技术架构图说明

1. 架构设计目标

➢必须满足的功能需求和非功能需求。

➢尽量利用成熟的开源技术，降低成本。

➢架构必须满足简单、安全升级扩展、伸缩。

➢全面、准确地界定系统架构的涉及的范围，清楚定义内部系统与外部系统的边界以及交互约定。

➢确定各子系统的具体职责，子系统间的交互协议和交互过程。

➢确定系统的开发、部署和运维平台和相应的规范。
➢为系统设计和后续编码、测试、维护提供原则、基础和规范。

2. 架构设计原则

➢**满足功能性需求和满足非功能需求。**这是一个软件系统最基本的要求，也是架构设计时应该遵循的最基本的原则。

➢**实用性原则。**就像每一个软件系统交付给用户使用时必须实用，能解决学校的实际问题，并且系统必须稳定可靠、性能良好。

➢**可复用性原则。**标准模块设计成可重复使用的中间件或标准组件，最大程度的提高开发人员的工作效率。

➢**单一职责原则。**各子系统专注于完成一个特定功能，不交叉耦合，系统间通过设计良好的接口交互。

➢**可扩展原则。**应用面向服务的思想，业务模块子系统通过提供服务的方式对外提供接口，以后有业务扩展可直接扩展服务即可。系统能够灵活的根据业务量的大小来灵活部署，即可集中式部署也可分布式部署。

3. 详细说明

3.1. APP 端及 Web 端与服务器通讯采用 https

1）认证服务器。客户端内置一个受信任的 CA 机构认证的证书。第一阶段服务器会提供经 CA 机构认证颁发的服务器证书，如果认证该服务器证书的 CA 机构，存在于客户端的受信任 CA 机构列表中，并且服务器证书中的信息与当前正在访问的网站（域名等）一致，那么客户端就认为服务端是可信的，并从服务器证书中取得服务器公钥，用于后续流程。否则，客户端将不会继续发起请求。

2）协商会话密钥。客户端在认证完服务器，获得服务器的公钥之后，利用该公钥与服务器进行加密通信，协商出两个会话密钥，分别是用于加密客户端往服务端发送数据的客户端会话密钥，用于加密服务端往客户端发送数据的服务端会话密钥。在已有服务器公钥，可以加密通讯的前提下，还要协商两

个对称密钥的原因，是因为非对称加密相对复杂度更高，在数据传输过程中，使用对称加密，可以节省计算资源。另外，会话密钥是随机生成，每次协商都会有不一样的结果，所以安全性也比较高。

3）加密通讯。此时客户端服务器双方都有了本次通讯的会话密钥，之后传输的所有 Http 数据，都通过会话密钥加密。这样网路上的其它用户，将很难窃取和篡改客户端和服务端之间传输的数据，从而保证了数据的私密性和完整性。

3.2. 为防止客户端证书被伪造，客户端使用证书锁定技术

如果用户手机中安装了一个恶意证书，那么就可以通过中间人攻击的方式进行窃听用户通信以及修改 request 或者 response 中的数据。

手机中间人攻击过程：

1）客户端在启动时，传输数据之前需要客户端与服务端之间进行一次握手，在握手过程中将确立双方加密传输数据的密码信息。

2）中间人在此过程中将客户端请求服务器的握手信息拦截后，模拟客户端请求给服务器（将自己支持的一套加密规则发送给服务器），服务器会从中选出一组加密算法与 HASH 算法，并将自己的身份信息以证书的形式发回给客户端。证书里面包含了网站地址，加密公钥，以及证书的颁发机构等信息。

3）而此时中间人会拦截下服务端返回给客户端的证书信息，并替换成自己的证书信息。

4）客户端得到中间人的 response 后，会选择以中间人的证书进行加密数据传输。

5）中间人在得到客户端的请求数据后，以自己的证书进行解密。

6）在经过窃听或者是修改请求数据后，再模拟客户端加密请求数据传给服务端。就此完成整个中间人攻击的过程。

防护办法：

1）公钥锁定

将证书公钥写入客户端 apk 中，https 通信时检查服务端传输时证书公钥与 apk 中是否一致。

2）证书锁定：

即为客户端颁发公钥证书存放在手机客户端中，在 https 通信时，在客户端代码中固定去取证书信息，不是从服务端中获取。

3.3. 应用网关具有对应用接口的负载均衡及熔断能力，并可进行失败重试

1）负载均衡

接口的请求压力有时候会超过单个服务实例的处理能力，这时候我们需要启动多个服务的实例，为了避免请求压力传导到其中一个实例，我们在网关层面执行轮询策略，即每次请求都分散到不同的服务实例上。

2）路由熔断

当我们的后端服务出现异常的时候，我们不希望将异常抛出给最外层，期望服务可以自动进行一降级。当某个服务出现异常时，直接返回我们预设的信息。
我们通过自定义的 fallback 方法，并且将其指定给某个 route 来实现该 route 访问出问题的熔断处理。
3）失败重试
有时候因为网络或者其它原因，服务可能会暂时的不可用，这个时候我们希望可以再次对服务进行重试。
我们通过自定义重试策略当某个服务出现异常时尝试调用其它服务实例或返回指定信息。开启重试在某些情况下是有问题的，比如当压力过大，一个实例停止响应时，路由将流量转到另一个实例，很有可能导致最终所有的实例全被压垮。

3.4. 缓存集群主要服务于后台及接口服务

后端接口服务总共分为 3 个层次：
1）网关层
主要处理鉴权，熔断，负载均衡，失败重试
2）应用服务层
负责业务逻辑处理
3）基础服务层
提供基础数据的增删改查及队列任务处理

一般情况下我们会将数据的缓存处理放在应用服务层，因为这一层最了解业务需求，可以根据业务的特点组织缓存数据的结构，甚至可以将从多个基础服务获取的数据组合起来缓存起来使用，更具灵活性。但是如果确实有必要，也可以由基础服务层提供，但前提是基础服务层必须提供非缓存的接口。

3.5. 应用层使用多数据源访问数据库

同一个项目有时会涉及到多个数据库，也就是多数据源。多数据源又可以分为两种情况：
1）两个或多个数据库没有相关性，各自独立，其实这种可以作为两个项目来开发。比如在游戏开发中一个数据库是平台数据库，其它还有平台下的游戏对应的数据库；
2）两个或多个数据库是 master-slave 的关系，比如有 mysql 搭建一个 master-master，其后又带有多个 slave；或者采用 MHA 搭建的 master-slave 复制；
3）采用 spring 配置文件直接配置多个数据源
针对两个数据库没有相关性的情况，可以采用直接在 spring 的配置文件中配置多个数据源，然后分别进行事务的配置
4）基于 AbstractRoutingDataSource 和 AOP 的多数据源的配置
基本原理是，我们自己定义一个 DataSource 类 ThreadLocalRountingDataSource，来继承 AbstractRoutingDataSource，然后在配置文件中向 ThreadLocalRountingDataSource 注入 master 和 slave 的数据源，然后通过 AOP 来灵活配置，在哪些地方选择 master 数据源，在哪些地方需要选择 slave 数据源。

3.6. 只能通过公共服务调用区块链及 IPFS

将文件存于 IPFS 网络，数据将会分散到各个网络节点。当需要获取文件时，只需要使用存放文件时返回的 hash 值，在 IPFS 网络里查找数据片，重新组装即可。

3.7. 数据库主从复制采用 GTID 方式

GTID 的概述：

1）全局事物标识：global transaction identifieds。

2）GTID 事物是全局唯一性的，且一个事务对应一个 GTID。

3）一个 GTID 在一个服务器上只执行一次，避免重复执行导致数据混乱或者主从不一致。

4）GTID 用来代替 classic 的复制方法，不在使用 binlog+pos 开启复制。而是使用 master_auto_postion=1 的方式自动匹配 GTID 断点进行复制。

5）MySQL-5.6.5 开始支持的，MySQL-5.6.10 后开始完善。

6）在传统的 slave 端，binlog 是不用开启的，但是在 GTID 中，slave 端的 binlog 是必须开启的，目的是记录执行过的 GTID（强制）。

GTID 的组成部分：

前面是 server_uuid：后面是一个序列号

例如：server_uuid：sequence number

7800a22c-95ae-11e4-983d-080027de205a:10

UUID：每个 mysql 实例的唯一 ID，由于会传递到 slave，所以也可以理解为源 ID。

Sequence number：在每台 MySQL 服务器上都是从 1 开始自增长的序列，一个数值对应一个事务。

GTID 比传统复制的优势：

1）更简单的实现 failover，不用以前那样在需要找 log_file 和 log_Pos。

2）更简单的搭建主从复制。

3）比传统复制更加安全。

4）GTID 是连续没有空洞的，因此主从库出现数据冲突时，可以用添加空事物的方式进行跳过。

GTID 的工作原理：

1）master 更新数据时，会在事务前产生 GTID，一同记录到 binlog 日志中。

2）slave 端的 i/o 线程将变更的 binlog，写入到本地的 relay log 中。

3）sql 线程从 relay log 中获取 GTID，然后对比 slave 端的 binlog 是否有记录。

4）如果有记录，说明该 GTID 的事务已经执行，slave 会忽略。

5）如果没有记录，slave 就会从 relay log 中执行该 GTID 的事务，并记录到 binlog。

6）在解析过程中会判断是否有主键，如果没有就用二级索引，如果没有就用全部扫描。