Advance Web Technology

高级Web技术

多人VR环境开发指北

Content

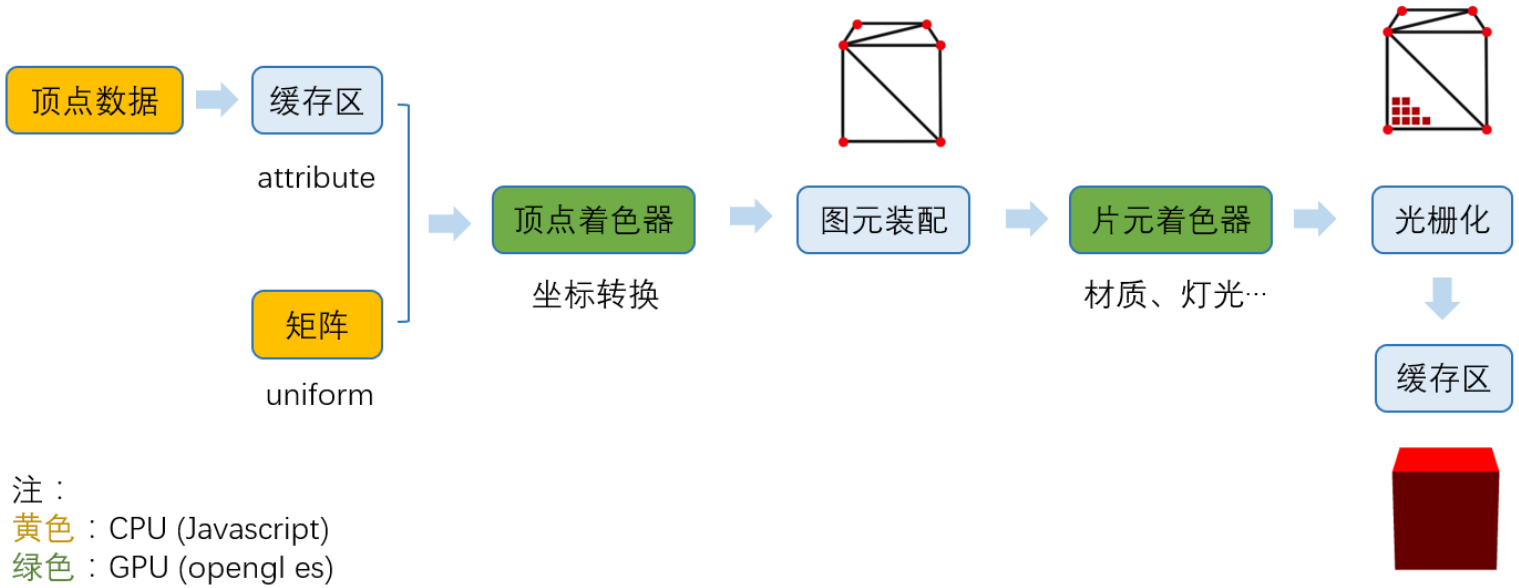
1. WebGL/three.js 实现本地VR环境

2. WebSocket/socket.io 实现多人联机功能

3. 一些架构建议

WebGL

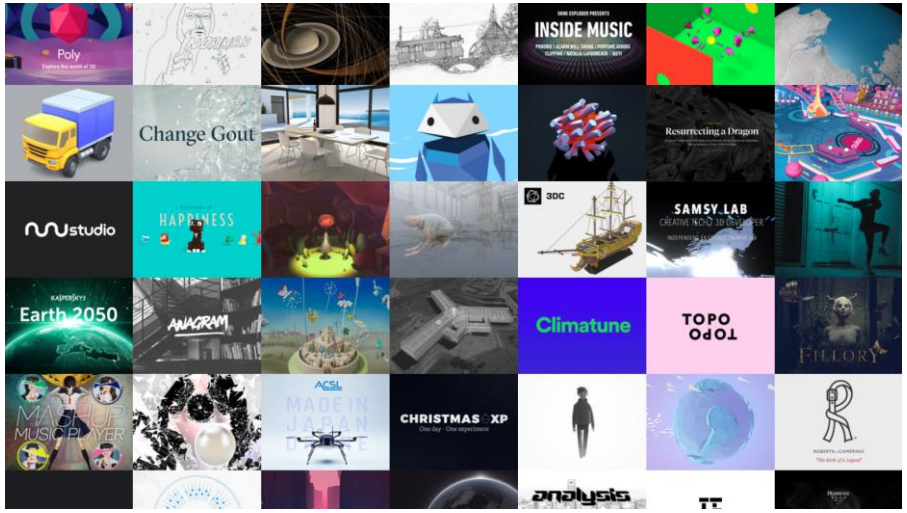
- **WebGL (Web图形库) 是一种JavaScript API，用于在任何兼容的Web浏览器中呈现交互式3D和2D图形，而无需使用插件。WebGL通过引入一个与OpenGL ES 2.0紧密相符合的API，可以在HTML5 <canvas> 元素中使用。**



three.js

- three.js 是一个 WebGL 库，对 WebGL API 进行了很好的封装。它库函数丰富，上手容易，非常适合 WebGL 开发。
- 官网：https://threejs.org/
- Github：https://github.com/mrdoob/three.js/

- three.js 的基础用法已在上个 lab 介绍



three.js 重要概念

- 参考官方文档：https://threejs.org/docs

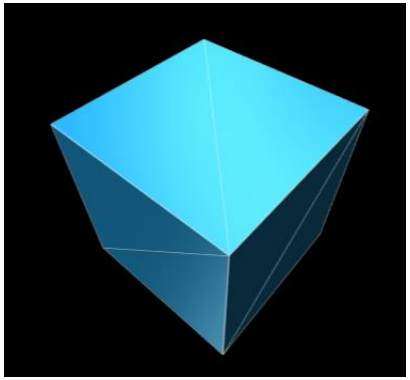
- 场景：Scene，所有游戏物体的容器
- 摄像机：Camera，看场景的角度
- 渲染器：Renderer，渲染场景到 canvas 上

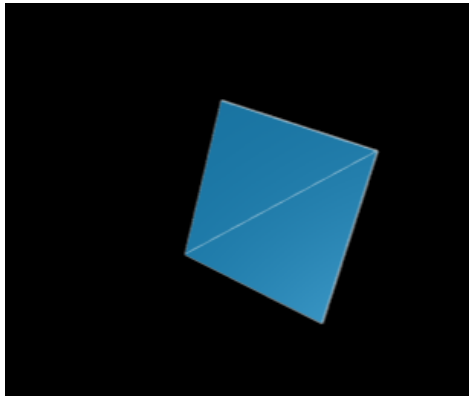
- 渲染静态场景：

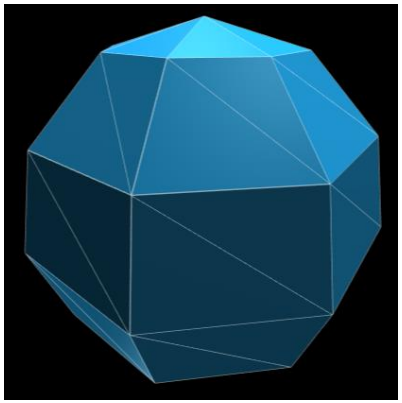
```
renderer.render( scene, camera );
```

three.js 重要概念

- 几何形状：Geometry
  - 立方体（BoxGeometry）
  - 平面（PlaneGeometry）
  - 球体（SphereGeometry）
  - 立体文字（TextGeometry）



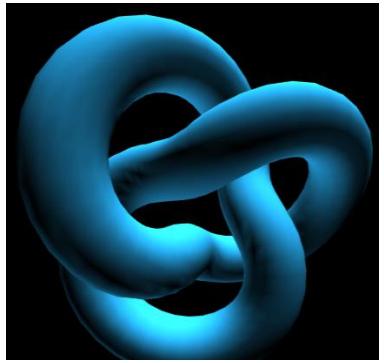




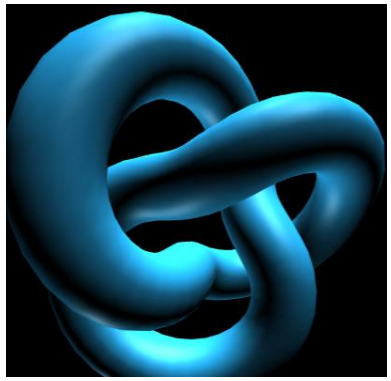
- 建议参考官方文档学习

three.js 重要概念

- **材质：Material**
  - 定义了物体的颜色、透明度、材质等等

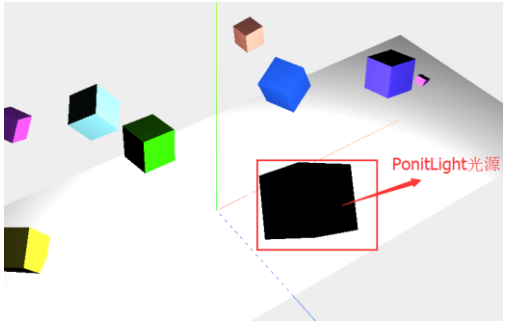


- **常用的材质：**
  - MeshBasicMaterial：
  - 对光照无感，给几何体一种简单的颜色或显示线框
  - MeshLambertMaterial：
  - 这种材质对光照有反应，用于创建暗淡的不发光的物体
  - MeshPhongMaterial：
  - 这种材质对光照也有反应，用于创建金属类明亮的物体

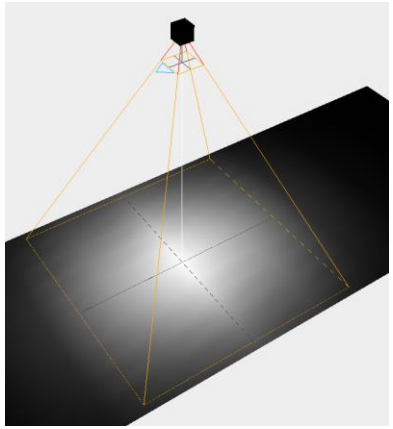


three.js 重要概念

- 光照：Light

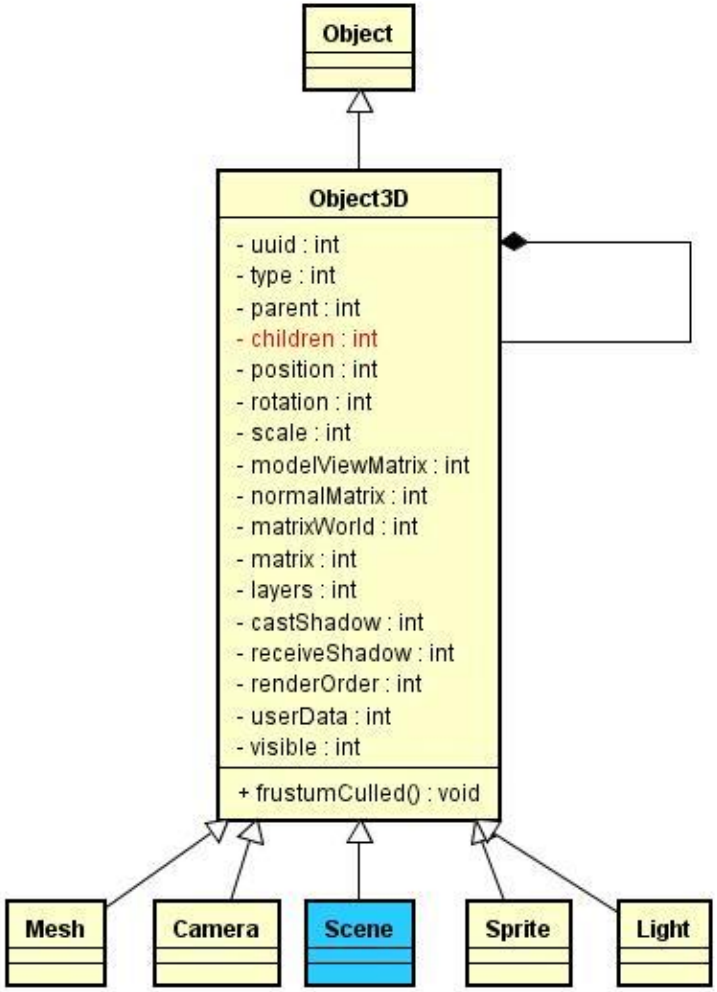


- 常用的光照：
  - AmbientLight:
  - 环境光，基础光源，它的颜色会被
  - 加载到整个场景和所有对象的当前颜色上。
  - PointLight：
  - 点光源，朝着所有方向都发射光线
  - SpotLight ：
  - 聚光灯光源：类型台灯，天花板上的吊灯，手电筒等
  - DirectionalLight：
  - 方向光，又称无限光，从这个发出的光源可以看做是平行光.



three.js 重要概念

- **各种3D物体的基类：Object3D**
  - 3D物体、光源等都是它的子类
  - 重要属性：
  - position、rotation、scale
  - 通过改变它们来实现各种交互效果
  - 参考官方文档：
    https://threejs.org/docs/index.html#api/core/Object3D



three.js 常用功能

天空盒 Skybox：

- 实际上是一个立方体对象。用户视角只在盒子内部活动，所以只需要渲染盒子内部表面。
- 天空盒子应当足够大，使得摄像机在移动时看天空仍然觉得足够远。但是，天空盒子不能超出摄像机最远可视范围。


```
var skyBoxGeometry = new THREE.BoxGeometry(500, 500, 500);
var skyBoxMaterial = new THREE.MeshBasicMaterial({
      color: 0x9999ff,
      side: THREE.BackSide
    });
var skyBox = new THREE.Mesh(skyBoxGeometry, skyBoxMaterial);
scene.add(skyBox);
```

three.js 常用功能

碰撞检测：

- Raycaster：检测射线与物体相交
- 可用于鼠标选择物体、简单的两物体间碰撞检测等

  https://threejs.org/docs/index.html#api/core/Raycaster

- **物理引擎**
- Physi.js、Cannon.js ... 参见各自 Github

加载外部模型：

- 参考 https://threejs.org/examples/ 下 loader 相关例子的源代码

加载音频：

- 参考 https://threejs.org/docs/index.html#api/audio/Audio

游戏循环

- 如何让场景与用户能够交互？
  - 典型的游戏循环：

```
while (true)
{
  processInput();
  update();
  render();
}
```

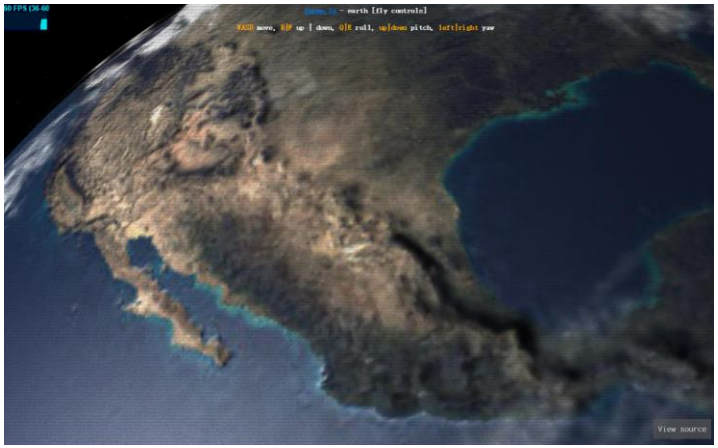
- JavaScript 的游戏循环
  - 使用 requestAnimationFrame 代替 while

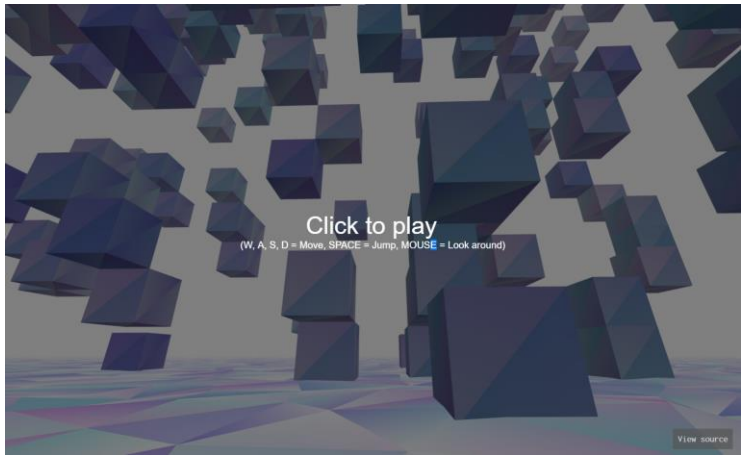
```
function loop() {
    processInput();
    update();
    render();
    requestAnimationFrame(loop);
}
```

输入控制

- 参考官方例子 https://threejs.org/examples/下 misc/controls 相关例子的源代码

- 例如：
- 模拟飞行： controls/fly
- 第一人称控制器： controls/pointerlock

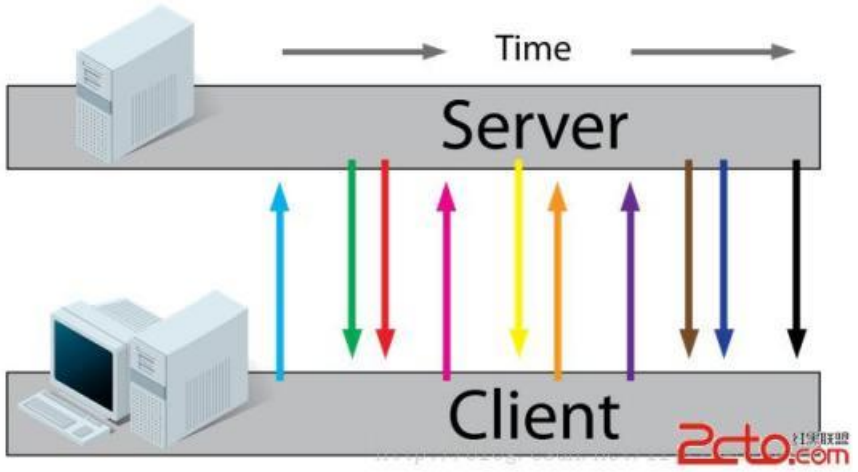




WebSocket

- WebSocket是HTML5开始提供的一种在单个 TCP 连接上进行全双工通讯的协议。

- 在WebSocket API中，浏览器和服务器只需要做一个握手的动作，然后，浏览器和服务器之间就形成了一条快速通道。两者之间就直接可以数据互相传送。



socket.io

- socket.io封装了websocket，同时包含了其它的连接方式，比如Ajax。原因在于不是所有的浏览器都支持websocket，通过socket.io的封装，你不用关心里面用了什么连接方式。

- 官网：https://socket.io/
- Github：https://github.com/socketio/socket.io/

学习资料：

- 官方文档：https://socket.io/docs/
- 聊天室教程：https://socket.io/get-started/chat/

socket.io

client：

建立连接：

```
<script src="/socket.io
<script>
  var socket = io();
</script>
```

发送/接收消息：

```
socket.emit('chat', 'xxxxxx');
```

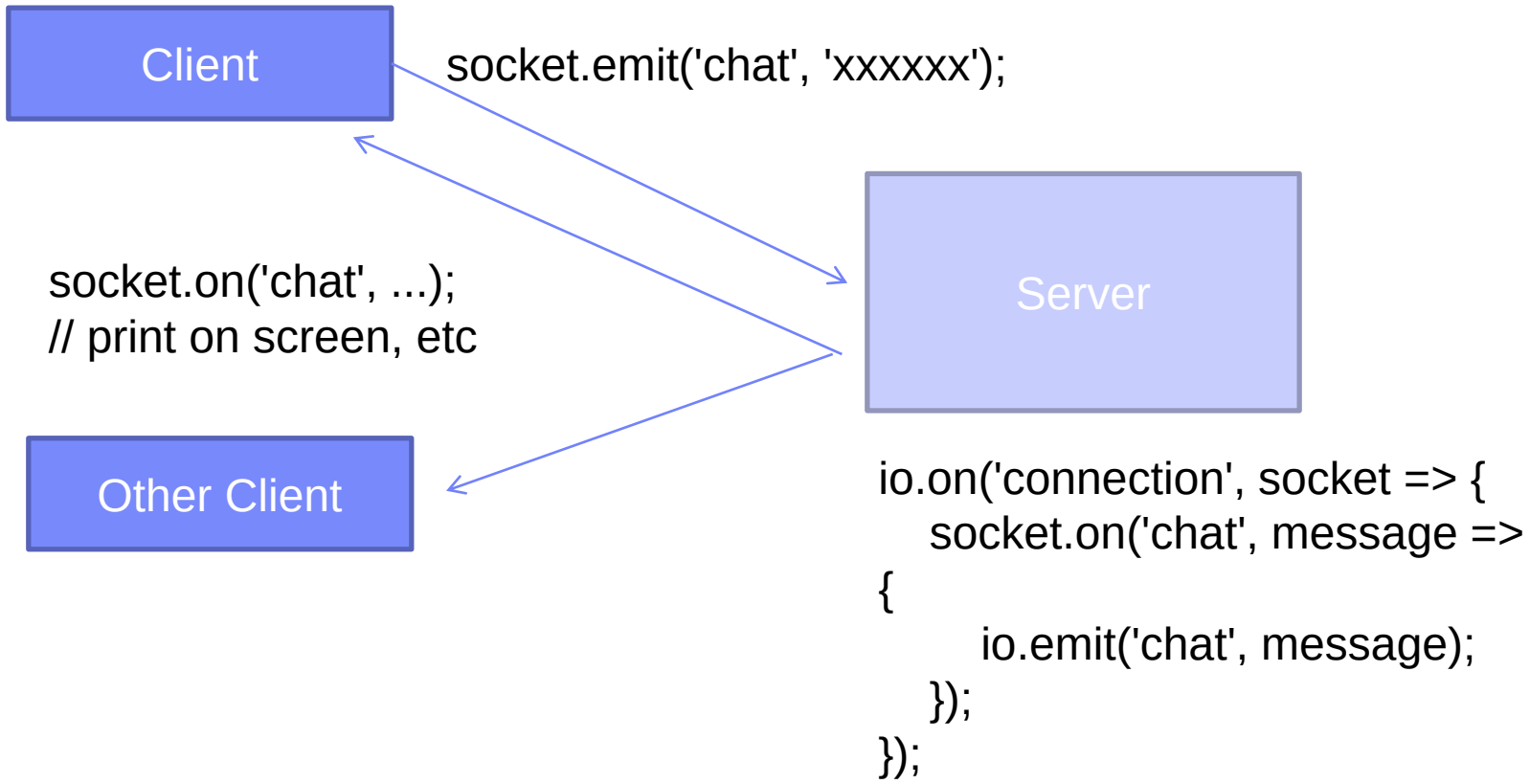
server：

```
io.on('connection', function(socket){
  console.log('a user connected');
  socket.on('disconnect', function(){
    console.log('user disconnected');
  });
});
```

```
socket.on('chat', message => {
        // do something
});
```

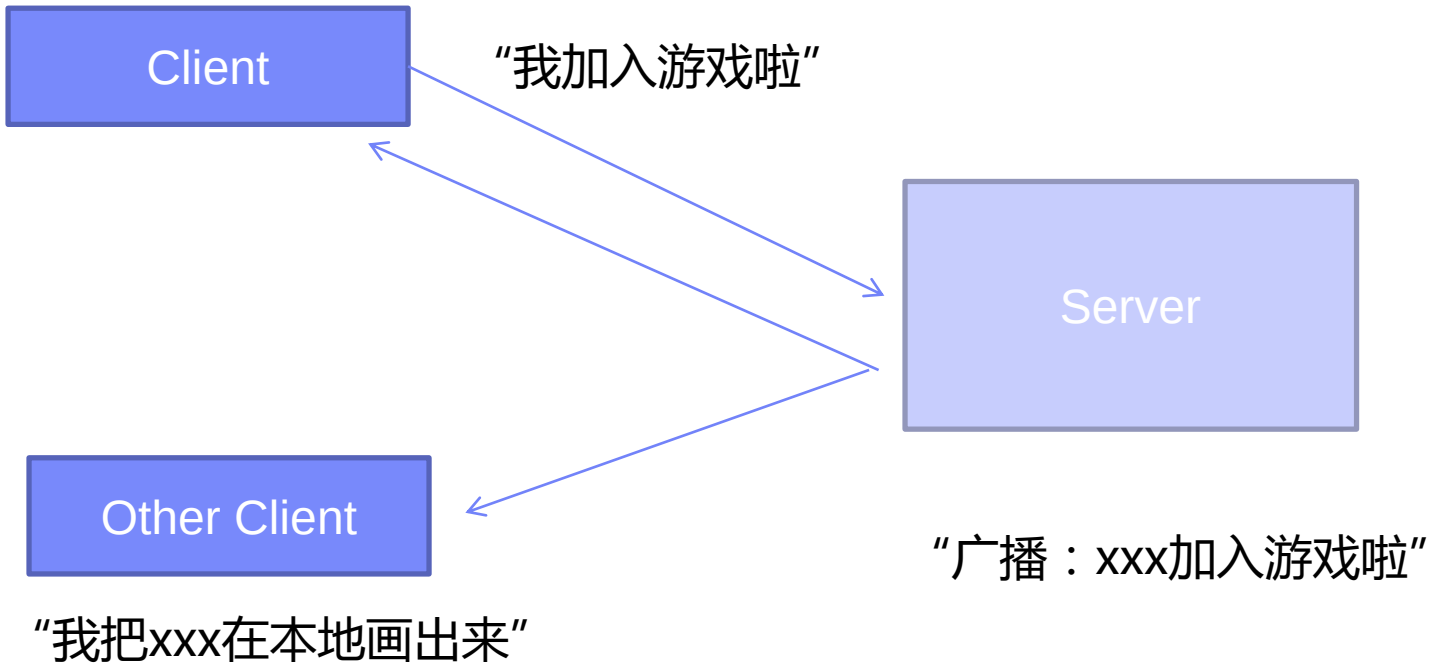
socket.io

- 聊天室例子：



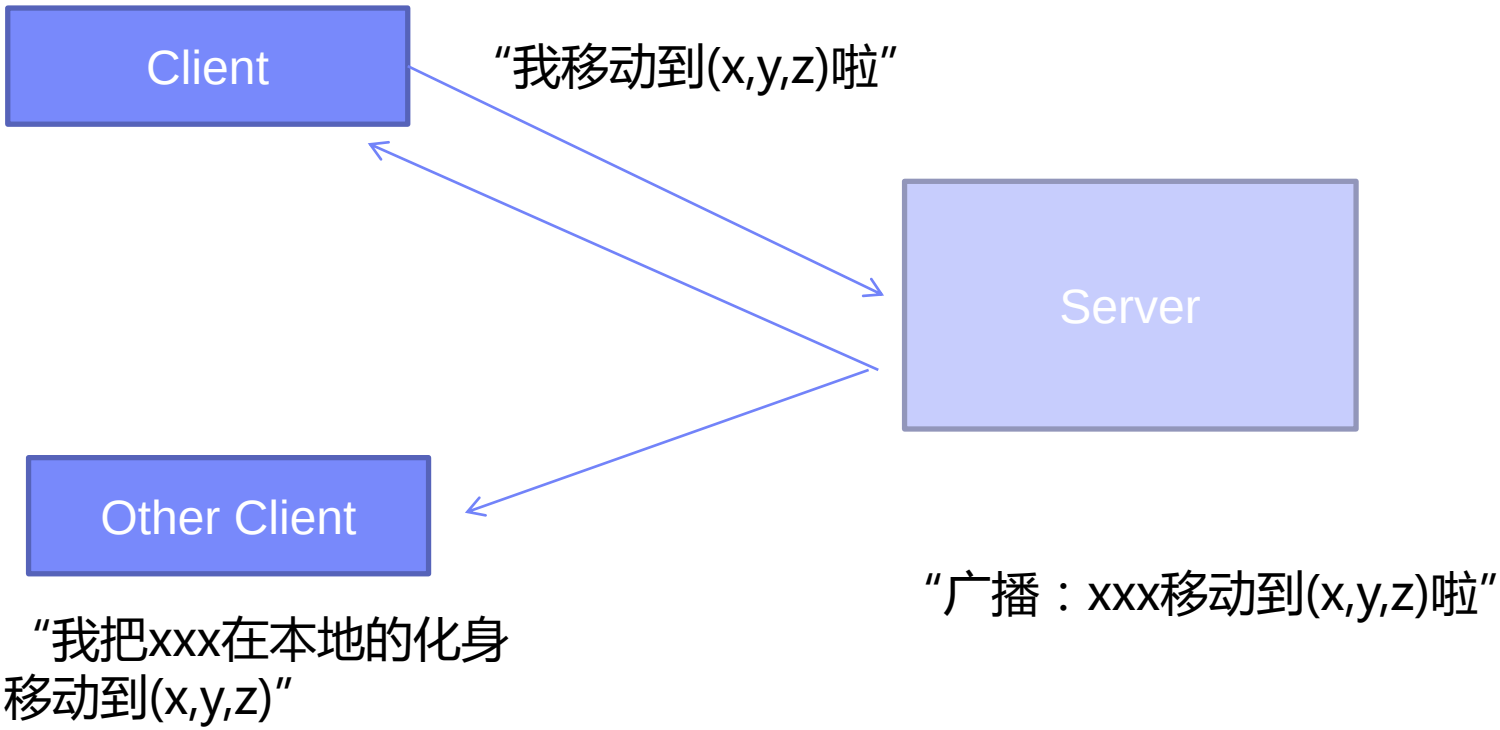
多人联机实现

- 需求分析：所有玩家相互可见



多人联机实现

- 需求分析：可以看到其他玩家的移动



多人联机实现

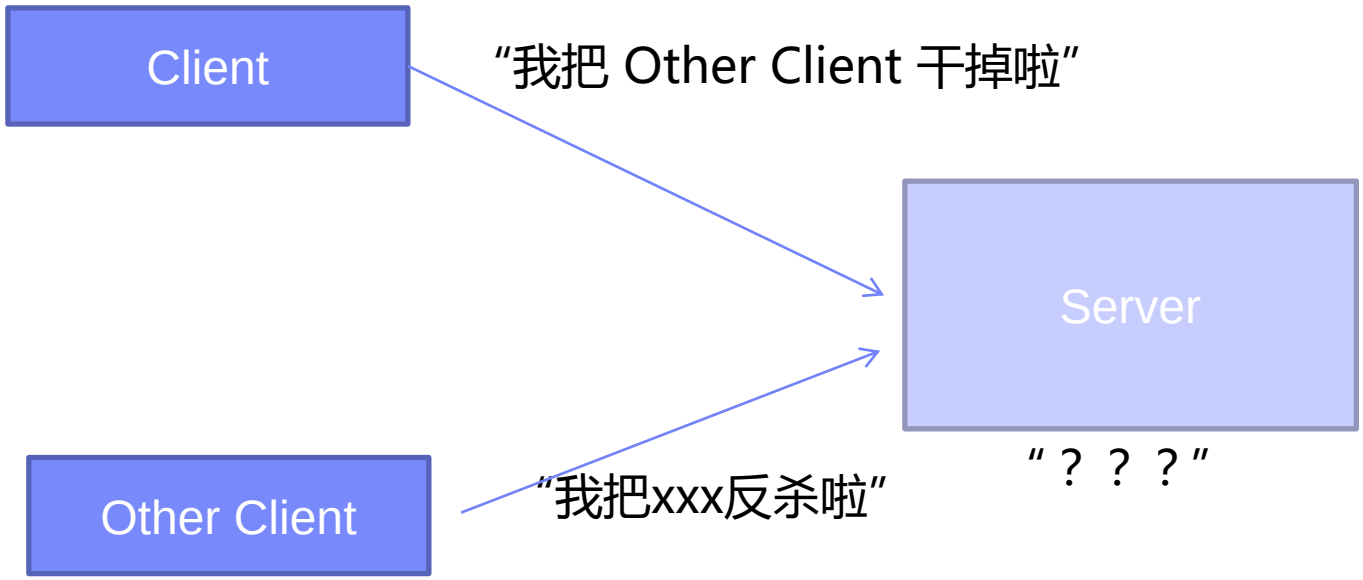
- 客户端需要定时上报自身的位置信息

```
function loop() {
    processInput();
    update();
    render();
    requestAnimationFrame(loop);
}
```

```
reportToServer() {
    // 判断上次上报时间
    // 大于一定的时间才上报移动
}
```

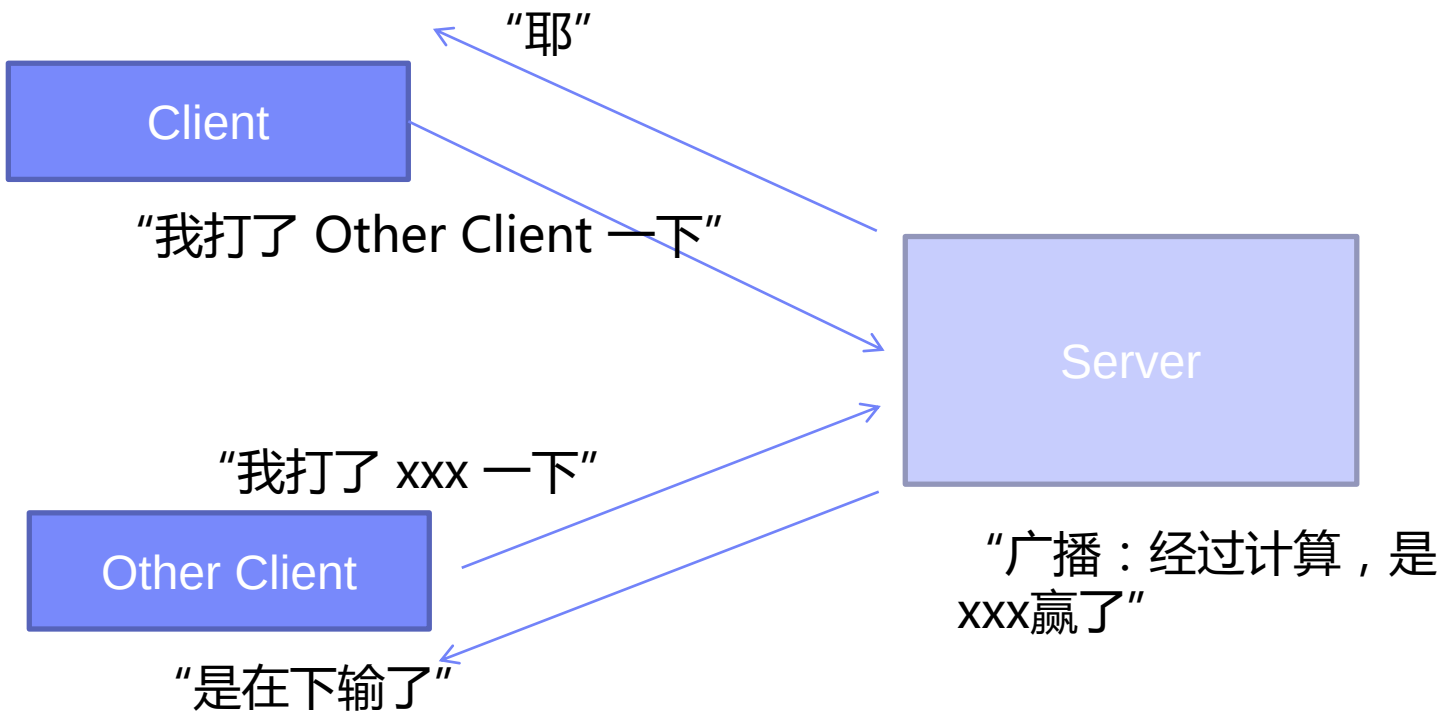
多人联机实现

- 需求分析：同步游戏的状态，与同步玩家状态类似
- 但可能存在冲突：



权威服务器*

- 服务器上也跑一个游戏循环
- 客户端只上报输入，服务端计算结果并广播



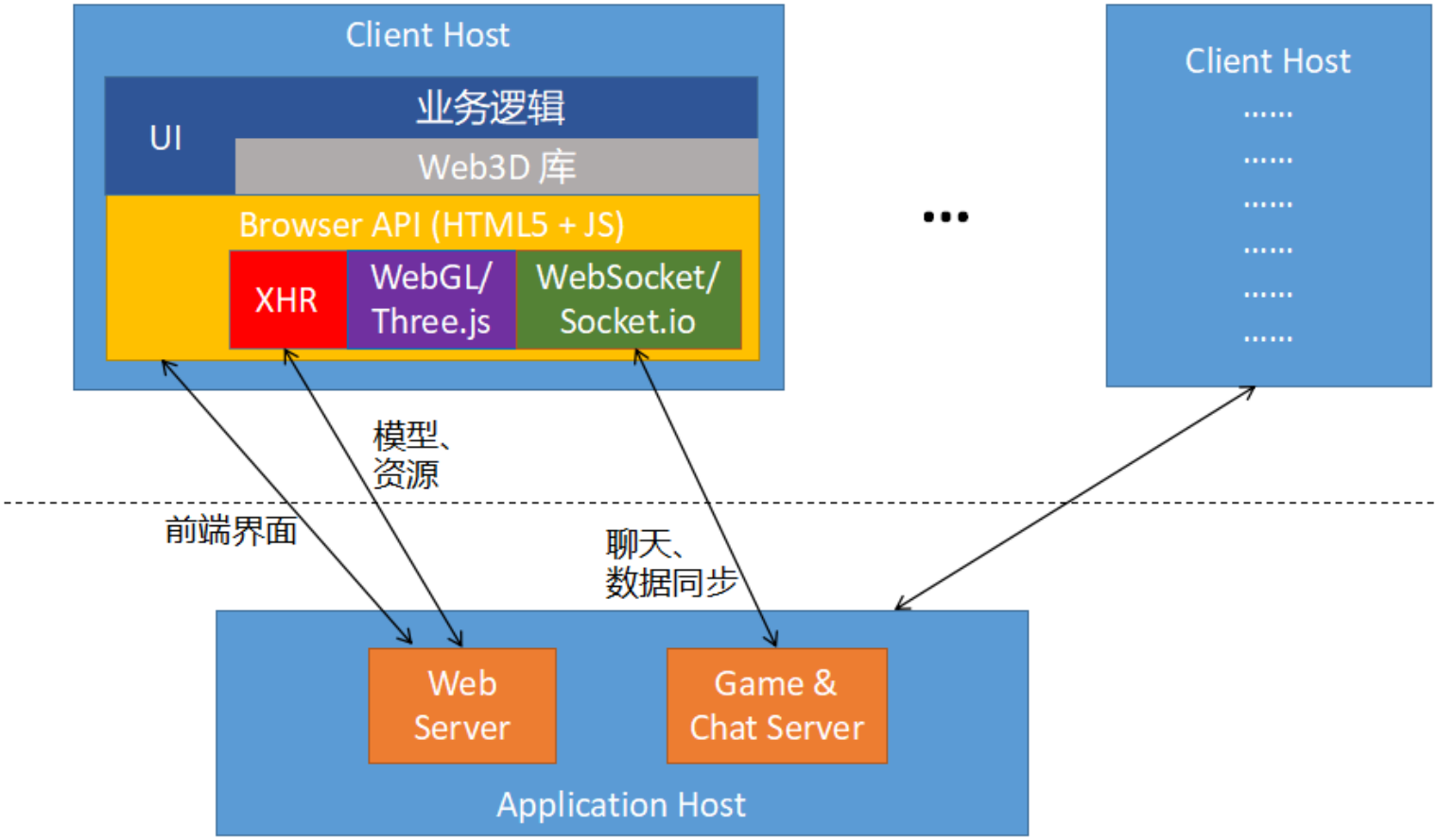
权威服务器*

- **服务端游戏循环：**
  - **定时向所有客户端发送场景的状态**

```
function loop() {
    processInput(); // 从客户端读到的输入
    update();
    render();
    setImmediate(loop);
}
```

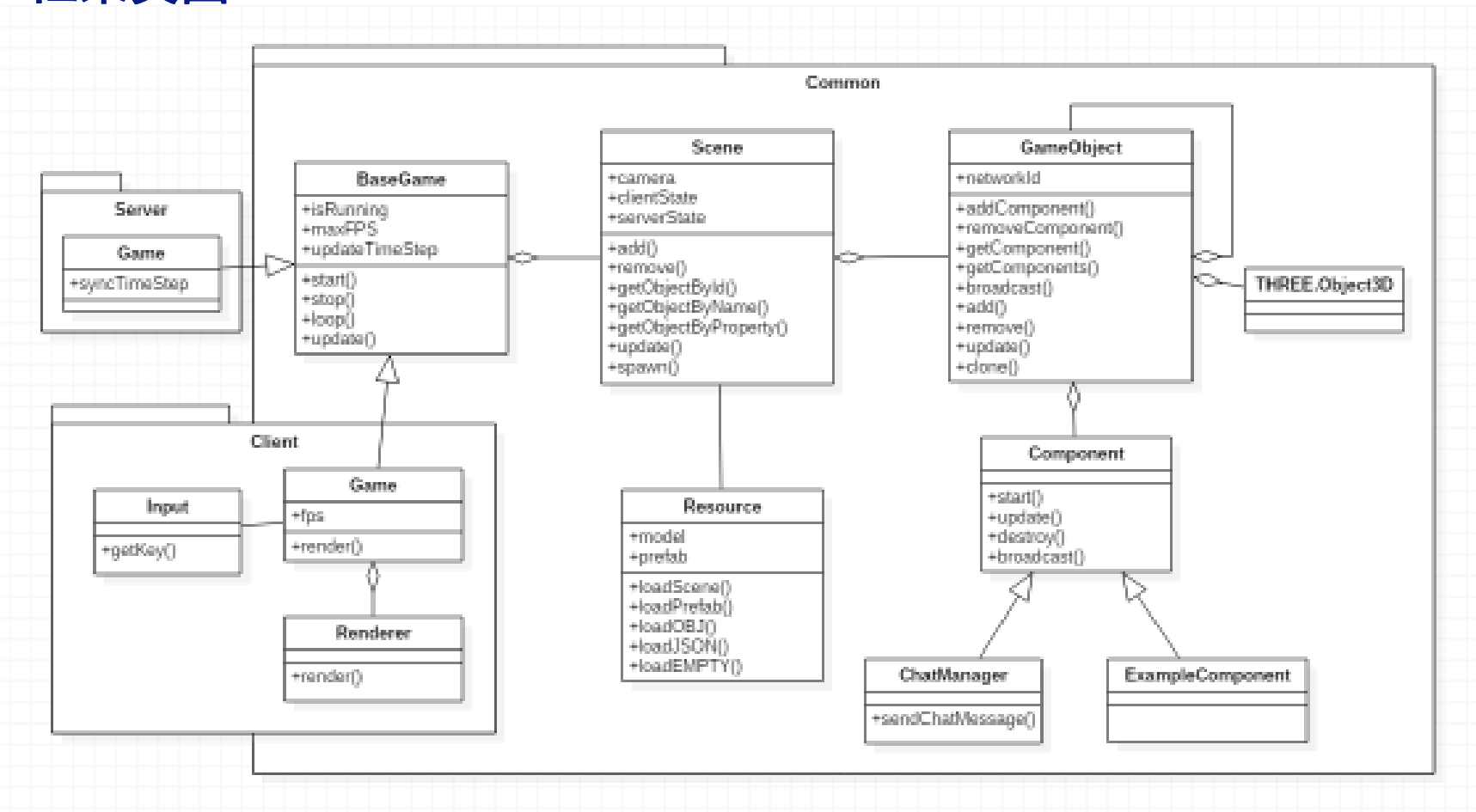
```
reportToClient() {
    // 判断上次同步时间
    // 大于一定的时间同步所有客户端
}
```

工程框架实例


Client Host
UI
业务逻辑
Web3D 库
Browser API (HTML5 + JS)
XHR
WebGL/
Three.js
WebSocket/
Socket.io
Client Host
.....
.....
.....
.....
.....
.....
模型、
资源
前端界面
聊天、
数据同步
Web
Server
Game &
Chat Server
Application Host

工程框架实例

- 框架类图


Common
Server
Game
+syncTimeStep
BaseGame
+isRunning
+maxFPS
+updateTimeStep
+start()
+stop()
+loop()
+update()
Scene
+camera
+clientState
+serverState
+add()
+remove()
+getObjectById()
+getObjectByName()
+getObjectByProperty()
+update()
+spawn()
GameObject
+networkId
+addComponent()
+removeComponent()
+getComponent()
+getComponents()
+broadcast()
+add()
+remove()
+update()
+clone()
THREE.Object3D
Client
Input
+getKey()
Game
+fps
+render()
Renderer
+render()
Resource
+model
+prefab
+loadScene()
+loadPrefab()
+loadOBJ()
+loadJSON()
+loadEMPTY()
Component
+start()
+update()
+destroy()
+broadcast()
ChatManager
+sendChatMessage()
ExampleComponent

工程框架实例

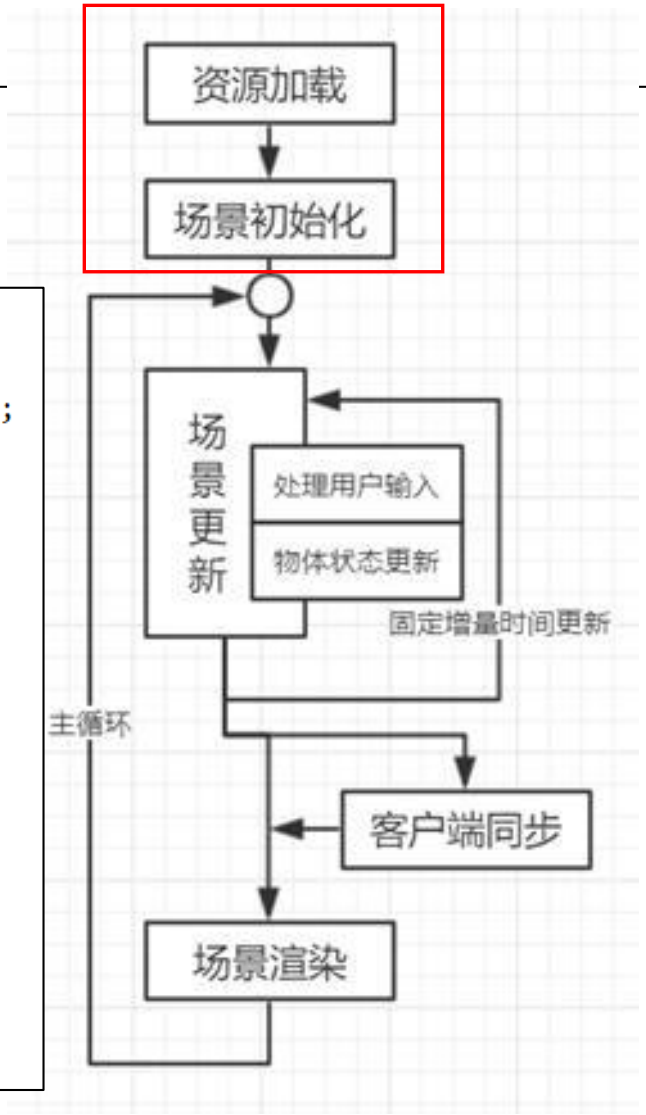
资源加载、场景初始化：

```
// 创建 three.js 渲染器，并附加到 div 元素下
var renderer = new THREE.WebGLRenderer();
renderer.setSize(container.offsetWidth, container.offsetHeight);
container.appendChild(renderer.domElement);

// 创建 three.js 场景
var scene = new THREE.Scene();

// 创建相机并加入场景
var camera = new THREE.PerspectiveCamera(45,
    container.offsetWidth / container.offsetHeight, 1, 4000);
camera.position.set(0, 0, 3.3333);
scene.add(camera);

// 创建一个长方体并加入场景
var geometry = new THREE.PlaneGeometry(1, 1);
var mesh = new THREE.Mesh(geometry,
    new THREE.MeshBasicMaterial());
scene.add(mesh);
```



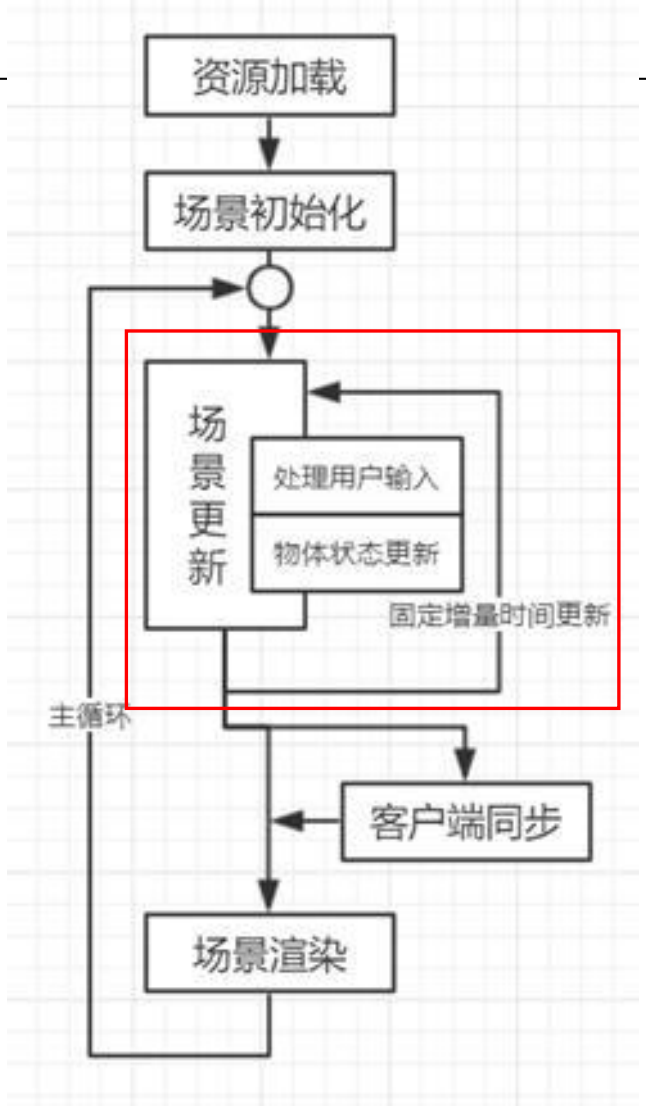
工程框架实例

场景更新：

```
function loop() {
    processInput();
    update();
    render();
    requestAnimationFrame(loop);
}
```

```
let frameTime = timestamp - this.lastFrameTime;
this.delta += frameTime;
this.lastFrameTime = timestamp;

while (this.delta >= this.updateTimeStep) {
    this.update(this.updateTimeStep);
    this.delta -= this.updateTimeStep;
}
```



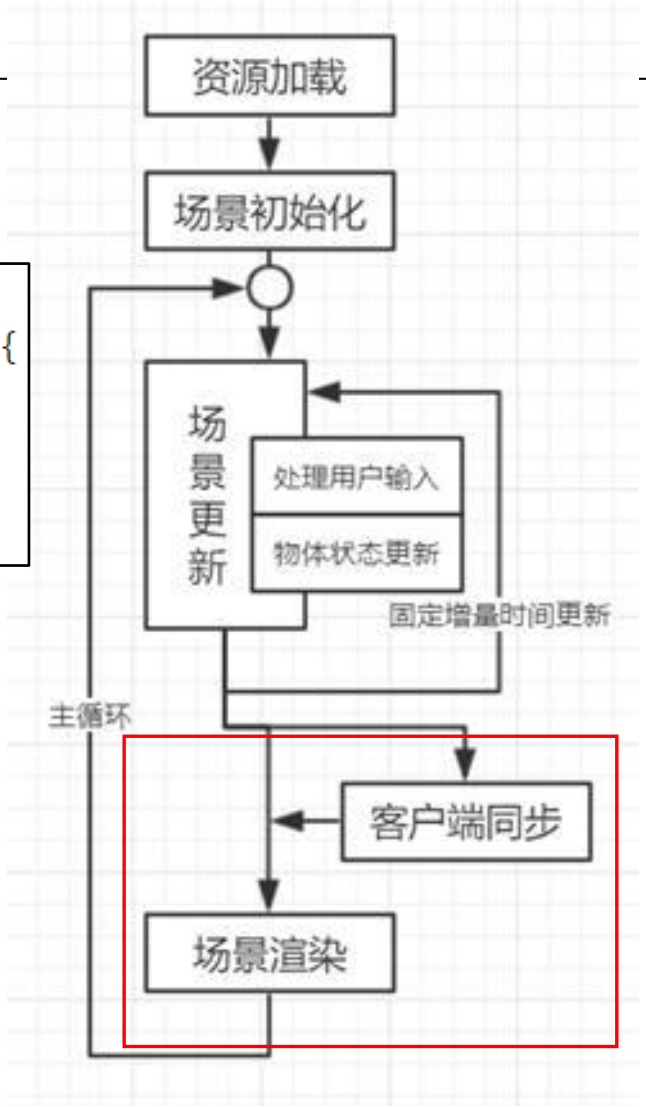
工程框架实例

客户端同步：

```
// sync all clients
if (timestamp - this.lastSyncTime > this.syncTimeStep) {
    io.emit(Event.SERVER_SEND_STATE,
        JSON.stringify(this.scene.serverState));
    this.lastSyncTime = timestamp;
}
```

场景渲染：

```
renderer.render( scene, camera );
```



其他资源

- ECMAScript 6 标准：
- http://es6.ruanyifeng.com/

- 游戏编程模式：
- http://gameprogrammingpatterns.com/ （en）
- http://gpp.tkchu.me/ （cn）

- 简单的多人 Web3D Demo：
- src：https://gitee.com/maliut/aweb-web3d
- demo： http://maliut.gitee.io/aweb-web3d