**分布式Session共享（一致性）**

使用Nginx+Tomcat进行负载均衡时，希望使用轮询方式进行负载。

但是如果使用轮询方式的话，可能会访问不同的Tomcat，此时如果不进行Session共享，则相当于是一个新的Session。

就比如现有系统都是需要认证登录的系统，如果没有Session共享，则会导致用户退出登录。

当分布式的系统部署起来了，session就需要一致性，方便验证已经登陆的用户信息。

session一致性解决方法：

# ：基于Nginx的ip\_hash策略来做负载均衡。

原理：根据IP做hash计算，同一个IP的请求始终定位到同一台tomcat

缺点：如果这台机器挂掉了，那这些IP就不能访问了。

Ip\_hash是在upstream配置中定义的:

upstream nginx.example.com{

server 192.168.74.235:80;

server 192.168.74.236:80;

ip\_hash;

}

server{

listen 80;

location /{

proxy\_pass

http: //nginx .example.com;

}

}

Ip\_hash机制缺陷:

(1).nginx不是最前端的服务器

ip\_hash要求nginx一定是最前端的服务器,否则nginx得不到正确ip,就不能根据ip作hash. Eg: 使用的是squid为最前端.那么nginx取ip时只能得到squid的服务器ip地址,用这个地址来作分流肯定是错乱的

(2).nginx的后端还有其它负载均衡

假如nginx后端还有其它负载均衡,将请求又通过另外的方式分流了,那么某个客户端的请求肯定不能定位到同一台session应用服务器上,这么算起来,nginx后端只能直接指向应用服务器,或者再搭一人squid,然后指向应用服务器. 最好 的办法是用location作一次分流,将需要session的部分请求通过ip\_hash分流,剩下的走其它后端去.

# ：Tomcat服务器Session复制

原理：Tomcat服务器创建Session后，会通过组播方式把session发送到组播地址中的其他服务器上。

缺点：这种情况会大量占用内存以及带宽，非常不划算

# ：使用数据库对Session集中统一管理

原理：Session不由单个tomcat管理，而是统一放到一个地方集中式管理，读取和写入Session都依赖第三方软件，如Redis

MongoDB, mysql

缺点：有侵入性

优点：可扩展能力强，高可用，适用于大型分布式环境。



## 实现方式：4 Spring-session+redis

# 学习Spring-Session+Redis实现session共享

### 1、添加依赖

<dependency>

<groupId>org.springframework.session</groupId>

<artifactId>spring-session-data-redis</artifactId>

<version>1.2.1.RELEASE</version>

</dependency>

<dependency>

<groupId>redis.clients</groupId>

<artifactId>jedis</artifactId>

<version>2.8.1</version>

</dependency>

### 2、配置

spring-mvc.xml:

<bean id="redisHttpSessionConfiguration"

class="org.springframework.session.data.redis.config.annotation.web.http.RedisHttpSessionConfiguration">

<property name="maxInactiveIntervalInSeconds" value="600"/>

</bean>

<bean id="jedisPoolConfig" class="redis.clients.jedis.JedisPoolConfig">

<property name="maxTotal" value="100" />

<property name="maxIdle" value="10" />

</bean>

<bean id="jedisConnectionFactory"

class="org.springframework.data.redis.connection.jedis.JedisConnectionFactory" destroy-method="destroy">

<property name="hostName" value="${redis\_hostname}"/>

<property name="port" value="${redis\_port}"/>

<property name="password" value="${redis\_pwd}" />

<property name="timeout" value="3000"/>

<property name="usePool" value="true"/>

<property name="poolConfig" ref="jedisPoolConfig"/>

</bean>

web.xml添加拦截器：

<filter>

<filter-name>springSessionRepositoryFilter</filter-name>

<filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>

</filter>

<filter-mapping>

<filter-name>springSessionRepositoryFilter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

### 3、使用spring-session

只要使用标准的servlet api调用session，在底层就会通过Spring Session得到的，并且会存储到Redis或其他你所选择的数据源中。

这里是我写的一个demo:

/\*\*

\* @author fengzp

\* @date 17/2/23下午3:19

\* @email fengzp@gzyitop.com

\* @company 广州易站通计算机科技有限公司

\*/

@Controller

@RequestMapping(value = "index")

public class IndexController {

private final Gson gson = new GsonBuilder().setDateFormat("yyyyMMddHHmmss").create();

@RequestMapping(value = "login")

public String login(HttpServletRequest request, String username){

request.getSession().setAttribute("user", gson.toJson(new User(username,"123456")));

return "login";

}

@RequestMapping(value = "index")

public String index(HttpServletRequest request, Model model){

User user = gson.fromJson(request.getSession().getAttribute("user").toString(), User.class);

model.addAttribute("user", user);

return "index";

}

}

index.jsp:

第一个tomcat

<html>

<body>

<h2>Hello World!</h2>

<p>${user.username}</p>

</body>

</html>

第二个tomcat

<html>

<body>

<h2>Hello World! i am the second!</h2>

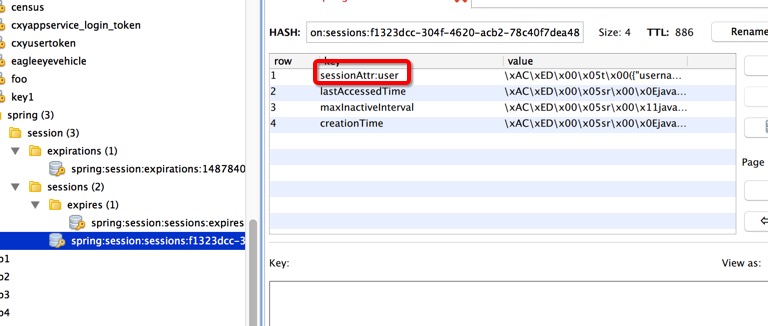
<p>${user.username}</p>

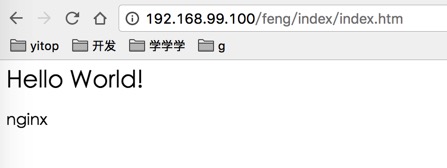
</body>

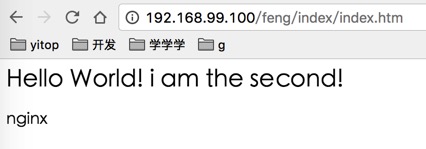
</html>

### 测试

这里利用上一篇nginx负载配置的两个tomcat来测试。  
首先访问 http://192.168.99.100/feng/index/login.htm?username=nginx 来触发生成session。

查看redis，发现session已经保存到redis。  


访问 http://192.168.99.100/feng/index/index.htm 来读取session, 并刷新多次。  




发现在负载的情况下读取session没问题，并且是同一个session，成功实现负载+session共享！



在session一致性中，是将tomact的session实现给覆盖，使用spring的session，就是上面的过滤器覆盖的，DelegetingFilterproxy;可以查看这个过滤器的实现：

最主要是这个方法：



实现第三种方法的使用情况：

https://www.cnblogs.com/andyfengzp/p/6434287.html

实现原理讲解：

https://blog.csdn.net/xlgen157387/article/details/60321984

https://blog.csdn.net/qq\_27529917/article/details/79169905

# Tomcat的Session共享（复制）的几种实现方案

## 为什么会提出这个问题？

使用Nginx+Tomcat进行负载均衡时，希望使用轮询方式进行负载。但是如果使用轮询方式的话，可能会访问不同的Tomcat，此时如果不进行Session共享，则相当于是一个新的Session。就比如现有系统都是需要认证登录的系统，如果没有Session共享，则会导致用户退出登录。

## 方案1：使用Tomcat内置的Session复制方案

具体配置如下：

1. *<!-- 第1步：修改server.xml，在Host节点下添加如下Cluster节点 -->*
2. *<!-- 用于Session复制 -->*
3. <Cluster className="org.apache.catalina.ha.tcp.SimpleTcpCluster" channelSendOptions="8">
4. <Manager className="org.apache.catalina.ha.session.DeltaManager" expireSessionsOnShutdown="false" notifyListenersOnReplication="true" />
5. <Channel className="org.apache.catalina.tribes.group.GroupChannel">
6. <Membership className="org.apache.catalina.tribes.membership.McastService" address="228.0.0.4"
7. port="45564" frequency="500" dropTime="3000" />
8. *<!-- 这里如果启动出现异常，则可以尝试把address中的"auto"改为"localhost" -->*
9. <Receiver className="org.apache.catalina.tribes.transport.nio.NioReceiver" address="auto" port="4000"
10. autoBind="100" selectorTimeout="5000" maxThreads="6" />
11. <Sender className="org.apache.catalina.tribes.transport.ReplicationTransmitter">
12. <Transport className="org.apache.catalina.tribes.transport.nio.PooledParallelSender" />
13. </Sender>
14. <Interceptor className="org.apache.catalina.tribes.group.interceptors.TcpFailureDetector" />
15. <Interceptor className="org.apache.catalina.tribes.group.interceptors.MessageDispatchInterceptor" />
16. </Channel>
17. <Valve className="org.apache.catalina.ha.tcp.ReplicationValve" filter="" />
18. <Valve className="org.apache.catalina.ha.session.JvmRouteBinderValve" />
19. <Deployer className="org.apache.catalina.ha.deploy.FarmWarDeployer" tempDir="/tmp/war-temp/"
20. deployDir="/tmp/war-deploy/" watchDir="/tmp/war-listen/" watchEnabled="false" />
21. <ClusterListener className="org.apache.catalina.ha.session.ClusterSessionListener" />
22. </Cluster>
24. *<!-- 第2步：在web.xml中添加如下节点 -->*
25. *<!-- 用于Session复制 -->*
26. <distributable/>

最终在Tomcat日志里可能会打印出如下日志信息：

1. 29-Mar-2018 15:06:22.181 INFO [localhost-startStop-1] org.apache.catalina.ha.session.DeltaManager.getAllClusterSessions Manager [/ydzwV3], requesting session state from org.apache.catalina.tribes.membership.MemberImpl[tcp://{192, 168, 1, 233}:4010,{192, 168, 1, 233},4010, alive=572778, securePort=-1, UDP Port=-1, id={-56 73 0 62 -31 -122 65 -50 -108 49 -1 -12 -84 -32 -7 -77 }, payload={}, command={}, domain={}, ]. This operation will timeout if no session state has been received within 60 seconds.
2. 29-Mar-2018 15:06:22.282 INFO [localhost-startStop-1] org.apache.catalina.ha.session.DeltaManager.waitForSendAllSessions Manager [/ydzwV3]; session state send at 3/29/18 3:06 PM received in 107 ms.

## **方案2：使用第三方（个人）基于Tomcat实现的Session管理**

这里github上的tomcat-redis-session-manager来实现。

项目地址：<https://github.com/jcoleman/tomcat-redis-session-manager>

具体配置方法，在上述站点中有详细说明。在此不再赘述。

注意：这种方式还不支持Tomcat8。尽管有人基于上述代码进行了修改，但不能保证可用性。

## 方案3：使用Spring Session实现

Spring Session提供了多种方式来存储Session信息，包括redis、mongo、gemfire、hazelcast、jdbc等。这里用redis来举例说明，首先进行依赖添加，然后进行配置即可。

### 添加依赖（gradle）

compile "org.springframework.session:spring-session-data-redis:1.3.2.RELEASE"

注意：当引入上述依赖包时，还会引入如下依赖：

org.apache.commons:commons-pool2:2.4.2

org.springframework.data:spring-data-redis:1.7.10.RELEASE

org.springframework.session:spring-session:1.3.2.RELEASE

redis.clients:jedis:2.8.1

项目中原来使用了Redis作为Spring Cache的实现，当时使用的spring-data-redis是1.4.2.RELEASE版本，现在使用1.7.10.RELEASE版本后，需要把cacheManager这个Bean做如下调整：

1. *<!-- 调整前 -->*
2. <bean id="cacheManager" class="org.springframework.data.redis.cache.RedisCacheManager"
3. [c:](https://www.baidu.com/s?wd=c%3A&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd)template-ref="redisTemplate"/>
5. *<!-- 调整后 -->*
6. <bean id="cacheManager" class="org.springframework.data.redis.cache.RedisCacheManager"
7. c:redisOperations-ref="redisTemplate">
8. </bean>

### 进行Spring Session配置

1. *<!--*
2. *第1步：在Spring配置文件中添加如下bean*
3. *以后在web.xml中配置session超时时间就无效了，如果需要指定session超时时间，则使用maxInactiveIntervalInSeconds来指定，默认是1800s=30min*
4. *-->*
5. <bean class="org.springframework.session.data.redis.config.annotation.web.http.RedisHttpSessionConfiguration"
6. p:maxInactiveIntervalInSeconds="1800"/>

??? 这里怎么没有Redis连接配置 ??? 请看RedisHttpSessionConfiguration类中的如下代码：

1. // 这里会自动注入connectionFactory，而项目中已经注入了jedisConnectionFactory
2. @Bean
3. public RedisTemplate<Object, Object> sessionRedisTemplate(
4. RedisConnectionFactory connectionFactory) {
5. RedisTemplate<Object, Object> template = new RedisTemplate<Object, Object>();
6. template.setKeySerializer(new StringRedisSerializer());
7. template.setHashKeySerializer(new StringRedisSerializer());
8. if (this.defaultRedisSerializer != null) {
9. template.setDefaultSerializer(this.defaultRedisSerializer);
10. }
11. template.setConnectionFactory(connectionFactory);
12. return template;
13. }

项目中注入的jedisConnectionFactory Bean如下：

1. <bean id="jedisConnectionFactory" class="org.springframework.data.redis.connection.jedis.JedisConnectionFactory"
2. p:host-name="${redis.hostname}"
3. p:port="${redis.port}"
4. p:database="${redis.database}"
5. p:poolConfig-ref="redispoolconfig"
6. p:use-pool="${redis.usepool}">
7. </bean>

所以，如果你项目中从来没有使用过Redis，也可以使用如下配置：

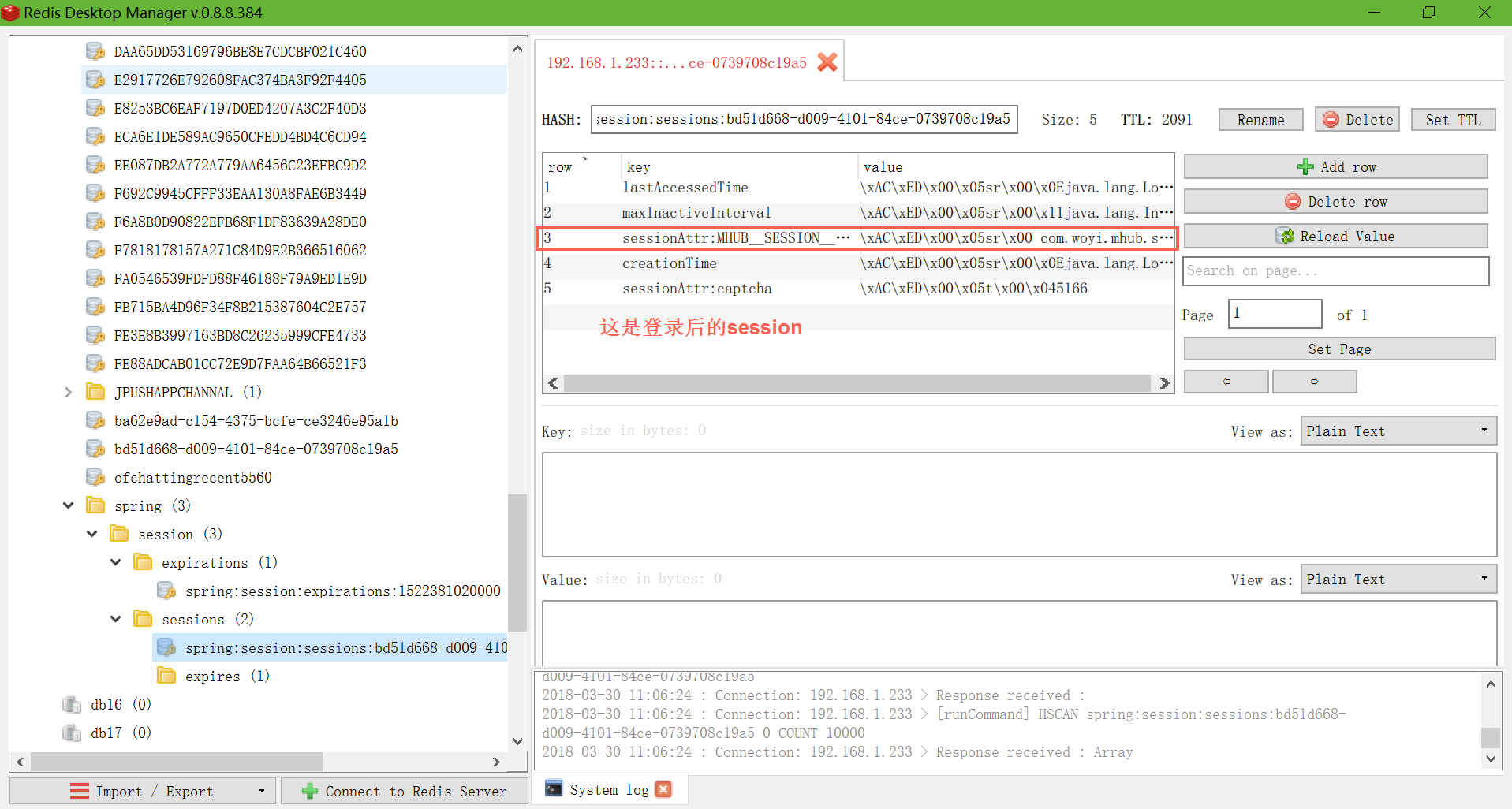
1. <bean class="org.springframework.session.data.redis.config.annotation.web.http.RedisHttpSessionConfiguration"/>
2. *<!--*
3. *Jdeis连接工厂Bean*
4. *注意：这种方式没有使用连接池，生产环境下务必需要使用连接池*
5. *-->*
6. <bean id="jedisConnectionFactory" class="org.springframework.data.redis.connection.jedis.JedisConnectionFactory"
7. p:host-name="192.168.1.233" p:port="6379" p:database="15" p:usePool="false">
8. </bean>

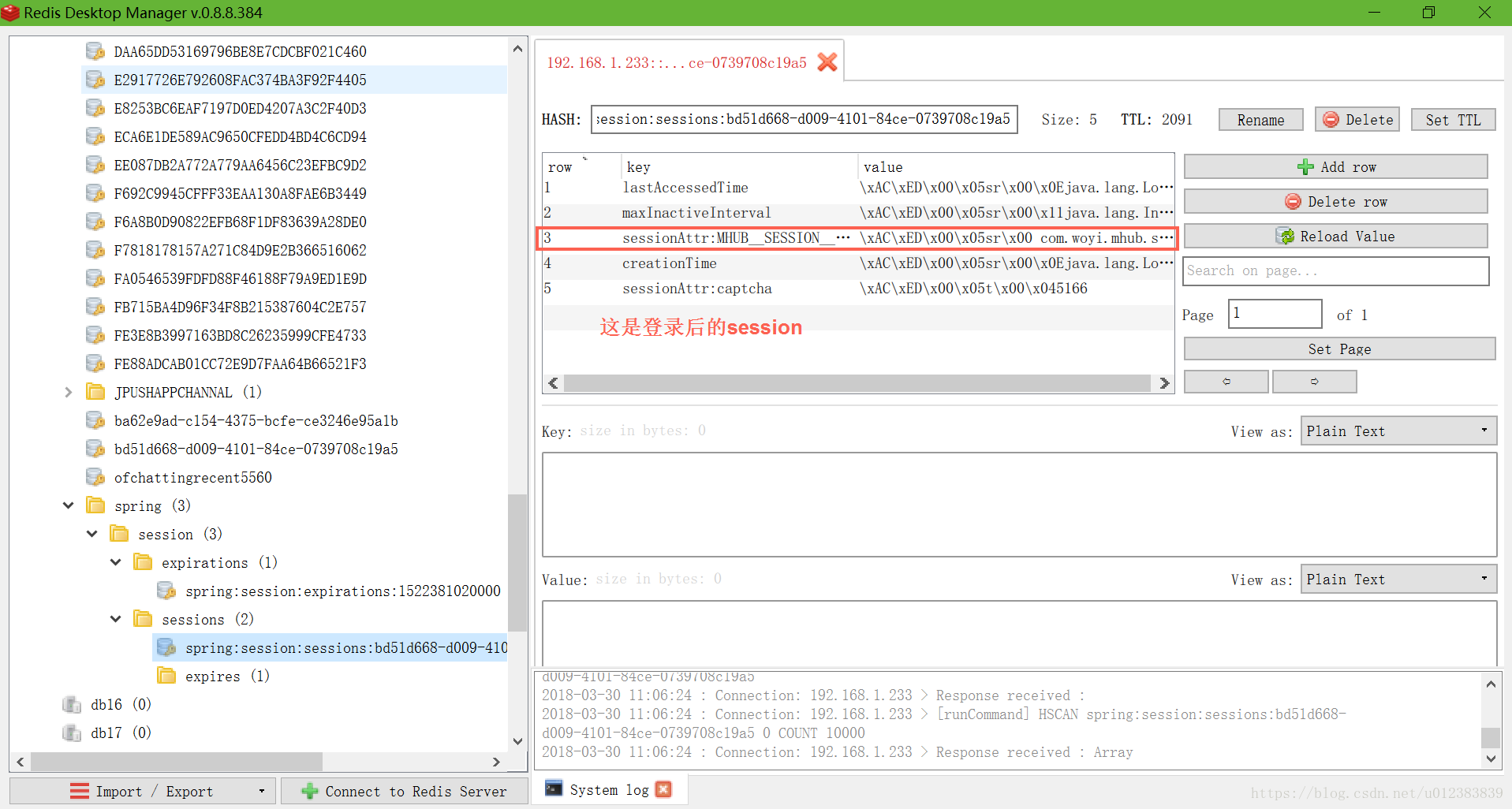
下面进行过滤器的配置：

1. *<!-- 第2步：在web.xml中添加如下过滤器 注意：此过滤器必须放在其他过滤器之前-->*
2. <filter>
3. <filter-name>springSessionRepositoryFilter</filter-name>
4. <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
5. </filter>
6. <filter-mapping>
7. <filter-name>springSessionRepositoryFilter</filter-name>
8. <url-pattern>/\*</url-pattern>
9. </filter-mapping>

至此，就配置完成啦！

在Redis中查看效果：如果运行成功的话，可以在Redis中查看到key为spring:session:sessions的信息，如下所示：





## 方案对比

针对上述3中方案，以下仅是个人见解。

|  |  |
| --- | --- |
| 方案1：使用Tomcat内置的Session复制方案 SimpleTcpCluster | 优点：内置 缺点：只适合Tomcat小集群，不适合大集群，因为session复制是all to all的方式 |
| 方案2：使用第三方（个人）基于Tomcat实现的Session管理 tomcat-session-manager | 优点：已经实现对tomcat7的支持 缺点：第三方支持，支持力度不够，尤其是不能提供对Tomcat8的支持 |
| 方案3：使用Spring Session实现 基于redis存储实现 | 优点：不依赖于特定容器，官方支持 缺点：未发现 |

所以，我认为还是Spring Session来实现Session共享更加好用。