# 1.场景管理物体树形结构

|  |
| --- |
|  |
|  |

# 2.组件化开发模式

|  |
| --- |
|  |

# 3.组件化开发流程详解

|  |
| --- |
|  |
|  |
|  |

# 4.演练/示例

## 1）新建一个项目，起名classFive，然后在assets里面新建3个文件夹，AssetsPackage，Scenes和Scripts并且把当前场景保存到Scenes文件夹里面

|  |
| --- |
|  |

## 2）把场景中默认的东西删除，然后添加一个立方体和一个摄像机

|  |
| --- |
|  |

## 3）慢慢调整摄像机的位置使得它能够看到立方体

|  |
| --- |
|  |

## 4）我们可以在立方体里面创建另外一个物体，如球体，然后在这个物体里面有可以创建另外一个物体如圆柱体

|  |
| --- |
|  |

### 此时，立方体是场景的孩子，球体是立方体的孩子，柱体的球体的孩子，是立方体的孙子，这种结构就叫做树形结构，这种结构的好处是，当我们移动父节点，他的所有子孙节点都会跟着移动。

## 5）在Scripts文件夹上面点击右键-》创建-》Typescript，然后起个名字，如：GameMgr

|  |
| --- |
|  |
|  |

## 6）注意，组件需要节点作为载体，所以你必须选择需要使用该组件的节点，如cube，效果如下

|  |
| --- |
|  |

## 7)然后点击添加属性检查器的添加组件按钮，点击自定义脚本，就会看见我们的组件类

|  |
| --- |
|  |

### 这个步骤就是下面第三节的步骤一，新建一个组件类被引擎识别，此时双击GameMgr类，就会打开vscode，因为我们前面配置了vscode作为默认的代码编辑器，

|  |
| --- |
|  |

### ，把一些不必要的代码删除，一个组件类的模板是这样子的

|  |
| --- |
|  |

### 选择一个或者组件，如\_decorator,点击右键，-》goto Definition，就可以进入这个类的声明，他是在一个叫做cc.d.ts文件

|  |
| --- |
|  |

### 我们发现，他是一个命名空间，它定义了很多装饰器

|  |
| --- |
|  |

## 8）当ts文件编译的时候，也是以cc.d.ts作为帮助来进行语法检查的，如我们故意写一些错误的代码，就会出现一条红色的波浪线，说明引擎检查出来错误，null和Vec3类型不匹配

|  |
| --- |
|  |

## 9）在第7个步骤的时候如果我们点击GameMgr脚本，就会在节点上面新建一个这个组件类的实例

|  |
| --- |
|  |

## 10）创建后你还可以继续创建组件实例

|  |
| --- |
|  |

## 11）组件类里面的onLoad方法会在引擎第一次调用组件的时候被调用，可以用来初始化数据，组件类的start方法也可以用来初始化数据，onLoad方法绘制start方法前面调用

## 12）如果你需要添加一个数据成员并且使他可以在引擎的编辑器里面可视化编辑，我们需要定义一个数据成员并且给他添加property装饰器，

### 如我们添加一个isDebug成员

|  |
| --- |
|  |

## 13）此时我们点击Cube节点，就会看见我们刚刚添加的数据成员

|  |
| --- |
|  |

## 14）我们在onLoad方法和start方法里面都添加一些输出代码，并且在start方法里面输出isDebug数据成员的值，此时我们并没有修改isDebug的值，运行场景，效果如下

|  |
| --- |
|  |

## 15）然后我们在编辑器中勾选isDebug，就相当于把它设置为true，此时刷新页面，效果如下

|  |
| --- |
|  |
|  |

## 16）我们打开update方法并且添加一条输出语句

|  |
| --- |
|  |

## 17）运行场景，效果如下

|  |
| --- |
|  |