## 延时类功能脚本对象回收、移除

Tween、Timer计时器、事件管理器 、 需要声明字段 在UI销毁时候将这些移除（Tween内部已经作了处理了）

## 数组

全局的数组容器管理的对象（渲染节点）被删除之后 数组给清空 （如果是挂在在被销毁对象就不需要处理）

        tempList.length = 0;// 容器 线性空间  数组还在还能用

        // tempList = null;

不固定数组 有数据平移 尽量避免增删改查会接近于 真正的array数组

在数据量庞大的时候List的性能比Array的性能低；

在数据量较小的时候List的性能和Array的性能基本上差不多；

在数据量小或者长度不可知的情况下推荐使用List，因为其长度是可变的；

在数据量大或者数据量的长度明确的情况下推荐使用Array，因为这样可以提高性能。

## 时间

跟时间挂钩的逻辑问题

   private timer: *number* = 0;//走的时间

    private intervalTimer: *number* = 0;//间隔时间

    private isDoWork: *boolean* = false;//执行

    update(*deltaTime*: *number*) {

        this.timer += *deltaTime*;

            }

        }

        if (this.timer > this.intervalTimer) {

            this.timer = 0;

        }

    }

这样是错误的

1. 累加的时间有精度问题，不停地累加会不断产生差值 导致误差
2. 致命错误:this.timer归零 重置，导致在原有误差上进一步误差，

正确的做法：

使用系统时间，转换成对应的毫秒 对比当前帧和下一帧走过的时间差，然后根据比例计算出应该显示的字符位置

## Update

游戏中理论存在不能过多的Update 生命周期函数，有一个地方进行管理 使用 否则引擎内部会进行递归叠加

## 异步UI 和数据 刷新逻辑

1. 数据驱动UI，在UI打开的时候 去调取UI要展示的数据

数据接受到之后 去调取UI刷新数据

1. 一定需要跳转的界面才需要先加载界面，再刷新对应的数据

## 数据信息的创建和移除

1. 请求数据
2. 打开UI

第一步和第二部是同时进行的

1. 打开UI通过数据展示展示
2. 数据请求下来，调用UI刷新方法

关闭UI后，同样要把数据字段重置不然下次数据没有到，UI先打开的时候展示的就是错误的信息了

## Lanmd表达式

    /\*\* 添加玩家到场景 并添加点击这个玩家的事件 \*/

    private AddRoomPlayerAndClick(*roomPlayerInfo*: RoomPlayerInfo) {

        SchoolMap.instance.addRoomPlayer(*roomPlayerInfo*, this.IsMaster, (*clickRoomPlayerInfo*: RoomPlayerInfo, *eventTouch*: EventTouch) *=>* {

            console.log(*eventTouch*.getLocation());

            if (this.IsMaster) {

                this.\_currentKickUid = *clickRoomPlayerInfo*.uid;

                UIManager.instance.uiEvent(UIConst.RoomUI, 3, true);

            }

        }, this);

    }

Lanmd表达式不属于任何一个类，普通函数在在类==null之后 所有的字段会从栈上移除，属于直进直出，Lanmd表达式的函数需要在所有有引用的地方不适用之后，交给GC来进行回收

## 行为树

一个链表结构，有开启 更新 和结束 的方法，类似状态机， 适用于剧本流程类型的游戏， 注册相应行为，在行为自己内部控制UI界面和逻辑，也可以监听异步的消息，同时可以从任意行为节点开始，保证同步进行，修改后可以执行按条件判断的行为树。

## 二叉树

最快的遍历就是没有遍历，走一步算一步