泛型

框架搭建 集合中 应用很多

在C#中 object类是所有类型的基类 子类类型可以被当做父类类型

所以在定义方法的时候 遇到可能出现的多种类型参数可以使用object类型来定义参数列表

但是 频繁的拆箱和装箱会对性能带来损耗 影响效率

解决这个问题必然需要使用到泛型

泛型方法

泛型方法的定义和调用

```
        static void Main(string[] args)
        {
            Fun<int>(100);

            string str = "helloworld";
            Fun<string>(str);

            Item item = new Item();
            Fun<Item>(item);
        }

        static void Fun<T>(T x)
        {
            //打印类型的名字
            Console.WriteLine(x.GetType());
            Console.WriteLine(typeof(T).Name);
            Console.WriteLine(x.ToString());
        }
```

多个泛型同时存在

```
         static void Main(string[] args)
        {
            Fun<int,string>(100,"game");
```

```
            string str = "helloworld";
            Fun<string,double>(str,5.2);

            Item item = new Item();
            Fun<Item,bool>(item,true);
        }

        static void Fun<T1,T2>(T1 x,T2 y)
        {
            //打印类型的名字
            Console.WriteLine(x.GetType());
            Console.WriteLine(typeof(T1).Name);
            Console.WriteLine(y.GetType());
            Console.WriteLine(typeof(T2).Name);
            Console.WriteLine(x.ToString());
            Console.WriteLine(y.ToString());
        }
```

泛型是延迟声明

在定义方法时 没有指定具体的参数类型 而是把参数类型的声明推迟到了调用时才指定

底层泛型的编译流程：

VS编译器编译源代码->遇到泛型生成占位符->运行时经过JIT编译器->把占位符替换成具体类型

方法性能的问题

由高到低

泛型方法->普通方法->object方法(包含拆箱装箱的)

where修饰符 泛型约束

```
        static void Main(string[] args)
```

```
        {
            Item item = new Item();
            Fun<Item>(item);

            //Fun<int>(200); 编译报错 泛型方法被约束成 只能传递Item类型或其子类类型
        }

        static void Fun<T>(T x) where T:Item
        {

        }
```

泛型约束	效果
where T:struct	必须是值类型
where T:class	必须是引用类型
where T:new()	必须有默认的构造函数
where T:Item	必须是Item或Item的子类类型
where T:接口名	必须是某个接口或者接口的子类类型

多约束

```
        static void Fun<T,T2>(T x,T2 y) where T:Item,new() where T2:class,new()
        {
            x = new T();
        }
```

泛型类

泛型除了可以表示方法 还可以表示类Class和interFace接口

```
public class Item<T1,T2>
{
    public T1 t1;
    public T2 t2;
}

public class ClassA : Item<int, float>
{
}
public class ClassB : Item<float, string>
{
}

static void Main(string[] args)
{
     Item<int, string> item = new Item<int, string>();
     Item<string, string> item2 = new Item<string, string>();
     Item<float, int> item3 = new Item<float, int>();
}
```

集合

非泛型集合

ArrayList 动态数组 实现了不固定长度的数组

```
    static void Main(string[] args)
    {
        //创建动态数组
        ArrayList list = new ArrayList();
        //添加元素的方法
        list.Add(true);
        list.Add(100);
        list.Add("helloworld");
        //移除所有元素
        list.Clear();

        //整体添加一个数组
        list.AddRange(new int[] { 20, 50, 90 });
```

```
            //整体添加另一个动态数组
            list.AddRange(list);

            //移除某个单个元素
            list.Remove(50);
            //移除索引值为2的元素
            list.RemoveAt(2);
            //移除范围内的所有元素 起始索引+个数
            list.RemoveRange(3, 1);

            //插入
            list.Insert(1, "插入的数据");

            if (!list.Contains("game"))
            {
                list.Add("game");
                Console.WriteLine(list.Contains("game"));
            }

            //数组的遍历
            for (int i = 0; i < list.Count; i++)
            {
                Console.WriteLine(list[i]);
            }

            //foreach遍历
            foreach(var a in list)
            {
                Console.WriteLine(a);
            }

            list.Reverse();//反转
            list.Sort();//升序排序
        }
```

Hashtable 哈希表

键值 key = value

```
            Hashtable hash = new Hashtable();

            //添加数据
            hash.Add("g", "game");
            hash.Add("x", "xyz");
            hash.Add("h", "helloworld");
            hash.Add("t", "ttt");
            // hash.Add("t", "www");不同出现重复的键 但可以有重复的值

            //通过键获取值
            Console.WriteLine(hash["g"]);
            //通过键修改值
            hash["x"] = "abc";
            Console.WriteLine(hash["x"]);

            //是否包含某个key
            if (!hash.ContainsKey("w"))
            {
                hash.Add("w", "www");
                Console.WriteLine(hash["w"]);
            }

            //使用foreach遍历所有的键
            foreach(string k in hash.Keys)
            {
                Console.WriteLine(k);
            }

            foreach (string v in hash.Values)
            {
                Console.WriteLine(v);
            }

            foreach(DictionaryEntry data in hash)
            {
                Console.WriteLine(data.Key);
                Console.WriteLine(data.Value);
            }
```

不管是ArrayList还是Hashtable 都将所有的元素看成了object类型 引用类型

使用Add添加元素时 如果是值类型元素 实际上都进行一个装箱的操作 而在获取数据时又进行了拆箱

装箱：

1 在托管堆里分配一个object

2 本来的值类型数据在栈中 在装箱时实际上了移动到了托管堆中的这块新内存

拆箱：

1 当进行拆箱的时候 堆中的数据需要移动到栈中

2 堆中的无用数据要进行垃圾回收

所以 装箱和拆箱过程操作简单 但是如果涉及大量的操作 必然会影响程序的性能

类型安全问题

```
 ArrayList list = new ArrayList();
 list.Add(1);//装箱
 list.Add(true);//装箱
 list.Add(20.2);//装箱

foreach(object it in list)
{
    //编译不出错 但是运行报错
    //使用ArrayList 进行类型转换时 即使类型转换不正确也不会提示
    Console.WriteLine((int)it);
}
```

泛型集合 List

特点：

1 可以通过索引访问集合中的元素

2 是ArrayList的泛型等效类

3 可以接受null类型

4 允许出现重复的元素

5 无需拆装箱 类型安全

```
            List<string> data = new List<string>();
            data.Add("apple");
            data.Add("orange");
            data.Add("banana");
            data.Add("grape");

            //data.AddRange(data);

            //使用索引访问元素
            Console.WriteLine(data[1]);
            //根据元素获取索引
            Console.WriteLine(data.IndexOf("apple"));

            //使用for循环遍历列表
            for (int i = 0; i < data.Count; i++)
            {
                Console.WriteLine(data[i]);
            }
            data.Sort();//升序排序

            //foreach遍历列表
            foreach (var it in data)
            {
                Console.WriteLine(it);
            }
            //反转
            data.Reverse();

            data.Remove("banana");
            data.RemoveAt(1);
            data.Clear();
```

练习：

一个整数数组 把数组中的奇数和偶数分别存储在不同的list集合中 再把两个list合并成一个list

int[] zheng={1,2,3,4,5,,6,7,8,9}

```
            int[] zheng = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
            List<int> list1=new List<int>();
            List<int> list2 = new List<int>();

            foreach (int data in zheng)
            {
                if (data % 2 == 0)
                    list1.Add(data);
                else
                    list2.Add(data);
            }

            list1.AddRange(list2);

            foreach (var a in list1)
            {
                Console.WriteLine(a);
            }
```

字典

Dictionary<T1,T2> 集合

特点：

从一组键（key）和一组值（value）的映射

任何键都必须是唯一的

键不能为空null　值如果是引用类型 可以为Null

key和value可以是任何类型(class struct等)

```
        static void Main(string[] args)
        {
            Dictionary<int, string> dic = new Dictionary<int, string>
();

```

```
            dic.Add(100, "game");
            dic.Add(200, "Hello");
            dic.Add(300, "C#");
            dic.Add(400, "Lua");

            //通过键获取值
            Console.WriteLine(dic[100]);

            //通过键修改值
            dic[100] = "C++";

            dic.Remove(200);

            //字典大小
            Console.WriteLine(dic.Count);

            Console.WriteLine(dic.ContainsKey(500));
            Console.WriteLine(dic.ContainsValue("C#"));

            string v;
            //尝试 使用键获取值
            Console.WriteLine(dic.TryGetValue(200, out v));
            Console.WriteLine(v);

            //遍历所有的key
            foreach (var k in dic.Keys)
            {
                Console.WriteLine(k);
            }

            foreach (var v1 in dic.Values)
            {
                Console.WriteLine(v1);
            }

            //遍历所有的字典元素 key和value
            foreach(KeyValuePair<int,string> data in dic)
            {
                Console.WriteLine(data.Key);
                Console.WriteLine(data.Value);
            }
            dic.Reverse();
```

```
            dic.Clear();
        }
```

字典 用到了两个占位符 k 和 v

在定义集合时 声明了元素的实际键值类型 保证了类型安全

字典和哈希表 对元素的操作方法相似 添加 获取 删除等方法基本相同

不同点：

在字典中进行操作无需拆箱和装箱 但是不能像哈希表一样可以添加任意类型

练习：

“Welcome to hina Haha”

打印字符串中每种字母出现的次数 区分大小写

结果: W 1 e 2

```
            string str = "Welcome to China Haha sfsda";

            Dictionary<char, int> dic = new Dictionary<char, int>();

            for (int i = 0; i < str.Length; i++)
            {
                if (str[i] >= 'a' && str[i] <= 'z' || str[i] >= 'A' && str[i] <= 'Z')
                {
                    if (!dic.ContainsKey(str[i]))
                        dic.Add(str[i], 1);
                    else
                        dic[str[i]]++;
                }
            }

            foreach (var it in dic)
            {
                Console.WriteLine(it.Key + " " + it.Value);
            }
```

作业：

把1234 转为 壹贰仨肆

例如：

控制台输入 1 输出壹

控制台输入 3 输出仨

"1=壹|2=贰|3=仨|..."