# 技术方案

#### **前端：**

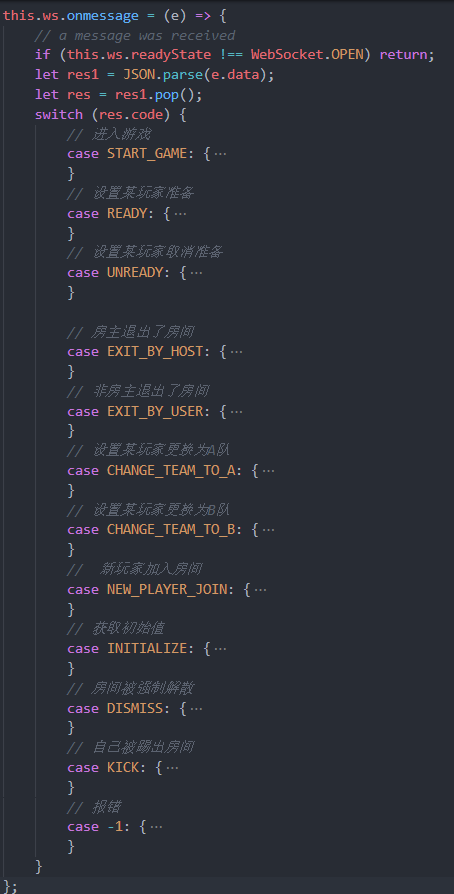
1. **react-native框架**

项目采用目前通用且流行的react-native框架进行安卓手机端的开发。开发过程中，前端强调组件功能的封装及不同功能组件的高独立性。同时可以利用react-native的原生组件及一些第三方依赖包（如rn-camera,rn-baidumap）等来搭建软件的界面。

1. **websocket连接**

在房间及游戏进程中，前后端的连通未采用使用广泛的http单向请求连接，而是使用了Socket，使每一个前端和后端服务器建立起全双工通信连接。这种连接方式取代了ajax轮询，提供了更好的性能以及更高的容错率，同时降低了前端组件的复杂程度。

在实际的使用当中，我们项目未使用已封装好的第三方依赖包，而是使用了原生的WebSocket类建立连接，设置状态参数及添加监听函数，使参数的传递更加自由。



1. **Axios+JSON前后端传值**

JSON字符串传值作为最方便、最快捷的前后端传值方式，同时受限于Sokcet连接只能传递字符串类型参数，项目中所有前后端参数传递均使用JSON传值。

axios作为react生态中封装好的ajax请求，其可以自行将键值对类型数据转化为json类型参数，为代码的提供了更简单的编写方式及更清晰的阅读方式。



1. **采用rn-camera来实现拍照射击功能**

项目使用react生态提供的第三方依赖包rn-camera来实现拍照射击的功能。通过camera对手机摄像头的调用来处理一系列与相机有关的需求。



#### **后端：**

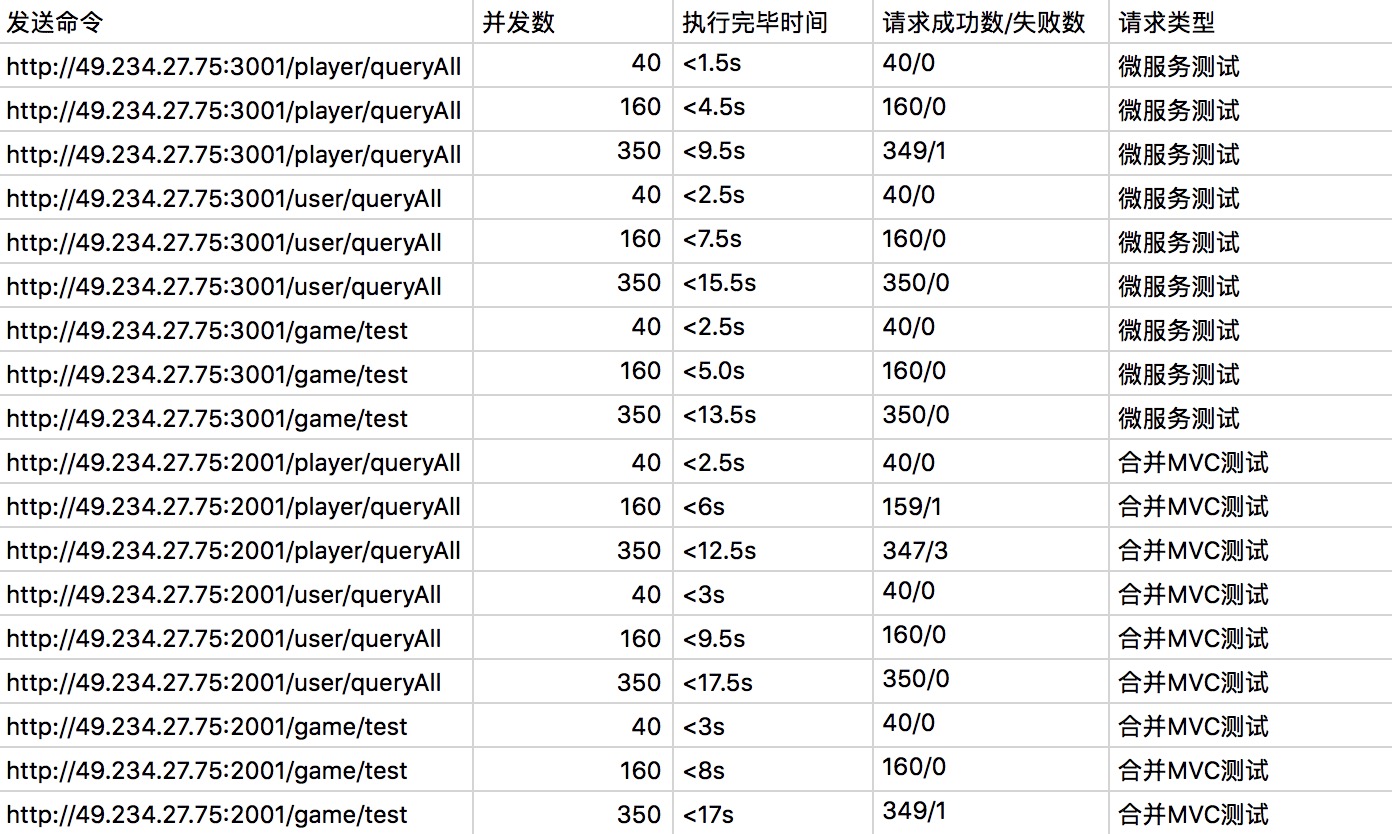
1. 微服务架构

#### 项目使用为服务技术，将user、room和game三个服务分散在三个不同的服务中。在微服务的架构中，采用传统的euraka架构，构建注册中心。将三者分散开的原因：

#### 1.user room 和game是相对独立的，耦合度比较低，而每一个服务内部的内聚很高，user中实现对用户信息的管理和用户的登录、修改密码等操作；room中实现游戏内房间的创建、加入和退出等工作；game中实现游戏内与前端的联通工作。

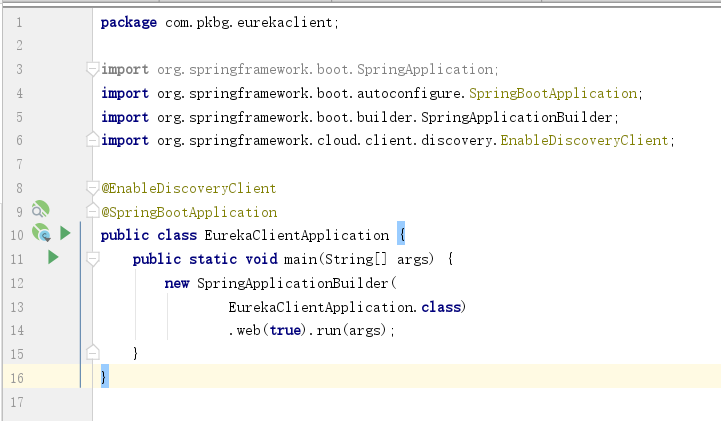
#### 2.三者分散开，登陆、创建加入房间和进行游戏三个行为分散在三个不同的服务（服务器）中，大大地缓解了服务器的压力，极大地解决了请求并行并发的问题，极大地提高游戏性能和游戏体验。

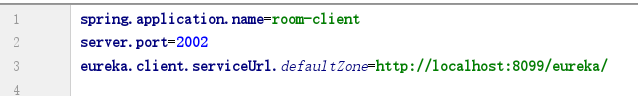
#### 将三者配置在同一个服务器上和使用微服务架构的技术参数对比：

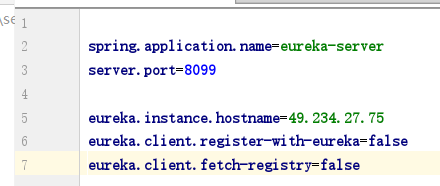


#### 实验证明微服务对减轻服务器压力有效。

#### 微服务euraka细节如下：





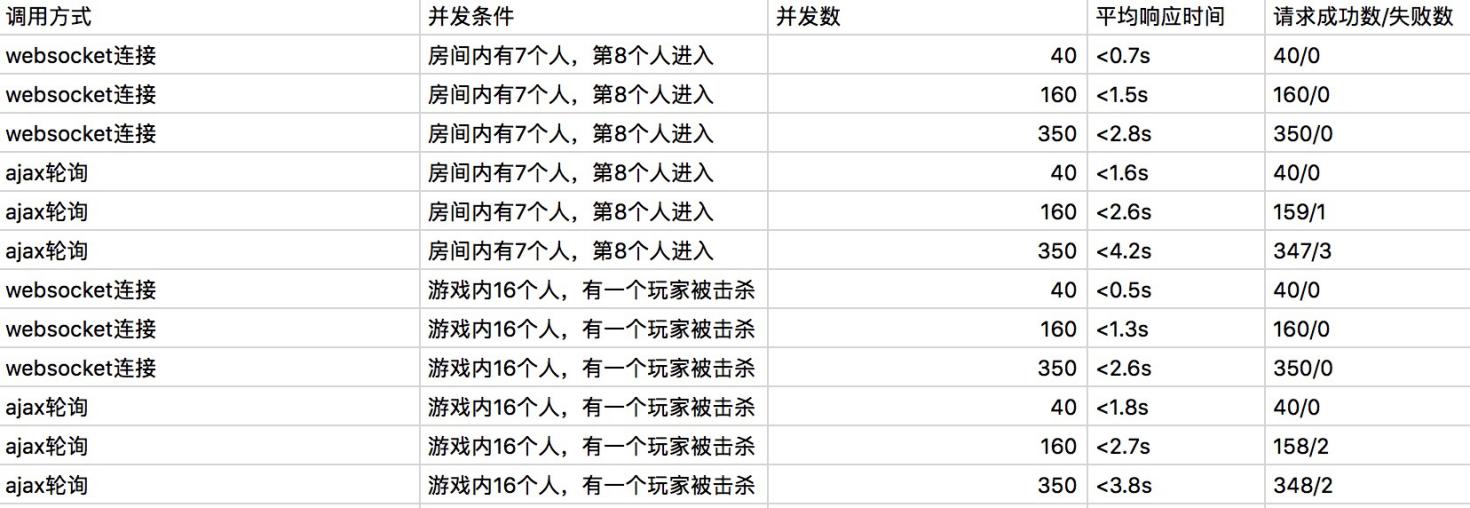


1. websocket实现真人对战

#### 后端实现PVP的游戏模式，放弃了消耗资源过多的ajax轮询技术，采用效率更高且对资源占用更小的websocket连接服务，在game服务和room服务中分别建立两次websocket连接，变成状态可控的前后端实时连接。其优势如下：

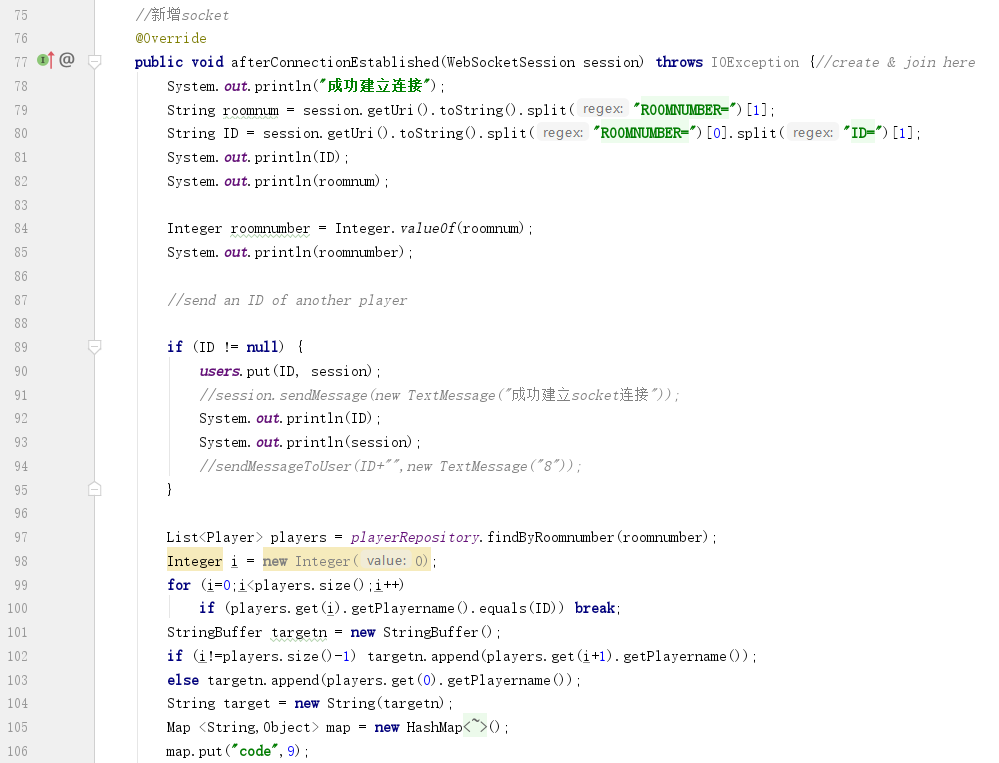
#### 1.当一个玩家进入房间后可以广播通知所有玩家，玩家的状态实现不依靠websocket轮询而实时共享

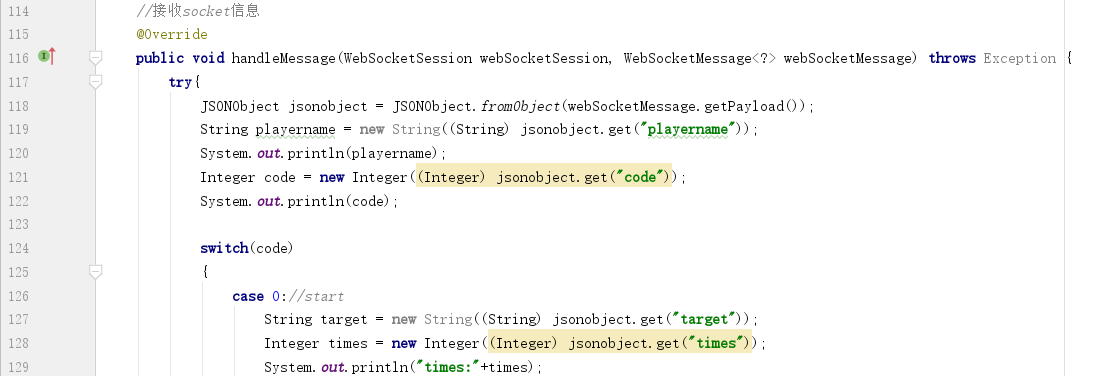
#### 2.不采用ajax的方式沟通，前后端联通效率更高，对比ajax轮询的实验对比如下：

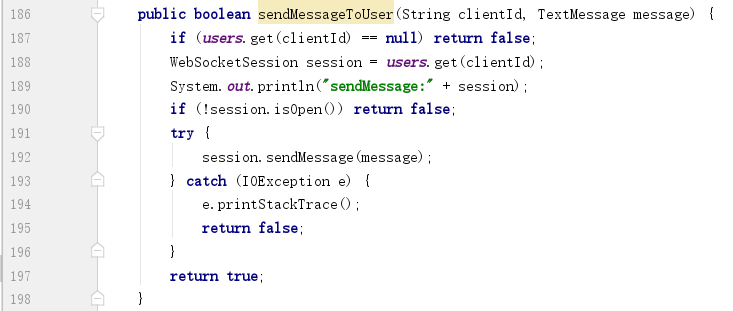


#### 实验证明websocket比ajax长轮询效率和精度更高。

#### 使用java的endpoint低层级进行websockt细节如下：





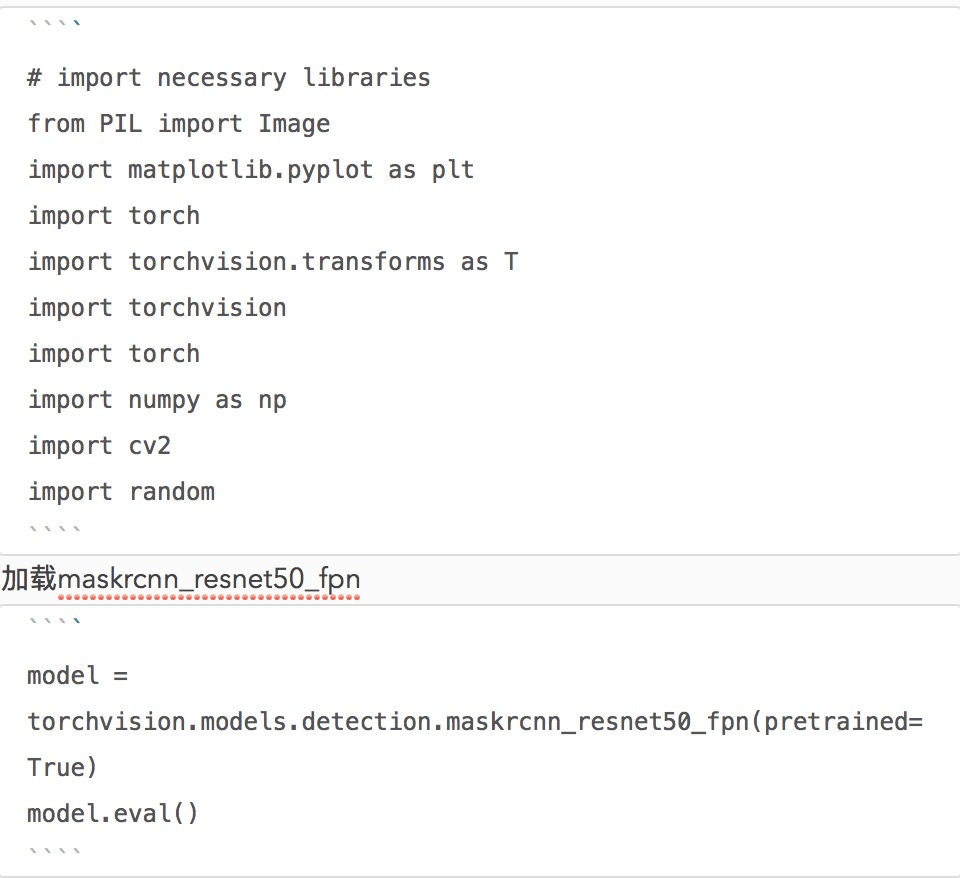


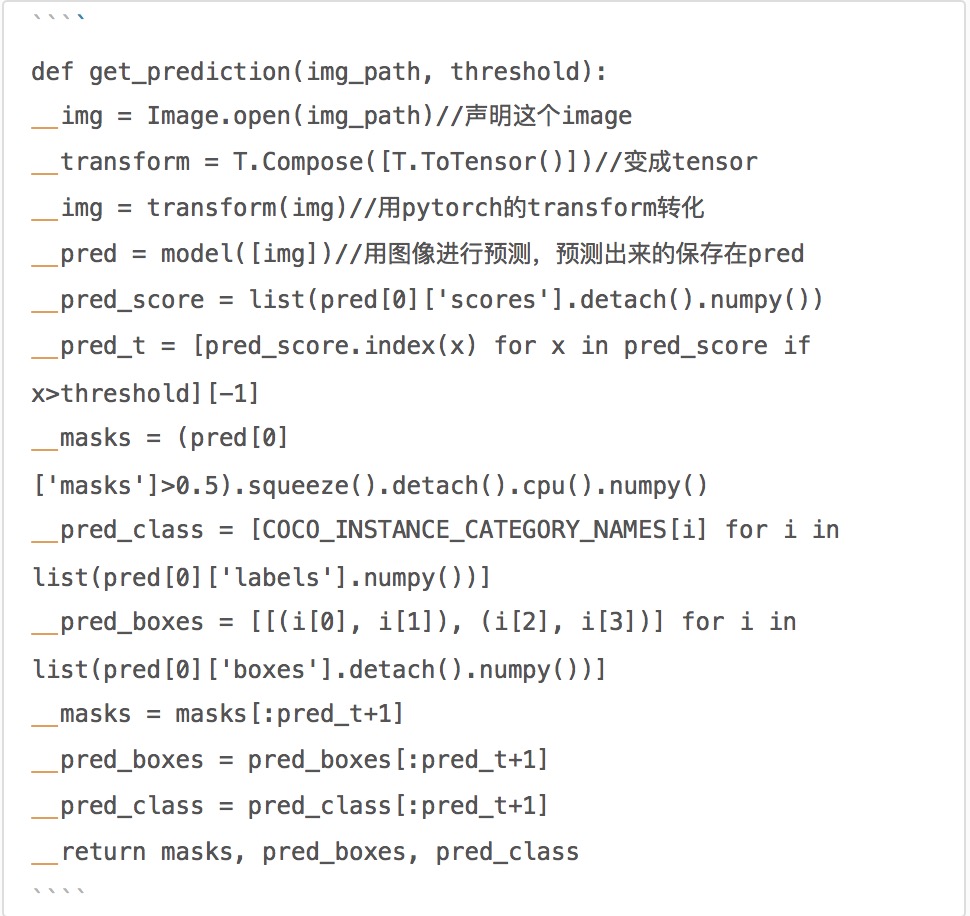
#### **算法：**

1. 人物识别算法

* 人物识别模型小组采用不同的现有模型进行试验，最终选定了采取以MASK R-CNN为baseline的Face++ api进行人物捕捉识别的功能，小组进行试验如下：

#### 我们进行试验的代码如下（仅贴出Mask R-CNN的训练过程，其他过程类似）：







* 采用不同网络进行训练如下：

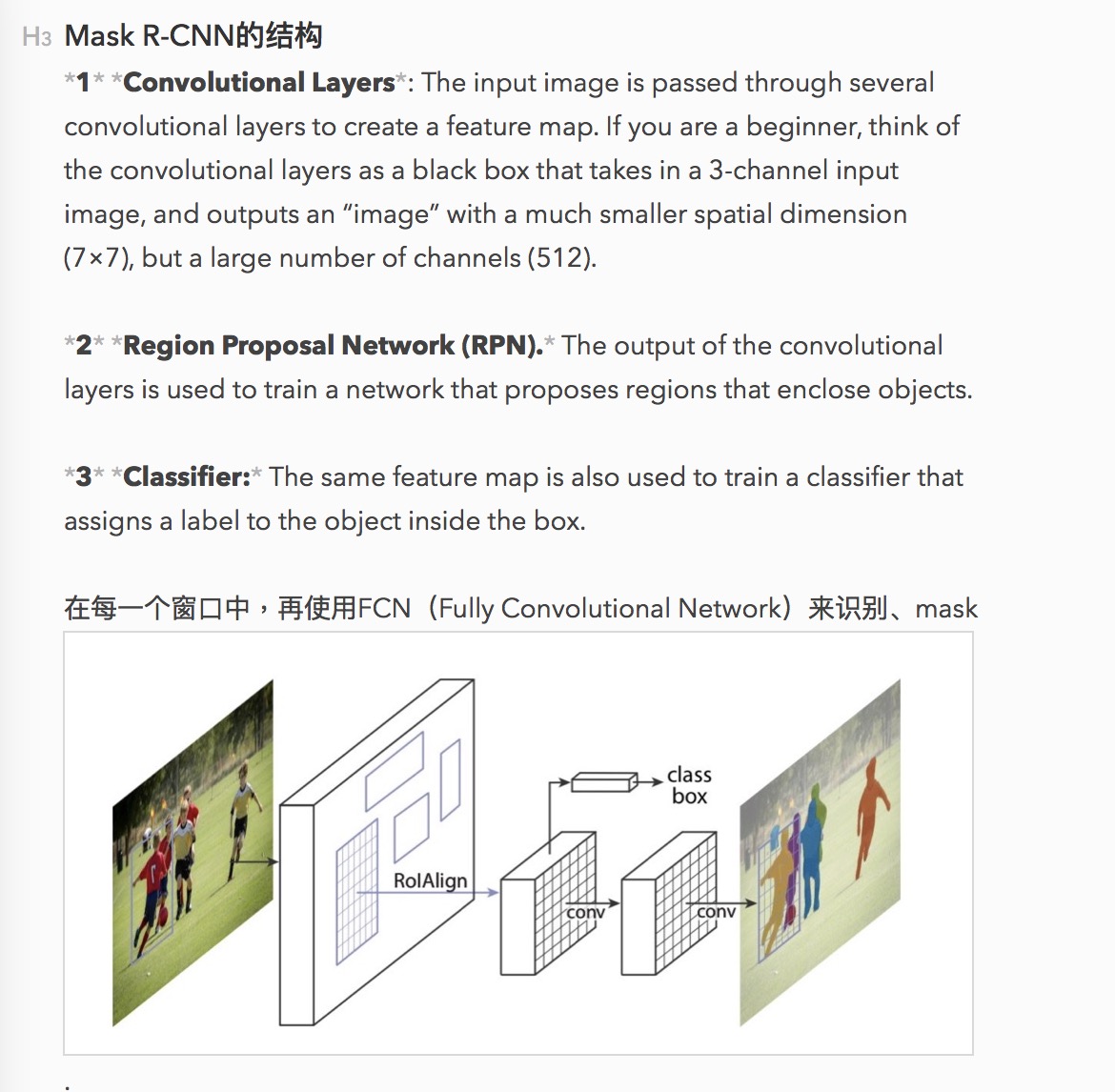


* 采用Face++ API的原因如下：

#### 1.我们通过实验发现pretrained的网络过于复杂庞大，有很多不必要的功能，因此现有的网络需要transfer learning进行微调，在微调的过程中由于现有的一些大型网络不是end to end的训练模式，需要多张极高水平的显卡进行训练。现有的计算资源只能f微调resnet101级别的网络精确度和识别时间不如大型网络，而Face++的API已经包含现有的大型的网络、返回正符合需要的数据，最终决定采用节省成本且效率最高的以MASK R-CNN为basline的Face++ api进行人物捕捉识别的功能。

* 最后我们研究了Mask R-CNN的结构：

#### 也就是Face++使用的baseline：



1. gps算法

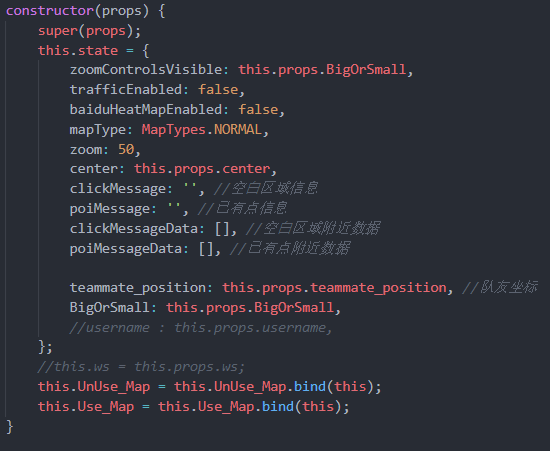
#### 我们采用百度地图进行gps的地图和标记队友的服务。

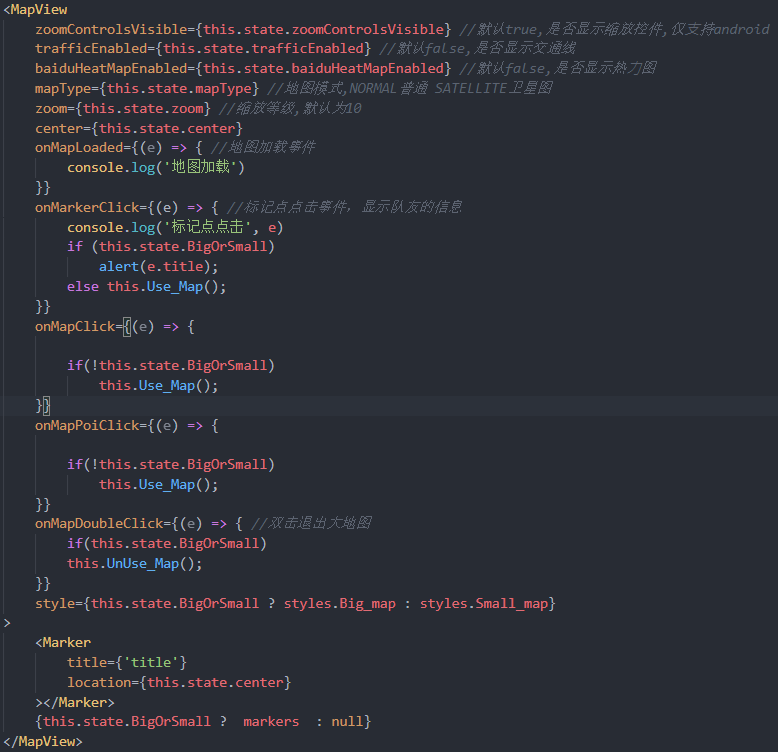
* 我们不使用Google或者高德的地图的原因：

#### 1.这二者的api和SDK建立在android的java语言环境下，而我们采用react—native的前端代码架构，不能使用现有的接口，于是使用baiduMap。

#### 2.BaiduMap的中文支持性更好，便于调试和玩家的理解，不再需要我们

#### 我们的实验代码如下，通过调用接口实现了人物标记、队友查看建立在websocket技术之上。







1. 开枪（屏幕捕捉）算法

由于每次拍照是需要一点无法避免的延迟及微小卡顿时间，如果每次点击都是即时拍摄一张图片并上传至Face++，那么短时间内的多次射击必然会造成极多次的游戏画面卡顿、一定时间的游戏画面延迟和巨大的上传图片流量，这是非常影响性能甚至导致程序卡死的。所以为了避免这种情况的出现，我们重新设计了射击时按键的算法：根据不同枪械的射速，在快速点击射击按钮或长按时，会根据最近拍摄保留下来的一张图片在短时间内判断是否命中，而不是多次拍摄极为相似的图片。相对于前者，这样可以一定程度上的优化画面的流畅性及玩家的操作手感。