MPush开发手册 -- Push IM 消息推送 Android 物联网, 免费, 开源, 安全, 加密, 集群 V1.0

第一章 系统介绍

系统介绍

mpush

是一款开源的及时消息推送服务，开发语言为java，基于Netty4开发。

发展状况

> 2015.12 项目启动
> 2016.2 1.0-beta
> 2016.3 1.0-release 上线
> 2016.5 开源预研,业务剥离
> 2016.8 正式开源v0.0.1
> 2016.9 v0.0.2, v0.0.3
> 2016.10 v0.0.4, v0.0.5
> 2016.11 v0.0.6
> 2016.12 v0.6.1
> 2017.1 v0.7.0
> 2017.2 v0.7.1

后续目标

因为Java的开源消息系统非常少，希望有兴趣的朋友可以一起参与进来，做一个最好的java开源消息推送系统。 有兴趣加入开发组的请加我QQ。

项目成员

夜色[QQ:251939168]、黄志磊、魏永霖、老佛爷

帮助与支持

QQ群: 114583699

捐助

图片地址：https://static.oschina.net/uploads/img/201703/09152814_k2YZ.png

功能特点

1. 源码全部开放，包括server、android、ios 等
2. 代码质量高，全部模块化设计，真正的商用级产品，考虑到推送中遇到的大部分场景
3. 安全性高，基于RSA精简的加密握手协议，简单，高效，安全
4. 支持断线重连，及弱网下的快速重连，无网络下自动休眠节省电量和资源
5. 协议简洁，接口流畅，支持数据压缩，更加节省流量
6. 支持集群部署，支持负载均衡，基于成熟的zookeeper实现
7. 用户路由使用redis集群，支持单写，双写，集群分组；性能好，可用性高
8. 支持http代理，一根TCP链接接管应用大部分请求，让http请求更加及时
9. 高度可配置化，基本上通过修改配置可满足大部分场景
10. 扩展性强，高度模块化，基于SPI模式的可拔插设计，以满足特殊需求
11. 监控完善，日志详细，可快速排查线上问题及服务调优

第二章 快速上手

源码地址

Github源码

- group https://github.com/mpusher/ 源代码空间
- server https://github.com/mpusher/mpush 服务端源码
- alloc https://github.com/mpusher/alloc 调度器源码
- mpns https://github.com/mpusher/mpns 个性化推送中心源码
- java-client https://github.com/mpusher/mpush-client-java 纯java客户端源码
- android sdk&demo https://github.com/mpusher/mpush-android 安卓SDK和DEMO源码
- IOS sdk(swift) https://github.com/mpusher/mpush-client-swift swift版客户端源码
- IOS sdk(OC) https://github.com/mpusher/mpush-client-oc Object C 客户端源码
- websoket(JS) https://github.com/mpusher/mpush-client-js websocket js 客户端源码

码云源码

- group http://git.oschina.net/mpush 源代码空间
- server http://git.oschina.net/mpush/mpush 服务端源码
- alloc http://git.oschina.net/mpush/alloc 调度器源码

• java-client http://git.oschina.net/mpush/mpush-client-java 纯java客户端源码
• android sdk&demo http://git.oschina.net/mpush/mpush-android 安卓SDK和DEMO源码
• IOS sdk(swift) http://git.oschina.net/mpush/mpush-client-swift swift版客户端源码

服务部署

部署提示：

1. 详细部署文档请点此查看(https://github.com/mywiki/mpush-doc/blob/master/SUMMARY.md)，以下仅仅是mpush server 本身的部署
2. mpush 服务只依赖于zookeeper和redis，当然还有JDK>=1.8

部署流程

1. 安装jdk 1.8 以上版本并设置%JAVA_HOME%
2. 安装zookeeper (安装配置步骤略)
3. 安装Redis (安装配置步骤略)
4. 下载mpush server最新的正式包 https://github.com/mpusher/mpush/releases(https://github.com/mpusher/mpush/releases)

(也可以自己根据源码构建:mvn clean package -Pzip,pub)

1. 解压下载的tar包tar -zvxf mpush-release-x.y.z.tar.gz到 mpush 目录, 结构如下

> drwxrwxr-x 2 shinemo shinemo 4096 Aug 20 09:30 bin —> 启动脚本
> drwxrwxr-x 2 shinemo shinemo 4096 Aug 20 09:52 conf —> 配置文件
> drwxrwxr-x 2 shinemo shinemo 4096 Aug 20 09:29 lib —> 核心类库
> drwxrwxr-x 1 shinemo shinemo 1357 May 31 11:07 LICENSE
> drwxrwxr-x 2 shinemo shinemo 4096 Aug 20 09:32 logs —> 日志目录
> drwxrwxr-x 1 shinemo shinemo 4096 May 31 11:07 README.md
> drwxrwxr-x 2 shinemo shinemo 4096 Aug 20 09:52 tmp

1. 修改 conf 目录下的 vi mpush.conf文件, mpush.conf里的配置项会覆盖同目录下的reference.conf文件

```
mp.log.level=debug
mp.core.min-heartbeat=10s
mp.core.max-heartbeat=10s
mp.core.compress-threshold=10k //启用压缩阀值
mp.zk.server-address="127.0.0.1:2181" //zookeeper地址
mp.redis {// redis 集群配置
  nodes:["127.0.0.1:6379"]//格式是ip:port,密码可以没有ip:port
```

```
}
mp.http.proxy-enabled=true //启用http代理
mp.net.gateway-server-net=udp //网关服务使用的网络类型tcp/udp
```

```
如果要修改其他配置请参照`reference.conf`文件
```

1. 给bin目录下的脚本增加执行权限chmod u+x *.sh
2. 执行./mp.sh start 启动服务, 查看帮助./mp.sh 目前支持的命令：

> Usage: ./mp.sh {start|start-foreground|stop|restart|status|upgrade|print-cmd}
>
> set-env.sh用于增加和修改jvm启动参数，比如堆内存、开启远程调试端口、开启jmx等

1. cd logs目录，cat mpush.out查看服务是否启动成功

集成开发

1. 添加Maven依赖到工程

```
<dependency>
  <groupId>com.github.mpusher</groupId>
  <artifactId>mpush-boot</artifactId>
  <version>x.y.z</version>
</dependency>
```

1. 启动入口类 com.mpush.bootstrap.ServerLauncher.java

> 启动调用 start 方法，停止调用 stop 方法

1. 在工程里添加classpath下添加application.conf配置文件，配置方式参照服务部署(http://mpush.mydoc.io?v=24639&t=134336)第6点
2. spring bean 方式配置

```
<bean class="com.mpush.bootstrap.ServerLauncher" init-method="start" destroy-method="stop"/>
```

源码测试

1. git clone https://github.com/mpusher/mpush.git
2. 导入到eclipse或Intellij IDEA
3. 打开mpush-test模块，所有的测试代码都在该模块下
4. 如果没有安装Redis和Zookeeper，请修改如下几个类的注解为@Spi(order = -1)

```
com.mpush.test.spi.SimpleCacheMangerFactory //模拟Redis
com.mpush.test.spi.SimpleMQClientFactory    //模拟MQ pub/sub
```

```
com.mpush.test.spi.SimpleRegistryFactory    //模拟ZK服务注册
com.mpush.test.spi.SimpleDiscoveryFactory   //模拟ZK服务发现
```

1. 修改配置文件src/test/resource/application.conf文件修改方式参照服务部署(http://mpush.mydoc.io?v=24639&t=134336)第6点
2. 运行com.mpush.test.sever.ServerTestMain.java启动长链接服务
3. 运行com.mpush.test.client.ConnClientTestMain.java 模拟一个客户端
4. 运行com.mpush.test.push.PushClientTestMain.java 模拟给用户下发消息
5. 可以在控制台观察日志看服务是否正常运行，消息是否下发成功
6. websocket 客户端测试地址 http://127.0.0.1:8080/index.html
7. alloc 启动入口 com.shinemo.mpush.alloc.AllocServerMainTest.java

第三章 架构分析

系统架构

图片地址：https://static.oschina.net/uploads/img/201610/29215003_BWQU.png

说明

1. 最左侧三大组件分别是日志系统、监控系统、控制台治理服务

• Log System 主要负责业务日志大输出，主要有链接相关日志、推送链路日志、心跳日志、监控日志等

• Monitor 主要用作系统状态监控，可用于系统调优，包括jvm内存，线程，线程池，系统堆栈，垃圾回收情况，内存泄漏情况等。

• AdminServer主要用于在控制台对单台机器进行控制与探查，比如参看连接数量，在线用户数，取消本级ZK注册，关闭服务等

1. 右侧三个分别是ZK服务，配置中心和安全工具箱

• ZK Client 主要负责注册长链接ip:port,网关ip:port以及监听各个节点变化，同时增加了缓存

• ConfigCenter 是MPUSH Server 配置化的关键，贯穿到各个模块，非常重要

• Sercutity Box 主要实现了RSA加密，DES加密，会话密钥生成及Session复用(用于快速重连)

1. Core模块分别是长链接服务，网关服务，Packet编解码及分发模块，Message序列化及处理模块

• ConnectServer用于维持和客户端之间的TCP通道，主要负责和客户端交互

• GatewayServer用于处理Mpush Server之间的消息交互比如踢人，发送PUSH

• Packet主要是协议部分的编解码和包的完整性校验，最大长度校验等

• PacketReceiver主要负责消息的分发，分发是根据Command来的

• Connection/ConnectionManager主要负责链接管理，定时检查链接空闲情况，是否读写超时，如果链接断开发出相应的事件给路由中心去处理

• Message部分是整个的业务核心处理了处理消息的序列化，还有压缩、加密等，MessageHandler会根据不同消息独立处理自己所属的业务，主要有：心跳响应、握手及密钥交换、快速重连、绑定/解绑用户、http代理、消息推送等

1. 路由中心主要包括：本地路由，远程路由，用户在线管理三大块

• LocalRouterManager负责维护用户 + 设备与链接(connection)之间的关系

• RemoteRouterManager负责维护用户 + 设备与链接所在机器IP之间的关系

• UserManager主要处理用户上下线事件的广播，以及单台机器的在线用户及数量的维护和查询

1. 是MPUSH的缓存部分，目前只支持Redis,支持双写，主备，hash 等特性

服务依赖关系

系统调用关系图

图片地址：https://static.oschina.net/uploads/img/201610/29222817_jMc7.png

说明

1. 业务系统是要发送业务消息的服务，所有要推送的消息直接转给MPNS

2. MPNS是我们的业务推送系统，负责消息推送，长链接的检查，离线消息存储，用户打标等

3. APNS、JPUSH、MPUSH等分别是我们的客户端已经接入的推送系统

4. MPNS主要是为了隔离业务系统和各种推送系统，用户使用哪个长链接服务，业务系统不需要感知，统一有MPNS去选择、去切换

5. Alloc负责调度维护MPushServer集群，提供查询可用机器列表的接口，详细参见[Alloc实现]

协议格式

协议说明

• mpush使用的为自定义私有协议，定长Header + body, 其中header部分固定13个字节。

• 心跳固定为一个字节，值为 -33。

Header 说明

名称	类型	长度	说明
length	int	4	表示body的长度
cmd	byte	1	表示消息协议类型
checkcode	short	2	是根据body生成的一个校验码

flags	byte	1	表示当前包启用的特性，比如是否启用加密，是否启用压缩
sessionId	int	4	消息会话标识用于消息响应
lrc	byte	1	纵向冗余校验，用于校验header

消息类型

mpush 目前支持如下消息类型

```
public enum Command {
    HEARTBEAT(1),          // 心跳
    HANDSHAKE(2),          // 握手
    LOGIN(3),
    LOGOUT(4),
    BIND(5),               // 绑定用户
    UNBIND(6),             // 解绑用户
    FAST_CONNECT(7),       // 快速重连
    PAUSE(8),
    RESUME(9),
    ERROR(10),             // 错误消息
    OK(11),                // 成功消息
    HTTP_PROXY(12),        // HTTP代理
    KICK(13),              // 踢人
    GATEWAY_KICK(14),
    PUSH(15),              // 推送
    GATEWAY_PUSH(16),
    NOTIFICATION(17),
    GATEWAY_NOTIFICATION(18),
    CHAT(19),
    GATEWAY_CHAT(20),
    GROUP(21),
    GATEWAY_GROUP(22),
    ACK(23),
    UNKNOWN(-1);
}
```

系统配置

```
#####################################################################
#################################################
#
# NOTICE：
#
# 系统配置文件，所有列出的项是系统所支持全部配置项
# 如果要覆盖某项的值可以添加到mpush.conf中。
#
# 配置文件格式采用HOCON格式。解析库由https://github.com/typesafehub/config提供
。
# 具体可参照器说明文档，比如含有特殊字符的字符串必须用双引号包起来。
#
#####################################################################
#################################################

mp {
   #基础配置
   home=${user.dir} //程序工作目前

   #日志配置
   log-level=warn
   log-dir=${mp.home}/logs
   log-conf-path=${mp.home}/conf/logback.xml

   #核心配置
   core {
      max-packet-size=10k //系统允许传输的最大包的大小
      compress-threshold=10k //数据包启用压缩的临界值，超过该值后对数据进行压缩
      min-heartbeat=3m //最小心跳间隔
      max-heartbeat=3m //最大心跳间隔
      max-hb-timeout-times=2 //允许的心跳连续超时的最大次数
      session-expired-time=1d //用于快速重连的session 过期时间默认1天
      epoll-provider=netty //nio:jdk自带，netty:由netty实现
   }

   #安全配置
   security {
      #rsa 私钥、公钥key长度为1024;可以使用脚本bin/rsa.sh生成, @see
com.mpush.tools.crypto.RSAUtils#main
      private-
```

```
key="MIIBNgIBADANBgkqhkiG9w0BAQEFAASCASAwggEcAgEAAoGBAKCE8JYKhsbyd
MPbiO7BJVq1pbuJWJHFxOR7L8Hv3ZVkSG4eNC8DdwAmDHYu/wadfw0ihKFm2gKDcL
Hp5yz5UQ8PZ8FyDYvgkrvGV0ak4nc40QDJWws621dm01e/INlGKOIStAAsxOityCLv0zm
5Vf3+My/YaBvZcB5mGUsPbx8fAgEAAoGAAy0+WanRqwRHXUzt89OsupPXuNNqBlCEq
gTqGAt4Nimq6Ur9u2R1KXKXUotxjp71Ubw6JbuUWvJg+5Rmd9RjT0HOUEQF3rvzEepKt
araPhV5ejEIrB+nJWNfGye4yzLdfEXJBGUQzrG+wNe13izfRNXI4dN/6Q5npzqaqv0E1CkC
AQACAQACAQACAQACAQA="
        public-
key="MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCghPCWCobG8nTD24juwSV
ataW7iViRxcTkey/B792VZEhuHjQvA3cAJgx2Lv8GnX8NIoShZtoCg3Cx6ecs+VEPD2fBcg
2L4JK7xldGpOJ3ONEAyVsLOttXZtNXvyDZRijiErQALMTorcgi79M5uVX9/jMv2Ggb2XAe
ZhlLD28fHwIDAQAB"
        aes-key-length=16 //AES key 长度
    }

    #网络配置
    net {
        connect-server-port=3000 //长链接服务对外端口, 公网端口
        admin-server-port=3002 //控制台服务端口, 内部端口
        gateway-server-port=3001 //网关服务端口, 内部端口
        gateway-client-port=4000 //UDP 客户端端口
        gateway-server-net=udp //网关服务使用的网络类型tcp/udp
        gateway-server-multicast="239.239.239.88" //239.0.0.0～239.255.255.255为本地管
理组播地址，仅在特定的本地范围内有效
        gateway-client-multicast="239.239.239.99" //239.0.0.0～239.255.255.255为本地管
理组播地址，仅在特定的本地范围内有效
        public-host-mapping { //本机局域网IP和公网IP的映射关系
            //"10.0.10.156":"111.1.32.137"
            //"10.0.10.166":"111.1.33.138"
        }
        traffic-shaping { //流量整形配置
            gateway-client {
                enabled:true
                check-interval:100ms
                write-global-limit:30k
                read-global-limit:0
                write-channel-limit:3k
                read-channel-limit:0
            }

            gateway-server {
                enabled:true
```

```
            check-interval:100ms
            write-global-limit:0
            read-global-limit:30k
            write-channel-limit:0
            read-channel-limit:3k
        }

        connect-server {
            enabled:false
            check-interval:100ms
            write-global-limit:0
            read-global-limit:100k
            write-channel-limit:3k
            read-channel-limit:3k
        }
    }
}

#Zookeeper配置
zk {
    server-address="127.0.0.1:2181"
    namespace=mpush
    digest=mpush //zkCli.sh acl 命令 addauth digest mpush
    local-cache-path=/
    retry {
        #initial amount of time to wait between retries
        baseSleepTimeMs=3s
        #max number of times to retry
        maxRetries=3
        #max time in ms to sleep on each retry
        maxSleepMs=5s
    }
    connectionTimeoutMs=5s
    sessionTimeoutMs=5s
}

#Redis集群配置
redis {
    write-to-zk=false
    password=""//your password
    cluster-model=single//single,cluster
    nodes:[]//["127.0.0.1:6379"]格式ip:port:password,密码可以不设置ip:port
```

```
        config {
            maxTotal:8,
            maxIdle:4,
            minIdle:1,
            lifo:true,
            fairness:false,
            maxWaitMillis:5000,
            minEvictableIdleTimeMillis:300000,
            softMinEvictableIdleTimeMillis:1800000,
            numTestsPerEvictionRun:3,
            testOnCreate:false,
            testOnBorrow:false,
            testOnReturn:false,
            testWhileIdle:false,
            timeBetweenEvictionRunsMillis:60000,
            blockWhenExhausted:true,
            jmxEnabled:false,
            jmxNamePrefix:pool,
            jmxNameBase:pool
        }
    }

    #HTTP代理配置
    http {
        proxy-enabled=false //启用Http代理
        max-conn-per-host=5 //每个域名的最大链接数, 建议web服务nginx超时时间设长一点,
以便保持长链接
        default-read-timeout=10s //请求超时时间
        max-content-length=5m //response body 最大大小
        dns-mapping { //域名映射外网地址转内部IP, 域名部分不包含端口号
            //"mpush.com":["127.0.0.1:8080", "127.0.0.1:8081"]
        }
    }

    #线程池配置
    thread {
        pool {
            boss { //netty server boss
                min:1 //boss 只需要一个线程即可
                max:1
                queue-size:0
            }
```

```
work { //netty server boss
    min:0 //0表示线程数根据cpu核数动态调整(2*cpu)
    max:0
    queue-size:0
}

event-bus {
    min:4
    max:4
    queue-size:10000 //大量的online，offline，
}

http-proxy {
    min:0 //0表示线程数根据cpu核数动态调整(2*cpu)
    max:0
    queue-size:0
}

biz { //其他业务
    min:1
    max:4
    queue-size:16
}

mq { //用户上下线消息, 踢人等
    min:2
    max:4
    queue-size:10000
}

push-callback { //消息推送
    min:2
    max:2
    queue-size:0
}

push-center { //消息推送
    min:4
    max:4
    queue-size:0
}
```

```
    }
  }

  #推送消息流控
  push {
    flow-control { //qps = limit/(duration)
        global:{ //针对非广播推送的流控，全局有效
            limit:5000 //qps = 5000
            max:0 //UN limit
            duration:1s //1s
        }

        broadcast:{ //针对广播消息的流控，单次任务有效
            limit:3000 //qps = 3000
            max:100000 //10w
            duration:1s //1s
        }
    }
  }

  #系统监控配置
  monitor {
     dump-dir=${mp.home}/tmp
     dump-stack=false //是否定时dump堆栈
     dump-period=1m  //多久监控一次
     print-log=true //是否打印监控日志
     profile-enabled=true //开启性能监控
     profile-slowly-duration=10ms //耗时超过10ms打印日志
  }

  #SPI扩展配置
  spi {
     thread-pool-factory:"com.mpush.tools.thread.pool.DefaultThreadPoolFactory"
     dns-mapping-
manager:"com.mpush.common.net.HttpProxyDnsMappingManager"
  }
}
```

功能模块

模块依赖关系图

图片地址：https://static.oschina.net/uploads/img/201610/29224431_zx6f.jpg

模块说明

mpush-client：服务端SDK,主要提供发送Push的接口给其他业务使用，比如MPNS

mpush-boot：是服务端启动入口模块，主要控制server启动、停止流程

mpush-api：定义了mpush相关核心接口及协议，还包括对外暴露的SPI接口

mpush-netty：主要提供netty相关的一些基础类，像NettyServer,NettyClient

mpush-tools：mpush用到的一些工具类，包括线程池，加密，配置文件解析等等

mpush-zk：zookeeper的client, 包括path的定义，节点定义，数据监听等

mpush-cache：redis缓存模块，支持单机模式和3.x集群模式，包括用户路由，上下线消息等

mpush-common：定义了mpush-client模块和mpush-core模块都会用到的类，主要是消息、路由等

mpush-core：sever核心模块，包括接入服务，网关服务，路由中心，推送中心等等

mpush-monitor：服务监控模块主要监控JVM，线程池，JMX，服务状态统计，性能统计等

SPI定制化

什么是SPI

建议直接百度 java spi, 官方介绍: http://docs.oracle.com/javase/tutorial/sound/SPI-intro.html

mpush 使用SPI的目的

肯定是为了扩展性，在不想修改源码的情况下，去替换系统原有的实现，代价最小也最灵活。

mpush目前支持的SPI列表

所有受支持的SPI都在mpuhs-api模块的com.mpush.api.spi包下面。

- ServerEventListenerFactory 监听Server 产生的相关事件
- BindValidatorFactory 绑定用户身份是校验用户身份是否合法
- ClientClassifierFactory 同一个用户多设备登录互踢策略，默认Android和ios会互踢
- PushHandlerFactory 接收客户端发送到服务端的上行推送消息
- ExecutorFactory 整个系统的线程池工厂，可以按自己的策略创建线程池
- CacheManagerFactory 系统使用的缓存实现，默认是redis，支持自定义
- MQClientFactory 系统使用的MQ实现，默认是redis的pub/sub，支持自定义
- ServiceRegistryFactory 服务注册组件，默认实现是zookeeper，支持自定义
- ServiceDiscoveryFactory 服务发现组件，默认实现是zookeeper，支持自定义
- PushListenerFactory 消息来源自定义，默认使用Gateway模块，支持自定义，比如使用MQ

开发流程

- 新建一个maven工程，并加入mpush-api模块的依赖
- 实现要定制的接口和其对应的Factory
- 在resources目录下创建META-INF.services目录
- 创建名字为com.mpush.api.spi.xxx.XXXFactory的文件，即对应的接口名
- 文件的内容为实现类的全名(packageName+className)
- 通过mvn打成jar包，并把其放到mpush/lib目录
- 重启mpush server 就会优先加载用户提供的实现来覆盖原有的默认实现

第四章 深度进阶

密钥交换

交换方案

1. 算法上同时使用了非对称加密算法(RSA)和对称加密算法(AES)。

2. 在客户端预埋好由服务端生成好的公钥。

3. 客户端生成随机数R1，并通过RSA公钥加密后发送给服务端。

4. 服务端用RSA私钥解密客户端发送的数据取得R1,同时生成随机数R2,并以R1为Key使用AES加密R2然后把加密后的数据发送到客户端。

5. 客户端以R1为Key使用AES解密服务发送的数据取得R2

6. 此时客户端和服务的同时都拥有随机数R1、R2，然后使用相同的混淆算法生成会话密钥(sessionKey), 之后传输的数据都以此值作为AES加密的密钥。

交换流程

图片地址：https://static.oschina.net/uploads/img/201610/29235425_gXHU.png

握手及快速重连

时序图

图片地址：https://static.oschina.net/uploads/img/201611/01184038_oHrJ.png

说明

1. tcp连接建立后，第一个消息就是握手或快速重连消息。

2. Handshake的目的是为了生成会话密钥，同时会把生成好的密钥存储到redis，并把key返回给客户端，为快速重连使用

3. FastConnect是基于Handshake生成的sessionId来配合使用的，目的是为了减少RSA加密的使用次数，特别是网络较差的情况，毕竟RSA加密想对还是比较耗时的，客户端只需把sessionId传给服务端，其就能从redis中取出上次会话信息，恢复到上次握手成功之后的会话状态，这个过程不需要任何加密和密钥交换，相对会比较快速。

给连接绑定用户

时序图

图片地址：https://static.oschina.net/uploads/img/201611/01185848_Uf42.png

说明

1. 握手成功后表示安全通道已经建立，但同时还要给连接设置用户甚至标签，只有这样业务才能更好的去识别用户（没有用户的业务是另外一回事）。

2. 设置用户非常简单，只需把其存储到session即可，但因为要支持集群的，就必须把用户的位置信息（或者叫路由信息）发布出去，让集群里的其他机器能够通过useId来查找用户的位置（在哪台机器），因为客户端的TCP连接在哪台机器，那么和这个客户端的所有数据传输都必须经过这台机器的这个连接！(很多同学会问为什么不把connection存储到redis)。

3. 路由中心有两部分组成：本地路由和远程路由，本地路由数据结构为userId->connection的map，数据存储在本机内存；远程路由数据结构为userId->connServer机器ip，数据存储在redis;所以要给一个用户发信息必须先查远程路由，找到用户在哪台机器，然后把消息发给这台机器，让其去本地路由查找connection并通过查找到的TCP连接把消息发给用户。

4. MessageCenter之前使用的redis提供的pub/sub实现的，也可以自己搭建MQ来实现，最新版踢人已经不再使用pub/sub，而是直接使用udp网关实现。

5. 踢人：之所以会有踢人的情况是根据业务需要来的，有些业务系统是不允许同一个用户在多个设备同时在线的，或者只允许不同类型的终端同时在线，比如QQ,手机和PC可以同时在线，但同一个帐号在两台PC登录时其中一个肯定会被踢下线，mpush的踢人要表达的也是同一个意思。

6. 顺便提一下关于同时在线的策略，或者说端的类型的定义mpush已经支持SPI定制化。

消息推送流程

流程图

图片地址：https://static.oschina.net/uploads/img/201611/01193442_ABeq.png

说明

1. PushSender是消息推送的入口，它的实现在mpush-client模块属于服务端SDK，主要包含有GatewayClient, RemoteRouterManager; RemoteRouterManager用于定位用户在哪台机器，有没有在线等，而GatewayClient用于把要发送的的业务消息发给用户TCP连接所在的机器。

2. GatewayServer负责接收GatewayClient发送过来的消息，然后到LocalRouterManager查找用户的Connection，然后把消息交由其下发。

3. ConnectionServer 负责维持所有连接到当前机器的客户端连接，所以消息最终还是尤其下发（图比较简单，但能表达核心流程）。

HTTP代理

使用场景

- 问题

> 前面有提到过http代理这个东西，但很多人不知道这个东西该怎么用，或者说有什么用？以及在什么场景下使用？

- 移动APP通信场景分析

> 从使用的链接情况来看，一般可以分为两大类：TCP长链接，HTTP短链接；长链接用于消息推送或IM等场景，HTTP用于业务数据的查询或修改。虽然不是所有的APP都需要IM功能，但大多应用都需要消息推送功能。为了推送消息，APP必须维持一根长链接，但大部分时间除了心跳这根链接上是没多少消息在传输的，特别是非IM类的APP，因为这类应用并没大量的消息要不停的推送，维持长链接只是为了消息的及时到达，这势必造成了很大的资源浪费！

- 解决方案

> 针对上述情况MPUSH提供了Http代理方案，目的一是充分利用push通道，而是提高数据传输效率节省电量，节省流量，提供比http更高的安全性。

- 实现原理

> MPushClient 提供了一个叫sendHttp的方法，该方法用于把客户端原本要通过HTTP方式发送的请求，全部通过PUSH通道转发，实现整个链路的长链接化；通过这种方式应用大大减少Http短链接频繁的创建，不仅仅节省电量，经过测试证明请求时间比原来至少缩短一倍，而且MPush提供的还有数据压缩功能，对于比较大的数据还能大大节省流量(压缩率4-10倍)，更重要的是所有通过代理的数据都是加密后传输的，大大提高了安全性！

使用方式

- 服务端

1. 修改mpush.conf增加mp.http.proxy-enabled=true启用http代理

2. 修改mpush.conf增加dns-mapping配置，示例如下

```
mp.http.dns-mapping={//域名映射外网地址转内部IP
  "api.jituancaiyun.com":["10.0.10.1:8080", "10.0.10.2:8080"]
}
```

> 说明：因为`mpush server`要做http代理转发，而客户端传过来的一般是域名比如`http://api.jituancaiyun.com/get/userInfo.json`为了不到公网上再绕一圈建议把mpush

> server 和业务服务(api.jituancaiyun.com)部署到同一个局域网，并增域名api.jituancaiyun.com到提供该服务的集群机器内网ip之间的一个映射，这样`mpush server`就可以通过局域网把请求转发到具体到业务服务，效率更高！

• 客户端

1. 设置ClientConfig.setEnableHttpProxy(true)来启用客户端代理。
2. 通过Client.sendHttp(HttpRequest request)方法来发送请求。

> > AndroidSDK通过`com.mpush.android.MPush#sendHttpProxy(HttpRequest request)`来发送比较合适。

流程分析

图片地址：https://static.oschina.net/uploads/img/201611/01195523_qymD.png

说明

1. Client代表App业务比如查询用户信息的接口
2. MPushApiProxy是一个工具类用于负责处理当前请求是使用普通的HTTP还是使用MPush长链接通道，这个类在SDK中说不存在的，是我们公司内部的业务，实现起来也很简单，建议Android工程中增加这么一个角色，而不是到处直接去依赖Mpush的代码，方便以后解耦。
3. MPushClient 这个SDK已经提供，用于把Http协议打包成mpush协议。
4. HttpProxyHandler包括后面的几个组件都是服务端业务组件。用于接收客户端传过来的请求并反解为Http协议，然后通过DNSMapping找到域名对应的局域网IP，再通过内置的HttpClient，把请求转发给业务WEB服务，并把业务服务的返回值(HttpResponse)打包成MPush协议发送到客户端。
5. DNSMapping负责通过域名解析成局域网IP，并具有负载均衡以及简单的健康检查功能(针对所配置的WEB服务)
6. HttpClient目前使用的是用Netty实现的全异步的一个HttpClient，负责通过http的方式请求业务服务。
7. Nginx是业务服务，也可以是Tomcat，特别需要建议的是链接超时时间配置长一些。

补充

为什么要这样实现？因为这样做对原有的业务系统侵入特别低，如果MPushApiProxy这个组件设计的好，对于最两边的业务组件/服务(Client,Nginx)，对请求方式应该是无感知的，这个角色是无法区分到底请求是通过普通的Http方式发送出去的还是通过长链接代理的方式发送的！！！

另附上通过Http Proxy 实现双向通信交互图

图片地址：https://static.oschina.net/uploads/img/201611/01195617_S2ll.png

消息分发

消息序列化

消息加密与压缩

线程池配置与定制

上行PUSH

接口产生背景

作为一个纯粹推送系统，最初是没有发送上行消息接口的只有下行的消息，毕竟MPush不是IM系统。

后来在很多同学的要求下，就增加了此接口用于cleint上报消息。

接口的使用

1. 此接口和服务端下行Push保持相同的协议，可以互发消息，默认服务端没有对消息做任何业务上的处理，收到后直接丢弃。

2. 同时MPush提供了SPI的方式来接管客户端发过来的消息，具体请参考第三章 系统架构 - SPI定制化(http://mpush.mydoc.io?v=24639&t=134851)。

3. 如果没有特殊需求客户端上行消息建议使用Http Proxy模式。

广播推送与流控

什么是广播推送？

按推送用户范围来划分，MPush目前支持三种方式的推送：

1. 单用户推送，推送消息给指定的某个用户。

2. 批量推送，业务自己圈定一批用户，推送同一条消息给圈定的用户。

3. 全网推送，推送消息给所有的在线用户。

这里所说的广播推送指的就是第三种用户范围的推送。

为什么广播推送要控制流量？

因为要推送消息给全网在线用户，用户量可能非常大，为了防止瞬时流量过大，所有加了入了防过载保护：流量控制。

流量控制的使用

1. 单任务流量控制

2. 全局流量控制

广播推送条件过滤

为什么要对用户进行过滤？

因为广播是针对所有在线用户，为了更精准的推送，必须对目标用户进行筛选，才能满足个性化的业务需求。

目前支持的筛选纬度

1. tag：业务自己打的标签
2. userId：用户登录ID
3. clientVersion：客户端版本
4. osName：客户端系统平台
5. osVersion：客户端系统版本

目前支持的表达式

目前只支持jvm内置的Nashorn脚本引擎，语法为javascript标准语法。

使用用例

具体请参照com.mpush.api.push.PushContext.java。

- 灰度20%的在线用户：userId % 100 < 20
- 包含test标签的用户：tags!=null && tags.indexOf("test")!=-1
- 客户端版本号大于2.0的安卓用户：clientVersion.indexOf("android")!=-1 && clientVersion.replace(/[^d]/g,"") > 20``

消息ACK

服务端启动流程分析

日志模块性能优化

JVM监控与性能分析

第五章 关联服务

Alloc服务

alloc 的作用

> * alloc 是针对client提供的一个轻量级的负载均衡服务
> * 每次客户端在链接MPUSH server之前都要调用下该服务
> * 以获取可用的MPUSH server列表,然后按顺序去尝试建立TCP链接,直到链接建立成功

对外提供的接口定义

> 接口类型　　：HTTP
> Method　　：GET
> 参数　　：无
> 返回值格式　：ip:port,ip:port
> content-type : text/plain;charset=utf-8

服务部署

1. 下载alloc最新包 https://github.com/mpusher/alloc/releases(https://github.com/mpusher/alloc/releases)
2. 解压下载的tar包tar -zvxf alloc-release-x.x.x.tar.gz
3. 修改 conf 目录下的 vi mpush.conf文件, 修改方式参照mpush server 部署
4. 给bin目录下的脚本增加执行权限chmod u+x *.sh
5. 执行./mp.sh start 启动服务, 查看帮助直接执行./mp.sh
6. cd logs目录，cat mpush.out查看服务是否启动成功

客户端建立连接流程

图片地址：https://static.oschina.net/uploads/img/201611/01201926_2SMA.png

实现讲解

1. 服务部署可以集成Tomcat或自己实现一个HttpServer比如基于Netty实现
2. mpush server 集群列表可以从Zookeeper查询，目前提供的有ZK查询客户端

3. 如果要实现负载均衡可以考虑使用以下几种方式实现：

> 随机，每次从mpush server列表随机选取一个地址返回给客户端
>
> 轮播，每次把mpush server列表依次返回给客户端
>
> 按链接数量排序，链接数少的排最前面

Alloc服务存在的意义

刚开始看MPUSH的童鞋可能会有疑问，这玩意有什么用，为什么不直接连mpush server？
如果直连你可能遇到一些问题，比如你的mpush server 可能不止一台，你怎么去选择连哪一台？
其中某台服务挂了怎么办？要更换机器又怎么办？这时你必然希望有一台前置服务来对整个mpush集群进行统一调度。

MPNS

个性化推送平台

第六章 SDK接入

服务端SDK

使用方式

1. 添加maven依赖

```
<dependency>
  <groupId>com.github.mpusher</groupId>
  <artifactId>mpush-client</artifactId>
  <version>x.y.z</version>
</dependency>
```

1. 在工程resources目录增加配置文件application.conf，并配置zookeeper

```
mp.zk.server-address="127.0.0.1:2181"
mp.redis.nodes=["127.0.0.1:6379"]
```

1. 使用com.mpush.api.push.PushSende.java进行推送，使用其create方法创建服务，start方法启动服务，stop方法停止服务。推送接口定义如下：

```
public interface PushSender extends Service {

 /**
```

```
     * 创建PushSender实例
     *
     * @return PushSender
     */
     static PushSender create() {
        return SpiLoader.load(PusherFactory.class).get();
     }

    /**
     * 推送push消息
     *
     * @param context 推送参数
     * @return FutureTask 可用于同步调用
     */
     FutureTask<PushResult> send(PushContext context);

   }
```

推送流程

流程图

图片地址：https://static.oschina.net/uploads/img/201611/01211537_dfNK.png

流程分析

1. PushSender启动后首先从ZK里获取可用的GatewayServer列表，然后创建相应的Client分别连接到对应的GatewayServer

2. 当调用send方法去推送时，PushSender首先会通过RemoteRouterManager查询要推送的用户当前所登录的机器IP，然后通过IP选择GatewayServer并通过第1中对应的Client把消息发送到该机器，因为该机器拥有用户的链接。

3. GatewayServer接收到Client发送过来的消息后，首先通过查询本地路由LocalRouterManager找到用户连接Connection，该链接是连接到ConnectionServer的。

4. 如果连接存在，ConnectionServer会通过此连接把消息下发到客户端。

5. 如果推送成功，GatewayServer会发送消息推送成功的消息给PushSender所持有的Client

6. PushSender收到推送成功消息后，会通过Callback#onSuccess回调调用方，整个推送流程结束。

7. 如果中间有任何失败则回调Callback#onFailure。

8. 如果用户不在线则回调Callback#onOffline。

9. 如果在一定时间内PushSender没有收到GatewayServer响应的消息则推送超时，回调Callback#onTimeout通知调用方。

源码解读

1. PushSender的实现类为com.mpush.client.push.PushClient.java

2. PushClient使用的是ConnectionRouterManager该类继承自RemoteRouterManager增加了本地缓存可在消息频繁时减轻Redis压力，但会存在一定情况的误判。

3. com.mpush.client.push.PushRequestBus.java用于维持异步推送任务，线程的调整可通过配置设置，任务的拒绝策略为在调用线程执行Callback。具体见DefaultThreadPoolFactory.java。线程池默认配置如下：

```
    mp.thread.pool.push-client=2
```

1. GatewayClient会根据GatewayServer的运行状态自行调整，如果有GatewayServer宕机对应的Client会及时销毁，如果有新的机器进来，对应Client也会自动创建。具体参见GatewayConnectionFactory.java。

MPushClient(Java)

MPushClient 架构图

图片地址：https://static.oschina.net/uploads/img/201611/01212400_lyC5.png

说明

1. MpushClient主要是为Android客户端设计，作为Android底层通信组件

2. MpushClient实现了所有Mpush的协议，也包加密，压缩等特性

3. MpushClient对SDK只暴露了3个对象：Client、ClientConfig、ClientListener

4. ClientConfig提供了所有支持对配置设置项，所有的配置都必须使用此对象设置

5. Client主要暴露一些往服务端上行的方法，常用的有：

```
public interface MPushProtocol {

    boolean healthCheck();//健康检查，读写超时检查，发送心跳

    void fastConnect();//快速重连

    void handshake();//握手交互密钥

    void bindUser(String userId);//绑定用户到链接

    void unbindUser();//解绑

    Future<HttpResponse> sendHttp(HttpRequest request);//发送HTTP代理亲求
}
```

1. 所有服务端下行的消息都直接转交给ClientListener来处理:

```
public interface ClientListener {

    void onConnected(Client client);

    void onDisConnected(Client client);

    void onHandshakeOk(Client client, int heartbeat);

    void onReceivePush(Client client, byte[] content);

    void onKickUser(String deviceId, String userId);
}
```

1. 定时发送心跳部分的定时功能，MpushClient没有去实现，因为Android有比Timer更好的实现AlarmManager, 所以定时功能由SDK收到onHandshakeOk事件时去触发。

2. 内部组件主要有:Connection、PacketReader、PacketWriter、PacketDispatcher、MessageHandler、ExecutorManager、AllotClient 等

3. Connection负责TCP链接创建、维护、关闭、重连及校验读写超时时间等

4. AllotClient负责从AllotServer获取最新的MpushServer IP地址列表

5. PacketReader 负责数据包的解码及沾包、半包的处理，独占一个线程负责消息的读取。

6. PacketWriter负责数据发送数据内部维护一个只有一个线程的线程池，如果发送失败，会尝试重发直到超时。

7. PacketReader、PacketWriter内部还分别维护了一个ByteBuf使用的是堆外内存(allocateDirect)该buffer只分配一次后续一直重复使用，避免反复的分配内存和垃圾回收，这点对于客户端还是非常有意义的。

8. PacketDispatcher负责根据Packet的Command把包分发到相应的MessageHander。

9. MessageHandler主要负责把Packet反序列化为业务Message包括解码和解压，并处理相应的业务逻辑，比如触发下一步的请求，或通知Listener。

10. ExecutorManager负责整个客户端的线程、线程池的分配和销毁。

Android SDK

系统架构图

图片地址：https://static.oschina.net/uploads/img/201611/01212307_WHqh.png

说明

1. 整个图非常简单清晰的分为Server、SDK、BIZ三部分。

2. MpushClient负责和server通信，屏蔽网络，协议，断线重连等所有和长链接相关的东西。

3. MpushService是常驻服务，持有MpushClient,并把自身作为MpushClient的ClientListener，监听MpushClient的变化事件。

4. MpushReceiver主要负责监听网络变化和AlarmManager，以便暂停和恢复推送服务以及健康检查。

5. 线1表示上行的请求，比如握手，心跳，绑定用户，业务HTTP代理请求等。

6. 线2表示下行响应或推送，比如握手成功，心跳响应，HTTP代理响应等。

7. 线3表示Client下发的事件，主要有：链接建立/断开，握手成功，收到PUSH，设备被踢下线等事件，其中PUSH和KICK_USER事件会广播出去，由业务(MyReceiver)接收；其他事件会通知给MpushReceiver以便其能更好的控制MpushClient的起停，而MpushService就比较轻量基本没有什么业务逻辑，只负责维持后台服务。

8. 线4表示由MpushService广播出去的PUSH消息，由于采用的是广播的形式，所有也可以分进程。

9. 线5表示消息有MyReceiver过滤处理后，转交给业务去显示或存DB等。

10. 线6表示业务可以直接调用MpushClient提供的接口发送消息，目前支持的有绑定usreId，发送Http请求等。

11. 线7表示一些不需要业务处理的消息都交由MpuhReceiver处理，比如握手成功后启动AlarmManager，当链接断开后取消AlarmManager。

12. 线8表示MpushReceiver接收到AlarmManager的提醒后去调用MpshClient的healthCheck方法发送心跳。

13. 线9表示MpushReceiver接收到网络变化后暂停或恢复MpushClient，这样做主要是为了省电，因为在网络断开后，MpushClient会去尝试重连而这时候去重连是没有意义的，因为没有网络。

MPushClient(Swift)

架构同 MPushClient(java)

MPushClient(Obj-C)

第七张 常见问题

常见问题

1.修改配置文件为什么不生效？

如果是源码测试，请修改mpush-test模块resources/application.conf文件

如果是独立部署，请修改conf/mpush.conf文件

请不要修改reference.conf文件，此文件只作为参考，列出系统支持哪些配置项及其默认值。

2.Alloc返回的外网地址不对

请在修改mpush server 里的mpush.conf和application.conf添加mp.net.public-host-mapping配置项，

格式查考reference.conf对应的默认配置，alloc 里的配置不需要修改

3.怎么开启http proxy?

请在mpush.conf和application.conf添加mp.http.proxy-enabled=true

4. redis.clients.jedis.exceptions.JedisMovedDataException: MOVED 3456

请设置redis为集群模式mp.redis.cluster-model=cluster

5. 广播推送收不到推送的消息，指定用户发送能收到

请修改网关模式为mp.net.gateway-server-net=tcp 因为udp网关模式下，广播推送使用的是组播，如果网络配置不正确，比如跨网段等是不能收到消息的。

6. 如何根据源码构建安装包？

构建命令为 mvn clean package -Pzip,pub 安装包输出路径为 ./mpush-boot/target/mpush-release-x.y.z.tar.gz。alloc构建命令同上。

7. Alloc源码启动报错

alloc 启动入口com.shinemo.mpush.alloc.AllocServerMainTest.java

系统压测