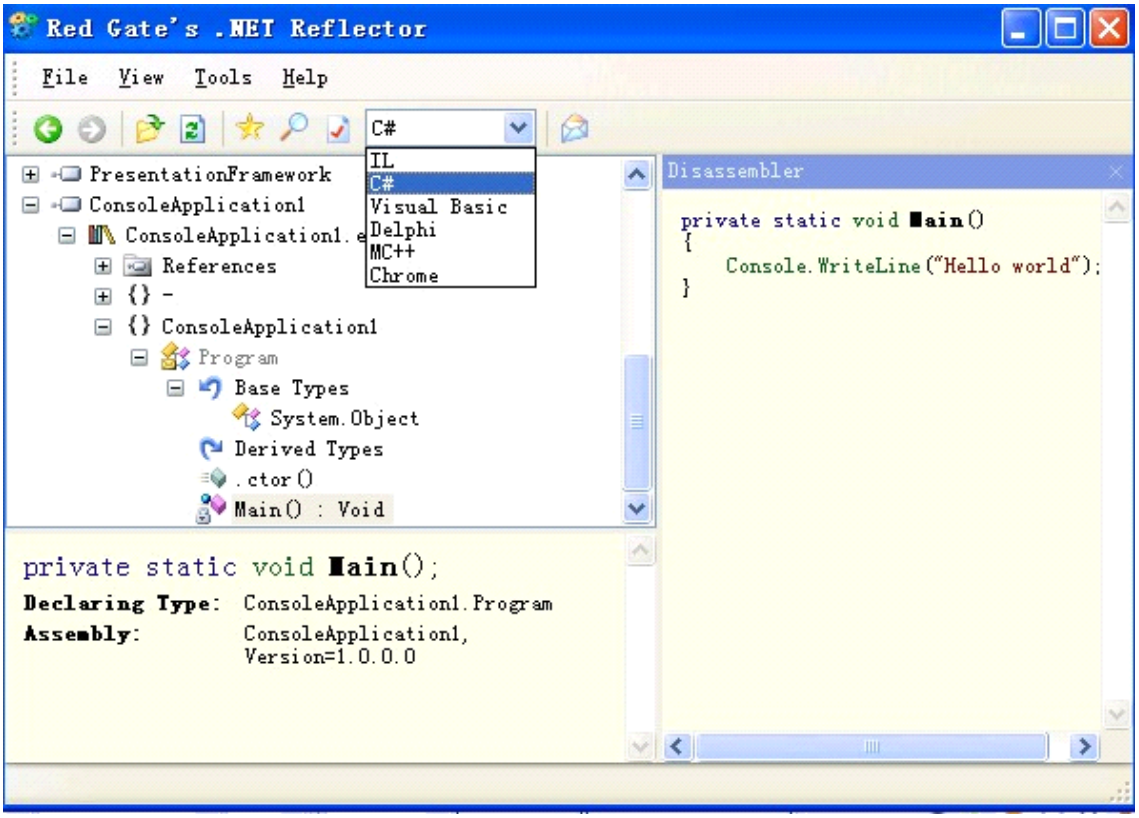
转载地址：
http://www.cnblogs.com/yangmingming/archive/2010/02/03/1662546.html

一：Ildasm.exe简介

这一微软VS自带工具，在上一篇博文《初识Ildasm.exe——IL反编译的实用工具》中已经做了详细介绍，这里不再鳌述；

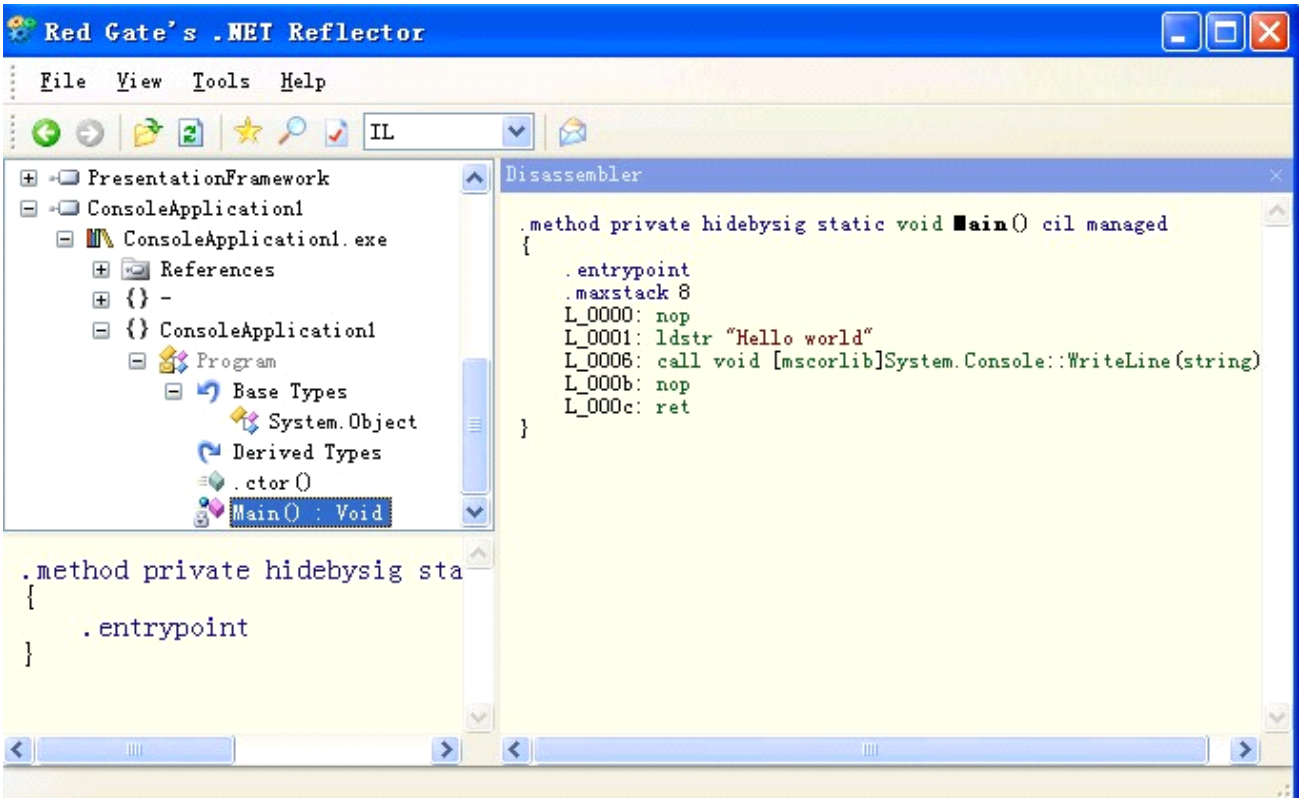
二：Reflector.exe简介

至此（10.02.03），从 http://www.aisto.com/roeder/dotnet 下载的最新版本为Latest version: 5.1.6.0 ，这里还以上篇中经典的"Hello World."示例演示，当用Reflector.ext打开可执行文件时，主界面如下：



可见Reflect.exe的功能要强于Ildasm.exe，它可以反编译IL、C#、Visual Basic、Delphi、MC++、甚至是Chrome，而且它的界面更加友好，所以他被冠称为深入IL的**最佳利器！**

我们先看它所反编译的IL代码是何模样？是否同于Ildasm.exe的效果？图示如下：



比较之后，可得Reflector.exe具有更容易理解的界面（没有Ildasm.exe中众多的标识符，见上篇博文的**MSDN帮助图**）。

当然，之所以称 Reflector.exe为深入IL的**最佳利器，是因为**它可以反编译IL语言，以及C#语言！这样我们：**可以在完全没有源代码的情况下研究所使用的程序集！！！**示例图见上上图，所示，我们发现果然编译为了源代码，且完全正确，这太令人惊讶了，而它的作用必定是意义深远的！

用IL揭示接口的本质：

一：接口本质

我们先看一个普通的接口定义：如图示：

```
interface IMyInterface
{
    void MyMethod();
}
```

而用Reflector.exe反编译的IL代码为：

```
.class private interface abstract auto ansi IMyInterface
{
   .method public hidebysig newslot abstract virtual instance void MyMethod() cil managed
   {
   }
}
```

可见接口**实质上**，被编译为**抽象类**，其中接口中的方法，被编译为**抽象方法**（abstract virtual），没有实现体。

二：接口的抽象实现原理

先来看最基本的类继承接口的示例，如下图：

```
class MyClass : IMyInterface
{
   void IMyInterface.MyMethod()
   {
   }
}
```

而其对应的IL代码，用Reflector.exe编译为：

```
.class private auto ansi beforefieldinit MyClass
   extends [mscorlib]System.Object
   implements InsideDotNet.OOThink.Interface.IMyInterface
{
   .method public hidebysig specialname rtspecialname instance void .ctor() cil managed
   {
   }
   .method private hidebysig newslot virtual final instance void InsideDotNet.OOThink.Interface.IMyI
nterface.MyMethod() cil managed
   {
      .override InsideDotNet.OOThink.Interface.IMyInterface::MyMethod
   }
}
```

这样可以看到该类中，对应接口的函数，在这里实现了**"覆盖"——Override！**因此接口的抽象机制，采用**多态**来实现的！

综述之，通过对Reflect.exe的实例应用，了解了更多的IL反编译工具。同时通过IL代码，实现了对接口本质的揭示，认识到其在保持接口**特有性**（这里不铺开论述。）的基础上， 用抽象类的实现机制及其多态性实现抽象的。

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5251795>