#### 1、C#规范

1. 变量 **smallCaml**
2. 函数 **BigCaml**
3. 私有字段前加下划线 private int **\_smallCaml**
4. 接口应该在其名称前加上大写字母 I，例如 **IInventoryHolder** 或 **IDamageable**
5. 运算符号赋值左右加空格

Godot

1. 场景 **BigCamlSence**
2. 节点 **smallCamlNode**

#### 2、变量类型

**char**-字符，单引号**'**创建，只能包含一个字符

**string**-字符串，双引号**“**创建,字符串本质是数组

Console.WriteLine('b');

Console.WriteLine(“b”);

**int**-整数

Console.WriteLine(123);

**float**-浮点数(精度6~9小数)，数字结尾f

Console.WriteLine(0.25F);

**double**-双浮点(精度15~17小数)

Console.WriteLine(0.25);

**decimal**-十进制(精度15~17小数),数字结尾m

Console.WriteLine(0.250m);

**bool**-布尔

Console.WriteLine(true);

Console.WriteLine(false);

**var**-自动获取值类型，仅用于局部变量。必须赋值，且第一次赋值后，无法赋新类型的值

var message = "Hello World!";

#### 2、字符串操作

**\t** - 制表符

**\ { }** - 转义,字符串中输出\或”或{或}等特殊含义字符

Console.WriteLine("\\和\"和{{和}}");

**@””** - 加逐字字符串文本，保留所有空格和字符

Console.WriteLine(@" c:\source\repos

(this is where your code goes)");

**\u** -Unicod转义符-以utf16转码\u后的4个字符

**$"{A}{B}"**字符串内插-字符串内输出A的值,AB间有空格

string message = $"{greeting} {firstName}!";

**$@"\{A}\"**-逐字字符串+字符串内插 共同作用

Console.WriteLine($@"C:\Output\{projectName}\Data");

1. **数值计算**

要赋值decimal变量且结果精确，除数中必须有一个为decimal类型

decimal decimalQuotient = 7.0m / 5;

**(类型)变量**-强制类型转换

int first = 7;

int second = 5;

decimal quotient = (decimal)first / (decimal)second;

**%**-取模运算符，求余

Int A = 200 % 5 //A=0

Int B = 7 % 5 // B=2

**A** **+=** **x** - 递增x

int value = 0; // value is now 0.

value = value + 5; // value is now 5.

value += 5; // value is now 5.

**A++** - 递增1

int value = 0; // value is now 0.

value = value + 1; // value is now 1.

value++; // value is now 2.

递乘递减递除格式同上，更换符号即可

1. **语法**

数组

**string[] array = new string[i] -** 字符串数组创建，i表示数组初始大小

**string[] array = new string[]{“a”,“b”,“c”}**

**string[,] 2darray = new string[i,j]{{},{}} - 二维数组**

**string[][] 2darray = new string[i][j]**

**枚举**

**enum 枚举{一 = 0,二 = 1，三 = 3}** - 用字符代表数字，默认从0开始

**列表**

**List<int> Numbers = new List<int>()**

**字典**

**Dictionary<keytype,valuetype> dic = new Dictionary<keytype,valuetype>()**

**{{,},{}}**

**Keyvaluepair<key,valye>**

**Dic.keys/dic.value**

**Dic.Trygetvalue(,)**

**Dic.Add(,)**

**栈 -** 先入后出

Stack<**string**> stackname = new stack<**string**>()

stackname.push() - 入

stackname.pop() - 出

stackname.trypop()

stackname.peek() -查看

**队列 -** 先入先出

queue<**string**> queuename = new queue<**string**>()

Queuename.Enqueue()

Queuename.Dequeue()

**哈希集 -** 同一元素只会出现一次

HashSet<**string**> hashsetname = new HashSet<**string**>()

Hashsetname.Add()

Hashsetname.count() - 计数元素

**结构体**

**是value type**

default为0

**Sortedlist**

**Sortedset**

**Sorteddictionary**

判断语句

**if(){}**

**else if{}**

**else{}**

**break** 结束循环

**continue** 进入下一循环

**switch ()**

**{ case "":**

**case "":**

**case "":**

**default:**//其余选项 **}**

**foreach(value** **in** **数组){}** -遍历数组

class ClassName : IEumerable<>{} - 允许自定义类可以被foreach遍历，详见7:19

yield return

**Value / reference type**

**循环**

**For(;;){}**

**While(){}**

**do{}while()**

**Property** - 属性

**string Name { get; set; } -** 自动实现

**string Name { get; }** - 只读

**string Name{set { name = value; }}** - 只写

//get，set可设置访问范围

**try-catch**

try{}catch(){}catch(){}finally{} - 尝试执行try，报错执行catch，可包含多个catch，总会执行finally即使catch里有return

throw

Expection

**可变参数**

**Void func(... , param string[] stringarray)-** param后跟随一个数组作为函数参数，可为函数传入任意数量参数(必须位于其他参数**末尾**且**唯一**)

**Void Func(int a = default,string b = default)** - 可为函数参数添加初始值

**Func(a:10)** - 调用函数时，指定参数加:添加初始值

**virtual / override -** 虚拟/重载修饰词

子类可以override和父类virtual名称相同的成员，实现与父类不同效果

**abstract** - 抽象修饰词,无法实例化

**sealed** - 密封修饰词,无法有子类

**destructor** 实例被摧毁时或找不到引用执行的函数

~classname(){}

**const -** 常量修饰词，让变量无法为空，且之后无法修改，并变成静态

**readonly -** 只读修饰词，可以在构造函数中设置值，而不用在编译时设置值

1. **注释**

**// - 行注释**

**/\* lotsofcode \*/ - 块注释**

1. **条件运算符**

**A ? b:c**三元条件运算符，当A为True时返回b，否则返回c

1. **范围**

**static** - 静态 属于类本身 无需实例化即可调用的成员

静态类所有成员都是静态的

**Protected -** 此类和其子类可访问

**Internal**

**public**

**private** - 只能在同一类中使用

**namespace** -

1. **接口interface**

**interface Iname{}**

**class Classname1 : Iname{} -** 有接口的类会继承接口所有成员

**class Classname2 : Iname{}**

**Iname name = new Classname1()**

**If(name is Classname1)**

**If(name.gettype() == typeof(Classname1))**

**If(name is Iname) -** 判断实例是否是某类

此时name接口实例无法调用Classname1的成员

类型转换

**Classname1 classname1 = name as Classname1** - 将name类型转换为**Classname1，** 如果失败返回null

**Classname1 classname1 = (Classname1)name** - 强制类型转换，失败报错

类型判断

**Switch(name){**

**case Classname1 classname1 when classname1.number > 20:**

**break;**

**case Classname1 classname1:**

**break;**

**case Classname2 classname2:**

**break;**

**default:**

**break**

**}**

1. **委托delegate** - 用值储存函数

**delegate void HelloDelegate();**

**delegate int SayHelloDelegate(string str);**

**static HelloDelegate helloAction1;**

**static Action helloAction2; -**与上一行等价(不用单独定义delegate了)

**static Action<int> helloAction2;-** Action 特殊委托，只会返回void

**static Func<int> helloAction3 -**Func 特殊委托，返回<>中的类型

**//lambda函数**

**Action<int> helloAction4 = (int a) => {};**

**Func<int,bool> helloAction5 = (int a) => {return false};**

**Func<int,bool> helloAction6 = (int a) => false;**

-花括号内可以直接写函数内容，不用再定义一个函数

**Static void Hello(){}**

**Static void Bye(){}**

**Static int SayHello(string name){return 0}**

**Void main(){**

**helloAction = Hello;**

**helloAction += Bye;**

**helloAction();** //执行Hello后再执行Bye

**helloAction -= Bye; //**删去Bye

**HellowDelegate sayHello = Hello**

**sayHello()** //执行Hello

**helloAction2 = Hello**

**helloAction2()** //执行Hello

**SayHelloDelegate sayHelloDelegate = SayHello**

**sayHelloDelegate()** //执行SayHello

**Local function** -功能与lambda函数相似，在另一个函数中被定义，只能在该函数中任意未知被调用，程序运行前已编译

-委托的lambda函数只能在定义后被调用，运行时才创建实例(花费内存)

1. **事件event** - 发生某事时提醒某类 解耦

-包含一个发布者和多个订阅/倾听者

-发布者和订阅者存在时间不同步时，记得取消订阅 -= （内存泄露）

class Player{

public event Action OnHelloSaid;

//-event修饰词会强迫事件出现在+=/-=旁边

public class OnHelloSaidEventArgs : EventArgs{} -传参

public event EventHandler<OnHelloSaidEventArgs> OnHelloSaid;

public void HeardSomeThing(){

if(OnHelloSaid != null){

OnHelloSaid();}}

//OnHelloSaid?.Invoke(this, new OnHelloSaidEventArgs{}

- ?左边不为null时执行右边

class HelloListener{

public HelloListener(Player player){

Player.OnHelloSaid += Player\_OnHelloSaid;}

private void Player\_OnHelloSaid(){

Console.WriteLine(“You heard : Hello”)；}

1. **泛型generics** 可以使用任意想要的类型 代码重用

public interface interface1<T>{

T Function()}

public struct struct1<T>{}

public class ClassName<T>{

public T arg;} 泛型类

ClassName<int> intInstance = new();

intInstance.arg = 5;

ClassName<string> stringInstance = new();

stringInstance.arg = “hello”;

public static void Function1<T>(T arg){

Console.WriteLine(arg)} 泛型函数

Function(1)

Function(“sb”)

Function(false)

public class ClassName<T> where T : class , new(){} - T只允许为class,并且它只有无参数的构造函数（：后为限制类型）

1. **设计模式**

**单例模式singleton -** 只允许一个类创建一个实例 7:30

懒汉模式 - 只在需要的时候创建单例

**观察者模式observer -** event

1. **reflection 反射** - 尝试调用可能不存在/没有权限的东西

typeof(something).getXXX(“”)

//获取特定修饰符成员，修饰符号自由选择，“|”表示“且”

BindingFlags flags = BindingFlags.Instance | BindingFlags.NonPublic;

//flags此时为非public实例成员

type.GetField(“”, flags);

GetValue()

GetProperty()

GetMethod()

1. **多线程&异步**

**Thread**

Thread t = new();

t.Start(); //开始线程

**Task**

Task.run(); //TPL(task并行库)，基于thread提供更多管理功能（线程）

await //等待修饰符，等待某段程序运行完后再继续运行

async //异步修饰符，使用await后自动为函数添加

List<Task> ts = new() //将多个task Add()入task泛型表，可以实现更多管理操作

Task.WhenAll(ts).ContinueWith(t => {}) //当ts task全部完成时继续执行其他程序

1. **网络**

**socket（服务端）**

private static List<Socket>clients = new();

private static List<String> userNames = new();

private static Socket Server;

...

//初始化服务端

Server = new Socket(AddressFamily.InterNetwork,//地址协议(ipv4)

SocketType.Stream,//套接字类型

ProtocolType.Tcp);//传输协议（TCP）

Server.Bind(new IPEndPoint(IPAddress.Any,12345)//ip地址，端口号

Server.Listen(10)//最大连接数

//Console.WriteLine(”服务端已开启”)

//接收连接

while(true){

Socket client = Server.Accept();

clients.Add(client);

//Console.WriteLine($”服务端来了个年轻人{client.RemoteEndPoint}”)

//创建线程处理连接

Thread th = new(() => HandleClient(client));

th.Start()}

//连接处理

static void HandleClient(Socket client){

byte[] buffer = new byte[1024] //缓冲区

string name = “”;

//防止未知情况

try{

int num = client.Receive(buffer);

name = Encoding.UTF8.GetString(buffer,0,num); //转换Socket

//Console.WriteLine($”{client.RemoteEndPoint}注册为{name}”)

BoardCasts($”{client.RemoteEndPoint}注册为{name}”,

client}；//向其他客户端广播

userNames.Add(name); //注册客户端名字

//接收并广播信息

while(true){

num = client，Receive(buffer);

if(num == 0)break; //没消息了

string message = Encoding.UTF8.GetString(buffer,0,num);

BoardCasts(name + “:” + message.client);//广播给客户端

//Console.WriteLine(name + “:” + message);} //服务端接收消息

catch(Execption ex){

//可以输出报错信息}

finally{

//注销客户端

client.Close();

cilents.Remove(client):

userNames.Remove(name);

BoardCasts(name + “已下线”, client);

//Console.WriteLine(name + “已下线”);}

//广播

static void BoardCasts(string msg, Socket client){

foreach(var i in clients){ //遍历所有客户端

if(i != client){ //排除自己(客户端)

i.Send(Encoding.UTF8.GetBytes(msg));}

**socket（客户端）**

private static Socket Client;

private static string serverAddress;

...

Client = new Socket(AddressFamily.InterNetwork,//地址协议(ipv4)

SocketType.Stream,//套接字类型

ProtocolType.Tcp);//传输协议（TCP）

Client.Connect(serverAddress,12345);

Console.Write(“已连接服务器，请注册name：”);

string name = Console.ReadLine();

Client.Send(Encoding.UTF8.GetBytes(name))；

Thread th = new(ReceiveMsg);

th.Start();

//发送信息

while(true){

string msg = Console.ReadLine();

Client.Send(Encoding.UTF8.GetBytes(msg));

//接收信息

static void ReceiveMsg(){

byte[] buffer = new byte[1024];//缓冲区

try{while(true){

int num = Client.Receive(buffer);

if(num == 0) break;

string message = Encoding.UTF8.GetString{buffer, 0 , num);

//Console.WriteLine(message);}}

catch(Exception){

//可以输出报错信息}

finally{

Client.Close();

//Console.WriteLine(“服务端已关闭”)}

**function extension 扩展方法**