**Lua**

**使用说明书**

文档信息：

|  |  |
| --- | --- |
| 作者： | 周忠年 |
| 创建日期（yyyy-mm-dd）： | 2019.08.19 |
| 审核者： |  |
| 审核日期（yyyy-mm-dd）： |  |
| 最后修订者： | 周忠年 |
| 最后修订日期（yyyy-mm-dd）： |  |
| 文档类型： |  |

文档修订历史：

|  |  |  |  |
| --- | --- | --- | --- |
| 版本号 | 修订日期 | 修订者 | 修订内容 |
| V1.0 | 2019.8.19 | 周忠年 | 初创 |
|  |  |  |  |
|  |  |  |  |

目录

[TimeLine的制作： 4](#_Toc16611005)

[基础包，基本设置 4](#_Toc16611006)

[TimeLine预制件 7](#_Toc16611007)

[TimeLine测试 8](#_Toc16611008)

[需要先初始化配置文件： 8](#_Toc16611009)

[开始下载和加载对应的预制件： 9](#_Toc16611010)

[加载的代码逻辑： 9](#_Toc16611011)

[制作范例 10](#_Toc16611012)

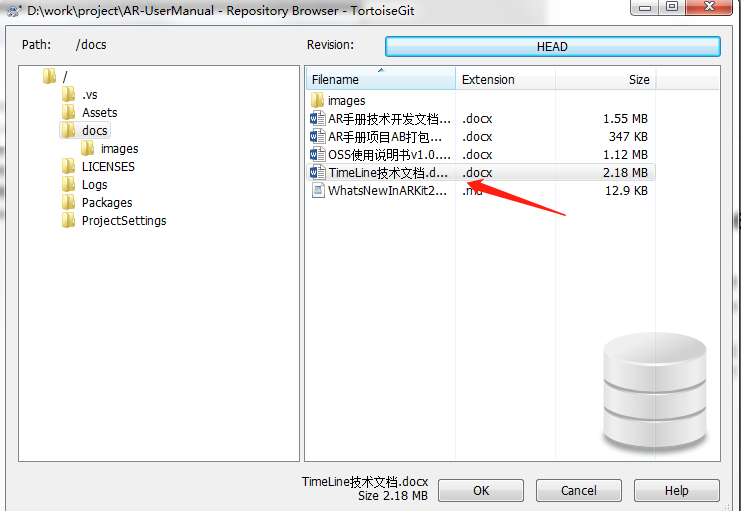
[TimeLine 使用反馈 28](#_Toc16611013)

# Lua热更简介：

鉴于可能在产品APP上线之后，会出现小的逻辑错误，但是又不能从新制作APP和从新上线，故而，使用，在线更新的方式，解决，在上线之后，需要的逻辑改动，此时，就需要Lua语言来实现该功能。

这个参考：

TimeLine技术文档



注意：以下的操作，需要在TimeLine制作完成后，再去完成一些操作。

# 基础包，基本设置

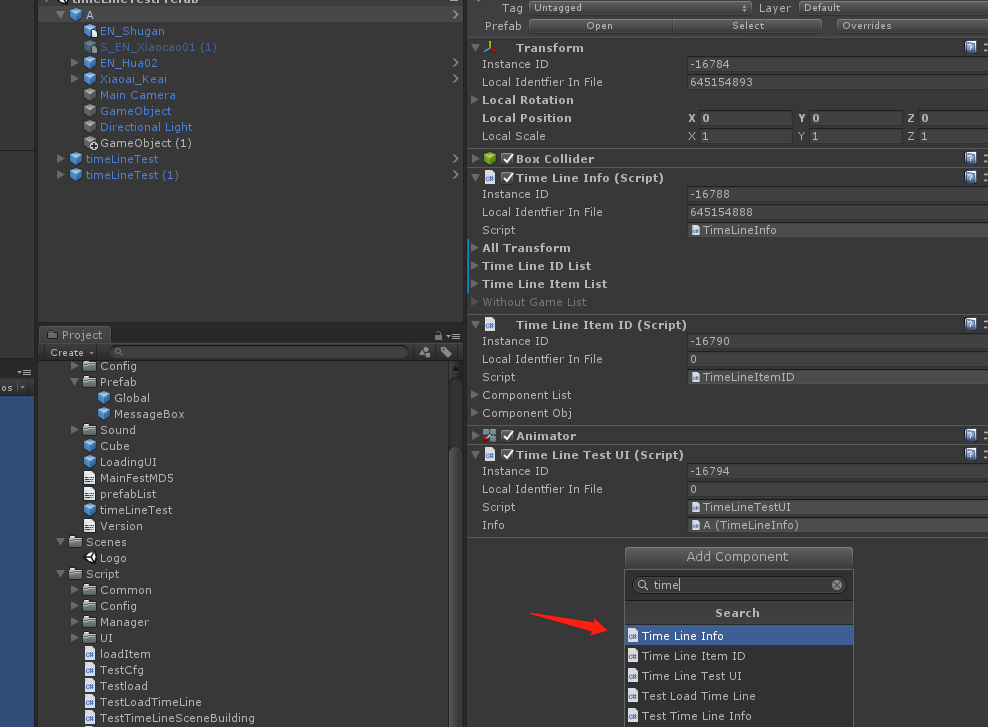
基础包，需要将Camera，AudioSource等GameObject放在同一个物体下面，便于管理与查看。

基于原理：

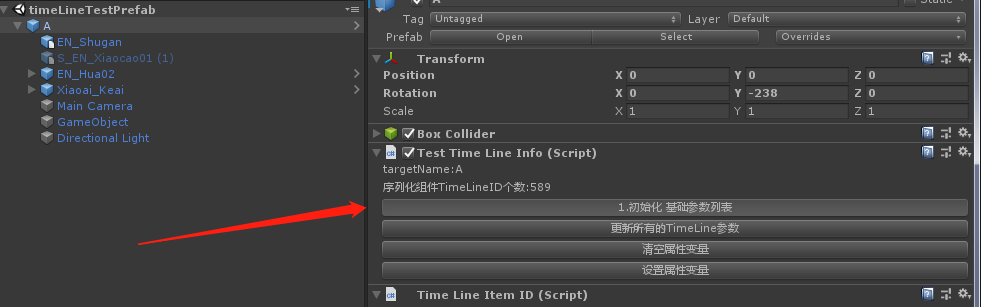
在每一个GameObject上，帮一个脚本，用于保存物体的ID信息。（由于InstanceID，在每次实例化基础包时，该值都会变，所以不宜采用InstanceID的方法）

选中基础包的父物体：

绑定，基础包的脚本：

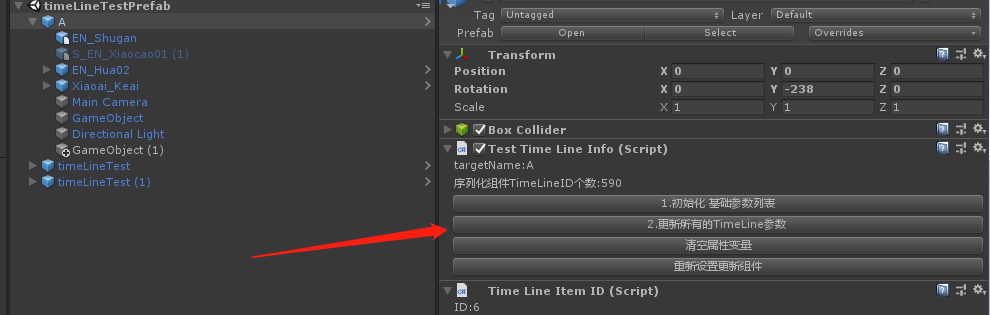


接下来，点击一下按钮



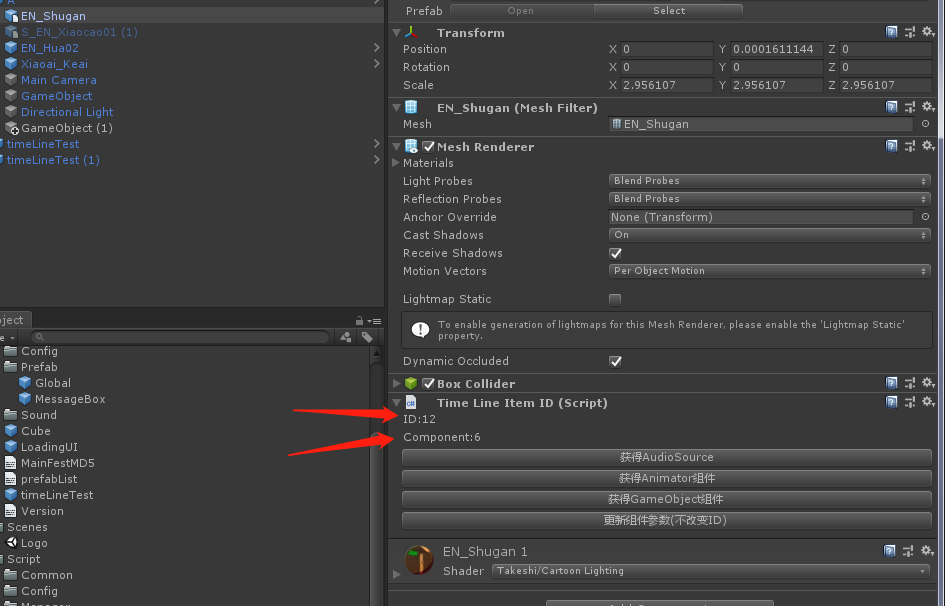
点击“1，初始化 基础参数列表”，速度可能会比较慢，等待加载完成后，即可。

如果这里，有正在编辑的TimeLine预制件，我们直接点击“2.更新所有的TimeLine参数”



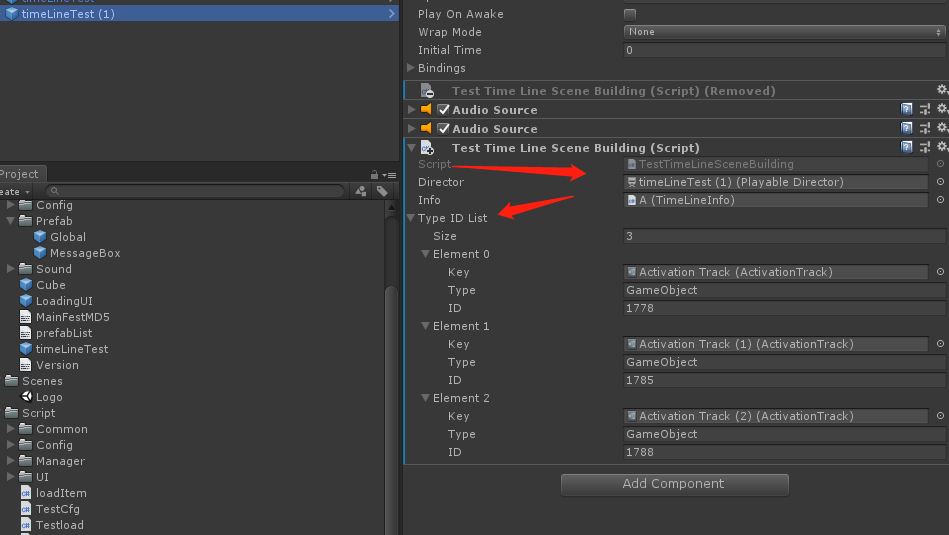
这个操作，就会把，所有的绑定的timeLine 都执行，参数的绑定。

以上两个操作，完成后，会在基础包中，添加了很多的TimeLineItemID的组件：



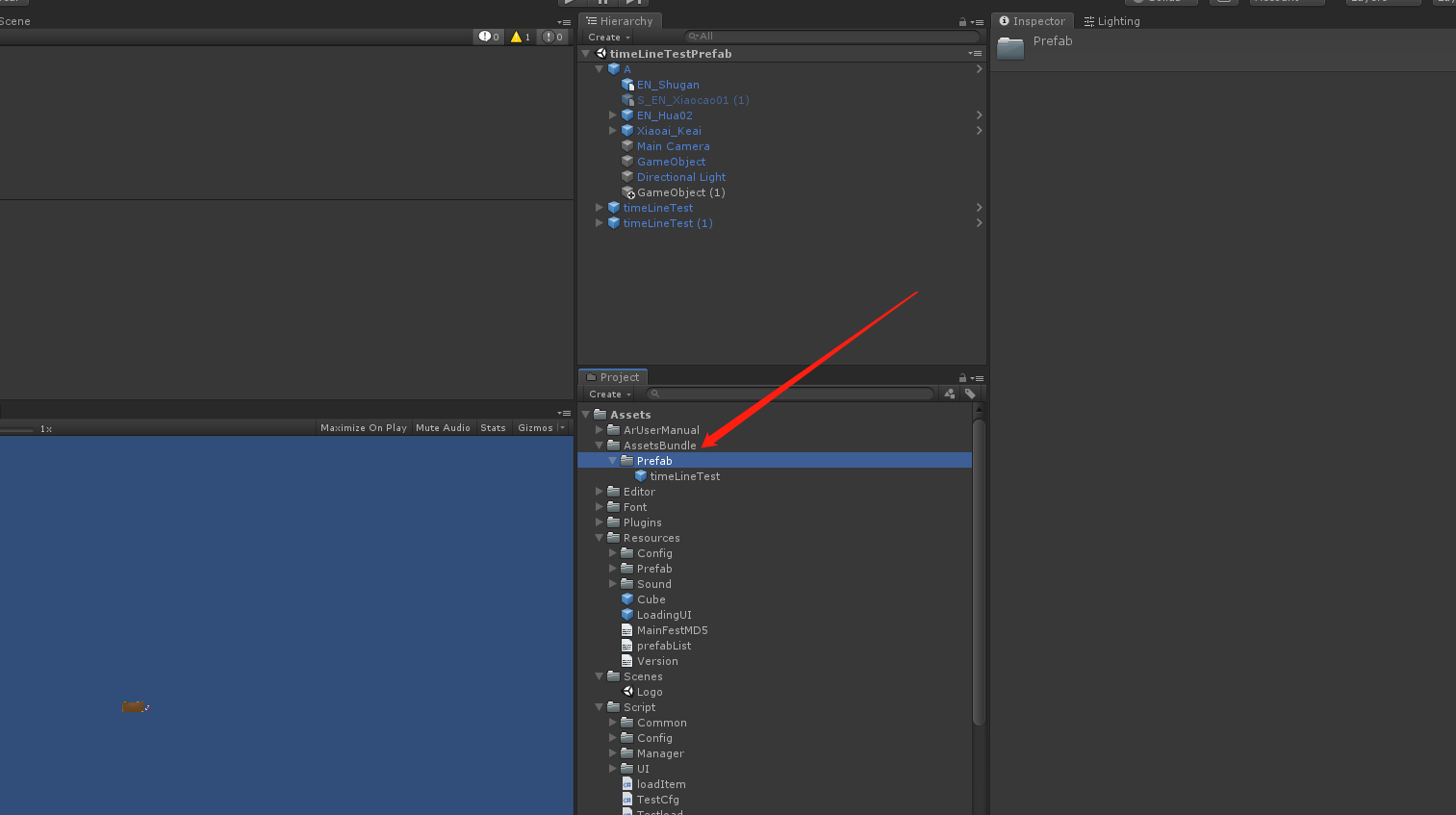
这里的ID，是不能更改的，是代码自动生成的

timeLine预制件：



基本上的设置，完成，将TimeLine保存 成预制件。

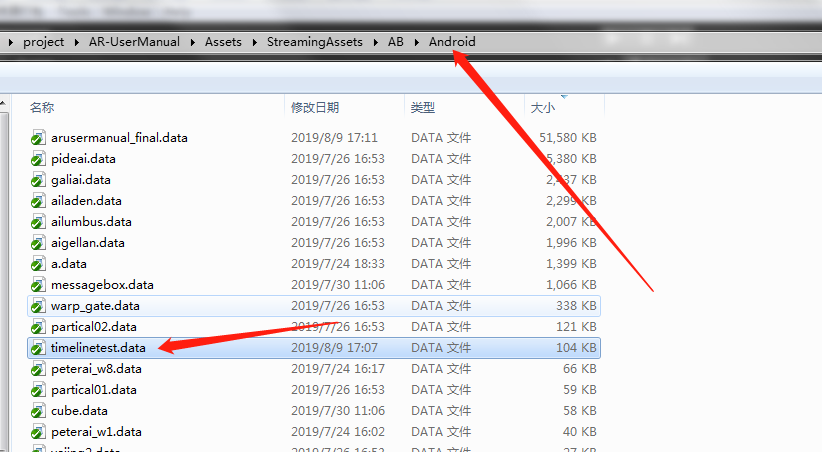
# TimeLine预制件



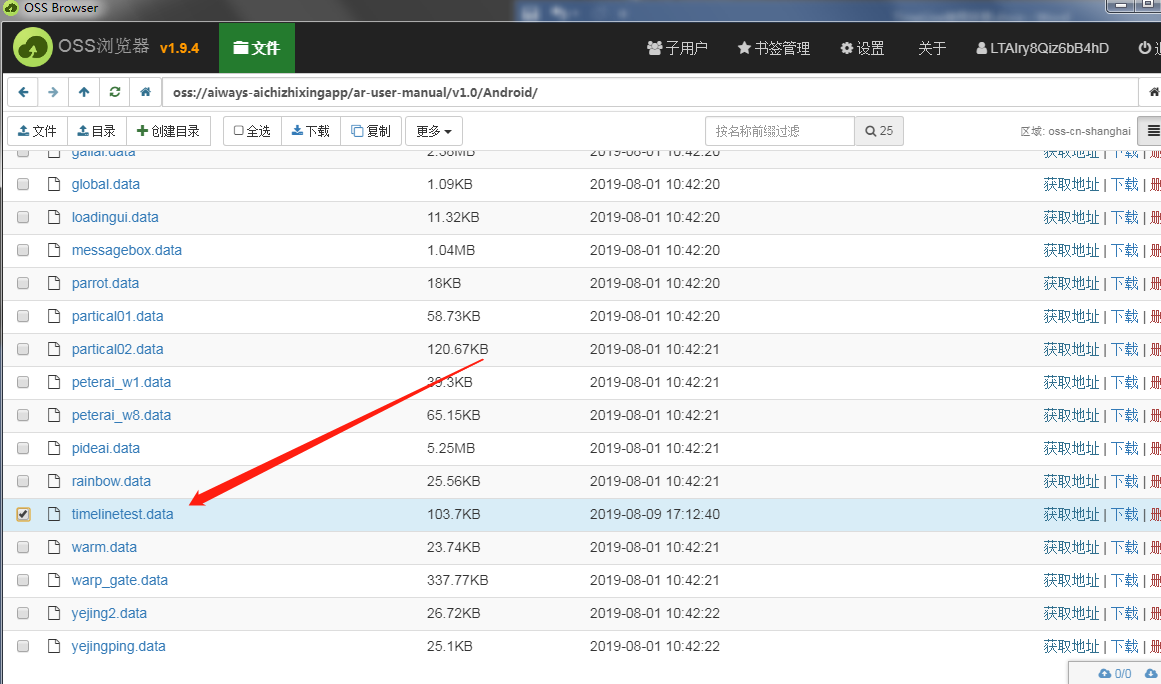
这个时候，可以执行 打包测试。（具体参考AB打包 OSS使用说明书中的打包）

跳过打包过程。

接下来，TimeLine的预制件，打包好了。



开始上传：



## TimeLine测试

### 需要先初始化配置文件：

AssetbundleEngine.instance.LoadTextByURL(GlobalUIManager.Instance, Common.versionUrl, (f) => { }, (version) =>

{

ConfigOverAll.Instance.version = version;

if (MainFestCfg.Instance == null)

{

//初始化，要放在Unity中的主线程中，实例化

MainFestCfg.Instance = new MainFestCfg();

MainFestCfg.Instance.Load(() =>

{

MainFestCfg.Instance.IsLoaded = true;

init();

});

}

});

### 开始下载和加载对应的预制件：

m\_data = MainFestCfg.Instance.Property("timeLineTest");

AssetbundleEngine.instance.LoadAssetToLocalByHttp(GlobalUIManager.Instance, m\_data.Address, (bo) =>

{

Debug.Log(m\_data.Name + ":" + " is ok");

text.text = m\_data.Name + ":" + " is ok";

\_isloading = false;

}, (f) =>

{

//Debug.Log(m\_data.Name + ":" + f);

text.text = ":" + f;//m\_data.Name +

m\_Fill.fillAmount = f;

\_isloading = true;

},m\_data.FileSize);

### 加载的代码逻辑：

MainFestItemData \_data = MainFestCfg.Instance.Property("timeLineTest");

AssetbundleEngine.instance.LoadModelByURL<GameObject>(this, \_data.Address, \_data.Name, (f) => { }, (go) => {

//获得参数

TestTimeLineSceneBuilding \_build = go.GetComponent<TestTimeLineSceneBuilding>();

if (\_build != null)

{

\_build.SetInfo(this.\_info);

}

});

测试的时候，（找开发）会根据需求，写个测试的代码，来实现

# 制作范例

# TimeLine 使用反馈