# Egret Engine 3D 教程文档

## 内测文档禁止外传

# 第一章 快速入门

Egret 3D 引擎是一款用 Typescript 语言编写的 HTML5 3D 引擎。

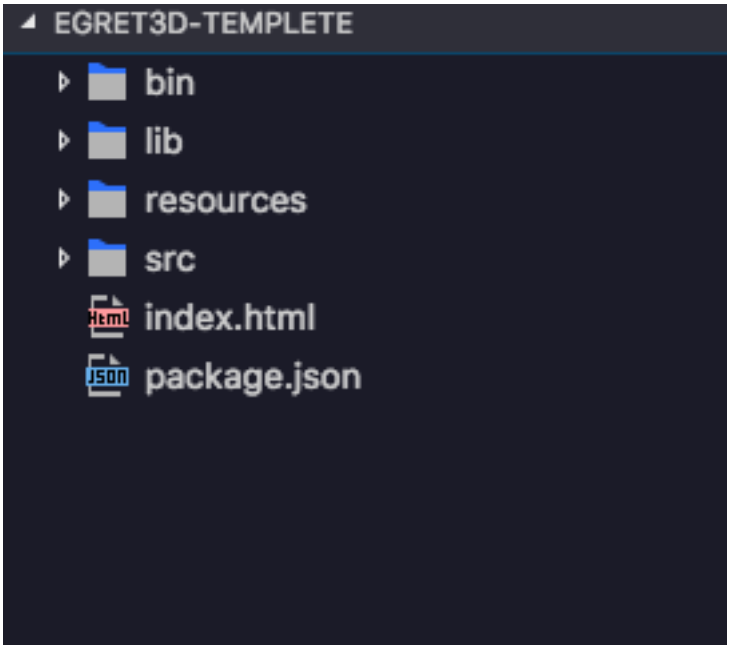
下面演示如何创建一个应用程序，加载 Egret 3D 引擎，并绘制一个立方体。

## 一、环境安装

Egret 3D 模版项目会基于 NPM 自动配置 TypeScript 运行环境。

开始之前只需要在您的计算机中安装 NPM 包管理工具。

## 二、启动引擎




1、egret-templete 文件夹中是 Egret 3D 的示例项目，拷贝此文件，用 Wing 打开

lib 是引擎库文件。

src 是用户代码。

bin 是 tsc 编译后的文件。

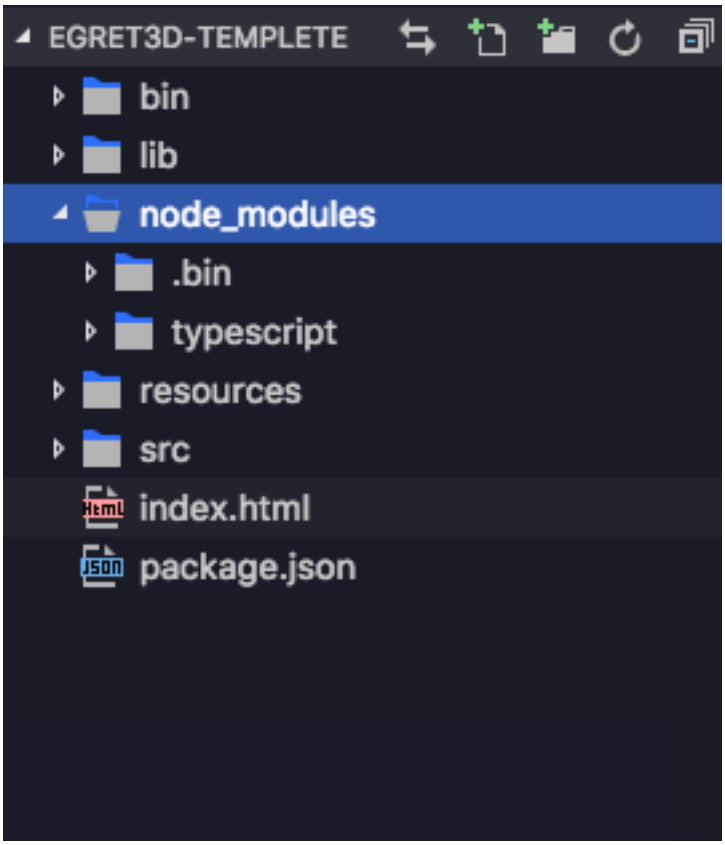
resouces 中存放游戏资源。

2、运行项目前，需要在当前根目录运行：

npm install

运行完成后，npm 包管理工具将自动为您安装好 TypeScript。

如果在命令行输入 tsc 运行报错的话是 TypeScript 没有安装成功，请单独安装




地址：http://www.typescriptlang.org/

在 package.json 中，我们为您内置了 tsc 编译脚本：

编译项目：npm run build

监听自动编译：npm run watch

当然，因为是标准的 TypeScript 项目，如果您熟悉 TypeScript 的话，也可以自己订制。

## 三、运行示例项目

编译项目，之后用浏览器打开项目，可以看到运行的结果：

index.html 文件中，引入了如下三个 js 文件：

Reflect.js 与 egret3d.js 是需要引入的引擎文件，app.js 是编译后的游戏逻辑代码。每次编译后，会更新这个 app.js 文件。

在文档最后的脚本中，启动游戏引擎。

使用命令 app.addUserCode(“Main”) 来添加用户逻辑。

```
<script src="lib/Reflect.js"></script>
<script src="lib/egret3d.js"></script>
<script src="bin/app.js"></script>
```

```
<script>
    window.onload = function () {
        var app = new egret3d.framework.Application();
        var div = document.getElementById("drawarea");
        app.start(div);
        app.showFps();
        app.bePlay = true;
        app.addUserCode("Main");
    }
</script>
```


用户逻辑需要被引擎识别和驱动，比如最基本的 update 驱动等，就是靠引擎来驱动的。这里用到了一个语言特性：装饰器。具体方法是：创建一个脚本，创建一个类，实现 egret3d.framework.IUserCode 接口，再对类增加 @egret3d.reflect.userCode 装饰器。然后通过类名或者类的实例就可以启动这个用户代码了。用户就可以舒服的使用 IUserCode 接口提供的方法了。

创建一个脚本 main.ts，实现 egret3d.framework.IUserCode 接口，补全接口方法。

```
/// <reference path="lib/egret3d.d.ts" />
//需加上这个反射标记，场景才能通过名字找到这个类，并自动创建他
@egret3d.reflect.UserCode
class Main implements egret3d.framework.IUserCode
{
    onStart(app: egret3d.framework.Application) {

    }
    onUpdate(delta: number) {
```

```
    }
    isClosed(): boolean {
        return false;
    }
}
```

启动用户代码有两种方法：

方法一：addUserCode("Main");通过类名启动。

方法二：addUserCodeDirect(new Main());通过类的实例启动。

所以在入口的地方调用下这个方法，就可以启动 Main 这个类了。

项目模版中已经默认添加了用户逻辑，我们打开 src 文件夹中的 Main.ts。

首先在 onStart 中加载 shader 资源包，没有 shader，什么都无法绘制。

```
// 加载shader
let shaderAssetUrl = "resources/shader/shader.assetbundle.json";
this.app.getAssetMgr().loadCompressBundle(shaderAssetUrl, (state) => {
    if (state.isfinish) {
        // 加载模型
```

在 shader 加载完成的回调里面，加载默认的 3D 模型。这里加载的是 Unity 导出的

```
// 加载模型
let baihuAssetUrl = "resources/prefabs/baihu/baihu.assetbundle.json";
this.app.getAssetMgr().load(baihuAssetUrl, egret3d.framework.AssetTypeEnum.Auto, (s) => {
    if (s.isfinish) {
        let prefab: egret3d.framework.Prefab = this.app.getAssetMgr().getAssetByName("baihu.prefab.json") as egret3d.framework.Prefab;
        this.baihu = prefab.getCloneTrans();
        this.scene.addChild(this.baihu);
        this.baihu.markDirty();
    }
});
```

Prefab，在 Unity 导出资源的方法将在后面介绍。

接着往场景中添加相机，只有被相机拍到的区域才会被渲染到屏幕上。

```
//添加一个摄像机
var objCam = new egret3d.framework.Transform();
objCam.name = "mainCamera";
this.scene.addChild(objCam);
this.camera = objCam.gameObject.addComponent("Camera") as egret3d.framework.Camera;
this.camera.near = 0.01;
this.camera.far = 100;
this.camera.backgroundColor = new egret3d.math.Color(0.5, 0.0, 0.0, 1.0);
objCam.localTranslate = new egret3d.math.Vector3(0, 0, -30);
objCam.markDirty();//标记为需要刷新
```

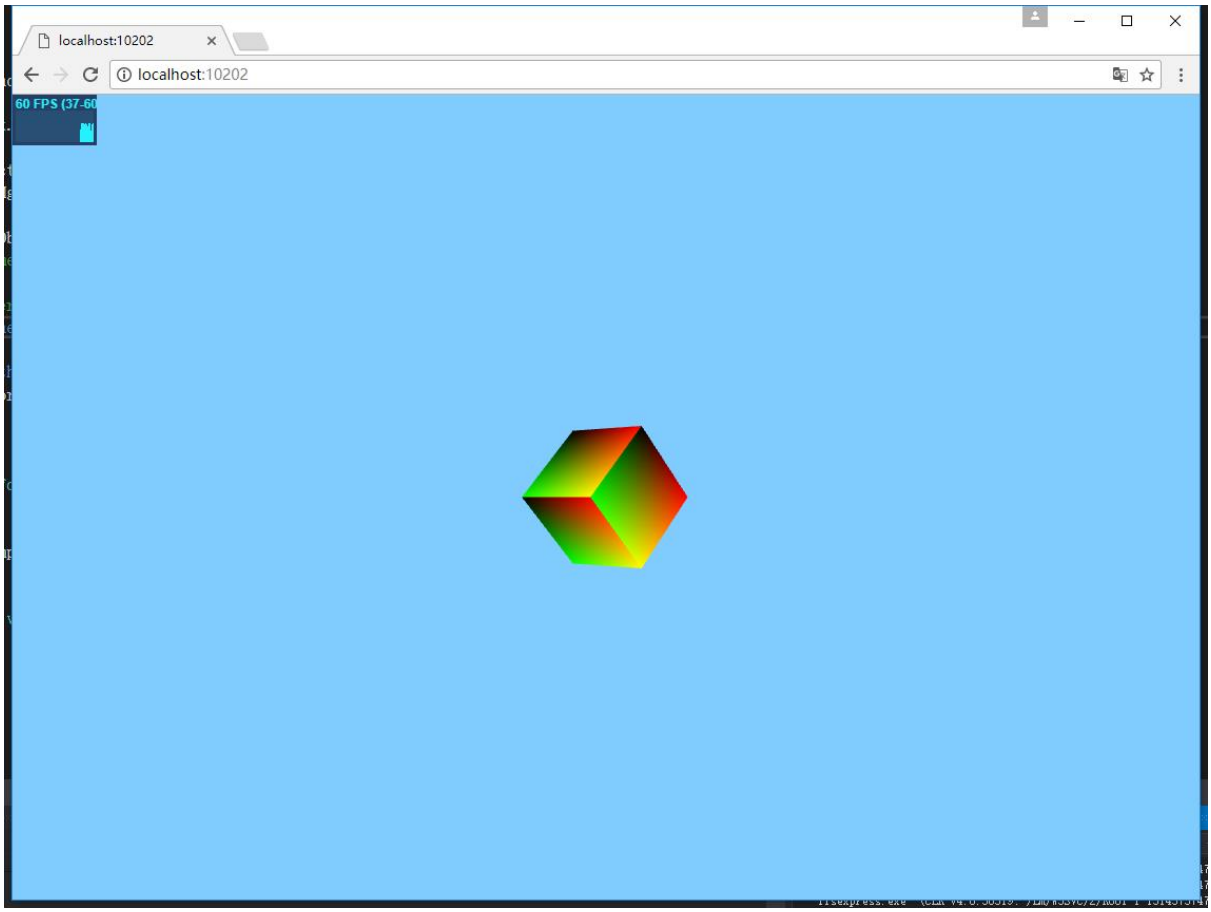
注意：一定要 markDirty()，改动才能生效。

下面我们动手修改一下工程，创建一个盒子。删除掉模型加载和显示的代码。

替换为如下代码：

```
// 显示内置cube
var cube = this.cube = new egret3d.framework.Transform(); //创建一个transform
cube.name = "cube"; //设定名字（可以不设定）
var mesh = cube.gameObject.addComponent(egret3d.framework.StringUtil.COMPONENT_MESHFILTER)
as egret3d.framework.MeshFilter; // 为其添加一个meshFilter组件
var smesh = this.app.getAssetMgr().getDefaultMesh("cube"); //获取引擎内置的cube模型
mesh.mesh = smesh; //赋值给meshFilter组件
var renderer = cube.gameObject.addComponent
(egret3d.framework.StringUtil.COMPONENT_MESHRENDER) as egret3d.framework.MeshRenderer;// 添加
meshRender组件，用来渲染盒子
cube.localEulerAngles = new egret3d.math.Vector3(45, 45, 0); //给盒子一定的旋转角度
cube.markDirty(); //标记dirty，这样引擎就会去去更新一次这个cube
this.scene.addChild(cube); // 将盒子添加到场景中去
```

运行后效果如下：




可以看到绘制出来一个彩色的盒子。如果细心的同学可能会注意到，我们上面并没有给盒子设置任何材质信息，我们在引擎中会检测如果一个 mesh 没有材质信息，就会

给它添加一个材质，给一个引擎内置的叫 def 的 shader，这个 shader 中会给顶点不同的颜色。绘制出来就是这个五彩的盒子。

meshrenderer.ts

```
02
03          private refreshLayerAndQue()
04          {
05              if (this.materials == null || this.materials.length == 0)
06              {
07                  this.materials = [];
08                  this.materials.push(new framework.material());
09                  this.materials[0].setShader(sceneMgr.app.getAssetMgr().getShader("shader/def"));
10              }
11
```

完整的代码如下在第一章示例附件中。

# 第二章 材质的创建和使用

我们在上一篇快速起步中绘制出了一个五彩的盒子。这一篇我们更改下盒子的材质，绘制一个带有 egret logo 的盒子。

1、首先我们把 egret 的 log 图片放到项目中的 resources 目录下。

2、接着在 onStart 中 shader 加载完成后，加载贴图，使用 AssetMgr 中的 load 方法进行加载：详见 api 文档:


对于第二个参数：AssetTypeEnum.Texture，也可以用 AssetTypeEnum.Auto，引擎中会根据后缀自动推算资源类型。

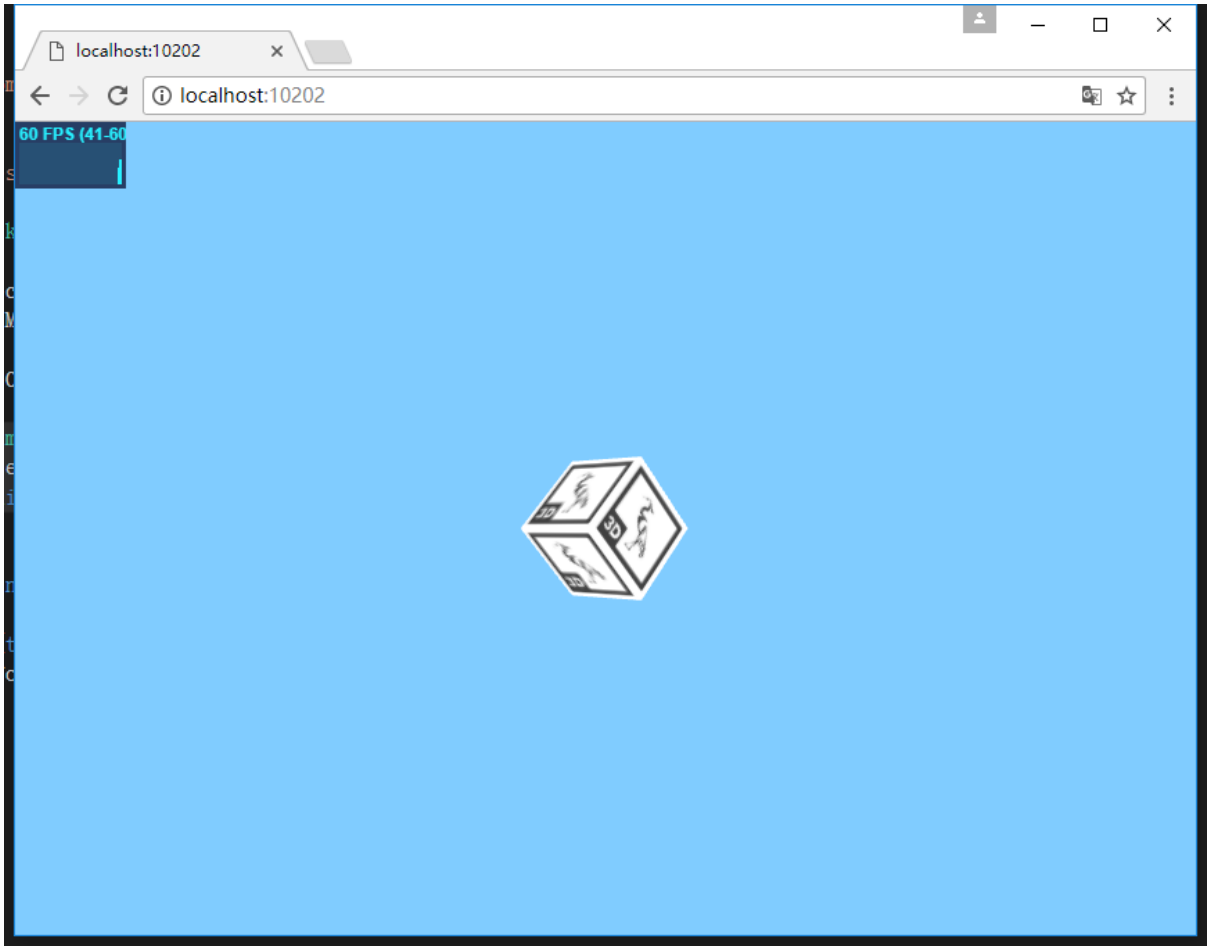
3、贴图加载完成后，创建一个材质，使用系统内置的 diffuse.shader，同时给材质加上贴图。

4、材质创建完成后，赋值给 meshRender 即可。

代码部分：

```
// 挂载材质贴图
this.app.getAssetMgr().load("resources/logo.png", egret3d.framework.AssetTypeEnum.Auto, (s1) => {
    let texture = this.app.getAssetMgr().getAssetByName("logo.png") as egret3d.framework.Texture;
    let mat = new egret3d.framework.Material();
    mat.setShader(this.app.getAssetMgr().getShader("diffuse.shader.json"));
    mat.setTexture("_MainTex", texture);
    renderer.materials = [];
    renderer.materials[0] = mat;
    cube.markDirty();
});
```

运行效果如下：




这样就完成了材质的替换。

但这里要注意的是：目前引擎支持的贴图格式是 png，jpg，而且 WebGL 限定，所有的贴图的长宽必须是 2 的 N 次方，否则加载出来的时候就会是一坨黑，控制台有警告提示。

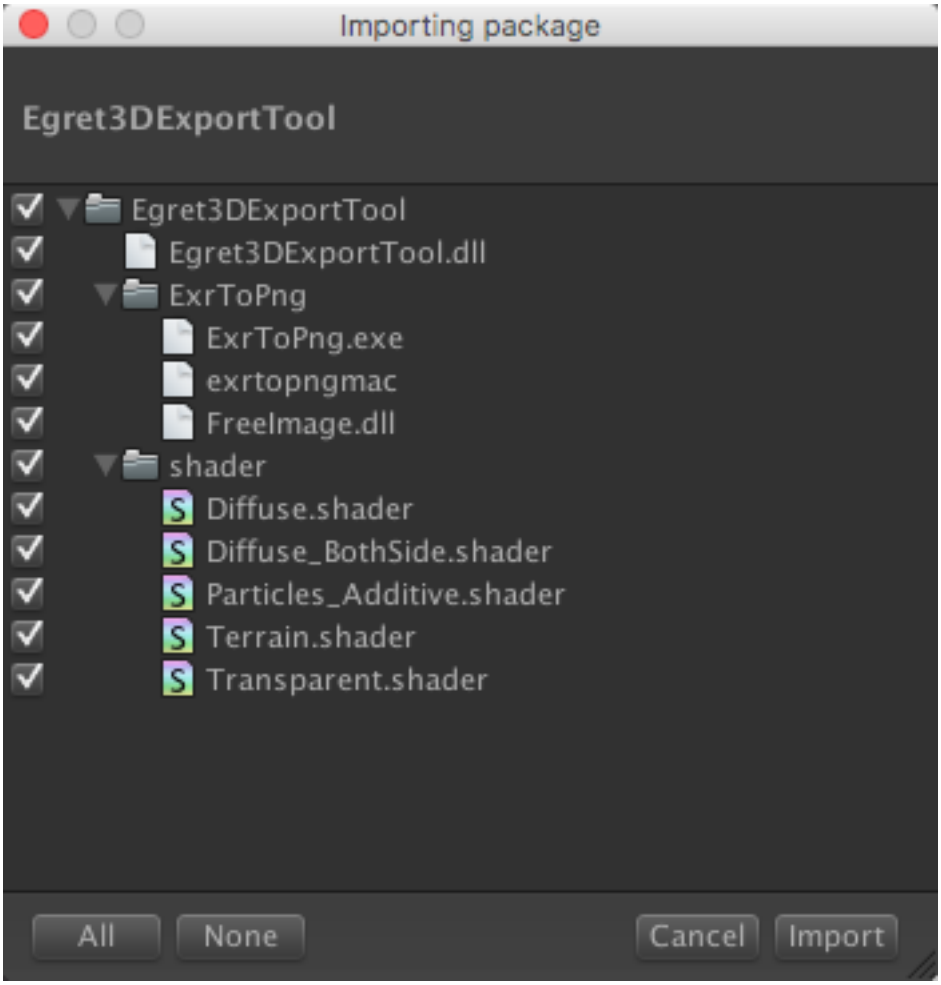
到目前位置我们可以用系统内置的模型+自定义的材质来绘制一个盒子了。下一篇我们来学习如何使用 unity 导出工具从 unity 中导出我们需要的资源来加载显示。

# 第三章 从 unity 导出资源

为了提升引擎的开发效率，我们提供了 Unity 导出工具，可以将 Unity 中的资源导出为 Egret 3D 引擎可用的格式，导出插件在第三章附件中。

这里以 Unity4.7.2 版本为例。其他版本由于 API 变更，可能会出现一些兼容问题。（我们之后会重点处理这些问题）

往 Unity 中导入工具。找到下载好的工具包并打开，将包中的所有内容导入到 Unity。



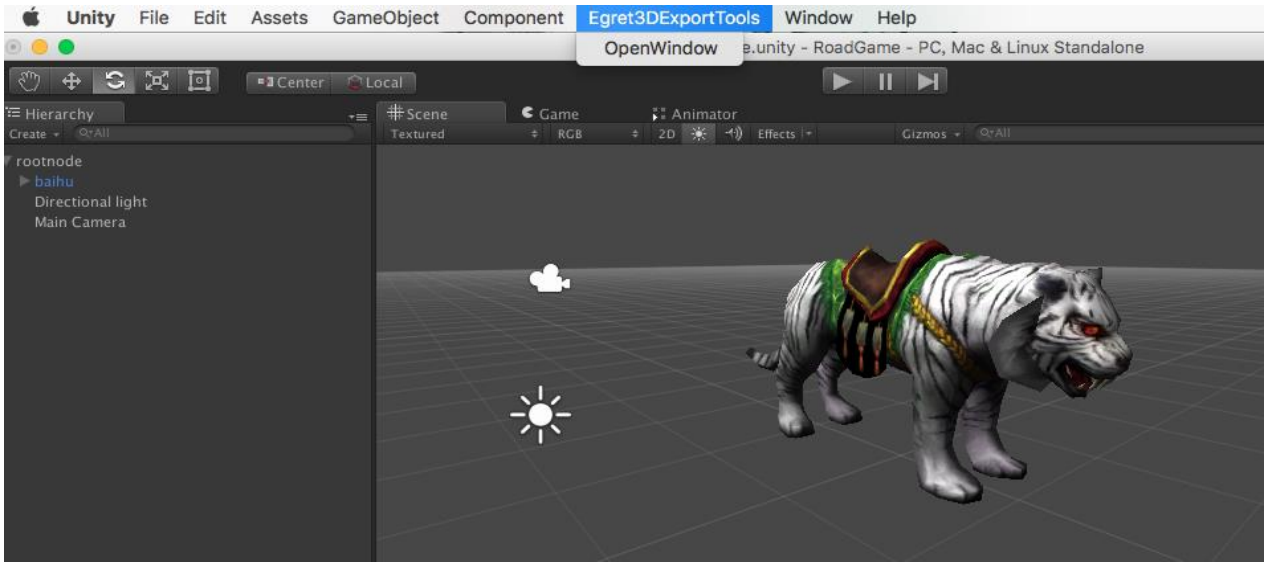

导入完成之后，就可以开始导出资源了。

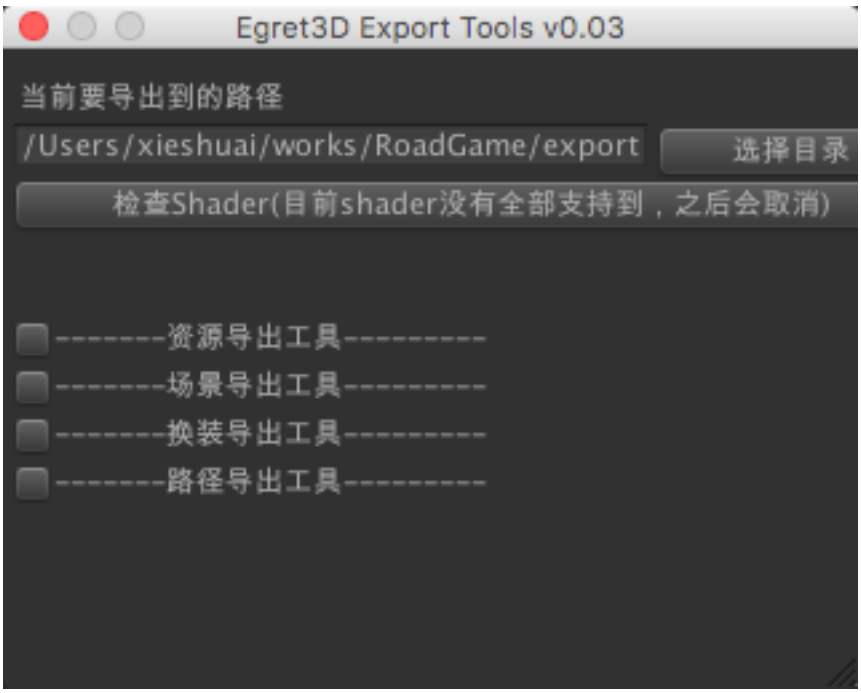
# 一、资源导出

## 1、静态模型的导出

我们首先在场景中摆一个模型。资源名称不要有特殊字符和空格


然后打开导出工具窗口



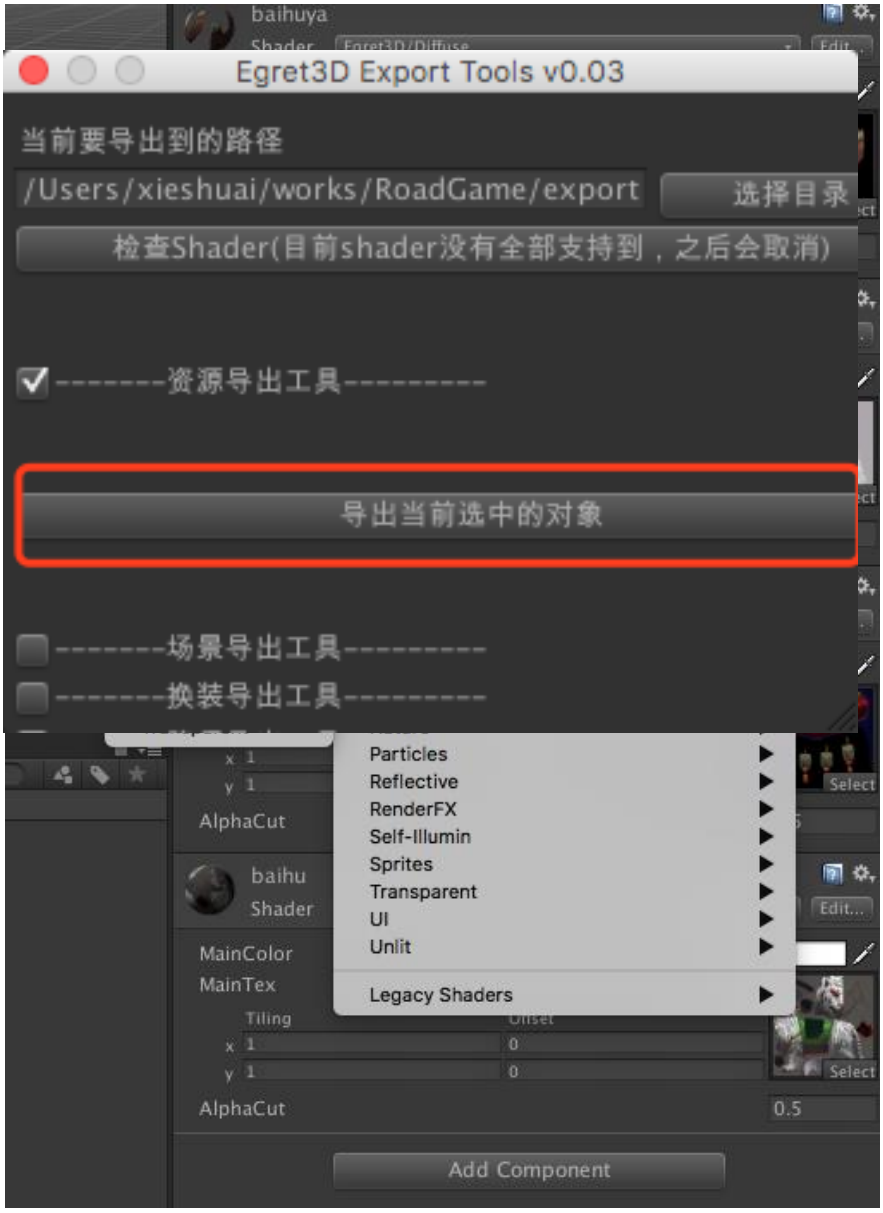


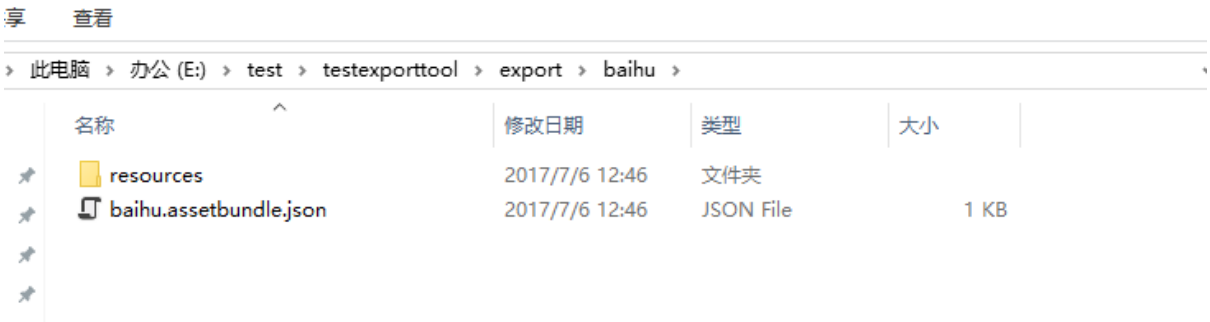

因为我们目前没有支持到全部 shader。为了保证导出的资源在引擎中能正常显示。在资源导出前要进行 shader 检查。检查之后，根据提示信息找到没有支持到的 shader 进行替换。目前支持的 shader 效果都是以 Egret 3D 开头。

全部更换完成之后，再次检测。如果没有提示了，就可以导出了

首先我们选中要导出的对象。然后勾选资源导出工具，点击 导出当前选中的对象

等待提示操作成功。资源就导出完成了。 如果导出未成功，请注意看 log



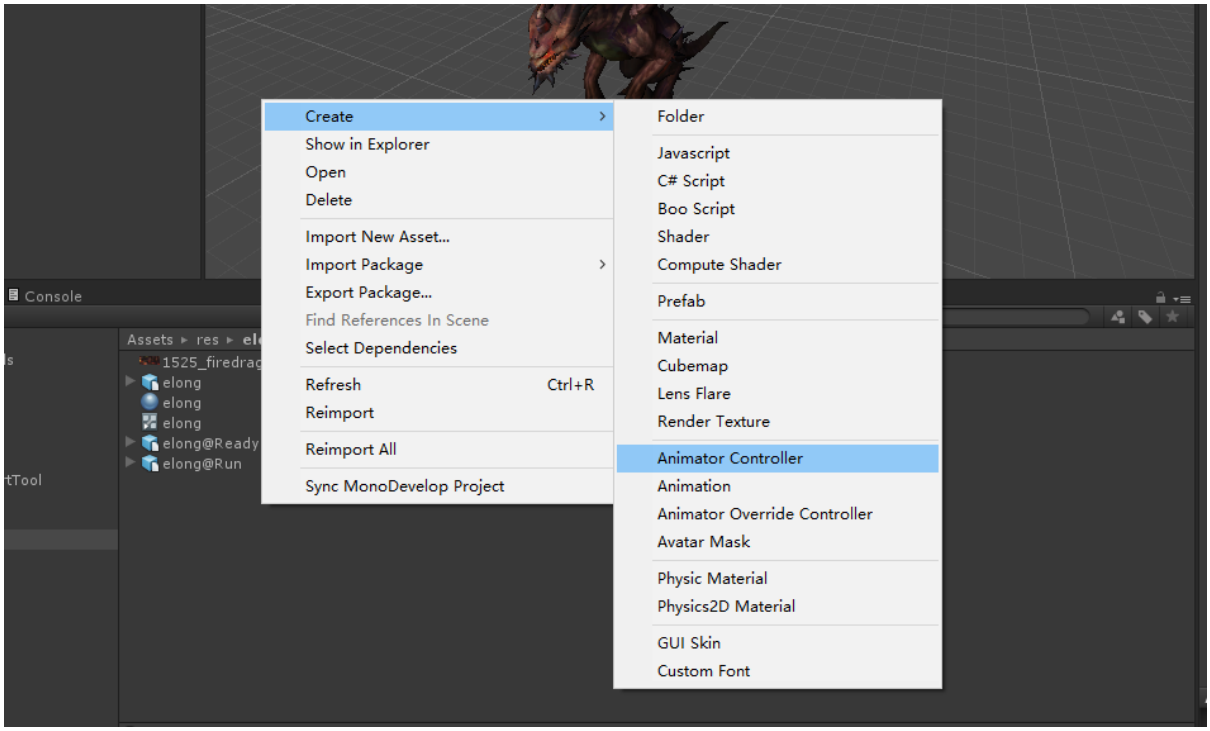




导出的资源就可以在 Egret 3D 引擎中使用了。在其他文章中会专门介绍导出的资源如何使用。

2、带骨骼动画的模型导出

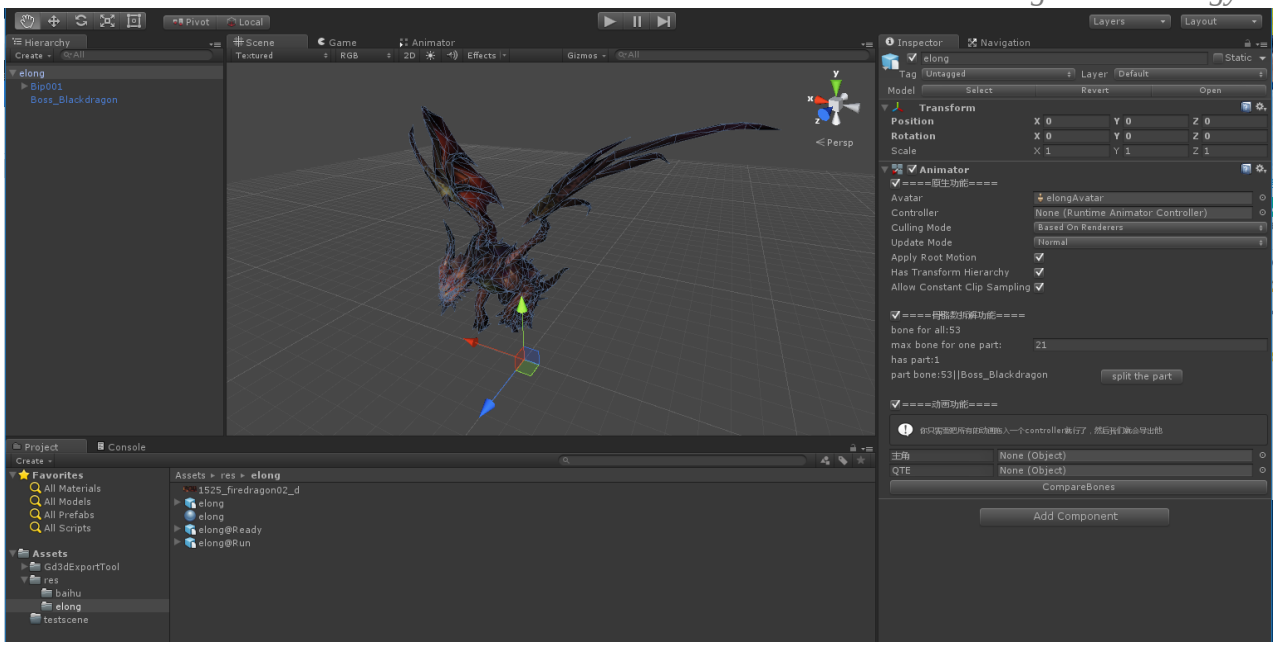
将动画的模型放到场景中。检查资源名称和 shader

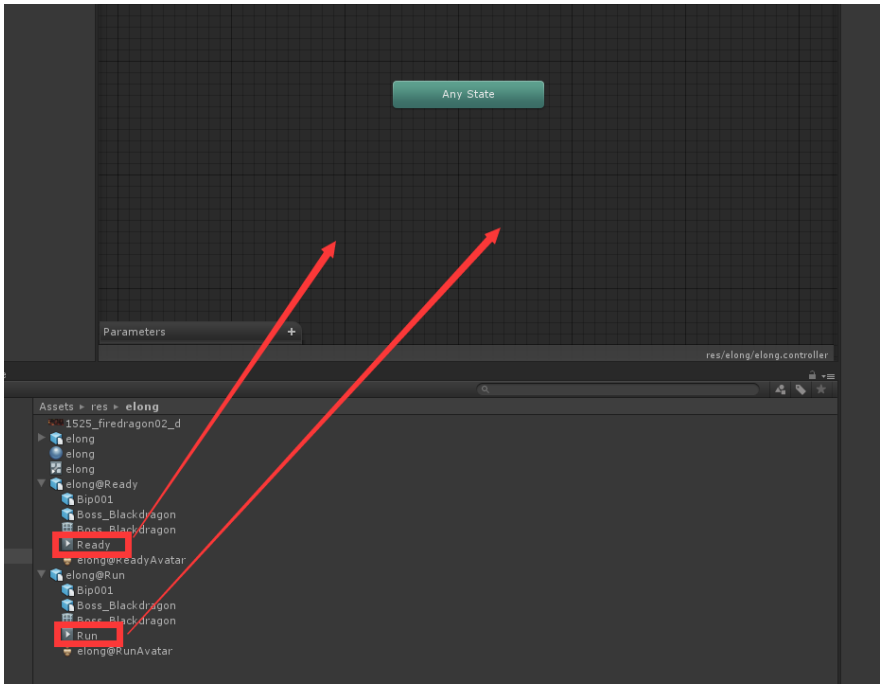
创建一个 Animator Controller。




双击创建好的 Animator Controller，进入编辑模式。

将模型的动画片段拖入进来

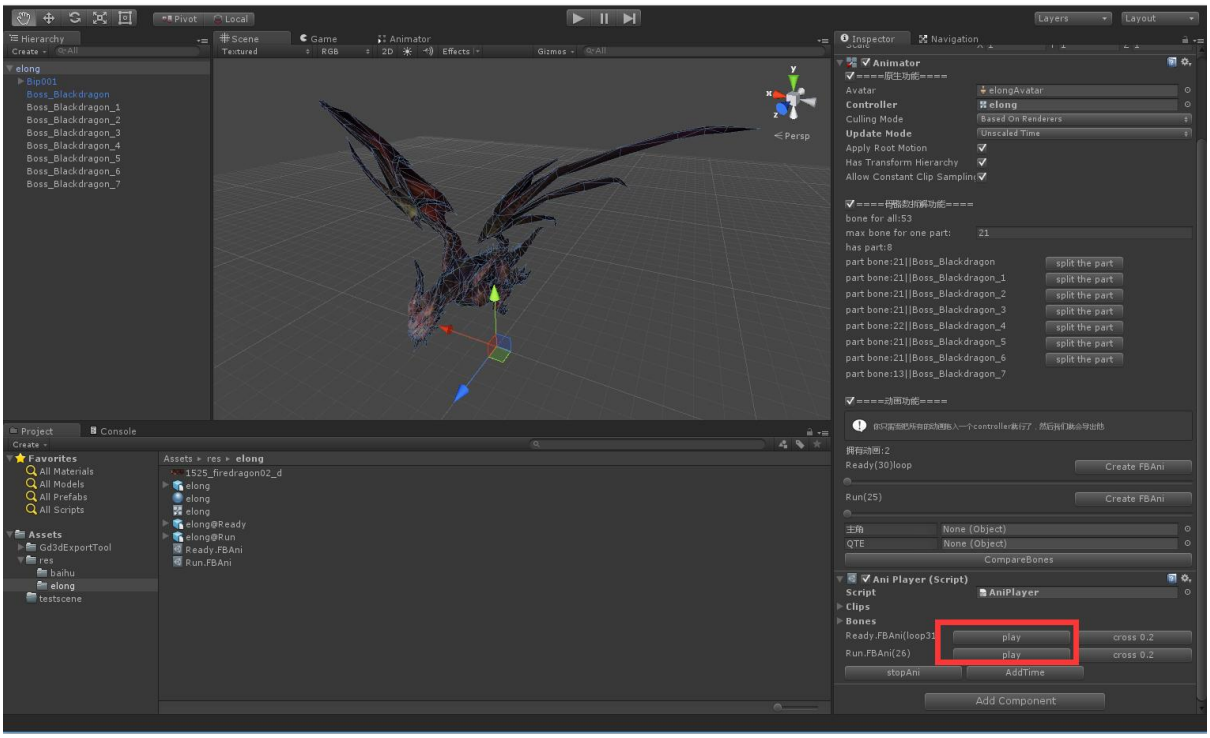




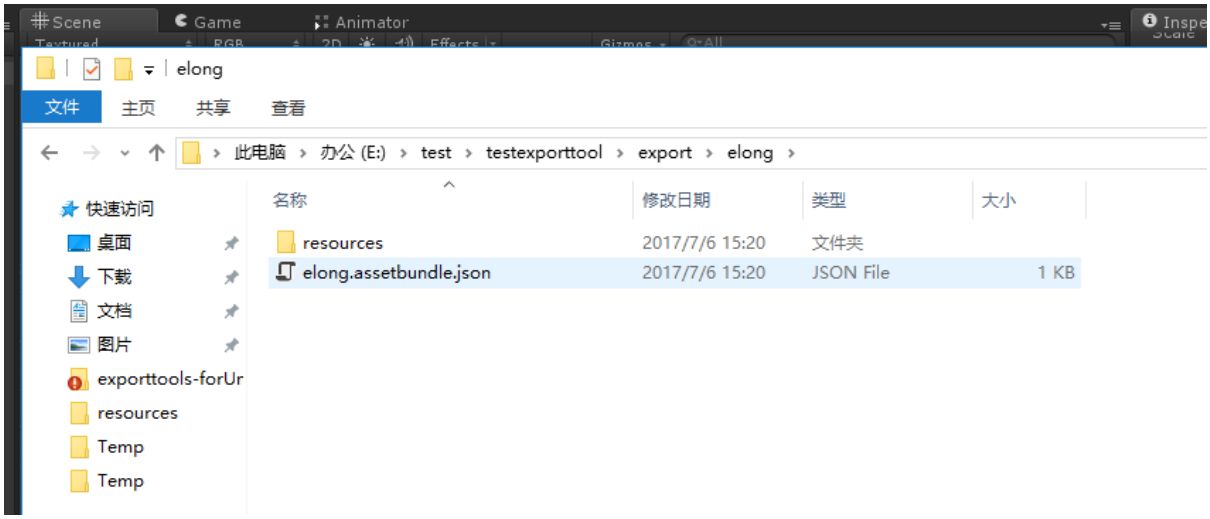

将编辑好的 Animator Controller 组件赋给模型的 Animator 组件。

选中要导出的对象。然后像之前导出资源那样，勾选资源导出工具，勾选「资源导出工具」，点击「导出当前选中的对象」

等待提示操作成功。资源就导出完成了。 如果导出未成功，请注意看 log 导出完成后，可以点击此处来预览动画。

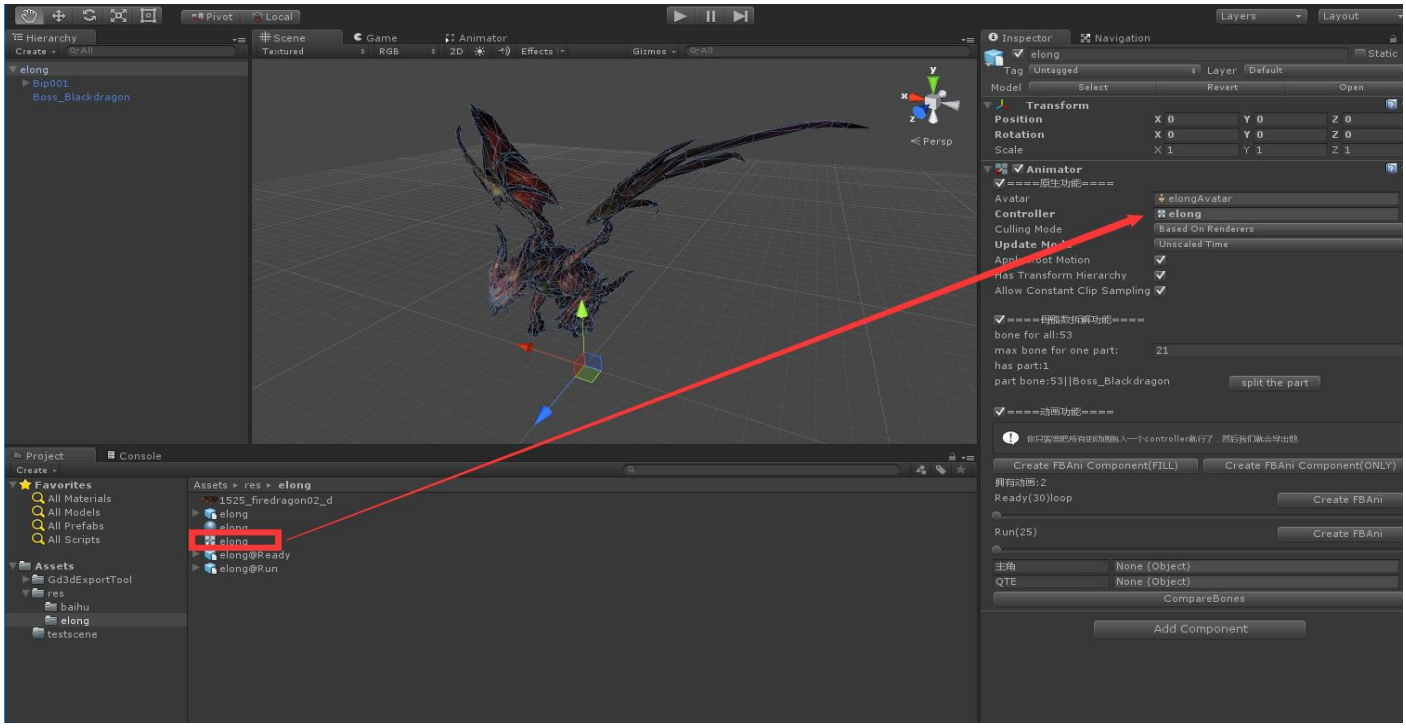


下面就是导出的骨骼动画资源。




在其他文章中会专门介绍导出的骨骼动画如何使用。

## 二、场景导出



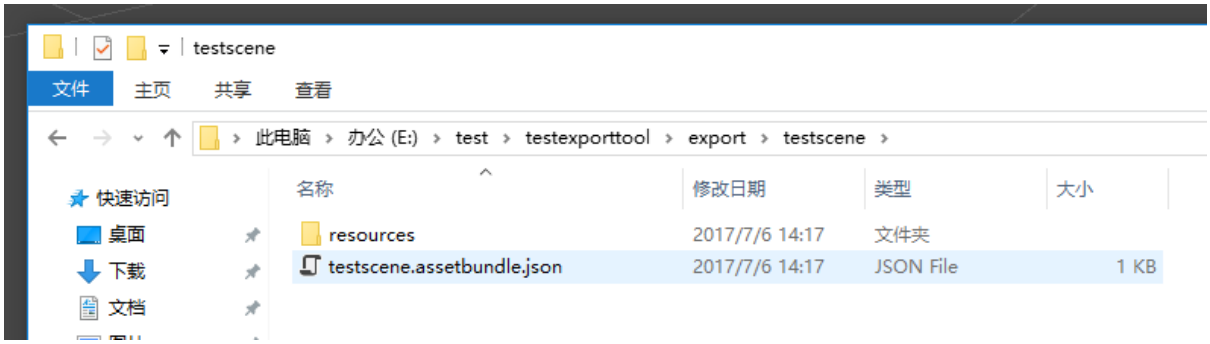
上面介绍的是场景中资源的导出。

如果要导出整个场景，场景中又烘焙了 lightmap 信息，那么就要用场景导出工具了。

场景导出和资源导出在导出前的处理上没什么差异，同样需要要注意材质的替换和资源的命名。

最后点击导出当前场景


下面就是导出的场景资源。



在下一章节会专门介绍导出的场景如何使用。

# 第四章 场景加载卸载

场景 Scene 做为 assetbundle 资源存储，加载为 rawscene 类型，包含了场景树结构、 lightmap 等信息

```
// 加载shader
let shaderAssetUrl = "resources/shader/shader.assetbundle.json";
this.app.getAssetMgr().loadCompressBundle(shaderAssetUrl, (state) => {
    if (state.isfinish) {
        // 加载并显示模型
        let assetUrl = "resources/scenes/test/test.assetbundle.json";
        this.app.getAssetMgr().load(assetUrl, egret3d.framework.AssetTypeEnum.Auto, (s) => {
            if (s.isfinish) {
                this.app.getAssetMgr().loadScene("test.scene.json", () => {
                    console.log("load over");
                })
            }
        });
    }
});
```

场景的加载:

效果如下：

场景的卸载分几步:

1.卸载当前场景，释放当前场景根节点，销毁场景树结构

```
this.app.getScene().getRoot().dispose();
```

2.释放 rawscene 资源，释放资源中场景树结构，lightmap 贴图计数减一

```
_scene.dispose();
```

3.卸载场景对应的 bundle，对应资源计数减一

```
this.app.getAssetMgr().unload("res/scenes/xxx/xxx.assetbundle.json");
```

4.调用 assetmgr 的资源释放接口，释放计数为 0 的资源

```
this.app.getAssetMgr().releaseUnuseAsset();
```

在上一篇中我们了解到，可以从 unity 导出场景，预设，以及骨骼动画，下一篇我们就来看下如何加载骨骼动画。骨骼动画导出后本质也是个预设，所以我们就放一个样例里面去学习。

# 第五章 静态模型以及骨骼动画资源的加载

```
// 加载shader
let shaderAssetUrl = "resources/shader/shader.assetbundle.json";
this.app.getAssetMgr().loadCompressBundle(shaderAssetUrl, (state) => {
    if (state.isfinish) {
        // 加载并显示模型
        let baihuAssetUrl = "resources/prefabs/baihu/baihu.assetbundle.json";
        this.app.getAssetMgr().load(baihuAssetUrl, egret3d.framework.AssetTypeEnum.Auto, (s) => {
            if (s.isfinish) {
                let prefab: egret3d.framework.Prefab = this.app.getAssetMgr().getAssetByName
                ("baihu.prefab.json") as egret3d.framework.Prefab;
                this.baihu = prefab.getCloneTrans();
                this.scene.addChild(this.baihu);
                this.baihu.markDirty();
            }
        });
    }
});
```

## 一、加载静态资源并添加到场景中

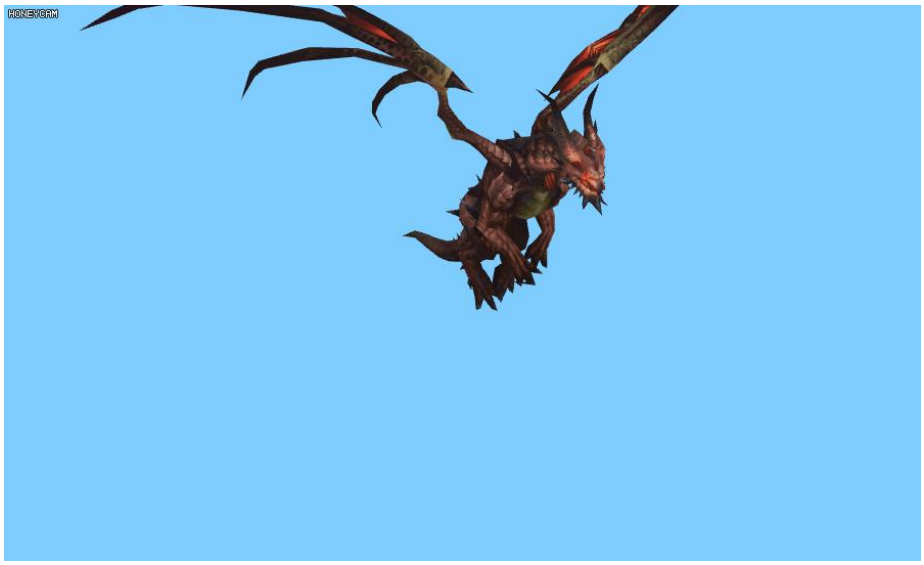
效果如下：

## 二、加载骨骼动画并播放

```
// 加载shader
let shaderAssetUrl = "resources/shader/shader.assetbundle.json";
this.app.getAssetMgr().loadCompressBundle(shaderAssetUrl, (state) => {
    if (state.isfinish) {
        // 加载并显示模型
        let baihuAssetUrl = "resources/prefabs/elong/elong.assetbundle.json";
        this.app.getAssetMgr().load(baihuAssetUrl, egret3d.framework.AssetTypeEnum.Auto, (s) => {
            if (s.isfinish) {
                let prefab: egret3d.framework.Prefab = this.app.getAssetMgr().getAssetByName
                ("elong.prefab.json") as egret3d.framework.Prefab;
                this.elong = prefab.getCloneTrans();
                this.elong.localEulerAngles = new egret3d.math.Vector3(0, 150, 0);
                this.elong.localScale = new egret3d.math.Vector3(0.5, 0.5, 0.5);
                this.scene.addChild(this.elong);
                this.elong.markDirty();

                // 动画控制
                let ap = this.elong.gameObject.getComponent
                (egret3d.framework.StringUtil.COMPONENT_ANIPLAYER) as egret3d.framework.AniPlayer;
                ap.playByIndex(0);
            }
        });
    }
});
```

效果如下：



# 第六章 特效的加载与释放

特效作为 assetbundle 包存在，使用正常的 assetbundle 加载流程即可

```
let name = "fx_ss_female@attack_04-";
//通过加载 assetbundle 的方式加载特效  这里使用的是加载压缩的 bundle
this.app.getAssetMgr().load("compressRes/particleEffect/" + name + "/" + name + ".assetbundle.json", egret3d.framework.AssetTypeEnum.Auto, (_state) =>
{
    //特效资源包括 1.特效配置  2.相关依赖图片 mesh 资源
    //加载完成后将特效配置提交给特效组件即可
    if (_state.isfinish)
    {
        this.tr = new egret3d.framework.Transform();
        //添加一个特效组件
        this.effect = this.tr.gameObject.addComponent(egret3d.framework.StringUtil.COMPONENT_EFFECTSYSTEM) as egret3d.framework.EffectSystem;
        //获取加载的特效配置文件
        var text: egret3d.framework.Textasset = this.app.getAssetMgr().getAssetByName(name + ".effect.json") as egret3d.framework.Textasset;
        //把特效配置提交给特效组件解析 释放
        this.effect.setJsonData(text);
        this.scene.addChild(this.tr);
        this.tr.markDirty();
        state.finish = true;
        this.effectloaded = true;
    }
}
);
```

到此已经讲完了所有资源的加载，下面我们看下关于事件的处理：鼠标拣选

# 第七章 鼠标拣选

引擎提供了射线类，可用于实现鼠标拾取操作

拾取分为 mesh 拾取和 collider 拾取 mesh 拾取等同于 meshcollider 的拾取

下面为 pick 示例代码 选中地面将 cube 移动过去

```
//添加一个盒子 作为地面
var cube = new egret3d.framework.Transform();
cube.name = "cube";

cube.localScale.x = 10;
cube.localScale.y = 0.1;
cube.localScale.z = 10;
this.scene.addChild(cube);
//添加渲染相关组件并设置 mesh
var mesh = cube.gameObject.addComponent("MeshFilter") as
egret3d.framework.MeshFilter;
mesh.mesh = (this.app.getAssetMgr().getDefaultMesh("Cube"));
var renderer = cube.gameObject.addComponent("MeshRenderer") as
egret3d.framework.meshRenderer;
//本次测试为 colliderpick 需要添加 collider
cube.gameObject.addComponent("Boxcollider") as egret3d.framework.Boxcollider;
cube.markDirty();

//添加一个盒子作为被移动物体
this.cube2 = new egret3d.framework.Transform();
this.cube2.name = "cube2";
this.scene.addChild(this.cube2);
this.cube2.localScale.x = this.cube2.localScale.y = this.cube2.localScale.z = 1;
this.cube2.localTranslate.x = -5;
this.cube2.markDirty();
var mesh = this.cube2.gameObject.addComponent("MeshFilter") as
egret3d.framework.meshFilter;
mesh.mesh = (smesh);
```

```
var renderer = this.cube2.gameObject.addComponent("MeshRenderer") as
egret3d.framework.meshRenderer;

//在 update 中检测按下事件并实现移动操作
timer: number = 0;
movetarget: egret3d.math.vector3 = new egret3d.math.Vector3();
inputMgr: egret3d.framework.inputMgr;//键盘、鼠标(触屏)事件管理类
pointDown: boolean = false;
update(delta: number)
{
        //上一帧不是按下状态  当前为按下状态 即为按下
        if (this.pointDown == false && this.inputMgr.point.touch == true)
        {
           //根据当前鼠标按下的屏幕位置创建射线
           var ray = this.camera.creatRayByScreen(new
egret3d.math.Vector2(this.inputMgr.point.x, this.inputMgr.point.y), this.app);
           //调用 pick 方法 默认为 pick collider
           var pickinfo = this.scene.pick(ray);
           if (pickinfo != null)
           {
                //pickinfo 中记录的交点的位置信息
                this.movetarget = pickinfo.hitposition;
                this.timer = 0;
           }
        }
        //作为上一帧的鼠标状态
        this.pointDown = this.inputMgr.point.touch;

        this.timer += delta;
        //把 cube 移动到拣选的位置上
        egret3d.math.vec3SLerp(this.cube2.localTranslate, this.movetarget,
this.timer, this.cube2.localTranslate);
        //必须 markdirty 一下才会更新
        this.cube2.markDirty();
}
```