# 客户端CSharp规范

## 基本

### 添加注释

1. 公开方法和接口必须添加注释
2. 类型文件名在头部或者类型简要描述处要进行足够的说明

### 修复警告

1. 当自己写的代码，刷新后在控制台出现黄色警告，如无特殊情况必须修复
2. 在打包资源包/发布完成后，也要留意控制台查看是否有警告，处理情况同上

### 代码文件格式

1. 格式采用UTF8，若不使用此格式，输出中文就会出现乱码，
2. 5.6以上默认采用此格式，但若发现不是，可在Unity的安装路径：

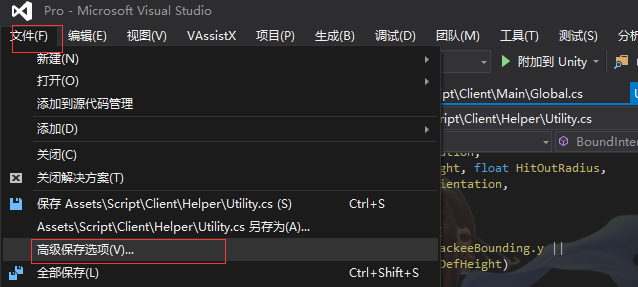
Editor\Data\Resources\ScriptTemplates\

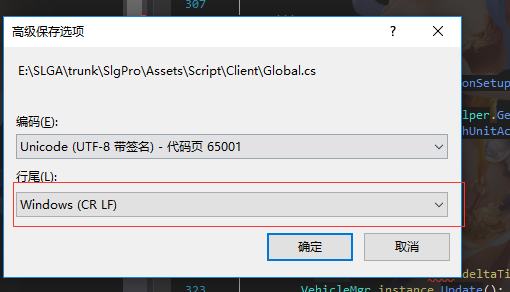
下找到 C# Script-NewBehaviourScript.cs.txt 文件

将此文件另存为 并选择格式为UTF8

### 文件行尾不一致问题

可通过Vs进行如下设置：





## 优化

### 严禁滥用匿名委托或者lambda表达式

能够使用对象方法或者静态方法时就一定要使用，如果需要捕捉额外的参数

可以自己定义一个中间类型进行回调传递

### 禁止在Update/LateUpdate/OnGUI或者其它频繁调用方法中不停调用GetComponent等耗时API方法

错误方式：

void Update()

{

YouCom com = gameObject.GetComponent<YouCom>();

}

合理方式：

而使用如下

private YouCom com = null;

void Start()

{

com = gameObject.GetComponent<YouCom>();

}

其它耗时方法有：GameObject.Find等

### 合理的日志输出

1. ，测试log，在测试完成后一定要记得删除
2. ，为了防止遗忘删除，最好是用编辑器预处理指令包裹，像这样

#if UNITY\_EDITOR

//日志输出

#endif

1. ，对于警告输出，出现一定要解决掉
2. ，对于错误输出，同上

### 使用对象池

概述：绝不允许出现频繁创建/销毁的情况

1. ，非引擎类型的对象池为ObjPool
2. ，游戏对象的对象池为GbjPool
3. ，可以在合适的时机将对象池进行清理，比如在场景切换完成后的某一刻，清理并调用GC以缓解对象池的内存压力

### 不要频繁的使用不可变字符串进行拼接

概述：尤其是不要对大字符串进行多次拼接

1. ，错误方式：

string str = str1 + str2 + “myStr” + str3 + str4+str5;

1. ，合理方式：

满足下列条件之一可使用可变字符串结构StringBuilder进行字符串拼接

1. 拼接条目超过5个以上

如：string str = str1 + str2 + “myStr” + str3 + str4+str5

1. 拼接条目中具有大字符串(长度>20)并条目超过3个(包含)以上

如：string str1 = “xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx”

String str = str1+ str2 + str3

### 在列表的遍历中删除元素时使用倒序遍历

错误方式：

type it = null

for (int i=0; i<list.Count; i++)

{

it = list[i];

it.Update()

}

不合理的因素:

假设列表长度10，当前遍历到索引5，此时要删除索引5所在元素，在删除索引5元素之前，下一个要遍历的索引是列表的第6位；在删除索引5元素之后，列表的第6位变成第5位，列表的第6位是删除之前的第7位，这样就跳过了删除之前的第6位

合理方式：

Int beg = lst.Count – 1

type it = null

for(int i= beg; i>-1; --i)

{

it = list[i]

it.Update()

}

### 在列表尾部删除元素

比如在上述遍历过程中，删除元素时，如果直接使用RemoveAt或者使用Remove

有可能会出现极其耗时的情况；因为列表的元素在内存中地址是连续分布的，使用上述两种方法删除后，会向上移动移除位置之后的元素，如果是在Update中频繁进行非尾部删除…

后果可想而知；所以此时在上述的应用场景中就可以使用将当前要删除的元素和尾部元素进行交换，并移除尾部元素，比如：

Int beg = lst.Count – 1

type it = null

for(int i= beg; i>-1; --i)

{

it = list[i]

it.Update()

//也可以使用 ListTool中的Remove方法进行移除

If (移除条件达成)

{

Int last=list.Count-1

var temp = list[last]

list[last] = it

list[i] = temp

list.RemoveAt(last)

}

}

### 用for/枚举器遍历代替foreach

每个foreach每次调用都会产生24字节垃圾内存；

尤其不能在Update等每一帧都调用的方法中调用foreach；

当无法使用for代替时，可使用枚举器遍历方式代替foreach,如:

{

List<object> collect = new List<object>;

var ie = collect.GetEnumerator();

while(ie.MoveNext())

{

var cur = ie.Current;

}

}

### 不允许对存在大量元素的列表，进行频繁Contains操作

极其耗时，可以考虑使用字典/哈希集等 集合

### 比较标签

错误方式：

If(go.tay == “YouTag”)

合理方式：

If(go.CompareTag(“YouTag”))

### 删除无用的引擎回调函数

错误方式：内部没有任何语句块

void Update()

{

}

也包括LateUpdate,FixedUpdate 等其它回调函数

合理方式：

删除以上无代码块的回调函数

### 如果接入lua，严禁在C#中事件反复注册lua回调

比如CS类型 Example，其有类型：ExampleDelegate，名为:ExampleEvent的事件

错误方式：

lua中A处注册 Example.ExampleEvent(ExampleDelegate(self.func1, self))

B处注册 Example.ExampleEvent(ExampleDelegate(self.func1, self))

………….

建议方式：

在lua中对这个C#回调合理唯一注册，在唯一注册位置，进行lua事件抛出。

比如：在Lua中有某类型LuaExample, 其具有lua事件eExample

* 1. ，先注册C#事件

function My:Init()

Example.ExampleEvent(ExampleDelegate(self.Example, self))

end

function My:Example()

self.eExample(…)

end

* 1. ，再注册lua事件

当需要时只往LuaExample.eExample中注册

## 存放位置

### 运行时代码：

必须存放在Asset/Script/Client下

### 编辑器代码：

必须存放在Asset/Script/Editor下

## CS热更

### 继承MonoBehavior的代码：

* 1. ，如果选择热更，必须动态添加/创建脚本，而不能直接挂在Prefab或者GameObject上
  2. ，如果不热更，则此段代码，不能引用热更部分代码

### 继承ScriptableObject的代码：

* 1. ，只能动态创建，不允许用来保存资源

### 不允许使用二进制序列化(BinaryFormat)：

* 1. ，原因：其序列化数据，会保存程序集的元数据信息，导致热更后程序集不匹配
  2. ，可替代方案：BinaryReader/BinaryWriter,xml,json(Unity原生json会有部分问题)

### CS代码的创建和使用需和前端主程序进行说明