hryou0922 Lv2
2018年07月30日 阅读 1238

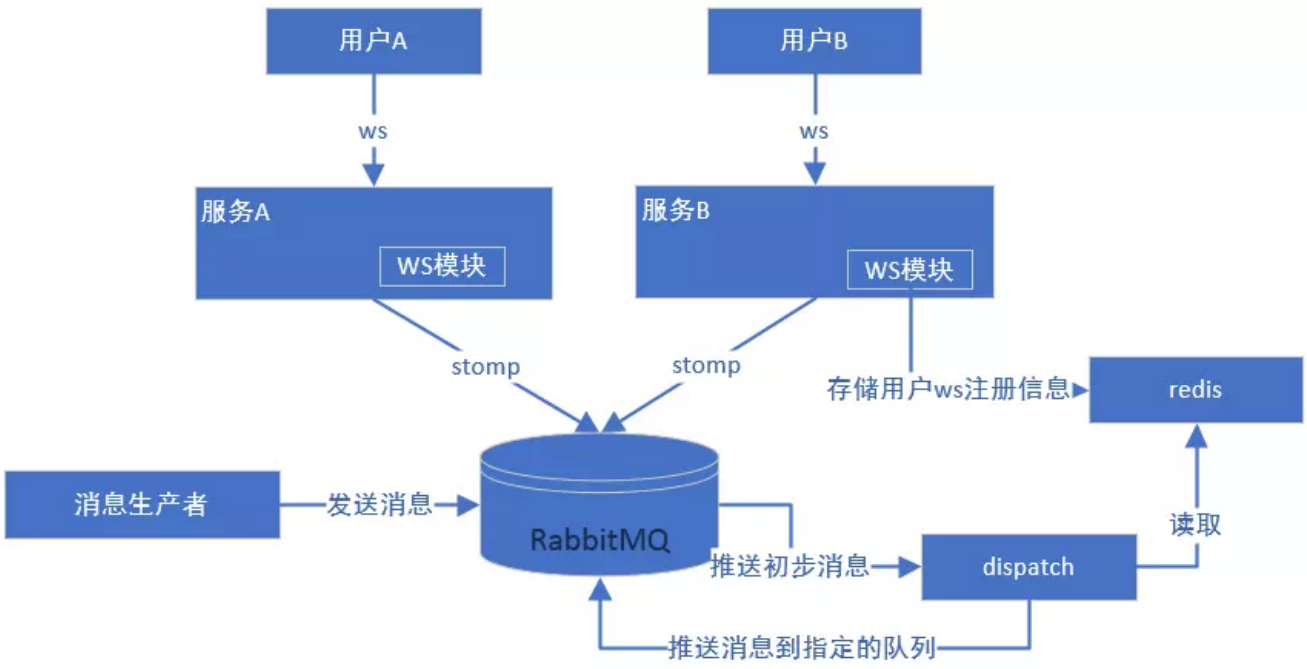
关注

# Spring Boot系列22 Spring Websocket实现websocket集群方案的Demo

## 概述

上一篇文章Spring Boot系列21 Spring Websocket实现websocket集群方案讨论里详细介绍了WebSocket集群的三种方案，并得出结论第三个方案是最好的，本文我们实现第三个方案。

第三个方案如下图




在方案一的基础进行如下修改，新的架构图流程如下：

1. 服务A增加WS模块，当websocket连接过来时，将此用户的连接信息（主要是websocket sesionId值）存储redis中
2. 消息生产者发送消息到的交换机，这些服务不直接推送服务A/B

订阅的队列上

4. 前端接收消息

## 详细实现的代码

工程名称：mvc 本文在Spring Boot系列20 Spring Websocket实现向指定的用户发送消息的基础进行修改。

### 在pom.xml中引入redis,rabbitmq相关的jar

```
<!--  webscoekt 集群 需要 引入支持RabbitMQ, redis-->
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-amqp</artifactId>
</dependency>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>
```

### rabbitmq, redis的配置

application-wscluster.properties

```
# websocket集群需要配置RabbitMQ
spring.rabbitmq.host:192.168.21.3
spring.rabbitmq.virtual-host: /icc-local
spring.rabbitmq.username: icc-dev
spring.rabbitmq.password: icc-dev

# 配置redis
spring.redis.database=0
spring.redis.host=192.168.21.4
# spring.redis.password=
spring.redis.port=7001
spring.redis.pool.max-idle=8
spring.redis.pool.min-idle=0
spring.redis.pool.max-active=8
spring.redis.pool.max-wait=-1
```

接口IRedisSessionService定义了对redis的操作 IRedisSessionService实现类将用户名称和websocket sessionId的关系存储到redis，提供添加、删除、查询 **IRedisSessionService**

```
public interface IRedisSessionService {
    void add(String name, String wsSessionId);
    boolean del(String name);
    String get(String name);
}
```

**SimulationRedisSessionServiceImpl** 将用户名称和websocket sessionId的关系存储到redis，提供添加、删除、查询

```
@Component
public class SimulationRedisSessionServiceImpl implements IRedisSessionService {

    @Autowired
    private RedisTemplate<String, String> template;

    // key = 登录用户名称,  value=websocket的sessionId
    private ConcurrentHashMap<String,String> redisHashMap = new ConcurrentHashMap<>(32);

    /**
     * 在缓存中保存用户和websocket sessionid的信息
     * @param name
     * @param wsSessionId
     */
    public void add(String name, String wsSessionId){
        BoundValueOperations<String,String> boundValueOperations = template.boundValueOps(n
        boundValueOperations.set(wsSessionId,24 * 3600, TimeUnit.SECONDS);
    }

    /**
     * 从缓存中删除用户的信息
     * @param name
     */
    public boolean del(String name){
        return template.execute(new RedisCallback<Boolean>() {

            @Override
            public Boolean doInRedis(RedisConnection connection)
                    throws DataAccessException {
                byte[] rawKey = template.getStringSerializer().serialize(name);
                return connection.del(rawKey) > 0;
            }
```

```
    /**
     * 根据用户id获取用户对应的sessionId值
     * @param name
     * @return
     */
    public String get(String name){
        BoundValueOperations<String,String> boundValueOperations = template.boundValueOps(n
        return boundValueOperations.get();
    }
}
```

## AuthWebSocketHandlerDecoratorFactory

装饰WebSocketHandlerDecorator对象，在连接建立时，保存websocket的session id，其中key为帐号名称；在连接断开时，从缓存中删除用户的sesionId值。此websocket sessionId值用于创建消息的路由键。

```
@Component
public class AuthWebSocketHandlerDecoratorFactory implements WebSocketHandlerDecoratorFacto
    private static final Logger log = LoggerFactory.getLogger(AuthWebSocketHandlerDecorator

    @Autowired
    private IRedisSessionService redisSessionService;

    @Override
    public WebSocketHandler decorate(WebSocketHandler handler) {
        return new WebSocketHandlerDecorator(handler) {
            @Override
            public void afterConnectionEstablished(final WebSocketSession session) throws E
                // 客户端与服务器端建立连接后，此处记录谁上线了
                Principal principal = session.getPrincipal();
                if(principal != null){
                    String username = principal.getName();
                    log.info("websocket online: " + username + " session " + session.getId(
                    redisSessionService.add(username, session.getId());
                }
                super.afterConnectionEstablished(session);
            }

            @Override
            public void afterConnectionClosed(WebSocketSession session, CloseStatus cl
                // 客户端与服务器端断开连接后，此处记录谁下线了
                Principal principal = session.getPrincipal();
```

```
                    redisSessionService.del(username);
                }
                super.afterConnectionClosed(session, closeStatus);
            }
        };
    }
}
```

## WebSocketRabbitMQMessageBrokerConfigurer

在Spring Boot系列20 Spring Websocket实现向指定的用户发送消息的基础上增加如下功能，将myWebSocketHandlerDecoratorFactory配置到websocket

```
@Configuration
// 此注解开使用STOMP协议来传输基于消息代理的消息，此时可以在@Controller类中使用@MessageMapping
@EnableWebSocketMessageBroker
public class WebSocketRabbitMQMessageBrokerConfigurer extends AbstractWebSocketMessageBroke

    @Autowired
    private MyPrincipalHandshakeHandler myDefaultHandshakeHandler;
    @Autowired
    private AuthHandshakeInterceptor sessionAuthHandshakeInterceptor;

    @Autowired
    private AuthWebSocketHandlerDecoratorFactory myWebSocketHandlerDecoratorFactory;

    @Override
    public void registerStompEndpoints(StompEndpointRegistry registry) {
        ….
    }

    @Override
    public void configureMessageBroker(MessageBrokerRegistry registry) {
          …
    }

    /**
     * 这时实际spring weboscket集群的新增的配置，用于获取建立websocket时获取对应的sessionid值
     * @param registration
     */
    @Override
    public void configureWebSocketTransport(WebSocketTransportRegistration registratic
        registration.addDecoratorFactory(myWebSocketHandlerDecoratorFactory);
```

### TestMQCtl：

在上文Spring Boot系列20 Spring Websocket实现向指定的用户发送消息的基础上，对此类进行修改

- sendMq2User()方法根据用户的帐号和websocket sessionId根据["web订阅队列名称+'-user'+websocket sessionId"]组合路由键。然后通过AmqpTemplate 实例向amq.topic交换机发送消息，路由键为["web订阅队列名称+'-user'+websocket sessionId"]。方法中websocket sessionId是从根据帐号名称从redis中获取 其它的方法，这里不一一列出

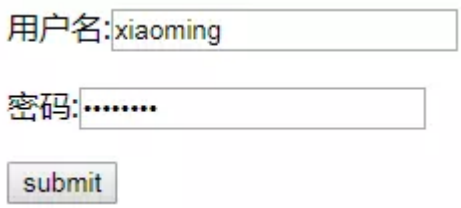
```
@Controller
@RequestMapping(value = "/ws")
public class TestMQCtl {
    private  static final Logger log = LoggerFactory.getLogger(TestMQCtl.class);

    @Autowired
    private AmqpTemplate amqpTemplate;
    @Autowired
    private IRedisSessionService redisSessionService;

     /**
     * 向执行用户发送请求
     * @param msg
     * @param name
     * @return
     */
    @RequestMapping(value = "send2user")
    @ResponseBody
    public int sendMq2User(String msg, String name){
        // 根据用户名称获取用户对应的session id值
        String wsSessionId = redisSessionService.get(name);
        RequestMessage demoMQ = new RequestMessage();
        demoMQ.setName(msg);

        // 生成路由键值，生成规则如下: websocket订阅的目的地 + "-user" + websocket的sessionId值。生
        String routingKey = getTopicRoutingKey("demo", wsSessionId);
        // 向amq.topi交换机发送消息，路由键为routingKey
        log.info("向用户[{}]sessionId=[{}]，发送消息[{}]，路由键[{}]", name, wsSessionId, wsSes
        amqpTemplate.convertAndSend("amq.topic", routingKey,  JSON.toJSONString(demoMQ));
        return 0;
    }
```

```
     *
     * @param actualDestination
     * @param sessionId
     * @return
     */
    private String getTopicRoutingKey(String actualDestination, String sessionId){
        return actualDestination + "-user" + sessionId;
    }
    ….
}
```

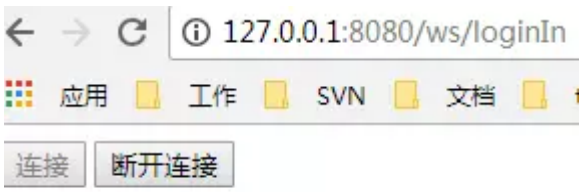
## 测试

**以不同端口启动两个服务** 启动服务类：WebSocketClusterApplication 以“--spring.profiles.active=wscluster --server.port=8081”参数启动服务A 以“--spring.profiles.active=wscluster --server.port=8082”参数启动服务B

**登录模拟帐号:xiaoming登录服务A，xiaoming2登录服务B** 使用xiaoming登录服务A，并登录websocket http://127.0.0.1:8081/ws/login 使用xiaoming登录，并提交




点击连接，如果连接变灰色，则登录websocket成功

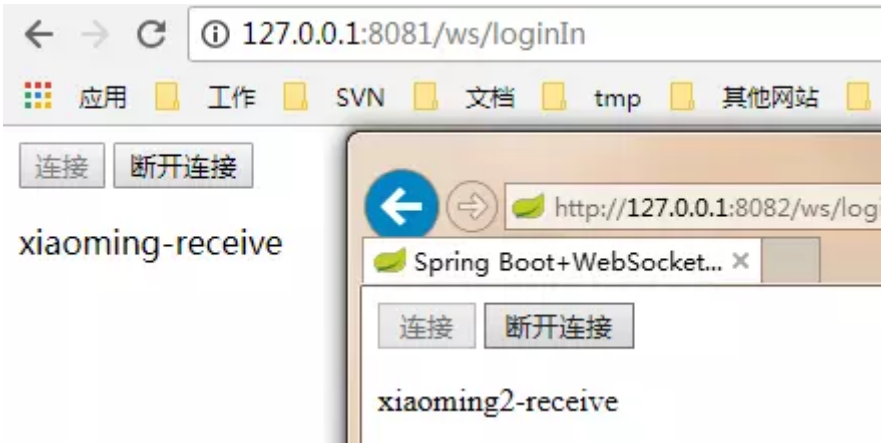



打开另一个浏览器，使用xiaoming2登录服务B，并登录websocket http://127.0.0.1:8082/ws/login 使用xiaoming2登录并提交，最后登录websocket

**登录服务A模拟发送页面** 登录http://127.0.0.1:8081/ws/send，发送消息

1. 向帐号xiaoming发送消息xiaoming-receive，只能被连接服务A的服务websocket收到 §

此时两个页面收到信息：




xiaoming帐号只收到xiaoming-receive xiaoming2帐号只收到xiaoming2-receive

**登录服务B模拟发送页面** 登录http://127.0.0.1:8082/ws/send，发送消息，和 http://127.0.0.1:8081/ws/send 一样发送相同消息，结果是一样

**结论** 无论用户登录的服务A，还是服务B，我们通过以上的代码，我们都可以发送消息到指定的用户，所以我们已经实现websocket集群

## 代码

所有的详细代码见github代码，请尽量使用tag v0.24，不要使用master，因为master一直在变，不能保证文章中代码和github上的代码一直相同

> **关注下面的标签，发现更多相似文章**

WebSocket Spring Spring Boot Redis

输入评论...

## 相关推荐

专栏 · 王磊的博客 · 46分钟前 · Redis / 后端

**Redis持久化的几种方式——深入解析RDB**

6

专栏 · zxiaofan · 6小时前 · Redis

**玩转Redis-如何高效访问Redis中的海量数据**

5

专栏 · 我妻礼弥 · 1天前 · Redis

**Redis 的命令详解 - Sorted Set 篇**

3

专栏 · shanyue · 4天前 · Node.js / WebSocket

**关于一个 websocket 多节点分布式问题的头条前端面试题**

47 4

专栏 · Noodles_Mars · 3天前 · Spring / Java

**手写Spring框架，加深对Spring工作机制的理解！**

17

专栏 · SnailClimb · 6天前 · Java / Spring Boot

**如何在 Spring/Spring Boot 中做参数校验？你需要了解的都在这里！**

51 7

荐 · 专栏 · yanglbme · 5天前 · Redis

**面试官：redis 的过期策略都有哪些？内存淘汰机制都有哪些？手写一下 LRU 代码实现？**

25 11

专栏 · 饿了么物流技术团队 · 1月前 · Redis / 后端

**Redis 到底是怎么实现“附近的人”这个功能的呢？**

专栏 · zimug · 1天前 · Spring Boot

## 详解Spring Security的formLogin登录认证模式