29 系统设计，前期规划很重要

更新时间：2020-08-19 09:44:29

> 学习这件事不在乎有没有人教你，最重要的是在于你自己有没有觉悟和恒心。 —— 法布尔

前言

你好，我是彤哥。

上一节，我们从需求收集、详细分析、可行性分析三个维度对本次实战项目做了完整的需求分析，最终得出了简化版的麻将需求。

本节，基于软件开发的基本步骤，并结合上一节需求分析的结果，我们将进行系统设计，让我们进入今天的学习吧。

系统设计

根据软件开发的基本步骤，系统设计，我们将分成技术选型、领域模型设计、接口设计、部署架构设计等四个部分来完成。

技术选型

技术选型，毫无疑问，我们选择的是 Netty，但是，基于 Netty，我们还需要做一些事情来构建系统，这些事情包含网络协议选型、数据协议设计、编解码设计等。

网络协议选型，也就是说请求是通过什么方式在客户端和服务端进行通信，常用的网络协议选型有 HTTP、HTTP2、TCP、WebSocket、UDP 等，让我们来对比一下：

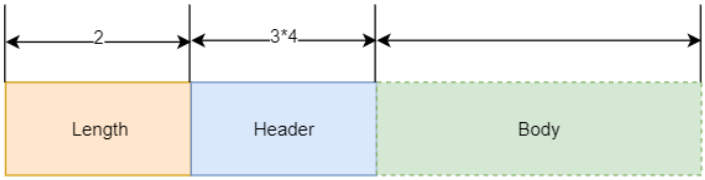
协议	层	长 / 短连接	基于	其它
HTTP	应用层	短连接	数据流	使用广泛，也可支持长连接，但无法支持服务端主动通信客户端
HTTP2	应用层	长连接	数据流	目前使用还不是很广泛
WebSocket	应用层	长连接	数据流	HTTP 协议的升级版，可以实现长连接，使用比较广泛
TCP	传输层	长连接	数据流	可靠，但数据流无界，需要自己分割，Netty 默认的协议
UDP	传输层	无连接	报文	不可靠，需要自己实现可靠性保证，但基于报文，没有粘包半包的问题

通过对比，可以发现，HTTP 无法支持服务端主动通信客户端，对于我们是不合适的，HTTP2 虽然支持长连接，但使用还不是很广泛，WebSocket 和 TCP 比较符合我们的要求，UDP 不可靠，还得自己保证可靠性，所以，我们的首要选择在 WebSocket 和 TCP 两者之间，考虑到未来可能会做小程序或者 Web 应用，所以，选择 WebSocket 协议是最佳选择，但是，选择 WebSocket 增加了复杂度和开发成本，第一个版本，我想做的简单点，所以，我们直接使用默认的协议就好了，即 TCP。

> 这里，我想说一下 Socket，Socket 它本身不是一种协议，翻译为套接字，它是应用层和传输层之间的一个抽象层，把 TCP/IP 层的复杂操作抽象成几个简单的接口供应用层调用以实现进程在网络中的通信。从设计模式的角度来看的话，Socket 就是一种门面模式，它把复杂的 TCP/IP 协议族隐藏在 Socket 接口后面。
>
> 另外，WebSocket 与 Socket 没有任何关系，就像 JavaScript 与 Java 的关系一样，纯属借名造势而已。

数据协议设计，主要是解决两个问题：粘包半包问题和协议内容。粘包半包的问题，在前面的章节我们也讨论过，这里肯定选择的是 长度 + 内容 的方式来解决。协议内容的问题就不是那么好确定了，参考业界比较优秀的协议，比如 Dubbo 和 RocketMQ，一般来说，数据协议都会分成消息头和消息体两个部分，消息体存储真正的数据内容，消息头存储着一些扩展信息，比如版本号、请求地址（操作码、命令字）、序列化方式、自定义的扩展字段等。

根据我们的业务场景，即麻将游戏场景，在请求头里面，可以存储版本号、命令字、请求 ID 等三个信息，每个字段占用一个 int 类型，最后，我们的数据协议大概是长这样子：



这里，Length 我选择 2 个字节，因为根据评估，我的请求内容不会超过（2^16-1=65535）个字节。

好了，网络协议和数据协议我们都弄好了，下面就是如何编解码了。

编解码设计，根据前面的内容，我们知道，编解码分为一次编解码和二次编解码。一次编解码，是对数据协议的编解码，这个在数据协议设计里面已经处理了，就是解决粘包半包的方式，也就是 长度 + 内容 法。二次编解码，是对 Java 对象的编解码，或者换个通俗的叫法为序列化 / 反序列化方式，对于消息头，格式是固定的，我们直接从 buffer 中读取或写入就好了，对于消息体，常见的序列化方式有 XML、JSON、Java 序列化、Protobuf 等，前面的章节我们也对比过，对于游戏这种对性能要求极高的应用，选择 Protobuf 无疑是最佳选择，但是，第一个版本，我想简单点，所以，我们先选择 JSON 来作为序列化方式，同时，JSON 在易读性方面也很有优势，也便于我们调试代码。

上面，我们从网络协议、数据协议、编解码三个维度对 Netty 这种技术选型做了分析，对于任何网络应用都要经过这些步骤，可以说，是通用化的解决方案，但是，对于具体的业务场景来说，领域模型设计就比较个性化了，下面我们就来看看如何对麻将这种个性化场景做领域模型方面的设计。

领域模型设计

领域模型设计，在不同的场合，可能也会称作数据库设计，或者数据结构设计，不过，它们多多少少还是有一些区别的，同学们可以自行体会一下。

对于麻将游戏的场景，我先给出下面一段描述：

玩家 A 打开 APP，登录到游戏，选择创建四个人的房间，玩家 B、C、D 同样地打开 APP，登录到游戏，同时，选择加入到玩家 A 创建的房间，此时，房间满足四人条件，游戏自动开始，玩家 A 作为房间创建者，自动成为庄家，开始发牌，玩家 A 获得 14 张牌，玩家 B、C、D 获得 13 张牌，玩家 A 开始出牌，玩家 A 出完牌后，玩家 B、C、D 可能会进行碰、杠、胡等操作，如果没有玩家操作，轮到玩家 B 摸牌、出牌，如此往复，直到摸完所有牌或者有玩家胡牌为止，发送结算消息给所有玩家，牌局结束，玩家离场。

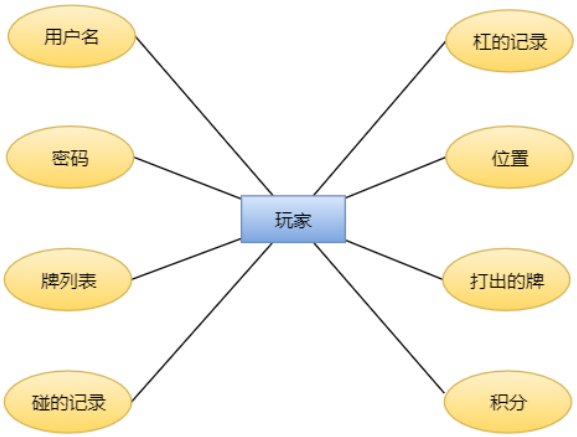
我们先抽出这段描述中所有的名词：玩家、APP、游戏、房间、庄家、牌、消息、牌局，然后去除一些干扰名词。

还剩下玩家、房间、牌、消息这几个名词，它们只是我们寻找的领域对象，让我们来分析一下它们应该具有的属性：

玩家

1. 玩家登录游戏需要什么？用户名和密码。
2. 玩家在游戏的时候操作的是什么？牌列表。
3. 碰、杠的牌要不要记录？要记录，补杠的时候需要看碰的牌，以后结算的时候可能会用到杠的牌，而且通常会把碰杠的牌放在玩家面前。
4. 玩家在房间内的位置要不要记录？要记录，不记录的话，每次要获取某个玩家的位置只能遍历房间中的所有玩家。
5. 玩家打出的牌要不要记录？打过线上麻将的同学应该都知道，玩家打出的牌是摆在自己面前的，所以，打出的牌也是要记录一下的。
6. 结算的时候按什么维度？暂且按积分，类似于斗地主那样，赢了加几分，输了扣几分。

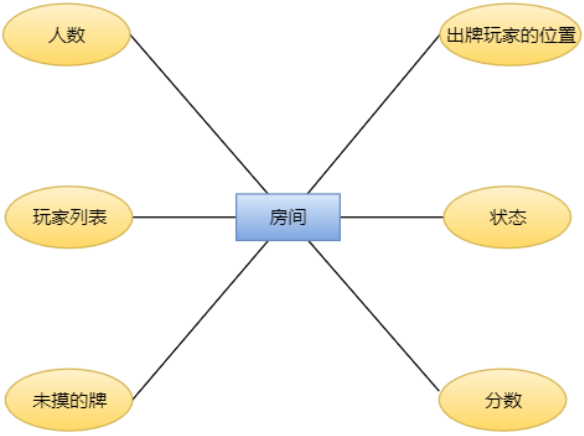
所以，玩家至少应该具有 用户名、密码、牌列表、碰的记录、杠的记录、位置、打出的牌、积分等几个属性。



房间

1. 玩家创建房间的时候指定了什么？人数。
2. 房间要不要记录玩家信息？要，玩家列表。
3. 玩家手里的牌从哪来的？房间的桌子上摸的，所以，还需要记录牌的信息，主要是未摸的牌。
4. 房间要不要记录庄家信息？暂时不需要，庄家位置，根据需求分析，现在还用不到庄家。
5. 房间要不要记录出牌玩家的位置？要，如果不记录，客户端随便发出牌消息，服务端无法比对。
6. 房间要不要状态信息？要，没有状态的话，怎么判断房间是不是在等待玩家加入？怎么判断是不是在游戏中？怎么判断是不是当前在等待玩家出牌，还是在等待其他玩家操作（碰杠胡）？
7. 结算的时候用的分数从哪来？在创建房间的时候传进来最好。

所以，房间至少应该具有 人数、玩家列表、未摸的牌、出牌玩家的位置、状态、分数等信息。



牌

1. 牌有哪些分类？万条筒，不考虑东南西北中发白、春夏秋冬梅兰竹菊的情况下。
2. 每种分类有哪些值？万条筒有 1 到 9 九种数值。

所以，关于牌，其实就这两个属性：类型和数值。

因为类型只有三种，数值只有九种，所以，我们使用一个 byte 就完全可以存储了，甚至还有点浪费，但是没办法，Java 中能操纵的最小单位就是 byte 了。

因此，关于牌，我们并不需要再单独定义一个类型了，直接用 byte 就可以了，不过，为了方便操作，最好定义一个工具类专门用来处理牌相关的信息。

消息

1. 消息是指什么？客户端与服务端之间的一次通信就是一次消息的传递过程，消息就是通信。
2. 通信又是什么？一次通信往往伴随着动作，比如，玩家登录，客户端发送登录请求给服务端，服务端处理完成响应客户端，这个过程是两次通信。
3. 动作具有什么特征？动作往往都是动词，所以，只要找上面描述中的动词就八九不离十了。
4. 动词有哪些？打开、登录、创建、加入、开始、成为庄家、发牌、获得、出牌、碰、杠、胡、操作、摸牌、发送、结束、离场。
5. 哪些动词不是客户端与服务端之间通信？打开。
6. 成为庄家是不是？成为庄家，服务端可以不给客户端发送请求，而且，我们这次的需求并没有使用到庄家，所以，暂且认为它不是。
7. 发牌是不是？发牌，发完牌之后，服务端要通知客户端哪个玩家拿了哪些牌或者多少张牌，所以，发牌可以认为是。
8. 获得是不是？同上，发完牌服务端通知客户端玩家获得了哪些牌，所以，获得也可以认为是。
9. 操作是不是？操作这个概念有点抽象，出牌、碰、杠、胡、摸牌都可以说是一种操作，所以，可以认为它是凌驾在这几个消息之上的消息，也就是抽象类的概念。
10. 发送是不是？发送本身不是消息，发送的东西才是消息，所以，发送不是。
11. 结束是不是？结束了，服务端要通知客户端，所以，结束，是。
12. 离场是不是？与结束一样，服务端要通知客户端玩家已离场。

经过上面的分析，还剩下 登录、创建、加入、开始、发牌、获得、出牌、碰、杠、胡、操作、摸牌、结束、离场。

这些消息还能不能再分类呢？

我认为可以分成三类：

1. 客户端请求服务端：登录、创建、加入。
2. 服务端通知客户端：开始、发牌、获得、结束、离场。
3. 包含以上两者，比如出牌先是服务端通知客户端，客户端玩家再出牌：出牌、碰、杠、胡、操作、摸牌。

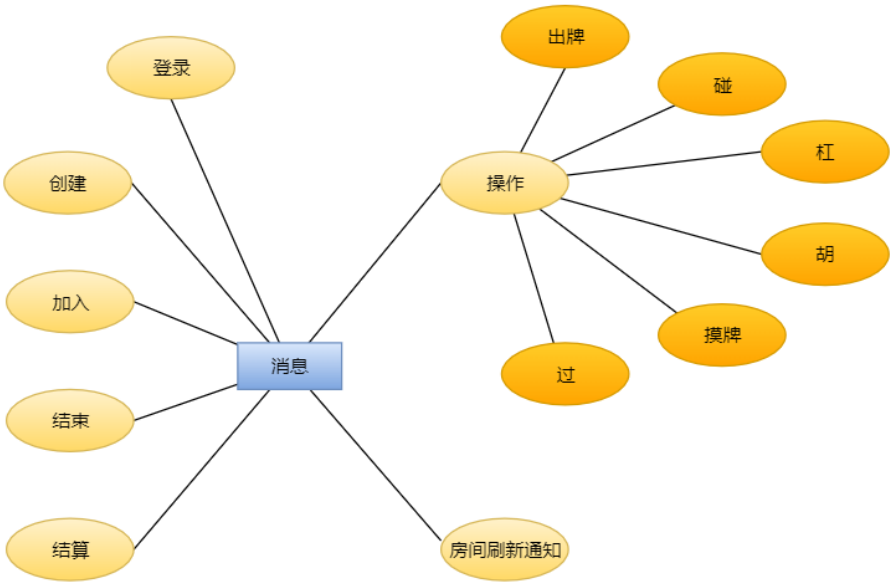
那么，每一种分类内部还能不能再抽象呢？

我认为是可以的：

1. 开始、发牌、获得，这三者都是在游戏开始的时候服务端主动通知客户端的，而且，是通知所有客户端，所以，这三条消息，我认为可以合并成一条，我们暂且称之为游戏开局通知。

2. 结束、离场，在我们的需求中，一局游戏结束，整个房间的游戏就结束了，所以，暂且也可以把这两个消息合并成一个。
3. 出牌、碰、杠、胡、摸牌，这些消息本身就是不同的操作，所以，我们可以把操作作为父类，把这几个消息作为子类来处理，其实还有个 “过” 的操作，即询问玩家碰不碰的时候，玩家不碰。
4. 另外，每一次操作之后都应该刷新牌局信息，比如，哪张牌打出了，谁摸牌了，等等，这样做的好处是防止某个客户端网络不稳定，依赖于客户端收到出牌消息来刷新牌局可能出现错乱的情况，而这个牌局的刷新其实与游戏开局通知是一样的，都是把房间的完整信息传递给客户端，所以，这两个可以合并为房间刷新通知。
5. 最后，上面的描述最后还有一个结算消息，这也是一个单独的消息。

综上所述，最后的消息有 登录、创建、加入、结束（离场）、操作（出牌、碰、杠、胡、摸牌、过）、房间刷新通知、结算等。



到这里，领域模型设计到这里基本就完成，下面，我们再来看看接口设计。

接口设计

接口设计，一般是针对 Web 系统来说的，在 Spring MVC 中，一般是指 Controller 层的设计，这个阶段，可以在领域模型设计完毕之后，立马就开始，接口设计完成之后交给客户端，两边就可以一起开发了。

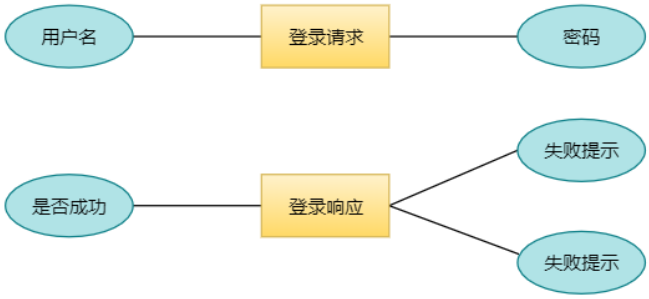
但是，我们这次的实战项目并不是 Spring MVC，接口设计在哪里呢？

其实，就是每一条消息的详细设计，在上面的领域模型设计中，我们已经归纳出了所有的消息类型，现在只需要把它们加上属性就可以了。真正意义上来说，上面的消息并不能算是领域模型设计，只是我们是第一次分析这个过程，所以，我觉得把它放在领域模型设计这一小节也是可以的，后面，我们熟悉了这个过程，可以直接把消息的设计拿到接口设计这里来。更通俗易懂地来说，在需求的描述中，名词一般就是领域模型，动词一般就是领域模型的行为，行为一般对应着接口。

好了，下面我们就来详细分析每一条消息应该具有的属性，为了方便，我们按照牌局进行的顺序来。

登录

登录分成登录请求和登录响应，登录请求需要输入用户名和密码，登录响应返回是否登录成功，同时返回玩家的信息，登录失败还需要有相应的提示，所以，对于登录，应当分化为两条消息：



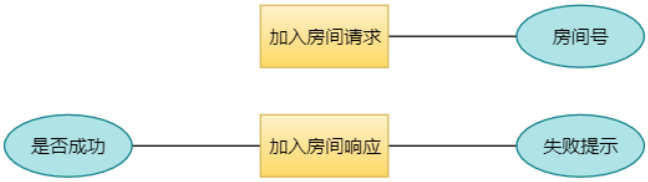
创建房间

根据上面房间的设计，应该在创建房间的时候传入人数和底分，还需要其它属性吗？

好像不需要了，需要响应吗？可以有响应，比如，服务端有检查人数和底分是否满足条件，此时，是需要给客户端一个响应告诉客户端是否创建房间成功的。

加入房间

加入房间，比较简单了，输入房间号，如果房间没满就可以加入，如果房间满了就加入失败，所以，加入房间是需要一个响应告诉客户端是否加入成功的。



房间刷新通知

房间刷新通知，顾名思义，就是将房间的信息发送给客户端，所以，直接把房间本身作为它的字段就可以了。

不过，有个问题，房间中是包含玩家信息的，每一个玩家是有牌列表的，所以，发送消息的时候要注意一下，针对不同的玩家看到的消息内容本身是不完全一样的，每个玩家都只能看到自己的牌列表，其他玩家的牌列表要隐藏起来。

那么，房间刷新通知在哪些情况下会触发呢？

1. 有个加入房间的时候；
2. 有人操作（出牌、碰、杠、胡、摸牌）的时候；

因此，还需要有个行为类型的字段，客户端拿到不同的行为类型做出不同的动画，比如，出牌就播放玩家出了一张牌的动画，摸牌就播放玩家摸了一张牌的动画。

另外，关于行为类型，除了加入房间，其它的都是牌局操作，所以，我们直接使用操作类型来表示行为类型，当没有操作类型的时候就表示为加入房间。

所以，房间刷新通知一共有两个属性：操作类型和房间。

操作消息

操作，在客户端与服务端之间是双向的，服务端首先通知客户端可以进行哪些操作（同时也要通知其他玩家），客户端操作完了，再通知回给服务端，最后，服务端再通知其他玩家，谁谁谁做了什么操作，所以，操作应当分成三种不同类型的消息：操作通知、操作请求、操作结果通知。

另外，针对不同的操作类型，需要传输的数据可能又不太一样，比如出牌通知和碰牌通知，出牌通知其他玩家是知道轮到谁出牌了的，碰牌通知其他玩家是不知道谁可以碰牌的，只有真正碰牌完毕了，其他玩家才知道，所以，这里根据不同的操作类型，又可以分化成不同的消息，比如出牌三种消息，碰牌三种消息，等等，不过，我们并不打算这么做，因为它们太像了，因为个别字段的不同就拆分成更细的消息，有点得不偿失，而且，非常不利于后期的扩展，比如，后面再加一种操作，吃，又要定义三种消息，这样就太费事了，所以，针对操作，我们一共只有三个消息，通过不同的操作类型来区分。

操作通知

经过上面的分析，操作类型是必不可少的，但是，有一种状况需要特别关注。

试想，玩家 A 出了一张牌，玩家 B 可能既能碰，也能杠，甚至还可以胡，这种情况下要怎么通知玩家 B 呢？

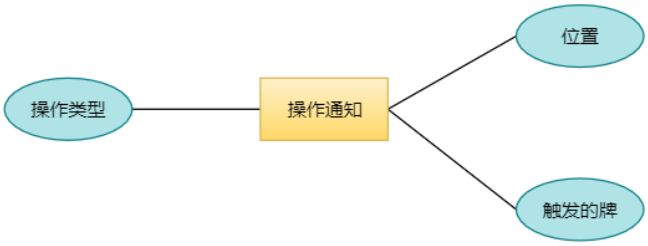
有两种方案，一种是通知里面维护一个操作列表，不过有点浪费空间，另一种方案是只使用一个 int，不同的位代表不同的操作，我们知道一个 int 有 32 位，所以，可以代表 32 种不同的操作，后期扩展也方便。

除了操作类型还需要什么字段呢？

对于出牌，还需要知道通知谁出牌，所以还需要一个位置的字段。

对于碰、杠、胡，还需要知道哪张牌触发的这些操作，所以还需要一个触发的牌的字段。

综上所述，操作通知一共需要三个字段：操作类型、位置、触发的牌。



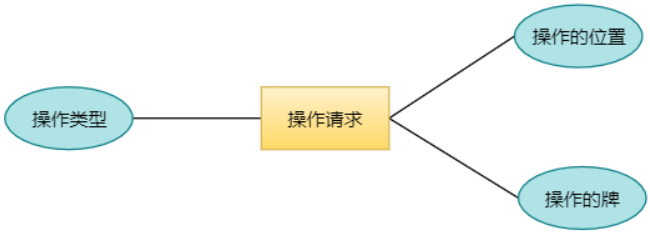
操作请求

操作类型也是必不可少的，还需要其它什么字段呢？

对于出牌，服务端需要知道哪个位置出了哪张牌。

对于碰、杠、胡，服务端也是需要知道操作了哪张牌，不过服务端似乎知道是哪张牌，因为是服务端通知客户端哪张牌触发的操作，所以，服务端应该找个地方记录这张触发的牌，放在哪里比较合适呢？我认为放在房间信息中再合适不过了。

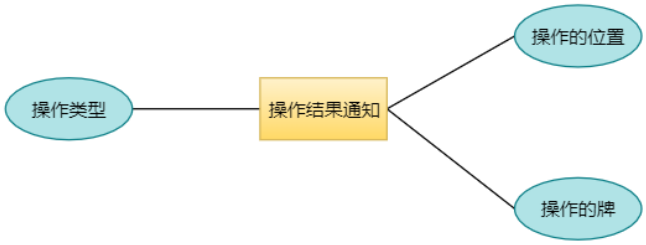
所以，操作请求一共需要两个字段：操作类型和操作的牌。



操作结果通知

同样地，操作类型也是必不可少的，还需要其它什么字段呢？

其他玩家需要知道哪位玩家做了什么操作，操作的牌是什么，所以，还需要知道操作的位置以及操作的牌。

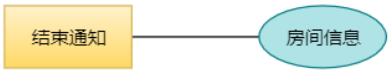


好了，操作的三个消息就设计完成了，当有玩家胡的时候，游戏就该结束了，此时，应该先发送结束消息，再发送结算消息。

结束通知

关于结束通知，也可以有两种设计，一种是与房间刷新通知放在一起，加一种结束的行为类型，同时，发送消息的时候也要注意，所有玩家的手牌都不用隐藏了，每个玩家都可以看到其他玩家的牌。不过，这样设计耦合度感觉有点高，所以，我更倾向于把结束通知单独当成一个消息进行通知。

通过上面的分析，可以知道，结束通知直接把房间的信息发送给客户端就可以了，所以，它只有一个属性：房间信息。

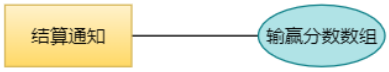


结算通知

关于结算通知，也有两种设计，一种是服务端只通知客户端谁赢了，客户端自行计算输赢的分数，另一种是服务端全部计算后返回给客户端。像我们的这次实战案例，规则比较简单，让客户端计算输赢也是可以的，不过，如果把规则弄复杂一些，加上各种番型，让客户端计算明显就不合理了，而且，服务端怎么都要计算的，所以，选择服务端全部计算完成再返回给客户端，比较合理。

那么，结算通知需要返回哪些信息呢？

针对本次的实战案例，我们需要简单地计算每位玩家的输赢的分数，而客户端拥有完整的房间信息，所以，我们只需要每个位置的输赢分数，不需要返回每个玩家的信息了，而位置可以通过数组的下标表示，因此，结算通知只需要一个输赢分数的数组，其中下标代表玩家位置。



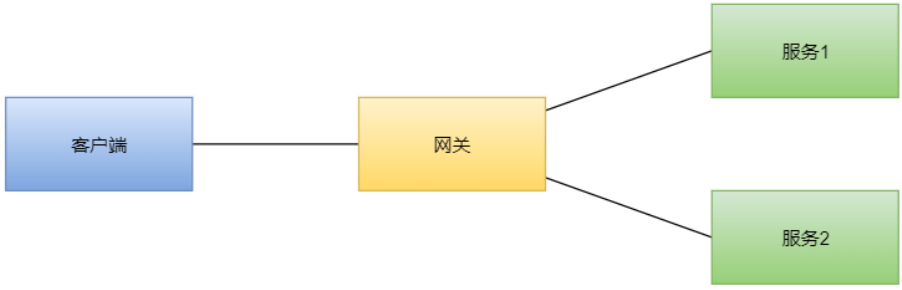
OK，到此，所有消息的属性我们都已经确定下来了，也就是完成了接口设计。

部署架构设计

有了上面的设计，研发同学就可以开工了，为什么在前期还要考虑部署架构的设计呢？

因为不同的部署架构，可能对代码的架构产生巨大的影响。

就拿我们这次的实战案例来说，如果只是单机部署，那部署架构就非常简单，可以说是没有，那么，如果改成多机部署呢？是不是需要一个网关层做流量转发？这个网关该如何设计？有没有现成的开源框架可以使用？



如果是传统的 Web 应用，我会跟你说，服务做成无状态的就可以了。

但是，我们这次做的是游戏，对于游戏应用，性能的要求非常高，所以，很多数据必须保存在本机的内存中，注意，是本机的内存中，而不是像 Redis 那种分布式缓存的内存中，因此，游戏应用的服务是不可能做成无状态的。

这时候网关就起到至关重要的作用了，对于不在牌局中的消息，玩家的请求随便分发到哪个服务都是可以的，但是，对于牌局中的消息，同一个房间所有玩家的所有消息必须分发到同一台机器的同一个 JVM，这样，这些消息才能共享内存来进行处理。但是，这样还不够快，有没有更快的方法呢？后面实现的时候我再告诉你。

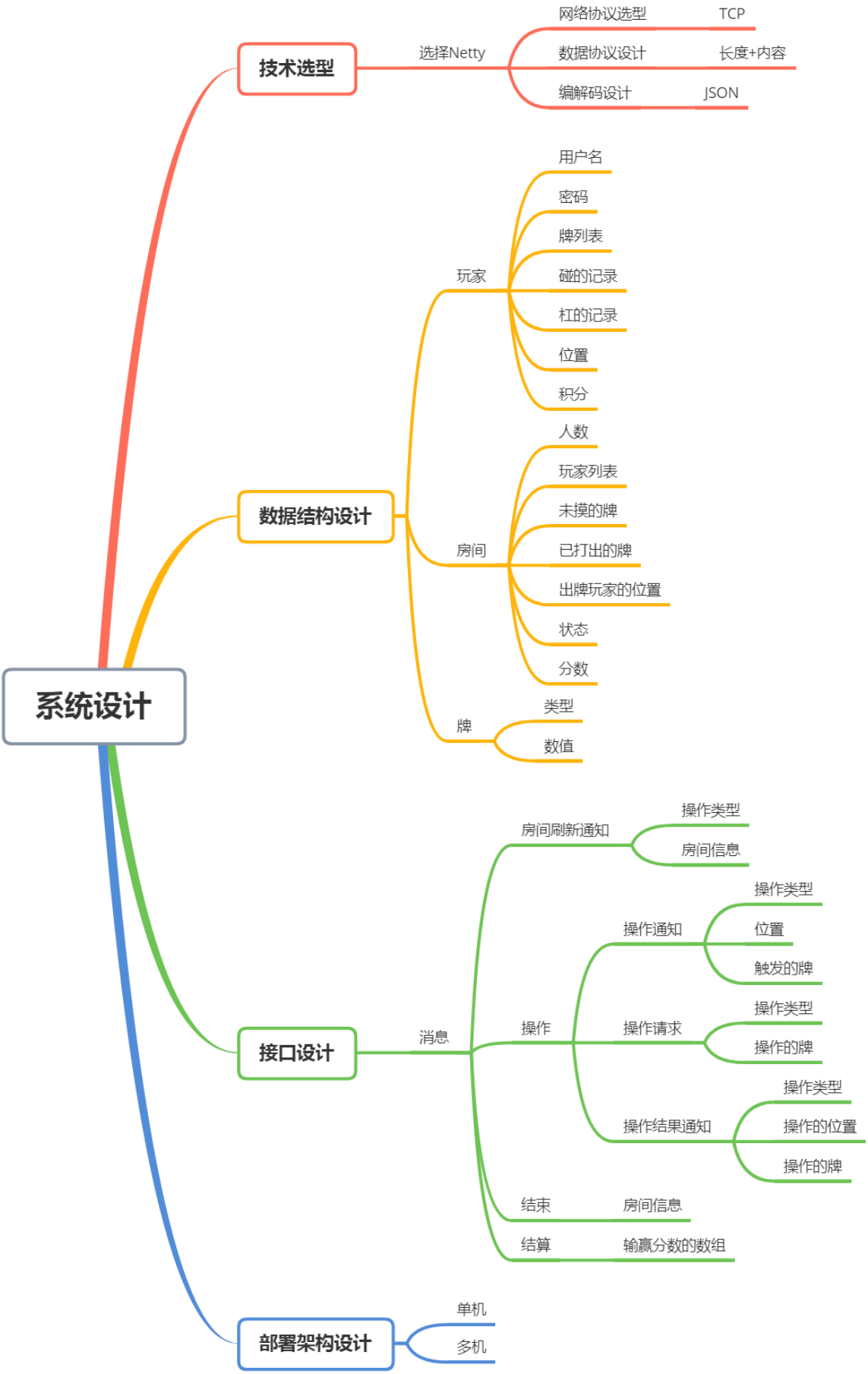
当然了，对于我们本次的实战案例，暂且只考虑单机部署的情况。

后记

本节，我们从技术选型、领域模型设计、接口设计、部署架构设计等四个方面非常全面地介绍了本次实战项目的系统设计，这些内容也是一个架构师应该具备的技能，也可以说是一个比较通用的系统设计过程，同样，也可以运用的我们的工作中。

有了这么全面的设计，我相信实现起来会变得非常容易，我已经迫不及待了，你呢？

思维导图


系统设计
技术选型
选择Netty
网络协议选型
TCP
数据协议设计
长度+内容
编解码设计
JSON
数据结构设计
玩家
用户名
密码
牌列表
碰的记录
杠的记录
位置
积分
房间
人数
玩家列表
未摸的牌
已打出的牌
出牌玩家的位置
状态
分数
牌
类型
数值
接口设计
消息
房间刷新通知
操作类型
房间信息
操作
操作通知
操作类型
位置
触发的牌
操作请求
操作类型
操作的牌
操作结果通知
操作类型
操作的位置
操作的牌
结束
房间信息
结算
输赢分数的数组
部署架构设计
单机
多机

}

28 需求分析，能砍则砍，最小化闭环

30 协议实现，双端打通