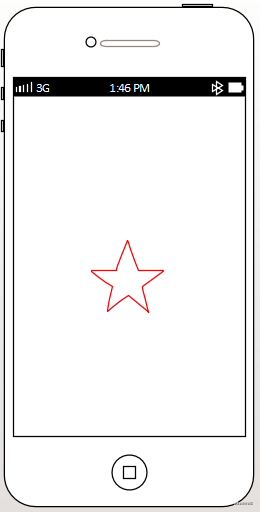
# 前言

* 前不久有兄弟给我留言，说是想了解游戏的渲染引擎，看了很多关于Cocos游戏渲染的分析，知道他们讲的是这个道理，但就始终还是串不上线。
* 熊猫根据自己的理解，在这里输出自己的分析，希望能有所帮助，也欢迎兄弟们留言任何需要一起分析的内容。
* 游戏渲染引擎一直以来都是各个游戏引擎的核心竞争力，包括Cocos、Laya、Unity游戏等，他们都自成一派拥有自己的渲染引擎。即使各有千秋，但深究后也发现他们其实都是一个派系——**拍戏，就是游戏渲染**。

# 游戏渲染，就是拍戏

人生如戏，戏如人生，而游戏本就是一出戏。Cocos游戏的渲染，就是一场戏场景（Scene）的拍摄，游戏场景一般包括：摄像机（Camera）摆放，场景物体（UI、Component）布置，模特（Model）选角，到演员动作剧本（Script)等，都是一场戏的元素（Node)，也是呈现给玩家所有游戏内容的载体（Item）。

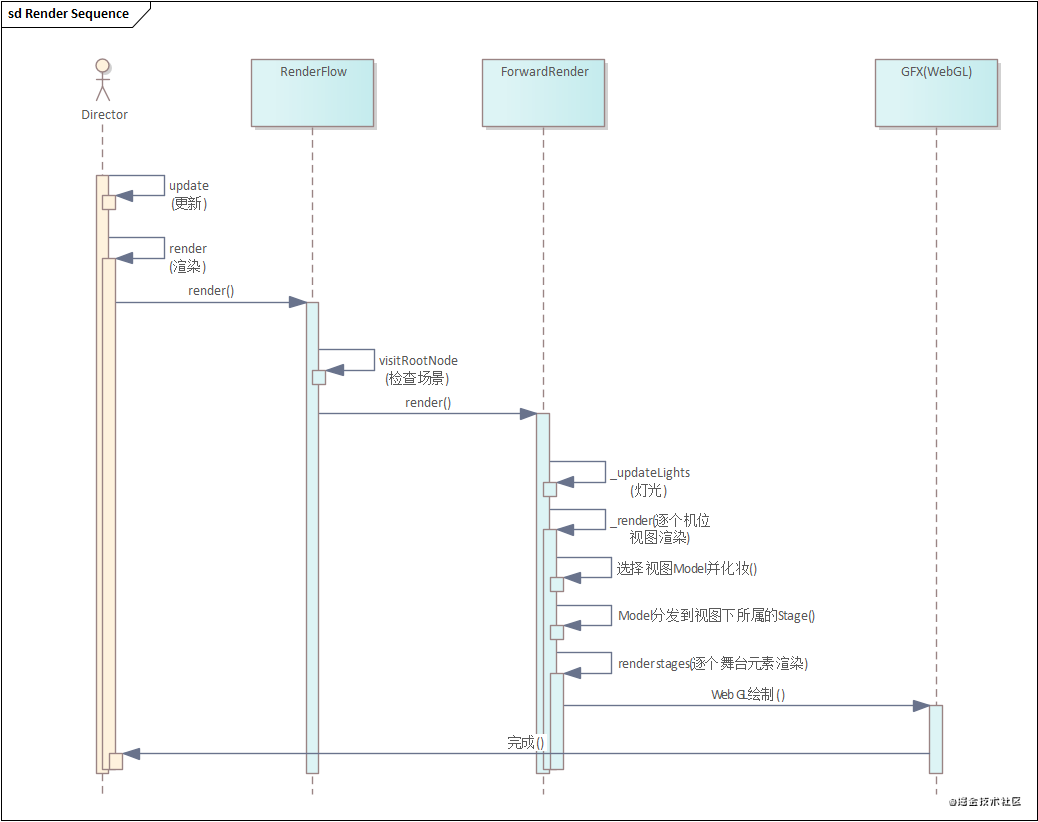
熊猫习惯做事，先要知道想达到什么目的。所以在分析渲染之前，先让大家知道，我们最终想在屏幕上渲染出这么一个图形：



#### 下面开始进入内容

# 游戏渲染

首先我们虽然会讲得跟现实挂钩，要抽象易懂，但是还是基于源码分析，所以先从整个Cocos源码角度看渲染流程。细节后面分解，逻辑如下图：



## 要素分解

正如前面提及的，下面我们会涉及到的游戏渲染元素有：

* **Director**：Cocos渲染引擎的导演类，负责统筹游戏渲染；
* **Camera**：Cocos渲染引擎的摄像机类，主要负责控制给玩家呈现的屏幕视觉效果。一个场景中，可以有多个Camera，比如我们玩英雄联盟时的小地图，就是第二个Camera；
* **View**：与Camera一一对应，决定当前渲染的视图大小；
* **Stage**：Cocos游戏引擎中的舞台类，是View的子节点，一个View可以有多个Stage。所有的渲染元素，都要先形成舞台元素，才能进行渲染。

### 1. 导演类Director

导演，是创作影视作品的组织者和领导者，是借助演员表达自己思想的艺术家，是把影视文学剧本搬上荧屏的总负责人。——百度百科

导演（Director）是游戏的组织者，Cocos游戏中的Director类严格按照帧率，更新（update）游戏场景状态，并进行渲染（render）呈现给玩家所需的视觉（View）效果，这个动作贯穿了整个游戏的渲染。另外，update、render的循环也几乎是所有游戏引擎的主循环（mainLoop）。

* 源码：Creator\2.4.5\resources\engine\cocos2d\core\CCDirector.js

/\*

\* Run main loop of director

\*/

mainLoop: CC\_EDITOR ? function (deltaTime, updateAnimate) {

...

// Update，反正这里在做着一些更新场景的事情

if (!this.\_paused) {

this.emit(cc.Director.EVENT\_BEFORE\_UPDATE);

this.\_compScheduler.startPhase();

this.\_compScheduler.updatePhase(deltaTime);

if (updateAnimate) {

this.\_scheduler.update(deltaTime);

}

this.\_compScheduler.lateUpdatePhase(deltaTime);

this.emit(cc.Director.EVENT\_AFTER\_UPDATE);

}

// Render

this.emit(cc.Director.EVENT\_BEFORE\_DRAW);

renderer.render(this.\_scene, deltaTime); // 场景更新好了，可以开始进行渲染

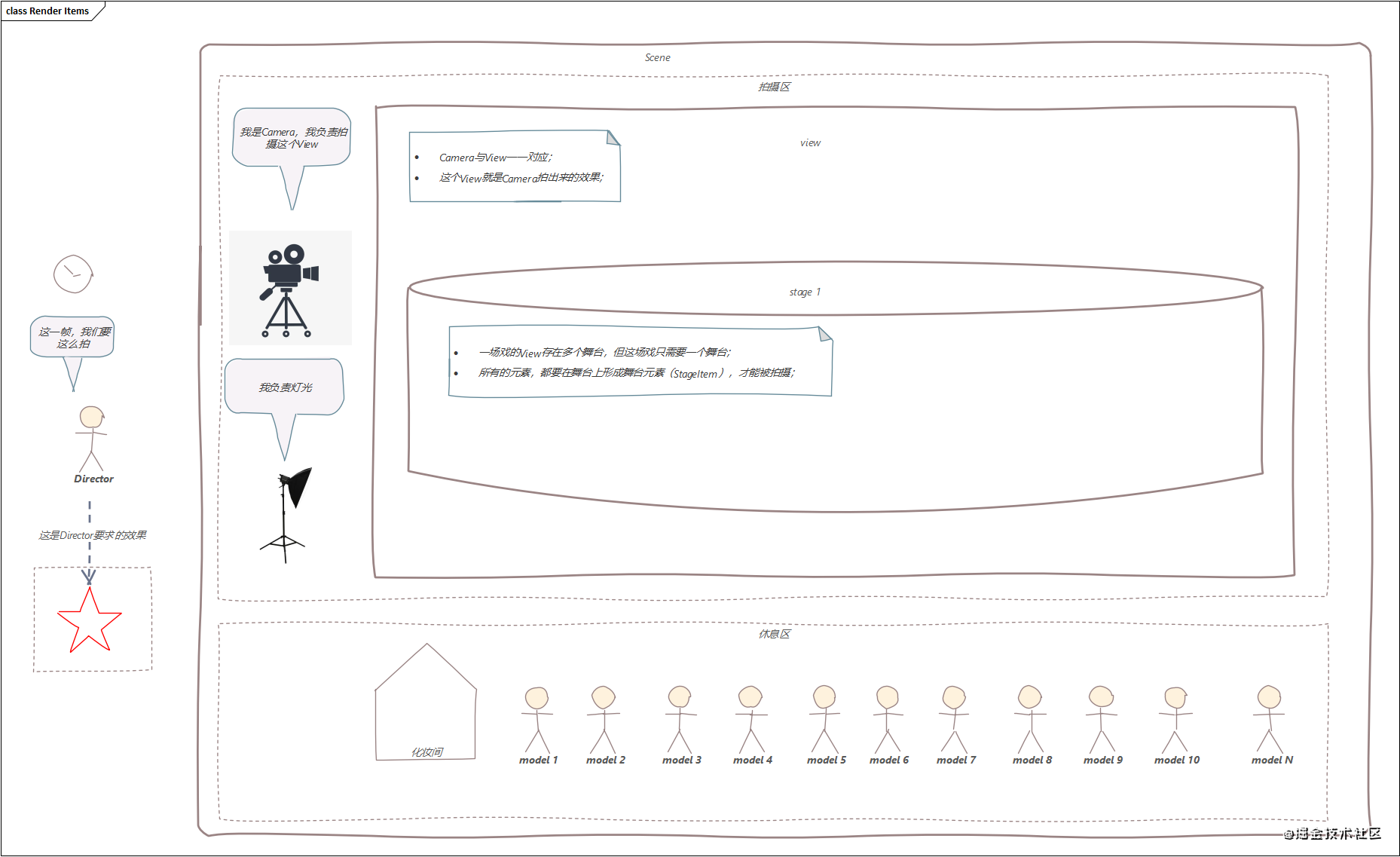
// After draw

this.emit(cc.Director.EVENT\_AFTER\_DRAW);

this.\_totalFrames++;

}复制代码

导演要求每次update后，都要把当前Scene的要素准备好，才能进行渲染。因为本章节，主要围绕渲染展开，所以我们默认已经拿到了update好的Scene，如下图：



### 2. 渲染流程RenderFlow

Cosos游戏有个专门的类负责驱动渲染流程RenderFlow，导演类Director拿到当前帧更新好的场景后，就丢给RenderFlow进行渲染。

* RenderFlow.render：场景渲染的入口，入参根节点rootNode就是场景Scene，所有场景有关的元素，都挂在该节点下；
* 源码：Creator\2.4.5\resources\engine\cocos2d\core\renderer\render-flow.js

RenderFlow.render = function (rootNode, dt) {

\_batcher.reset();

\_batcher.walking = true;

RenderFlow.visitRootNode(rootNode); // 从根节点开始遍历，准备场景

\_batcher.terminate();

\_batcher.walking = false;

\_forward.render(\_batcher.\_renderScene, dt); // Action~开始拍摄

};复制代码

* render方法主要做两件事：RenderFlow.visitRootNode遍历整个场景，根据update时做的一些标记Flag，准备场景需要。很明显，这也不是我们的重头戏，直接转至下一步；
* ForwardRender.render：这是本章节重点，从这里开始，导演就是拿着准备好的场景状态，要开始拍摄（render）了；

### 3. 渲染执行ForwardRender

#### 3.1 ForwardRender.render 准备视图

ForwardRender.render有三个步骤：

1. 场景打灯：\_updateLights(scene)，就我们前面说的，场景准备好后，就要把灯光打亮，让观众注意过来；
2. 设置摄像机（Camera）：每个Camera都对着一个视图（View）拍摄，在Cocos游戏里也是如此，因此要先从Cameras中的试图列举出来；
3. 逐个视图拍摄：\_render(view, scene)，这里我们先不提及WebGL相关的知识及操作，因此Device类相关的动作可以先忽略；

render (scene, dt) {

this.reset();

if (!CC\_EDITOR) {

if (dt) {

this.\_time[0] += dt;

this.\_time[1] = dt;

this.\_time[2] ++;

}

this.\_device.setUniform('cc\_time', this.\_time); // WebGL相关操作

}

this.\_updateLights(scene); // 1. 灯光师，打灯

const canvas = this.\_device.\_gl.canvas;

for (let i = 0; i < scene.\_cameras.length; ++i) {

let view = this.\_requestView();

let width = canvas.width;

let height = canvas.height;

let camera = scene.\_cameras.data[i];

camera.extractView(view, width, height); // 2. 摄像机就位

}

// render by cameras

this.\_viewPools.sort(sortView);

for (let i = 0; i < this.\_viewPools.length; ++i) {

let view = this.\_viewPools.data[i];

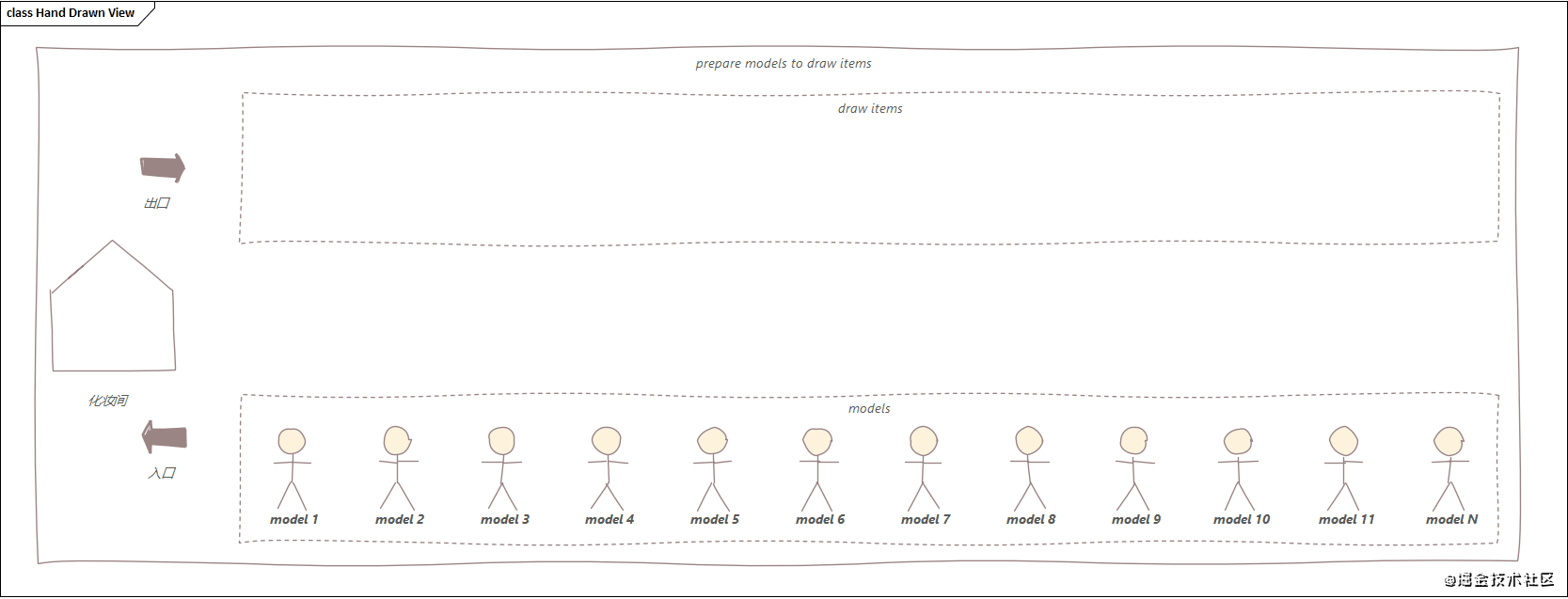
this.\_render(view, scene); // 3. 开始拍摄

}

}复制代码

#### 3.2 ForwardRender.\_render 逐一渲染

根据前面拿到的机位拍摄视图列表，Cocos游戏引擎会逐个视图进行渲染。本例中，我们只看一个视图的渲染。视图渲染时，导演要求先把当前几位要拍摄的演员筛选出来，并进行化妆，符合拍摄元素（drawItem）的要求。



* 源码：Creator\2.4.5\resources\engine\cocos2d\renderer\core\base-renderer.js

\_render (view, scene) {

...

// get all draw items

this.\_drawItemsPools.reset();

for (let i = 0; i < scene.\_models.length; ++i) {

let model = scene.\_models.data[i];

// filter model by view

if ((model.\_cullingMask & view.\_cullingMask) === 0) {

continue; // 筛选模特，不符合的就下一位

}

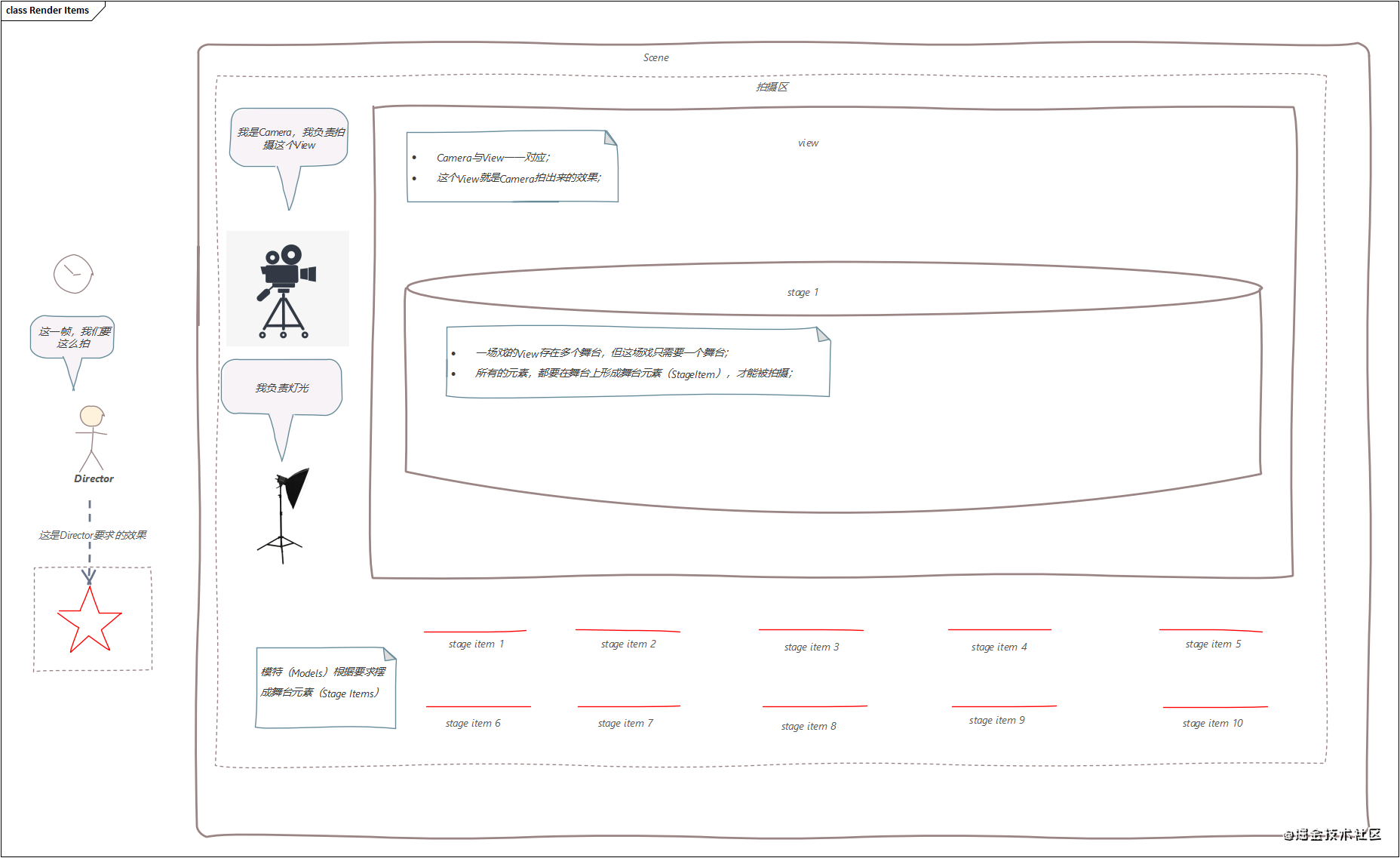
let drawItem = this.\_drawItemsPools.add();

model.extractDrawItem(drawItem); // 模特化妆

}

复制代码

演员（Model）就位后-->（DrawItem），导演把每个拍摄的视图分为几个小舞台（Stage），每个拍摄元素根据场景需要，到对应的舞台摆好姿势，形成舞台元素（StageItem），并对当前场景进行编排。



在本例中，因为Scene需要形成一个红色的五角星，所以形成需要10个短横线的舞台元素（StageItem），并编排组成场景需要的图案。

* 源码：Creator\2.4.5\resources\engine\cocos2d\renderer\core\base-renderer.js

\_render (view, scene) {

...

// dispatch draw items to different stage

\_stageInfos.reset();

for (let i = 0; i < view.\_stages.length; ++i) {

let stage = view.\_stages[i];

let stageItems = this.\_stageItemsPools.add();

stageItems.reset();

for (let j = 0; j < this.\_drawItemsPools.length; ++j) {

let drawItem = this.\_drawItemsPools.data[j];

let passes = drawItem.effect.stagePasses[stage];

if (!passes || passes.length === 0) continue;

let stageItem = stageItems.add();

stageItem.passes = passes;

stageItem.model = drawItem.model;

stageItem.node = drawItem.node;

stageItem.ia = drawItem.ia;

stageItem.effect = drawItem.effect;

stageItem.defines = drawItem.defines;

stageItem.sortKey = -1;

stageItem.uniforms = drawItem.uniforms;

}

let stageInfo = \_stageInfos.add();

stageInfo.stage = stage;

stageInfo.items = stageItems;

}

// render stages

for (let i = 0; i < \_stageInfos.length; ++i) {

let info = \_stageInfos.data[i];

let fn = this.\_stage2fn[info.stage];

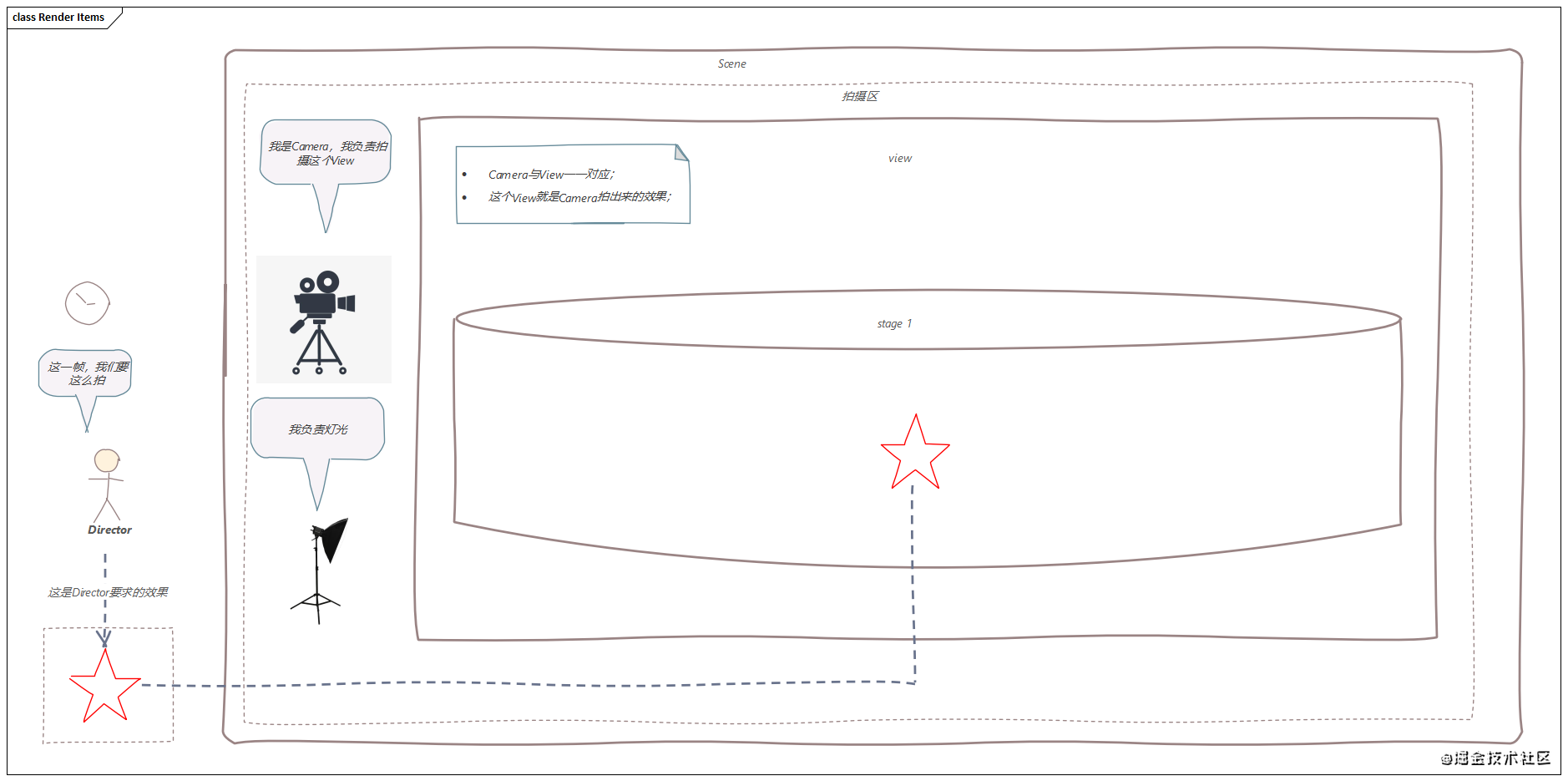
fn(view, info.items);

}

}复制代码

#### 3.3 ForwardRenderer.\_drawItems

最后在轮询每个Stage，\_drawItems方法负责将StageItems逐个画出来，形成场景需要呈现的效果



* 源码：Creator\2.4.5\resources\engine\cocos2d\renderer\renderers\forward-renderer.js

\_drawItems (view, items) {

let shadowLights = this.\_shadowLights;

if (shadowLights.length === 0 && this.\_numLights === 0) {

for (let i = 0; i < items.length; ++i) {

let item = items.data[i];

this.\_draw(item);

}

}

else {

for (let i = 0; i < items.length; ++i) {

let item = items.data[i];

for (let shadowIdx = 0; shadowIdx < shadowLights.length; ++shadowIdx) {

this.\_device.setTexture('cc\_shadow\_map\_'+shadowIdx, shadowLights[shadowIdx].shadowMap, this.\_allocTextureUnit());

}

this.\_draw(item);

}

}

}复制代码

#### 3.5 ForwardRenderer\_draw

从这里开始，就开始调用WebGL接口的特性进行绘制了。这块特性，我们放在下节说。

以上，整个一帧的场景渲染流程，就如我们所说的，跟拍戏一样，有流程、有章法、有逻辑。

#### 你鞋废了吗？

#### 原创作品，转载请注明出处！

#### 更多精彩，请持续关注！

作者：pandafactory  
链接：https://juejin.cn/post/6964779636412645384  
来源：掘金  
著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。