**Unity3D代码及性能优化**

# 代码可读性

* 杜绝MagicNumber（魔法数字）
* 代码格式，详见代码格式规范

# C#代码

* 队列循环中去删除自身导致莫名的问题
* C#引用机制带来的对象(容器)释放问题
* 回调的滥用 典型的如 回调中去删除对象本身
* 合理使用容器 常用的 List、Dictionary以及可能的替代方案 Queue/Stack

SortedList/SortedDictionary

Dictionary中Key的选择 尽可能使用值类型如int，而少使用复杂类型如string

List各接口时间复杂度

Add O(1),当元素个数超过Capacity时，则为O(n) Capacity可以在初始化时指定

Insert O(n)

Remove O(n)

RemoveArt(intdex) O(n) n = Count-index

批量删除建议 RemoveRange或者最后Clear

减少Dictionary的冗余访问 多使用dic. TryGetValue

* 使用内建的数组，比如用Vector3.zero而不是new Vector(0, 0, 0);
* 函数参数的优化：善于使用ref关键字 主要针对复杂的值类型，主要是struct 像Vector

# 内存

* 臭名昭著的GC

典型 foreach -> for

string+ string+= 使用stringBuilder或者string.format

* 高频调用函数 返回string对象 使用 ref或者out
* 循环中使用的string对象等 最好在循环外创建好，不要在循环中不停地创建
* 高频调用函数尽量多使用成员变量 或者使用objPool 避免堆内存对象的创建
* 不要使用LINQ命令
* Unity相关函数， 不可调频调用 使用缓存

GetComponent() 、go.tag

* 对象/容器释放

在合适的时机 将对象置空 obj = null;

容器 调用Clear 再置null

# 渲染优化

* 控制顶点数量
* 降低shader复杂度
* 贴图大小，降低显卡带宽
* 批次合并