机构图标

Java+Python方向中软信息集成移动平台

WebSocket长轮询及Redis技术预研报告

|  |  |  |
| --- | --- | --- |
| 文件状态：  [√] 草稿  [ ] 正式发布  [ ] 正在修改 | 文件标识： | ChinaSoftware-Project-JP-REPORT |
| 当前版本： | 1.1 |
| 作 者： | 张雲淞 |
| 完成日期： | 2018-7-6 |

机构公开信息

版 本 历 史

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 版本/状态 | 作者 | 参与者 | 起止日期 | 备注 |
| 1.0 | 张雲淞 | 张雲淞 | 2018/7/5-2018/7/6 | 初版 |
| 1.1 | 张雲淞 | 张雲淞 | 2018/7/6- | 增加原理说明字段 |
|  |  |  |  |  |

目 录

[0. 文档介绍 4](#_Toc518647224)

[0.1 文档目的 4](#_Toc518647225)

[0.2 读者对象 4](#_Toc518647226)

[0.3 参考文档 4](#_Toc518647227)

[1. 背景介绍 5](#_Toc518647228)

[2. 技术预研目标 5](#_Toc518647229)

[3. 技术预研取得的工作成果 5](#_Toc518647230)

[4. 技术的研究报告 5](#_Toc518647231)

# 0. 文档介绍

## 0.1 文档目的

说明WebSocket长轮询技术，以及Redis队列通信技术。

## 0.2 读者对象

1.项目经理

2.所有开发人员

## 0.3 参考文档

*[1]* [*http://www.runoob.com/redis/redis-tutorial.html*](http://www.runoob.com/redis/redis-tutorial.html)*，Redis 命令简介*

*[2]* [*https://blog.csdn.net/cxfly957/article/details/79297650/*](https://blog.csdn.net/cxfly957/article/details/79297650/)*，JavaWeb SpringMVC整合WebSocket(注解实现)*

*[3]* [*http://www.ruanyifeng.com/blog/2017/05/websocket.html*](http://www.ruanyifeng.com/blog/2017/05/websocket.html)*，为什么需要 WebSocket？*

*[4]* [*https://blog.csdn.net/yejingtao703/article/details/78484151*](https://blog.csdn.net/yejingtao703/article/details/78484151)*，三张图秒懂Redis集群设计原理*

# 1. 背景介绍

需求中提出了各个用户收发消息的功能，项目经理推荐使用WebSocket框架来实现前端对后端的长轮询，以及使用Redis队列储存消息来快速读取未读信息。于是我就针对这两个技术进行研究，制作了一个融合这两种技术的demo。

# 2. 技术预研目标

主要内容与目标：使用WebSocket的onMessage方法来实现前端对后端的长轮询。用户发的消息会通过Redis的lpush方法放入特定频道的list中，接收信息的用户能够通过这个特定的频道来读取所有的信息，也可以通过Redis的lrem的方法来从list中取出数据来实现已读的需求。

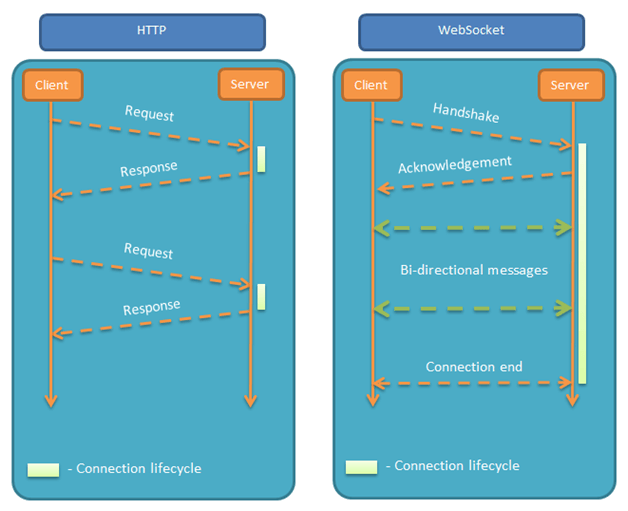
# 3. 技术预研取得的工作成果

成果：一个简单的demo，可以实现多个用户的消息收发和阅读。

# 4. 技术的研究报告

4.1 WebSocket技术

4.1.1 原理说明



WebSocket 协议在2008年诞生，2011年成为国际标准。所有浏览器都已经支持了。

它的最大特点就是，服务器可以主动向客户端推送信息，客户端也可以主动向服务器发送信息，是真正的双向平等对话，属于服务器推送技术的一种。

其他特点包括：

（1）建立在 TCP 协议之上，服务器端的实现比较容易。

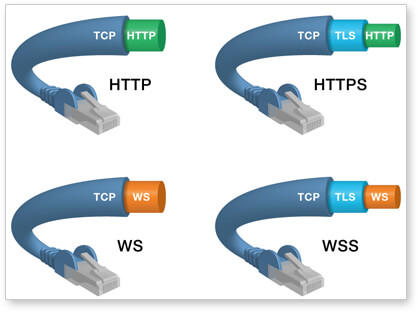
（2）与 HTTP 协议有着良好的兼容性。默认端口也是80和443，并且握手阶段采用 HTTP 协议，因此握手时不容易屏蔽，能通过各种 HTTP 代理服务器。

（3）数据格式比较轻量，性能开销小，通信高效。

（4）可以发送文本，也可以发送二进制数据。

（5）没有同源限制，客户端可以与任意服务器通信。

（6）协议标识符是ws（如果加密，则为wss），服务器网址就是 URL。例如：ws://example.com:80/some/path



4.1.2 WebSocket需要的jar包

1) org.springframework.spring-websocket

2) org.springframework.spring-messaging

注意：这两个包的版本最好与项目的spring相关的包的版本相同。

4.1.3 WebSocket服务器端实现

详见com.etc.websocket.WebSocketServer类，以下部分对重要部分进行说明。

4.1.3.1 注解说明

|  |  |  |  |
| --- | --- | --- | --- |
| 注解名称 | 参数 | 位置 | 说明 |
| @ServerEndPoint | value=“/websocket/{a}/{b}/{c}” | 类前 | 注解的值将被用于监听用户连接的终端访问URL地址，客户端可以通过这个URL来连接到WebSocket服务器端，其中{a},{c},{c}中的a、b、c是客户端开始连接服务器端时就向服务器端传递的参数值，参数在URL的位置应与前端相同，可以在@OnOpen的方法中读取。 |
| @OnOpen | 无 | 方法前 | 连接建立成功时调用的方法 |
| @PathParam | “URL传的参数” | @OnOpen方法中的参数位置 | 用来取URL中的参数 |
| @OnClose | 无 | 方法前 | 连接关闭时调用的方法 |
| @OnMessage | 无 | 方法前 | 收到客户端消息后调用的方法 |
| @OnError | 无 | 方法前 | 发生错误时调用 |

4.1.3.2 服务器端的编写思路

1）每开启一个WebSocket连接，就会创建一个新的WebSocketServer类，因此可以在每个服务器中创建一个Session变量来识别不同的连接。

2）可以通过创建static类型的变量，来保存或读取不同的连接。

3）变量CopyOnWriteArraySet<WebSocketServer> webSocketSet用来存放每个客户端对应的WebSocketServer对象，可以通过存多个对象来实现与多个客户端的通信。

3）变量ConcurrentHashMap<Session,Object> webSocketMap用来实现服务端与指定客户端的通信，其中Session可以作为不同用户的标识。

4）变量ConcurrentHashMap<String,Session> webSocketUserMap用来存用户名和不同的Session，也可以采用其他类型的Key。

5）从客户端取得信息，并对取得的信息进行处理，取得发送信息的目标的Session（target），将所要发送的信息（message）通过sendMessage(target,message)。其中，这个方法的调用需要从变量ConcurrentHashMap<Session,Object> webSocketMap中取出自己的WebSocketServer对象。

6）方法public void sendMessage(Session mySession,String message)使用的是mySession.getBasicRemote().sendText(message)来给mySession发送信息。

4.1.4 WebSocket前端实现

4.1.4.1 关键代码

$(document).ready(function(){

// 创建websocket，其中a、b、c是向服务器端传送的初始变量，不是必须的，“/websocket”应与服务器端的注解@ServerEndPoint的参数相同。

var websocket = new WebSocket(“ws://localhost:8080/项目名称/websocket/{a}/{b}/{c}”);

// 打开时

websocket.onopen = function(evnt) {

console.log(" websocket.onopen ");

};

// 处理消息时

websocket.onmessage = function(evnt) {

$("#msg").append("<p><font color='red'>" + evnt.data + "</font></p>");

console.log(" websocket.onmessage ");

};

// 出错时

websocket.onerror = function(evnt) {

console.log(" websocket.onerror ");

};

// 关闭时

websocket.onclose = function(evnt) {

console.log(" websocket.onclose ");

};

// 向服务器端发送信息

websocket.send(msg);

})

4.1.4.2 前端的编写思路

1）利用websocket.onmessage的方法来对获取信息后前端的内容进行处理。

2）可以通过websocket.send（）发送JSON格式的数据，来发送例如表格内的各种信息。

4.2 Redis技术

4.2.1 原理说明

4.2.1.1 简介

1）Redis支持数据的持久化，可以将内存中的数据保存在磁盘中，重启的时候可以再次加载进行使用。

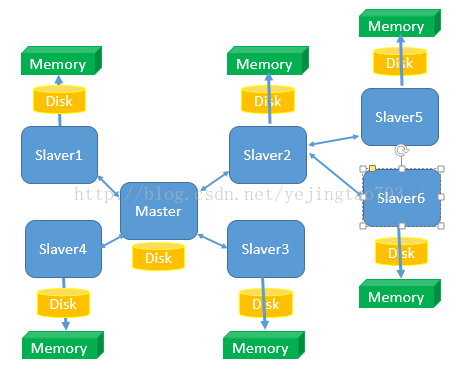
2）Redis不仅仅支持简单的key-value类型的数据，同时还提供list，set，zset，hash等数据结构的存储。

3）Redis支持数据的备份，即master-slave模式的数据备份。

4）Redis具有性能极高、丰富的数据类型、操作原子性等优势。

4.2.1.2 Redis集群设计原理

1) 节点主从：



1. 只有1个Master，可以有N个slaver，而且Slaver也可以有自己的Slaver，由于这种主从的关系决定他们是在配置阶段就要指定他们的上下级关系，而不是Zookeeper那种平行关系是自主推优出来的。
2. 读写分离，Master只负责写和同步数据给Slaver，Slaver承担了被读的任务，所以Slaver的扩容只能提高读效率不能提高写效率。
3. Slaver先将Master那边获取到的信息压入磁盘，再load进内存，client端是从内存中读取信息的，所以Redis是内存数据库。
4. 当一个新的Slaver加入到这个集群时，会主动找Master来拜码头，Master发现新的小弟后将全量数据发送给新的Slaver，数据量越大性能消耗也就越大，所以尽量避免在运行时做Slaver的扩容。

简单总结下主从模式的设计：

优点：

读写分离，通过增加Slaver可以提高并发读的能力。

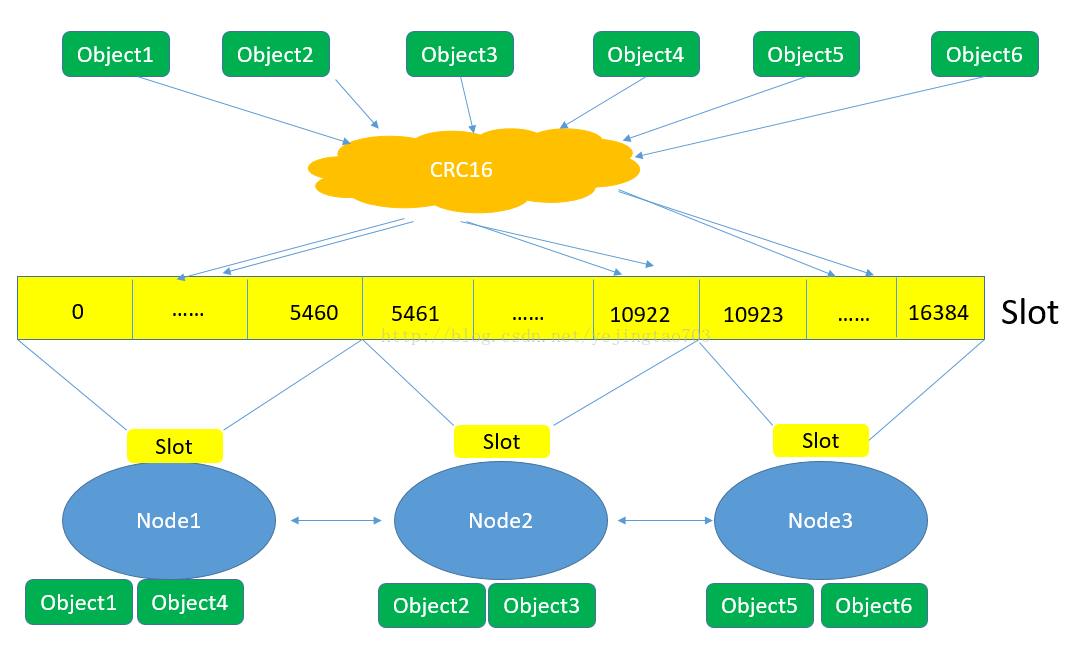
缺点：

Master写能力是瓶颈。

虽然理论上对Slaver没有限制但是维护Slaver开销总将会变成瓶颈。

Master的Disk大小也将会成为整个Redis集群存储容量的瓶颈。

2) 哈希Slot：



1. 对象保存到Redis之前先经过CRC16哈希到一个指定的Node上，例如Object4最终Hash到了Node1上。
2. 每个Node被平均分配了一个Slot段，对应着0-16384，Slot不能重复也不能缺失，否则会导致对象重复存储或无法存储。
3. Node之间也互相监听，一旦有Node退出或者加入，会按照Slot为单位做数据的迁移。例如Node1如果掉线了，0-5460这些Slot将会平均分摊到Node2和Node3上, 由于Node2和Node3本身维护的Slot还会在自己身上不会被重新分配，所以迁移过程中不会影响到5461-16384Slot段的使用。

简单总结下哈希Slot的优缺点：

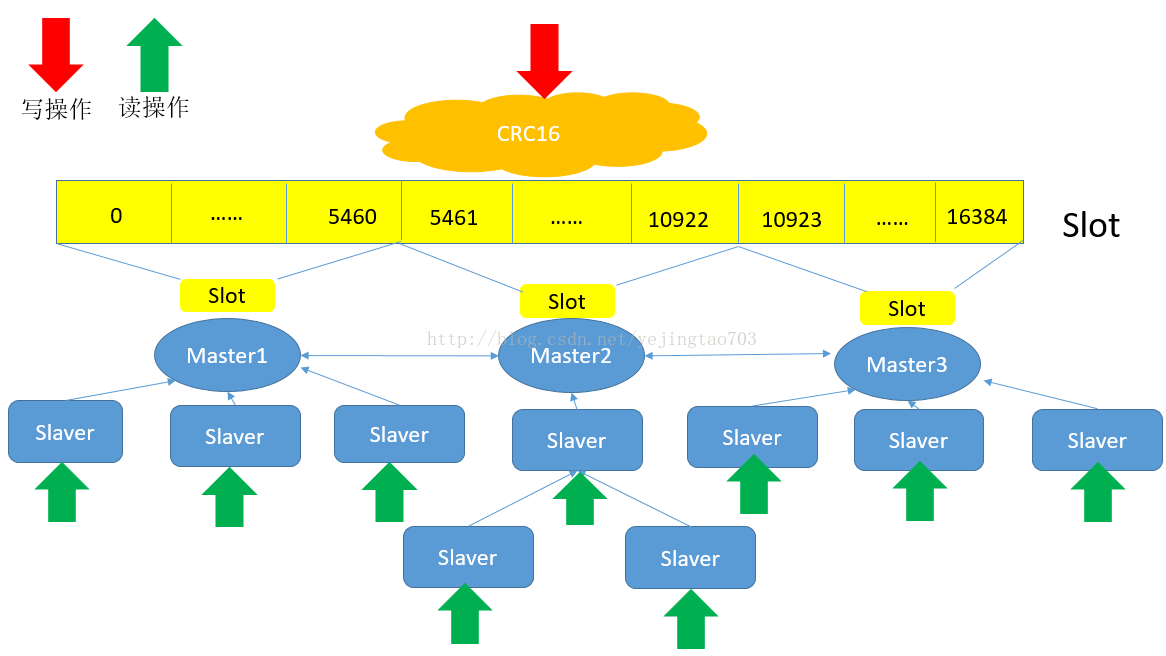
缺点：

每个Node承担着互相监听、高并发数据写入、高并发数据读出，工作任务繁重

优点：

将Redis的写操作分摊到了多个节点上，提高写的并发能力，扩容简单。

3) 二合一



主从和哈希的设计优缺点正好是相互弥补的，将图一每一套主从对应到图二中的每一个Node，就是Redis集群的终极形态，先Hash分逻辑节点，然后每个逻辑节点内部是主从。

想扩展并发读就添加Slaver，想扩展并发写就添加Master，想扩容也就是添加Master，任何一个Slaver或者几个Master挂了都不会是灾难性的故障。

4.2.2 Redis配置

4.2.2.1 依赖的jar包

1) jedis.jar

2) spring-data-redis.jar

4.2.2.2 xml配置与properties文件

1) properties文件参数：

|  |  |
| --- | --- |
| 字段 | 描述 |
| host | 服务器地址，一般为本地ip（127.0.0.1） |
| port | 端口，默认为6379 |
| password | 密码，默认没有密码 |
| maxIdle | 最大空闲数 |
| maxActive | 最大连接数 |
| maxWait | 最大建立连接等待时间 |
| testOnBorrow | 指明是否在从池中取出连接前进行检验,如果检验失败,则从池中去除连接并尝试取出另一个 |
| timeout | 客户端超时时间，单位是毫秒 |
| fep.local.cache.capacity | 本地内存最大占用量 |

2）redis-config.xml配置

具体内容请看resources中的redis-config.xml。

注意：别忘记在applicationContext.xml中添加

<context:property-placeholder location="classpath:redis.properties" />以及<import resource="redis-config.xml"/>

4.2.3 Redis的使用

4.2.3.1 开启Redis

Jedis jedis = new Jedis();

4.2.3.2 Redis常用方法

对Redis进行操作的方法用Jedis创建的对象来调用。

在此仅列出有关list的最基本操作，其余想了解的内容请到<http://www.runoob.com/redis/redis-commands.html> 查看。

|  |  |
| --- | --- |
| 方法 | 描述 |
| lpush(String key, String value) | 将value插入列表key |
| llen(String key) | 返回列表key的长度 |
| lindex(String key, int index) | 返回列表key的索引为index的value |
| lrem(String key, int count, String value) | 删除列表key的内容为value的对，其中count为删除的条数。 |

4.2.4 Redis+SpringMVC+WebSocket的整合思路

1）利用WebSocket给指定用户发信息，这个时候就会调用WebSocketServer的OnMessage方法，在这个方法里可以用Redis把传过来的信息存到list中。

2）前端读取信息的时候，可以用springMVC的Controller来把Redis指定list中的信息读出来，并用ajax和vue接收，显示在页面上。

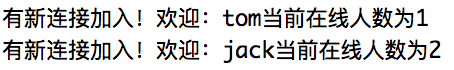
3）实现已读的过程，可以利用Controller把特定的message从Redis指定的list中取出来并删除掉，再把这条信息储存到数据库中。

4）存到Redis的list中的value可以是json格式的字符串，这样方便进行多个参数的传输。

4.3 Demo

1） 使用url：localhost:8080/项目名称/SubSocket.jsp?uname=用户名 来访问demo页面。通过开启多个页面，来模拟有多个用户连接。

2）每开启一个页面，控制台上都会打出相关的信息，例如，“有新连接加入”。



3) demo有两个输入框，一个用来输入信息，一个用来输入姓名。



4）点击发送数据后，相应的用户会收到信息，并得到一条提示。



5）用户可以点击“显示所有信息”按钮来查看所有未读信息。



6）用户可以点击“已读”按钮，来标记信息为已读，此条信息就会从Redis缓存中被删除。



注意：此demo没有实现把已读的数据添加到数据库中，具体功能需要开发者独自添加。

4.4 建议

1）要实现一对多的信息传输，需要在前端向后端传输的json数据中添加多个用户的信息，再在WebSocketServer中的OnMessage方法中来把用户信息解析出来，在Redis中循环发送。

2）新消息的个数可以直接读取Redis中相关list的个数，并通过WebSocket的前端的onmessage来接收。

3）从Redis缓存中读取信息的速度要远远快于从数据库中读取，所以要仔细斟酌是选择从Redis中读取还是选择从数据库读取。已读的信息不一定必须要从数据库中读取，也可以选择建立一个新的Redis的list用来存已经读过的信息，其他信箱，例如草稿箱，也是如此。