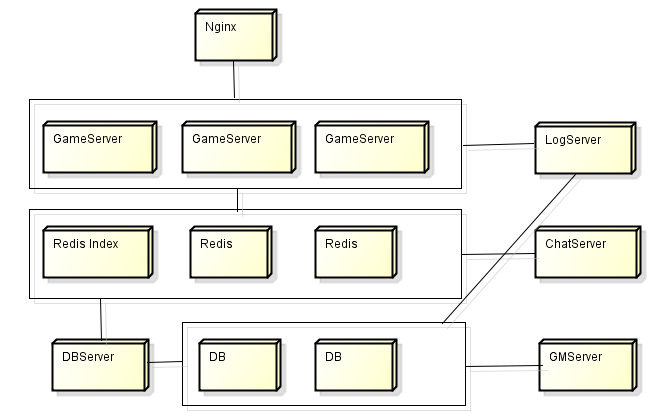
**服务器总体架构**

1. **服务器架构概述**
2. 采用大世界架构。
3. Nginx服务器直接与客户端交互，负责客户端数据的接收和返回。
4. 游戏服务器是WEB容器，多线程结构。所有游戏服务器都可以处理所有的游戏业务请求。单个游戏服务器的崩溃不会影响其他游戏服务器的运行。不保存数据，所有请求都是从Redis中取数据。战斗逻辑主要在客户端计算，没有服务器刷怪。
5. 使用同一个独立的DBS将Redis中的数据存入Mysql
6. 使用同一个独立的聊天服务器，支持所有游戏服玩家聊天。
7. 使用唯一的日志服务器保存所有服务器产生的日志
8. 使用唯一的GM服务器管理游戏世界
9. **服务器结构**



1. Nginx负责分发客户端的访问，根据客户端cookie定向到固定的GameServer中。
2. GameServer是没有状态的，多个GameServer的游戏内容都是相同的，可以处理游戏的任意请求。每次请求GameServer首先从缓存中读取数据，如果没有，再从数据库中读取。GameServer添加/更新/删除缓存中的数据。
3. Redis作为数据缓存服务器，在起服或者用户上线时会加载数据到Redis中，如果下次请求直接会从缓存中读取数据。Reids分为索引缓存和数据缓存，数据缓存可拓展为多个。
4. DBServer负责将缓存中的数据更新到数据库。为保证性能，DBS采用非实时的异步更新方式，避免玩家频繁操作的数据库造成压力，允许有一定的延时（可配），延时的数据如果丢失，可以容忍。
5. DB（Mysql数据库）也可以扩充为多个。
6. **负载均衡策略**

Nginx在接收到客户端请求后，使用以下策略分发到GameServer：

* 1. 用户在登录时，这个时候客户端携带信息(cookie)无状态，nginx按照默认规则分发到一个GameServer容器中，进行登录。如果处理登录的GameServer中已经记录该玩家所在GameServer，此时处理登录的GameServer重新写入cookie并返回重新登录消息到客户端，客户端再次请求登录。
  2. 如果玩家登录时，客户端携带信息(cookie)有状态，nginx根据cookie值分发到固定的一个GameServer容器中。

根据上面的nginx分发策略，一个玩家的数据交互都是在同一个GameServer容器中，这样就大大减少对玩家加锁的操作，只需要在容器中相同用户保证数据处理有序即可。

GameServer，Redis，DB负载均衡的策略都是选择玩家数量最少的服务器分配给玩家。

1. **通讯协议**

详情参照《[协议格式.docx](../2-通讯协议/协议格式.docx)》。玩家登录时下发的玩家基本信息、战报信息是两个会比较费流量的消息，需要进行压缩处理。

1. **存储设计**

详情参照《[Redis缓存.docx](数据存储/Redis缓存.docx)》、《[DBS服务器.docx](数据存储/DBS服务器.docx)》、《[数据库框架.docx](数据存储/数据库框架.docx)》

1. **应急能力**
2. 游戏服务器出现数据错误情况，如大规模刷钱等数据异常时，可以通过GM后台查询日志服务器记录的玩家数据变化日志，排查具体出错原因。然后可以直接上传一个JSP文件进行紧急修复。
3. 游戏服务器出现程序异常情况，可以通过游戏服务器上的Log4j日志排查问题。
4. 更新版本可以不需要停服即可进行。热部署时，动态增加一个GameServer，直接在Nginx中配置，新玩家将会登录到负载最小的新GameServer中，老用户还会在之前的GameServer。
5. **安全**
   1. 客户端不可信，客户端传输过来的东西，关键的验证点都放在服务器。
   2. 协议采用二进制格式，并用异或的方式加密。
6. **潜在问题**
7. Redis Index、DBServer、ChatServer都存在单点故障问题，其中Redis Index可以采用主从方式部署，规避单点问题。
8. DBS更新延迟，可能造成短时间的回档，需要控制在玩家可容忍范围内。
9. 分布式架构可能会带来的程序复杂性问题。