Atitit it 简化工作 轻量级mq

目录

[1.1. 要用Redis实现轻量级MQ？ 1](#_Toc3218)

[1.2. 开启监听 "loginEvtSubject2" 1](#_Toc15293)

[1.3. 发送消息 2](#_Toc4539)

[2. Ref 2](#_Toc9845)

## 要用Redis实现轻量级MQ？

在业务的实现过程中，就算没有大量的流量，解耦和异步化几乎也是处处可用，此时MQ就显得尤为重要。但与此同时MQ也是一个蛮重的组件，例如我们如果用RabbitMQ就必须为它搭建一个服务器，同时如果要考虑可用性，就要为服务端建立一个集群，而且在生产如果有问题也需要查找功能。在中小型业务的开发过程中，可能业务的其他整个实现都没这个重。过重的组件服务会成倍增加工作量。

所幸的是，Redis提供的list数据结构非常适合做消息队列。

但是如何实现即时消费？如何实现ack机制？这些是实现的关键所在。

各种轻量级的组件框架

Redis 轻量级mq

Sql +json字段，方便join查询。。

Mongodb也可以join，，es嘛就多表联查不方便了。。

jedis 发布代码比较简单，只需要调用 Jedis 类的 publish 方法。

*// 生产环境千万不要这么使用哦，推荐使用 JedisPool 线程池的方式*  
Jedis jedis = new Jedis("localhost", 6379);  
jedis.auth("xxxxx");  
jedis.publish("pay\_result", "hello world");

订阅的代码就相对复杂了，我们需要继承 JedisPubSub实现里面的相关方法，一旦有其他客户端往订阅的频道上发送消息，将会调用 JedisPubSub 相应的方法。

## 开启监听 "loginEvtSubject2"

**private** **static** **void** redisListern() {

Jedis jedis = **new** Jedis("localhost");

System.***out***.println("set lister start");

//is block thread..so only use by async thread

**new** Thread(**new** Runnable() {

@Override

**public** **void** run() {

jedis.subscribe(**new** JedisPubSub() {

@Override

/\*\*

\* JedisPubSub类是一个没有抽象方法的抽象类,里面方法都是一些空实现

\* 所以可以选择需要的方法覆盖,这儿使用的是SUBSCRIBE指令，所以覆盖了onMessage

\* 如果使用PSUBSCRIBE指令，则覆盖onPMessage方法 当然也可以选择BinaryJedisPubSub,同样是抽象类，但方法参数为byte[]

\*\*/

**public** **void** onMessage(String channel, String message) {

System.***out***.println(

Thread.*currentThread*().getName() + "-接收到消息:channel=" + channel + ",message=" + message);

}

}, "loginEvtSubject2");// 第一个参数是处理接收消息，第二个参数是订阅的消息频道

}

}).start();

System.***out***.println("set lister ok");

}

## 发送消息

// 连接本地的 Redis 服务

Jedis jedis = **new** Jedis("localhost");

// 如果 Redis 服务设置来密码，需要下面这行，没有就不需要

// jedis.auth("123456");

System.***out***.println("连接成功");

// 查看服务是否运行

System.***out***.println("服务正在运行: " + jedis.ping());

Long publish = jedis.publish("loginEvtSubject2", svr1.toString());// 返回订阅者数量

System.***out***.println(" puiblish:" + publish);

# Ref

redis实现消息队列&发布/订阅模式使用 - QiaoZhi - 博客园