|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **修改记录** | | | | |
| **序号** | **版本号** | **修改人** | **修改时间** | **修改备注** |
| 1 | V1.0.0 | 史鹏刚 | 2014-07-02 | 初稿 |
|  |  |  |  |  |
|  |  |  |  |  |

# 前期预言

## 难点分析

* 学生排队等待

如果学生排队时间过长，难免会影响用户体验。

* 高并发

假如有10W活跃用户，1W并发，如果使用tomcat作为容器，单节点部署，则并发请求将成为最大的瓶颈。

* 多应用长链接

如果每个应用都保持一个长链接到服务端，设备耗电将成为最大痛点。

* 富媒体实时传输

如何做好图像、音频、视频的实时传输将是最大难点。

* 安全问题

所有对外接口都会被别人攻击或非法访问。

* 消息可靠性

学生提交的所有请求消息是否需要入库保存？老师的回答是否需要？

* 负载均衡

如何应对单节点请求受限问题？

## 对应解决方案、思路

* 对于学生长时间排队问题，最好的解决办法肯定是增加更多老师做支持，我们无须考虑，但是设计时应该考虑排队优先级，因为如果学生加入会员等级机制，排队优先级是难免的。
* 作为互联网应用，高并发问题肯定是在做难免的，所以可考虑使用netty框架，netty可以脱离tomcat等web容器单独部署，支持高并发、高性能、高可靠性，基于NIO实现的异步IO通信框架。
* 客户端在SDK内嵌入单独服务，此服务主要负责客户端所有应用与服务器之间的通信，只需要此应用与服务器保持一个单独的长链接。
* 富媒体要做到实时性传输不仅需要很好的网络环境做支持，更需要高性能的代码来处理响应的业务，所有对代码质量需要做好严格把关。
* 所有的对外接口应该都进行基础的签名校验。
* 所有请求消息和应答消息可根据实际需求或消息类型来做可靠性存储。如某些请求可能本身就属于垃圾请求，此类消息可选择丢弃（可能需要人工参与），回复消息可根据题目对应存储。
* 采用Zookeeper做分布式多节点管理。

# 系统架构设计

## 整体架构设计说明



**说明：**系统整体分为简单的服务层和客户端层，客户端即我们的pad终端；服务端主要分为配置中心、推送中心和Zookeeper集群节点管理。

各模块职责

* 配置中心：配置中心主要负责公共数据的配置以及推送消息入口。
* Zookeeper：集群管理节点，主要负责集群数据管理。
* 推送中心：主要负责接受配置中心消息，同时根据需求将消息提送到客户端。

## 系统序列图



流程说明：

1. 各个推送节点启动的时候将自己的连接信息（IP/PORT）上报到Zookeeper，Zookeeper保存各个推送节点的信息。
2. 当APP请求链接时，先在zookeeper上获取推送中心链接信息，然后根据返回结果去链接实际的推送中心，连接成功后一直保持此连接。
3. 当需要给客户端推送消息时，通过配置中心直接给推送中心发送消息，然后由推送中心统一处理消息，分发到各个客户端。

# 通信协议

## 自定义协议

对所有的消息传递，为了避免协议解析的压力，所以，所有的请求和相应都采用统一的协议进行消息传送，报文头在前，消息数据在后。

如：

报文头组成：编码方式（1byte）、加密（1byte）、扩展1（1byte）、扩展2（1byte）、会话ID（4byte）、命令或者结果码（4byte）、数据包长（4byte）

数据包：Object

## 编码方式

数据编码：编码方式可采用netty中现有集成的编码解码器进行编解码（如StringDecoder、StringEncoder，ObjectEncoder、ObjectDecoder等）。

# Netty简单介绍（来自网络）

## Netty是什么？

Netty是一个基于JAVA NIO类库的异步通信框架，它的架构特点是：异步非阻塞、基于事件驱动、高性能、高可靠性和高可定制性。

## 使用Netty能做什么？

* 开发异步、非阻塞的TCP网络应用程序；
* 开发异步、非阻塞的UDP网络应用程序；
* 开发异步文件传输应用程序；
* 开发异步HTTP服务端和客户端应用程序；
* 提供对多种编解码框架的集成，包括谷歌的Protobuf、Jbossmarshalling、Java序列化、压缩编解码、XML解码、字符串编解码等，这些编解码框架可以被用户直接使用；
* 提供形式多样的编解码基础类库，可以非常方便的实现私有协议栈编解码框架的二次定制和开发；
* 基于职责链模式的Pipeline-Handler机制，用户可以非常方便的对网络事件进行拦截和定制；
* 所有的IO操作都是异步的，用户可以通过Future-Listener机制主动Get结果或者由IO线程操作完成之后主动Notify结果，用户的业务线程不需要同步等待；
* IP黑白名单控制；
* 打印消息码流；
* 流量控制和整形；
* 性能统计；
* 基于链路空闲事件检测的心跳检测
* ……

## 为什么选择Netty？

Netty是业界最流行的NIO框架之一，它的健壮性、功能、性能、可定制性和可扩展性在同类框架中都是首屈一指的，它已经得到成百上千的商用项目验证，例如Hadoop的RPC框架Avro使用Netty作为通信框架。很多其它业界主流的RPC和分布式服务框架，也使用Netty来构建高性能的异步通信能力。

Netty的优点总结如下：

* API使用简单，开发门槛低；
* 功能强大，预置了多种编解码功能，支持多种主流协议；
* 定制能力强，可以通过ChannelHandler对通信框架进行灵活的扩展；
* 性能高，通过与其它业界主流的NIO框架对比，Netty的综合性能最优；
* 成熟、稳定，Netty修复了已经发现的所有JDK NIO BUG，业务开发人员不需要再为NIO的BUG而烦恼；
* 社区活跃，版本迭代周期短，发现的BUG可以被及时修复，同时，更多的新功能会被加入；
* 经历了大规模的商业应用考验，质量得到验证。在互联网、大数据、网络游戏、企业应用、电信软件等众多行业得到成功商用，证明了它完全满足不同行业的商用标准。
* 正是因为这些优点，Netty逐渐成为Java NIO编程的首选框架。

附：http://www.blogjava.net/yongboy/archive/2013/05/13/399203.html