C#进阶笔记

简单数据结构类——放Object

- Arraylist
  - 定义：是一个C#封装好的类，本质上是可以自动扩容的Object数组，万物之父的数组
  - 声明：new一个，实例化对象
  - 操作
    - 增：.Add(object)/.AddRange(Arraylist)/Insert(int,object) 在索引位置插入元素
    - 删：.Remove(object) 正向查找并移除元素/.RemoveAt(int)/.Clear()
    - 查：数组名[index]/.Contains(object) 判断是否存在，返回一个bool值/.IndexOf(object) 正向查找并返回索引/.LastIndexOf(object) 反向查找并返回索引
    - 改：直接赋值
  - 遍历
    - 长度：Count；容量：Capacity——避免产生过多的垃圾
    - 常规遍历：for循环
    - 迭代器遍历：foreach循环
  - 装箱拆箱
    - 装箱：值类型->引用类型，栈内存->堆内存
    - 拆箱：引用类型->值类型，从object到其他类型要强转，堆内存->栈内存
- Stack
  - 定义：C#封装好的类，本质是Object数组，只是封装了特殊的存储规则——先进后出
  - 声明：new一个，实例化对象
  - 操作：
    - 压栈：.Push(object)
    - 弹栈：.Pop() 返回栈顶的object
    - 查：.Peek() 不弹栈只是看一下，栈无法查看指定位置的元素，只能查看栈顶元素；.Contains(object) 查询某个元素是否存在，返回一个bool值
    - 改：栈无法改变指定位置的元素
    - 清空：.Clear()
  - 遍历：
    - 长度：.Count
    - 常规遍历：**由于栈没有提供索引器**，因此无法用for循环遍历；将栈转换为object数组.ToArray() 顺序是从栈顶到栈底
    - 迭代器遍历：foreach循环，**顺序是从栈顶到栈底**
    - 循环弹栈：while循环，条件是Count>0
- Queue
  - 定义：C#封装好的类，本质是Object数组，只是封装了特殊的存储规则——先进先出
  - 声明：new一个，实例化对象
  - 操作
    - 增：.Enqueue(object)
    - 取：.Dequeue() 取出队列的第一个object
    - 查：.Peek() 查看队首元素但不取出；.Contains(object) 查询元素是否存在，返回

        - 改：由于队列没有提供索引器，因此无法修改某个指定元素
        - 清空：.Clear()
    - 遍历
        - 长度：.Count
        - 常规遍历：**由于队列没有提供索引器**，因此无法用for循环遍历；将队列转换为object数组.ToArray() 顺序是从队首到队尾
        - 迭代器遍历：foreach循环，**顺序是从队首到队尾**
        - 循环出队：while循环，条件是Count>0
- Hashtable
    - 定义：又称散列表，是哈希代码组织起来的键值对，主要作用是提高数据查询的效率
    - 声明：new一个，实例化对象
    - 操作
        - 增：.Add(object,object) 键和值可以是任意类型，但不能有相同的键
        - 删：.Remove(object) 只能通过键来删除键值对，删除不存在的键没反应
        - 查：用索引器[object]查询，通过键来查找值，若键不存在就返回null；.Contains(object) 通过键查询是否存在键值对，返回bool；.ContainsKey(object)和上面的方法一样；.ContainsValue(object) 通过值查询是否存在键值对，返回bool
        - 改：通过直接赋值修改键对应的值内容，但无法修改键
        - 清空：.Clear()
    - 遍历
        - 长度：.Count 键值对的数目
        - 遍历所有键：foreach(var item in hashTable.Keys)
        - 遍历所有值：foreach(var item in hashTable.Values)
        - 键值对一起遍历：foreach(DictionaryEntry item in hashTable) item是一个结构体，.key和.value分别表示它的键和值，不常用
        - 迭代器遍历：IDictionaryEnumerator接口

泛型

- 定义：泛型实现了**类型参数化**，通过类型参数化来实现同一份代码上操作多种类型；泛型相当于类型占位符，使用替代符代表变量类型，当真正使用类或方法时再具体指定类型；**泛型占位字母可以有多个**，用逗号隔开
- 泛型类
    - class 类名
- 泛型接口
    - interface 接口名
- 泛型函数
    - 函数名()
- 作用
    - **不同类型对象的相同逻辑处理**可以用泛型
    - 使用泛型可以一定程度**避免装箱拆箱**

泛型约束

- 定义：让泛型的类型有所限制
- 分类——6种，关键字where；泛型约束可以组合使用（可能会报错）
    - 值类型 where T: struct
    - 引用类型 where T: class
    - 有无参public构造函数的值类型/非抽象类 where T: new()
    - 某个类本身或其派生类 where T: 类名
    - 某个接口的派生类型 where T: 接口名
    - 另一个泛型类型本身或者派生类型 where T: 另一个泛型字母

常用泛型数据结构类

- List
    - 定义：是一个C#为我们封装好的类，本质是一个**可变类型的泛型数组**
    - 声明：List<类型名> 列表名 = new List<类型名>()
    - 操作：增删查改的API和Arraylist一样；遍历操作的方法和Arraylist一样
- Dictionary
    - 定义：可以理解为拥有泛型的Hashtable，它也是基于哈希代码组织起来的键值对
    - 声明：Dictionary<键类型名,值类型名> 字典名 = new Dictionary<键类型名,值类型名>();
    - 操作：增删查改的API和Hashtable一样；遍历操作的方法和Hashtable一样；键值对一起遍历要用KeyValuePair<键类型名,值类型名>泛型
- LinkedList
    - 定义：是一个C#封装好的类，本质是一个可变类型的泛型双向链表
    - 声明
        - 链表类：LinkedList<类型名> linkedList = LinkedList<类型名>();
        - 节点类：LinkedListNode<类型名> linkedList = LinkedListNode<类型名>();
    - 操作
        - 增：.AddFirst(元素) 在头部加元素；.AddLast(元素) 在尾部加元素；.AddAfter(元素,元素) 在指定元素后面插入节点；.AddBefore(元素,元素) 在指定元素前面插入节点
        - 删：.RemoveFirst() 删除第一个节点；.RemoveLast() 删除最后一个节点；.Remove(元素) 正向查找元素值并删除；.Clear() 清空
        - 查：.First 头部元素；.Last 尾部元素；.Find(元素) 正向查找并返回该元素，没有就返回null；.Contains(元素) 查询元素是否存在，返回bool
        - 改：.Value直接赋值
    - 遍历
        - 迭代器遍历：foreach
        - 节点遍历：while循环，临时节点每一次指向当前节点的next节点
            - 从头到尾
            - 从尾到头
- 泛型栈
    - 定义：是一个C#为我们封装好的类，本质是一个可变类型的泛型栈
    - 声明：Stack<类型名> stack = new Stack<类型名>()
    - 操作：API和Stack一样
- 泛型队列
    - 定义：是一个C#为我们封装好的类，本质是一个可变类型的泛型队列
    - 声明：Queue<类型名> queue = new Queue<类型名>()

- 操作：API和Stack一样

顺序存储和链式存储

- 顺序存储：用一组**地址连续**的存储单元依次存储线性表的各个元素
- 链式存储：用一组**任意的存储单元**存储线性表种的各个数据元素
- 顺序存储和链式存储的比较：
  - 增和删：链式存储计算上优于顺序存储
  - 查和改：顺序存储使用上优于链式存储

委托

- 定义
  - **委托是函数/方法的容器**
  - 可以理解为表示函数/方法的变量类型
  - 用来存储、传递方法
  - **委托的本质是一个类，用来定义方法返回值和参数的类型**
  - 不同的方法必须对应和各自“格式”一致的委托
- 语法
  - 关键字：delegate
  - 声明：访问修饰符 delegate 返回值 委托名(参数列表);
  - 位置：一般在namespace中，同一命名空间中的委托不能重名
  - 实例化：①委托名 对象 = new 委托名(函数名); ②委托名 对象 = 函数名; 两种写法，装载的函数返回值和参数要符合委托所定义的
  - 调用：.Invoke(参数); (参数);
- 使用
  - 作为类的成员
  - 作为函数的参数，自定义逻辑处理的顺序
  - 委托变量可以存储多个函数，**+= 函数名**，调用该委托时每一个函数都会依次被调用
  - 委托变量也可以删除函数，**-= 函数名**；若找不到要减去的函数则不处理；=null就是清空委托容器
- 系统自带的定义好的委托——Action无返回值Func有返回值
  - Action 无参无返回值
  - Func<类型名> 无参自定义返回值类型的**泛型委托**
  - Action<类型名,类型名,类型名...> **自定义参数类型和个数**且无返回值的泛型委托
  - Func<类型名,类型名,类型名...,类型名> **自定义参数类型和个数**且**自定义返回值类型**的泛型委托

事件

- 定义
  - 是基于委托的存在
  - **是委托的安全包裹**，让委托的使用更具有安全性
  - 是一种特殊的变量类型
- 使用
  - 声明：访问修饰符 event 委托类型 事件名; 相当于在委托的声明前面加了一个event关键字
  - **事件的使用和委托的使用方法几乎一样**
    - 调用方法

        - 增加和减少方法
        - 置空
    - 事件和委托的区别
        - 事件不能在类外部赋值；但是可以增加、减少方法，且只能用复合运算符
        - 事件不能在类外部调用
        - 事件只能作为成员存在于类、接口或结构体中
- 作用
    - 防止外部随意置空委托
    - 防止外部随意调用委托
    - 对委托进行封装，让委托更安全

匿名函数

- 定义
    - 没有名字的函数
    - 配合委托和事件使用，脱离委托和事件将无法使用
    - **声明匿名函数后必须用委托或事件容器去接收**
    - 若匿名函数有返回值，**不需要在声明时写出返回值**
- 语法
    - delegate (参数列表){ // 函数体 }
- 使用
    - 作为**参数**：函数中传递委托参数（实参）时，直接在参数列表中写出匿名函数
    - 作为**返回值**：作为委托或事件类型的返回值时，直接在return后写出匿名函数
- 缺点：由于没有名字，添加到委托或事件容器后无法单独删除

Lambda表达式

- 定义
    - 匿名函数的简写
    - 使用上和匿名函数一模一样
    - 需要配合委托和事件使用，脱离了委托和事件将无法使用
- 语法
    - (参数列表) => { // 函数体 };
    - 参数列表的参数类型可以省略，参数类型和委托/事件容器一致
- 使用
    - 使用方法和匿名函数一样
    - 为了方便和简写
- 闭包
    - 内层函数可以引用它外层函数的变量，即使外层函数的执行已经终止
    - **该变量的声明周期被改变**
    - 该变量的值并非变量创建时的值，而是在外层函数范围内的最终值

List排序

- 自带的排序方法
    - .Sort() // 默认升序，可选参数填false就可以降序
- 自定义排序
    - 要继承IComparable泛型接口

    - 排序规则：和other比较，排在右边的return正整数，排在左边的return负整数
    - 三目运算符可以用一行代码解决
- 通过委托函数进行排序——比上面更方便，代码更少
    - .Sort(委托名) F12进去发现Sort的其中一个重载的参数可以是委托
    - 委托格式：public delegate int Comparison(T x, T y); 可以看出返回值是int，参数类型是泛型的类型，参数个数为2
    - 排序规则：两两比较，x<y返回负整数，x>y返回正整数
- 通过匿名函数/Lambda表达式来写委托——现写现用，代码更少

协变逆变

- 协变的定义
    - 和谐、自然的变化
    - 子类变成父类感受是和谐的（儿子总会当爹，自然而然）
    - **关键字：out**
- 逆变的定义
    - 逆常规的变化
    - 父类变成子类感受是不和谐的（爸爸变成儿子感觉很怪）
    - **关键字：in**
- 用法
    - **用于修饰泛型字母占位符**
    - 只能在泛型**接口**和泛型**委托**中使用
    - 用协变**out修饰的泛型只能作为返回值**
    - 用逆变**in修饰的泛型只能作为参数**
- 协变的作用
    - son - > father：用父类返回值的委托装子类返回值的委托
    - 返回值的类型从子类“变成”了父类
- 逆变的作用
    - father - > son：用子类参数的委托装父类参数的委托
    - 参数的类型从父类“变成”了子类
- **协变和逆变的本质：里氏替换原则——父类容器装子类对象**

多线程

- 进程
    - 进程Process是计算机中的程序关于某数据集合上的一次运行活动，是操作系统结构的基础
    - 打开了一个应用程序就是在操作系统上开启了一个进程
    - 进程之间相互独立运行；也可以相互访问、操作
- 线程
    - 操作系统能够进行运算调度的最小单位，它被包含在进程之中，是进程中的**实际运作单位**
    - 一条线程指的是进程中的一个单一顺序的控制流，一个进程中可以并发多个线程
    - 简单理解：代码从上到下执行的一条管道
- 多线程
    - 同时运行代码的多条“管道”就叫多线程
    - 一般用来处理一些复杂的可能影响主线程流畅度的逻辑
- 线程的类——Thread，new一个**实例化后要传入一个委托/方法（名）作为变量封装要在该线程中执行的代码**
    - 启动线程：.Start()，默认开启的是前台线程

    - 设为后台线程 .IsBackGround = true
    - 终止线程：①.Abort(); ②设置一个bool标识
    - 休眠线程：Thread.Sleep(float t)；让线程休眠t毫秒，在哪个线程中写就休眠哪个线程
- 上锁lock
    - 多个线程之间共享内存
    - 同时执行可能会造成问题，比如代码语句顺序乱套
    - 可以通过加锁的形式避免问题：lock(obj){ // 逻辑处理}；obj相当于一把锁，给需要分开执行的代码逻辑分别**上同一把锁**，就可以避免两个代码块之间的语句顺序乱套
- 作用
    - 可以用副线程专门处理一些复杂耗时的逻辑，比如寻路、网络通信
    - 保持主线程运行流畅

反射

- 程序集
    - 是经由编译器编译得到的，供进一步编译执行的**中间产物**
    - 在Windows系统中一般是后缀为.dll（代码库文件）或.exe（可执行文件）的格式
    - 是一个代码的集合，所有写的代码都会编译器翻译为一个程序集供别人使用
- 元数据
    - 用来描述数据的数据
    - 程序中的类、方法、变量等信息就是程序的元数据
    - 元数据保存在程序集中
- 反射的概念
    - 程序正在运行时，可以查看其他程序集或者自身的元数据
    - **一个运行的程序查看本身或者其他程序的元数据的行为就是反射**
- 反射的作用
    - 因为反射可以在程序编译后获得信息，它提高了程序的拓展性和灵活性
- 反射的过程
    - 程序运行时得到所有元数据，包括元数据的特性
    - 程序运行时实例化对象、操作对象
    - 程序运行时创建新对象，并用这些对象执行任务

反射有关的类

- **Type——类的信息类**
    - 定义
        - 反射功能的基础
        - 访问元数据的主要方式
        - 使用Type的成员获取有关类型声明的信息
    - 方法
        - **变量.GetType()**；所有的对象都可以通过这个方法得到类型信息
        - **typeof(类名或变量名)**；常用于获得自己程序集中的类
        - **Type.GetType(字符串)**；字符串是类的FullName，必须包含命名空间
        - **.GetMembers()**；Type类型的变量调用该方法可以**获得类中的所有公共成员**，返回值是MemberInfo[]
        - **.GetConstructors()**；Type类型的变量调用该方法可以**获得类中的所有公共构造函数**，返回值是ConstructorInfo[]
        - **.GetConstructor(Type数组)**；得到某一个构造函数

        - **.GetFields()**；Type类型的变量调用该方法可以**获得类中的所有公共变量成员**，返回值是FieldInfo[]
        - **.GetField(字符串)**；Type类型的变量调用该方法可以**获得类中的指定名称的公共变量成员**，返回值是FieldInfo
        - **.GetValue(类名)**；获得某个类对象中的某个变量成员的值，调用者是FieldInfo类型的变量
        - **.SetValue(类名，要设置的值)**；设置某个类对象中的某个变量成员的值，调用者是FieldInfo类型的变量
        - **.GetMethods()**；Type类型的变量调用该方法可以**获得类中的所有公共方法成员**，返回值是MethodInfo[]
        - **.GetMethod(方法名字符串, Type数组)**；得到某一个方法
    - 注意
        - **相同类型的type在堆中的内存空间是一样的**
- **Activator——用于快速实例化对象的类**
    - 方法
        - **Activator.CreateInstance(Type type, 参数列表)**；实例化一个对象，调用的构造函数根据传入的参数列表自动匹配
- **Assembly——程序集类**
    - 定义
        - 主要用来加载其他程序集
        - 加载后才能用Type来获得并使用其他程序集中的信息
    - 方法
        - **Assembly.Load(程序集名称)**；加载在同一个文件夹下的其他程序集
        - Assembly.LoadFrom(包含程序集清单的文件的名称或路径)；加载不在同一个文件夹下的其他程序集
        - Assembly.LoadFile(程序集的绝对路径)；加载不在同一个文件夹下的其他程序集
        - **.GetTypes()**；Assembly类型的变量调用，可以获得该程序集中包含的类
        - **.GetType(类名)**；Assembly类型的变量调用，可以获得该程序集中指定的**类**

特性

- 定义
    - 一种允许我们向程序的程序集添加元数据的语言结构
    - 是用于保存程序结构信息的某种特殊类型的类
    - 特性与程序实体关联后，可在运行时使用反射查询特性信息
- 自定义特性
    - 继承**特性的基类：Attribute**
- 使用
    - [特性名(参数列表)]
- 方法
    - **.IsDefined(typeof(特性类名), false)**；Type类型的参数调用该方法，可以查看该类是否拥有某个特性，第一个参数是特性的类型，第二个参数表示是否搜索继承链，属性和事件忽略此参数
    - **.GetCustomAttributes(false)**；获得Type元数据中的所有的特性，返回的是object[]
- 作用
    - 在Unity中常用于注解，添加额外信息
    - 通过反射可以获取额外信息

- 系统自带的特性
  - Obsolete——提示方法过时的特性
  - CallerFilePath/CallerLineNumber/CallerMemberName——调用者信息特性
  - Conditional——条件编译特性，配合#define使用
  - DllImport——用来调用C/C++的Dll包写好的方法

迭代器

- 定义
  - iterator/cursor（光标）——是程序设计的软件设计模式
  - 迭代器模式提供一个方法**顺序访问一个聚合对象中的各个元素**而又**不暴露其内部的标识**
  - 从表现效果上看，是可以在容器对象，如链表或数组上遍历访问的接口，设计人员无需关心内存分配的实现细节
  - **可以用foreach遍历的类都是实现了迭代器的类**
- 传统方式实现迭代器
  - 关键接口：IEnumerator和IEnumerable
  - 同时继承上述两个接口并实现其中的属性和方法
    - Current属性——返回当前元素
    - MoveNext()方法——光标+1，返回true或false
    - Reset()方法——光标复原到-1，写在GetEnumerator方法中，用于遍历之前重置光标位置
  - foreach遍历时调用GetEnumerator方法，得到IEnumerator对象后执行其中的MoveNext方法，如果返回true，就把Current赋值给item，如果返回false就会跳出循环
- **用yield return语法糖实现迭代器**——只需要继承一个接口
  - 语法糖
    - 将复杂逻辑简单化，增加程序的可读性
    - 减少代码出错的可能
  - 关键接口：IEnumerable；这个接口支持被**泛型类**继承
  - 实现方法
    - 用for循环遍历元素
    - 每一次循环中yield return返回元素
  - yield关键字
    - 配合迭代器使用
    - 可以理解为暂时返回并**保留当前状态**

Var隐式类型

- 定义
  - 是一种特殊变量类型
  - 可以用来表示任意类型的变量
- 用法
  - **不能作为类的成员**，只能用于声明临时变量
  - 一般写在函数语句块中
  - var类型的变量**必须初始化**
- 作用
  - 偷懒，少写代码
  - 不确定类型时可以装载任意类型的变量

可空类型

- 关键字：？
- 用法
  - 在值类型名后加?就可以让该类型的变量为空
  - **在引用类型的变量后加?相当于安全校验——判空**
    - 是一种语法糖，若不是null才会执行后面的代码
    - 相当于if(go!=null){// 代码逻辑}，省代码
  - 也可以在委托类型的变量后加?简化安全校验的代码
  - **空合并操作符：??**
    - 相当于三目运算符
    - 可空类型才可以使用空合并操作符
    - 左边值 ?? 右边值，如果左边值为null就返回右边值
- 注意
  - 值类型可空时，类型变成了一种Nullable的泛型结构体
  - 安全获取可空类型的值：.GetValueOrDefault()