分布式系统的登录状态一致性解决方案

目录

任务目标

- 使用 SpringBoot 完成基本的 Redis 操作
- 登录状态一致性解决

知识点

1. Redis：
   - 性能极高 – Redis能读的速度是110000次/s,写的速度是81000次/s 。
   - 丰富的数据类型 – Redis支持二进制案例的 Strings, Lists, Hashes, Sets 及 Ordered Sets 数据类型操作。
   - 原子 – Redis的所有操作都是原子性的，意思就是要么成功执行要么失败完全不执行。单个操作是原子性的。多个操作也支持事务，即原子性，通过MULTI和EXEC指令包起来。
   - 丰富的特性 – Redis还支持 publish/subscribe, 通知, key 过期等等特性。

需掌握：

1. Redis安装：单机和集群
2. Redis的5种数据类型的使用：String / Hash / List / Set / Ordered Set
3. Redis事物：注意和我们之前接触的不同，单个 Redis 命令的执行是原子性的，但 Redis 没有在事务上增加任何维持原子性的机制，所以 Redis 事务的执行并不是原子性的。
4. Redis持久化：RDB和AOF
5. Redis配置：主从、性能测试等

参考学习：http://www.runoob.com/redis/redis-tutorial.html

2. Spring Data Redis（SDR）

SDR是Spring官方推出，可以算是Spring框架集成Redis操作的一个子框架，封装了Redis的很多命令，可以很方

便的使用Spring操作Redis数据库。

- Redis是用ANSI C写的一个基于内存的Key-Value数据库；
- Jedis是Redis官方推出的面向Java的Client，提供了很多接口和方法，可以让Java操作使用Redis，
- Spring Data Redis是对Jedis进行了封装，集成了Jedis的一些命令和方法，可以与Spring整合。

3. RedisTemplate

Spring封装了RedisTemplate对象来进行对Redis的各种操作，它支持所有的Redis原生的api。

- Redis的5种数据结构类型：

结构类型	结构存储的值	结构的读写能力
String	可以是字符串、整数或者浮点数	对整个字符串或者字符串的其中一部分执行操作；对象和浮点数执行自增(increment)或者自减(decrement)
List	一个链表，链表上的每个节点都包含了一个字符串	从链表的两端推入或者弹出元素；根据偏移量对链表进行修剪(trim)；读取单个或者多个元素；根据值来查找或者移除元素
Set	包含字符串的无序收集器(unorderedcollection)，并且被包含的每个字符串都是独一无二的、各不相同	添加、获取、移除单个元素；检查一个元素是否存在于某个集合中；计算交集、并集、差集；从集合里卖弄随机获取元素
Hash	包含键值对的无序散列表	添加、获取、移除单个键值对；获取所有键值对
Zset	字符串成员(member)与浮点数分值(score)之间的有序映射，元素的排列顺序由分值的大小决定	添加、获取、删除单个元素；根据分值范围(range)或者成员来获取元素

- RedisTemplate中定义了对5种数据结构操作

  redisTemplate.opsForValue();//操作字符串

  redisTemplate.opsForHash();//操作hash

  redisTemplate.opsForList();//操作list

  redisTemplate.opsForSet();//操作set

  redisTemplate.opsForZSet();//操作有序set

- StringRedisTemplate继承RedisTemplate。

  两者的数据是不共通的；也就是说StringRedisTemplate只能管理StringRedisTemplate里面的数据，RedisTemplate只能管理RedisTemplate中的数据。

SDR默认采用的序列化策略有两种，一种是String的序列化策略，一种是JDK的序列化策略。

StringRedisTemplate默认采用的是String的序列化策略，保存的key和value都是采用此策略序列化保存的。

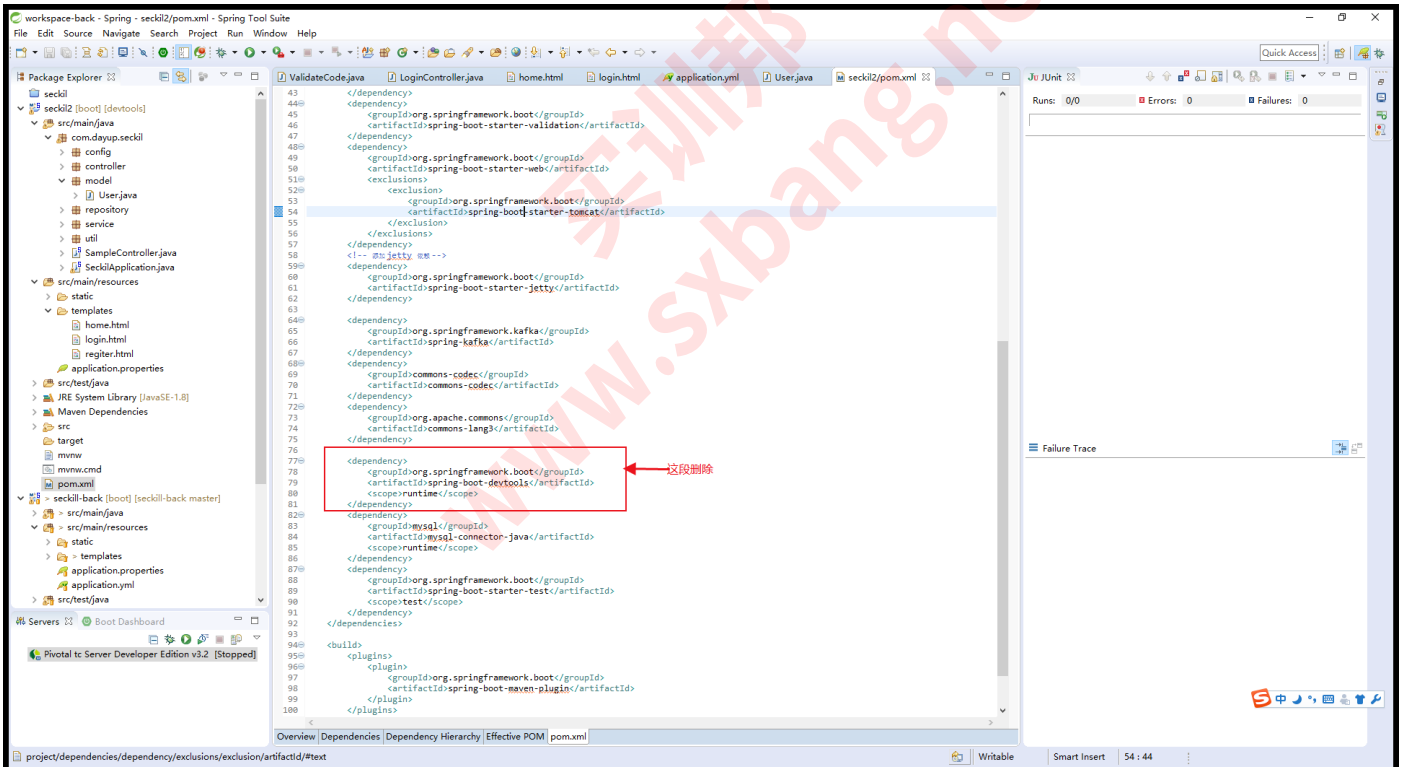
RedisTemplate默认采用的是JDK的序列化策略，保存的key和value都是采用此策略序列化保存的。

参考学习：https://www.cnblogs.com/EasonJim/p/7803067.html

热部署导致的问题：

java.lang.ClassCastException: com.dayup.seckil.model.User cannot be cast to com.dayup.seckil.model.User

解决办法：注释掉热部署的配置



4. 不用每次都写代码获取user对象：自定义参数解析器HandlerMethodArgumentResolver

https://blog.csdn.net/lovesomnus/article/details/73650336

为什么Controller方法上竟然可以放这么多的参数？而且都能得到想要的对象，比如HttpServletRequest或HttpServletResponse， 各种注解@RequestParam、@RequestHeader、@RequestBody、

@PathVariable、@ModelAttribute等。

这其实都是org.springframework.web.method.support.HandlerMethodArgumentResolver的功劳。

核心：

- supportsParameter

通过该方法我们如果需要对某个参数进行处理 只要此处返回true即可，

通过MethodParameter可以获取该方法参数上的一些信息， 如方法参数中的注解信息等

- resolveArgument

该方法就是对参数的解析，返回的Object会自动赋值到参数对象中

实现思路

1. Session 机制

首先 cookie 机制采用的是在客户端保持状态的方案，而 session 机制采用的是在服务器端保持状态的方案。服务器使用一种类似于散列表的结构来保存信息。

但程序需要为某个客户端的请求创建一个 session 的时候，服务器首先检查这个客户端的请求里是否包含了一个 session 标识 - 称为 session id,如果已经包含一个 session id 则说明以前已经为此客户创建过 session，服务器就按照 session id 把这个 session 检索出来使用(如果检索不到，可能会新建一个，这种情况可能出现在服务端已经删除了该用户对应的 session 对象，但用户人为地在请求的 URL 后面附加上一个 JSESSION 的参数)。

如果客户请求不包含 session id，则为此客户创建一个 session 并且生成一个与此 session 相关联的 session id，这个 session id 将在本次响应中返回给客户端保存。

保存 Session id 的几种方式：

A．保存 session id 的方式可以采用 cookie，这样在交互过程中浏览器可以自动的按照规则把这个标识发送给服务器。

B．由于 cookie 可以被人为的禁止，必须有其它的机制以便在 cookie 被禁止时仍然能够把 session id 传递回服务器，经常采用的一种技术叫做 URL 重写，就是把 session id 附加在 URL 路径的后面，附加的方式也有两种，一种是作为 URL 路径的附加信息，另一种是作为查询字符串附加在 URL 后面。网络在整个交互过程中始终保持状态，就必须在每个客户端可能请求的路径后面都包含这个 session id。

C．另一种技术叫做表单隐藏字段。就是服务器会自动修改表单，添加一个隐藏字段，以便在表单提交时能够把 session id 传递回服务器。

2. 你要理解 JSESSIONID 与 cookie 是什么关系，session 与 cookie 到底有什么关系：

简单来说，当第一次 request server 时，server 产生 JSESSIONID 对应的值 1，通过 http header set-cookie，传递给 browser，browser 检测到 http response header 里带 set-cookie，那么 browser 就会 create 一个 cookie，key=JSESSIONID，value=值 1，而后的每次请求，browser 都会把 cookie 里的键值对，放到 http request header 里，传递给 server。

当在 server 端调用 http.getSession()方法时，server 会先从 http request header 里解析出来 JSESSIONID 的值，再从一个 Map 容器里去找有没有 value，如果没有，就会产生一个 HttpSessioon 对象，放到这个 Map 容器里，同时设置一个最大生存时间。HttpSession 你也可以把它想象成是一个 Map，可以 getAttribute()，可以 setAttribute()。

2. 分布式 Session 如何实现。

最前端的负载均衡器，如 nginx，会把 request 原封不动的平均分配到集群中的一台机器，所喂原封不动，就是指的 request header 里的内容不变。

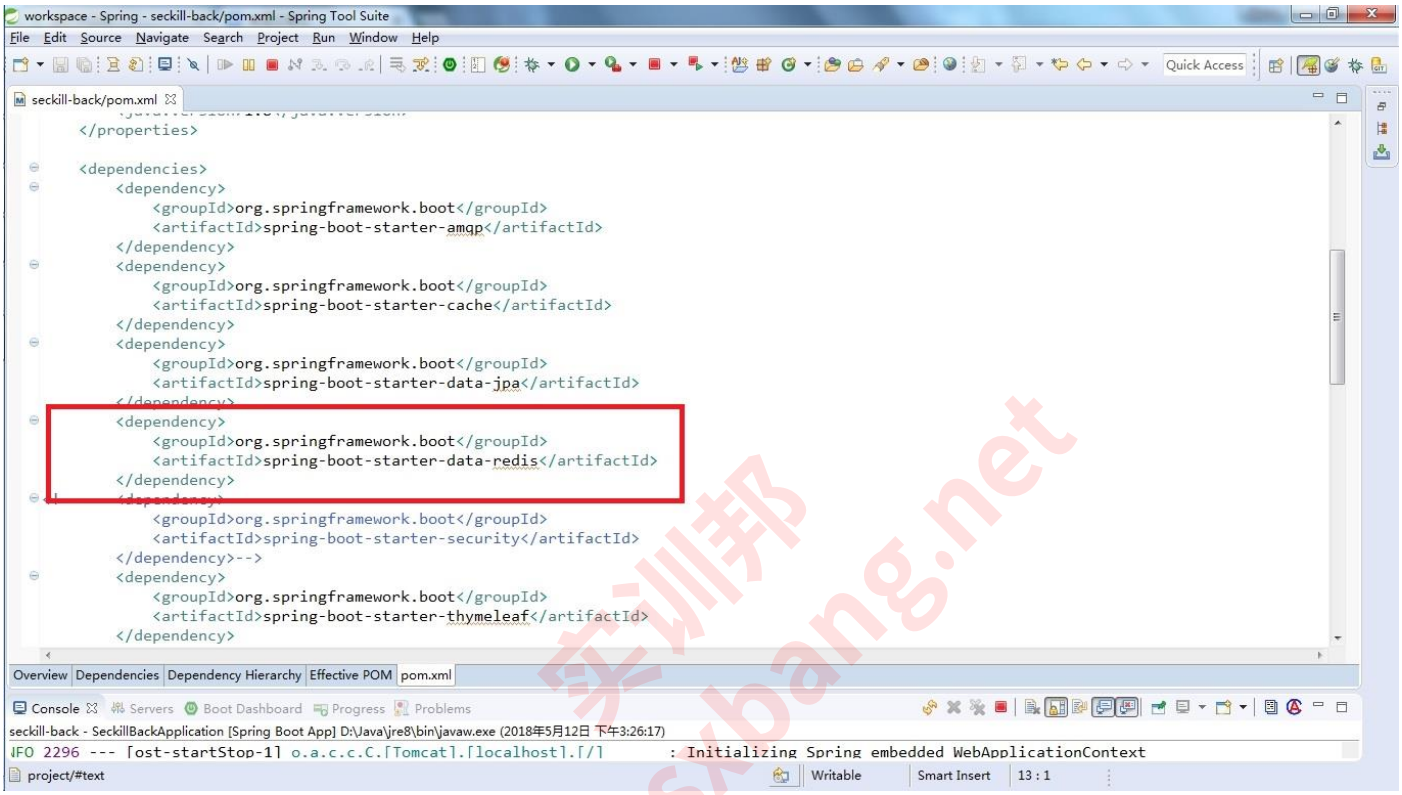
Tomcat 处理单机 session 其实就是 JSESSIONID 来获取 session 信息，所以要实现分布式 Session，只要得到 JSESSIONID 的值就可以了，剩下的操作无非就是根据 key 去 redis 或 memcache 中存取值，并缓存到本机内存中。

编码步骤

1. 在 SpringBoot 中调用 Redis 服务

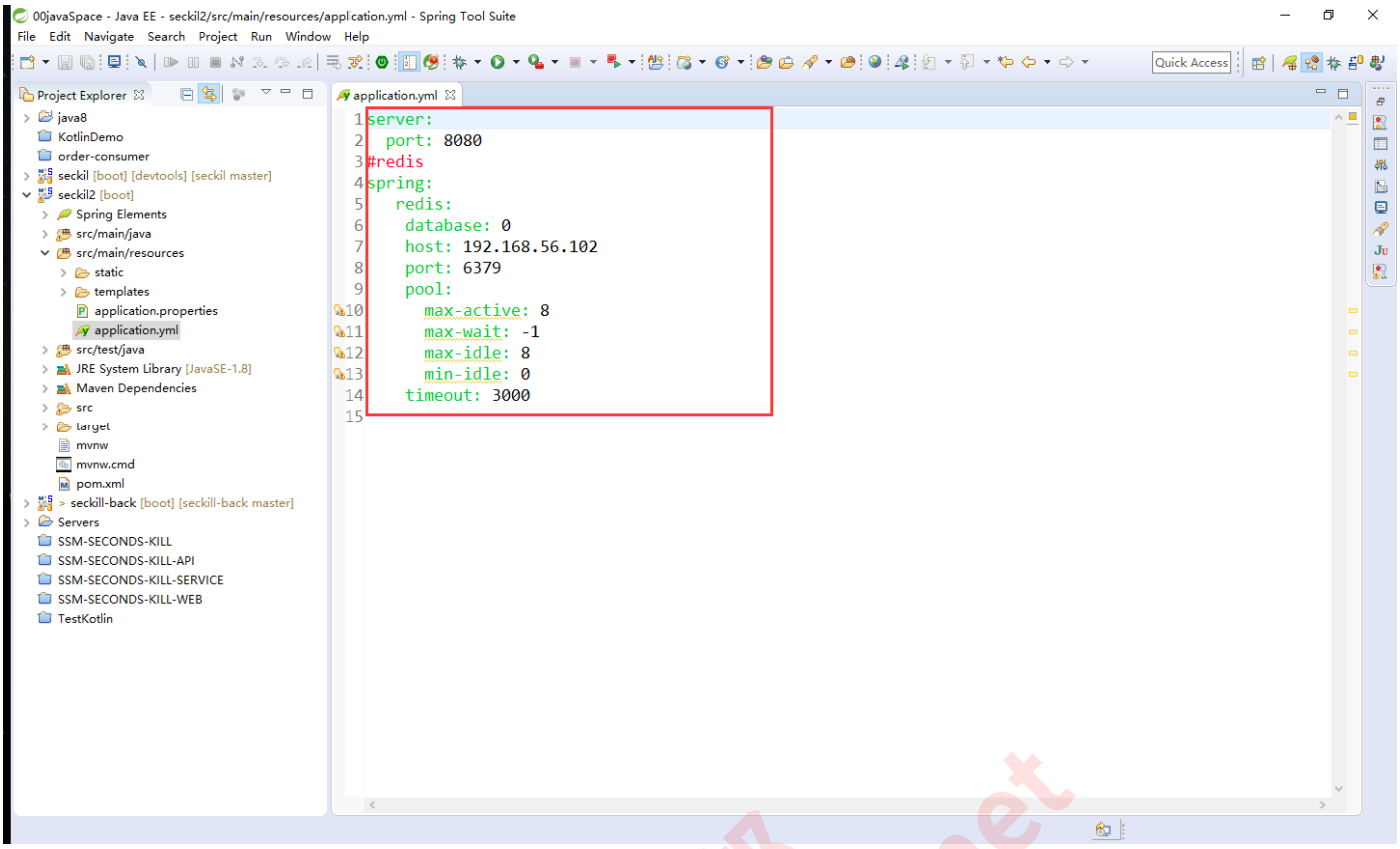
   Step1 引入 Redis 服务

   在 pom.xml 中添加依赖，引入 Redis

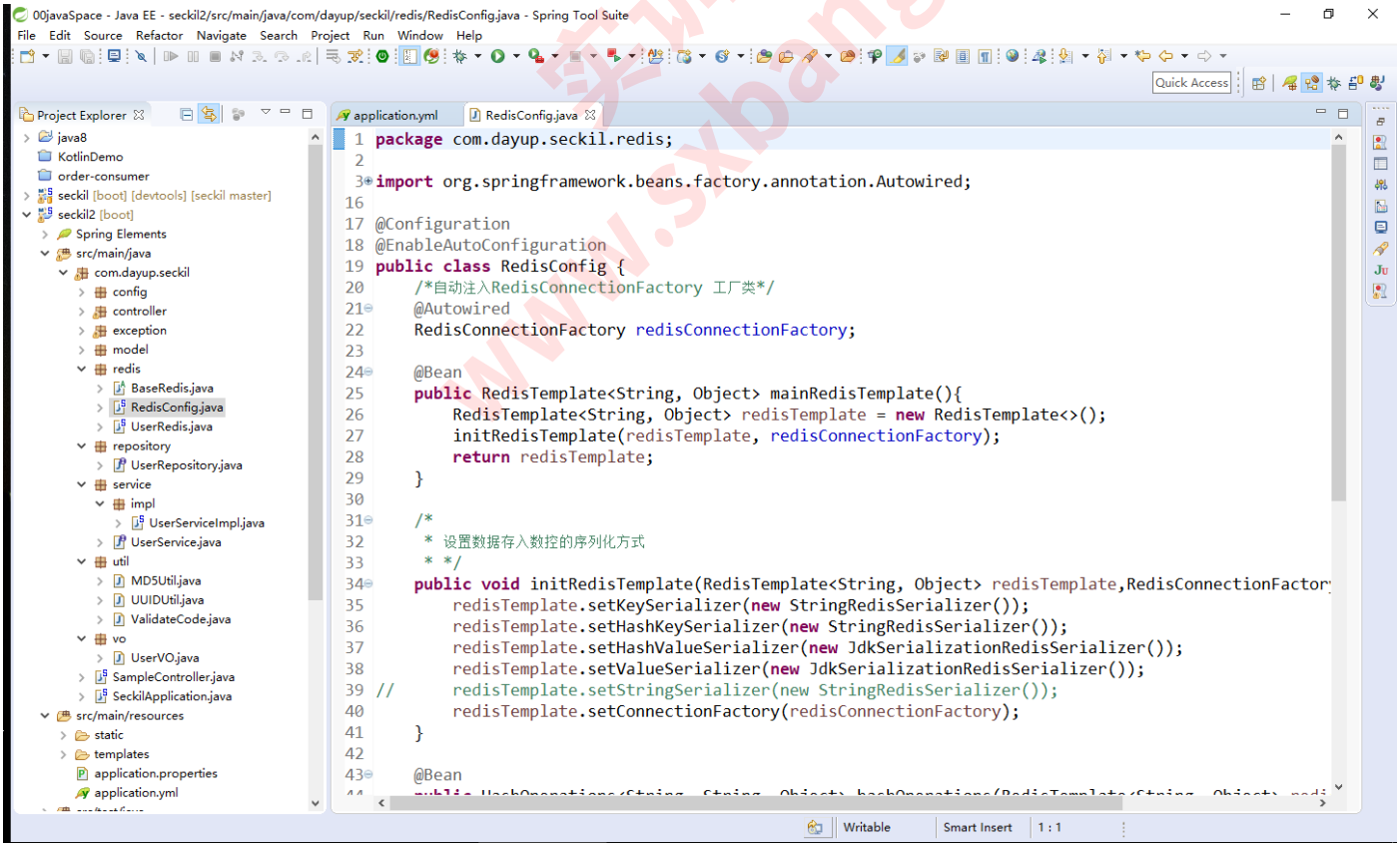


Step2.配置 Redis 连接池：yml 方式

```
#redis
spring:
   redis:
    database: 0
    host: 127.0.0.1
    port: 6379
    pool:
      max-active: 8
      max-wait: 1
      max-idle: 8
      min-idle: 0
  timeout: 100
```



Step3 配置 RedisConfig



代码展示：

```
package com.dayup.seckil.redis;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.autoconfigure.EnableAutoConfiguration;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.data.redis.connection.RedisConnectionFactory;

import org.springframework.data.redis.core.HashOperations;

import org.springframework.data.redis.core.ListOperations;

import org.springframework.data.redis.core.RedisTemplate;

import org.springframework.data.redis.core.SetOperations;

import org.springframework.data.redis.core.ValueOperations;

import org.springframework.data.redis.core.ZSetOperations;

import org.springframework.data.redis.serializer.JdkSerializationRedisSerializer;

import org.springframework.data.redis.serializer.StringRedisSerializer;


@Configuration

@EnableAutoConfiguration

public class RedisConfig {

    /*自动注入 RedisConnectionFactory  工厂类*/

    @Autowired

    RedisConnectionFactory redisConnectionFactory;


    @Bean
```

```
    public RedisTemplate<String, Object> mainRedisTemplate(){
        RedisTemplate<String, Object> redisTemplate = new RedisTemplate<>();
        initRedisTemplate(redisTemplate, redisConnectionFactory);
        return redisTemplate;
    }
    /*
     * 设置数据存入数控的序列化方式
     * */
    public void initRedisTemplate(RedisTemplate<String, Object> redisTemplate,RedisConnectionFactory
redisConnectionFactory){
        redisTemplate.setKeySerializer(new StringRedisSerializer());
        redisTemplate.setHashKeySerializer(new StringRedisSerializer());
        redisTemplate.setHashValueSerializer(new JdkSerializationRedisSerializer());
        redisTemplate.setValueSerializer(new JdkSerializationRedisSerializer());
//        redisTemplate.setStringSerializer(new StringRedisSerializer());
        redisTemplate.setConnectionFactory(redisConnectionFactory);
    }
    @Bean
    public HashOperations<String, String, Object> hashOperations(RedisTemplate<String, Object>
redisTemplate){
        return redisTemplate.opsForHash();
    }
    @Bean
```

```
    public ListOperations<String, Object> listperations(RedisTemplate<String, Object> redisTemplate){
        return redisTemplate.opsForList();
    }
    @Bean
    public SetOperations<String, Object> setOperations(RedisTemplate<String, Object> redisTemplate){
        return redisTemplate.opsForSet();
    }
    @Bean
    public ValueOperations<String, Object> valueOperations(RedisTemplate<String, Object> redisTemplate){
        return redisTemplate.opsForValue();
    }
    @Bean
    public ZSetOperations<String, Object> zSetOperations(RedisTemplate<String, Object> redisTemplate){
        return redisTemplate.opsForZSet();
    }
}
```

到这里，Redis 的配置就完成了。详细的源码，和其他的实例在下面的源码参考中。

Step4.编写公共类，定义 Redis 公共的抽象类来完成一些常规操作：

```
public abstract class IRedisDao<T> {

    @Autowired
    protected RedisTemplate<String, Object> redisTemplate;
    @Resource
    protected HashOperations<String, String, T> hashOperations;

    /**
     * 存入redis中的key
     *
     * @return
```

```
     */
    protected abstract String getRedisKey();

    /**
     * 添加
     *
     * @param key    key
     * @param doamin 对象
     * @param expire 过期时间(单位:秒),传入 -1 时表示不设置过期时间
     */
    public void put(String key, T doamin, long expire) {
        hashOperations.put(getRedisKey(), key, doamin);
        if (expire != -1) {
            redisTemplate.expire(getRedisKey(), expire, TimeUnit.SECONDS);
        }
    }

    /**
     * 删除
     *
     * @param key 传入key的名称
     */
    public void remove(String key) {
        hashOperations.delete(getRedisKey(), key);
    }

    /**
     * 查询
     *
     * @param key 查询的key
     * @return
     */
    public T get(String key) {
        return hashOperations.get(getRedisKey(), key);
    }

    /**
     * 获取当前redis库下所有对象
     *
     * @return
     */
    public List<T> getAll() {
        return hashOperations.values(getRedisKey());
    }

    /**
     * 查询查询当前redis库下所有key
```

```
     *
     * @return
     */
    public Set<String> getKeys() {
        return hashOperations.keys(getRedisKey());
    }

    /**
     * 判断key是否存在redis中
     *
     * @param key 传入key的名称
     * @return
     */
    public boolean isKeyExists(String key) {
        return hashOperations.hasKey(getRedisKey(), key);
    }

    /**
     * 查询当前key下缓存数量
     *
     * @return
     */
    public long count() {
        return hashOperations.size(getRedisKey());
    }

    /**
     * 清空redis
     */
    public void empty() {
        Set<String> set = hashOperations.keys(getRedisKey());
        set.stream().forEach(key -> hashOperations.delete(getRedisKey(), key));
    }
}
```

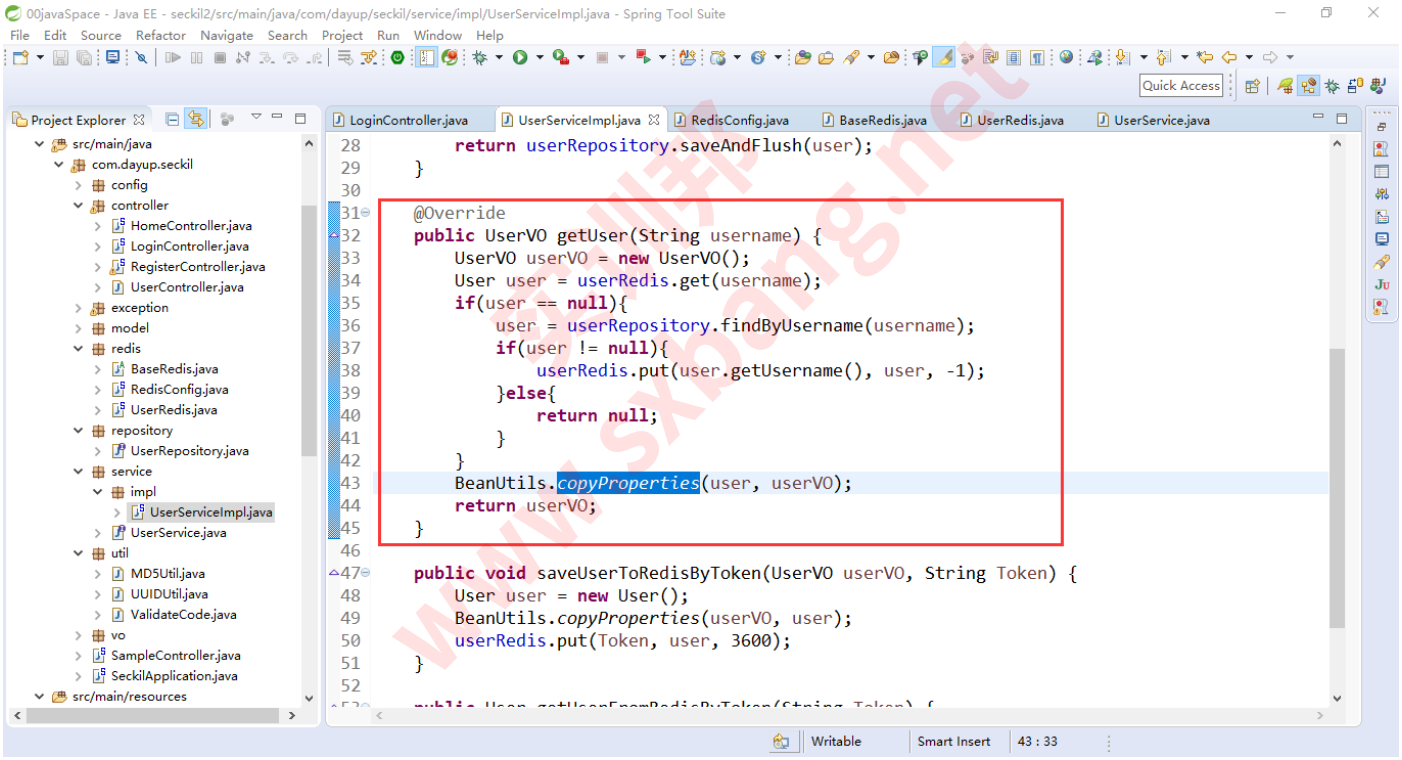
Step5.完成 User

```
package com.dayup.seckil.redis;

import java.util.concurrent.TimeUnit;
import org.springframework.stereotype.Repository;
import com.dayup.seckil.model.User;
import com.google.gson.Gson;

@Repository
public class UserRedis extends BaseRedis<User>{

    private static final String REDIS_KEY = "com.dayup.seckil.redis.UserRedis";
```

```
    @Override
    protected String getRedisKey() {
        return REDIS_KEY;
    }

    public void add(String key, Long time, User user){
        Gson gson = new Gson();
        redisTemplate.opsForValue().set(key, gson.toJson(user), time, TimeUnit.SECONDS);
    }
}
```

2. 完成基于 Redis 的登录功能

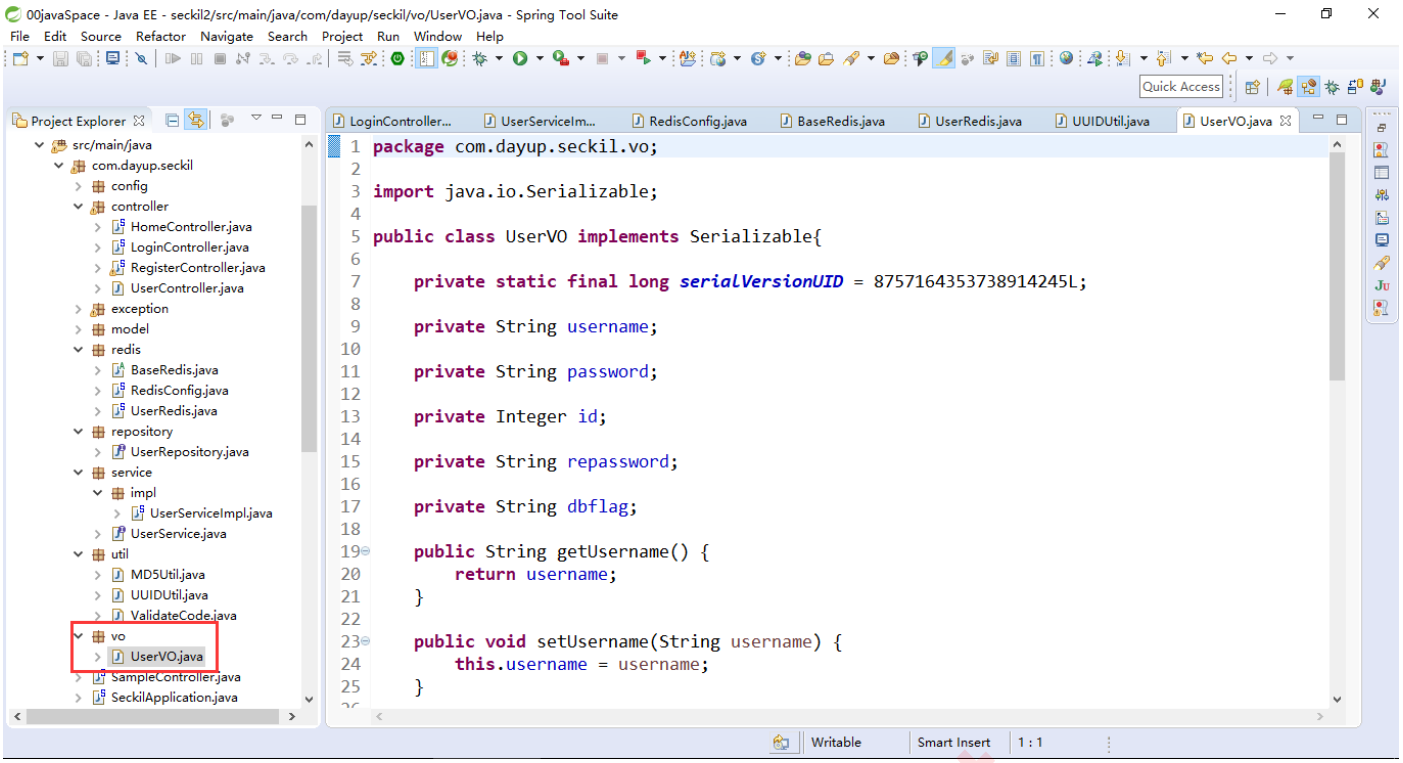
Step1.重写 getUser 方法:引入 Redis 缓存



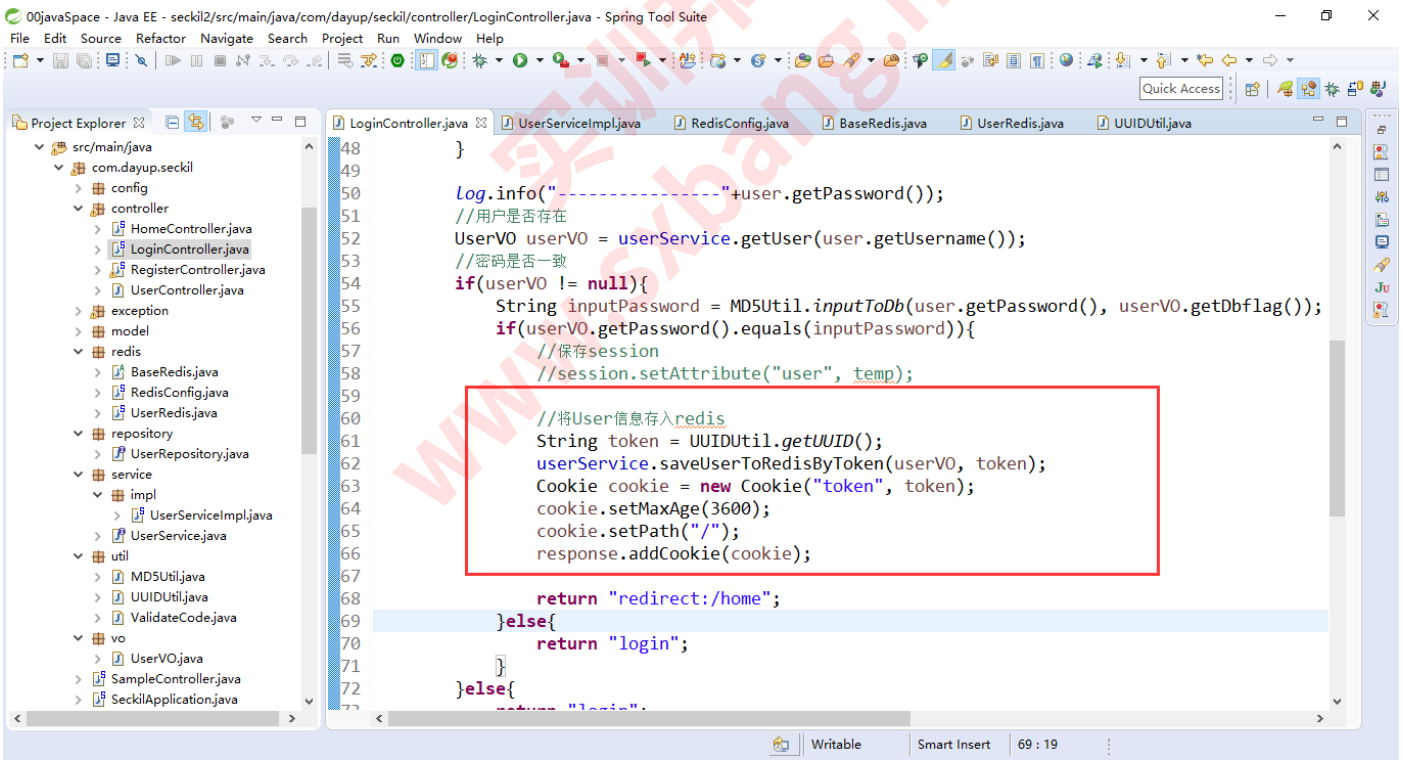
Step2.生成 token 作为 Redis key

```
package com.dayup.seckil.util;
import java.util.UUID;
public class UUIDUtil {
    public static String getUUID(){
        return UUID.randomUUID().toString().replaceAll("-", "");
    }
}
```

Step3.创建 UserVO 对象



Step4.将登陆成功的 User 对象存入 redis 中, key:token,value:user.同时将 token 写入到 cookie 中



3. 为了方便在后续的代码中直接使用 User 对象,我们采用 Spring 的自定义参数解析器

(HandlerMethodArgumentResolver)来完成参数的传值

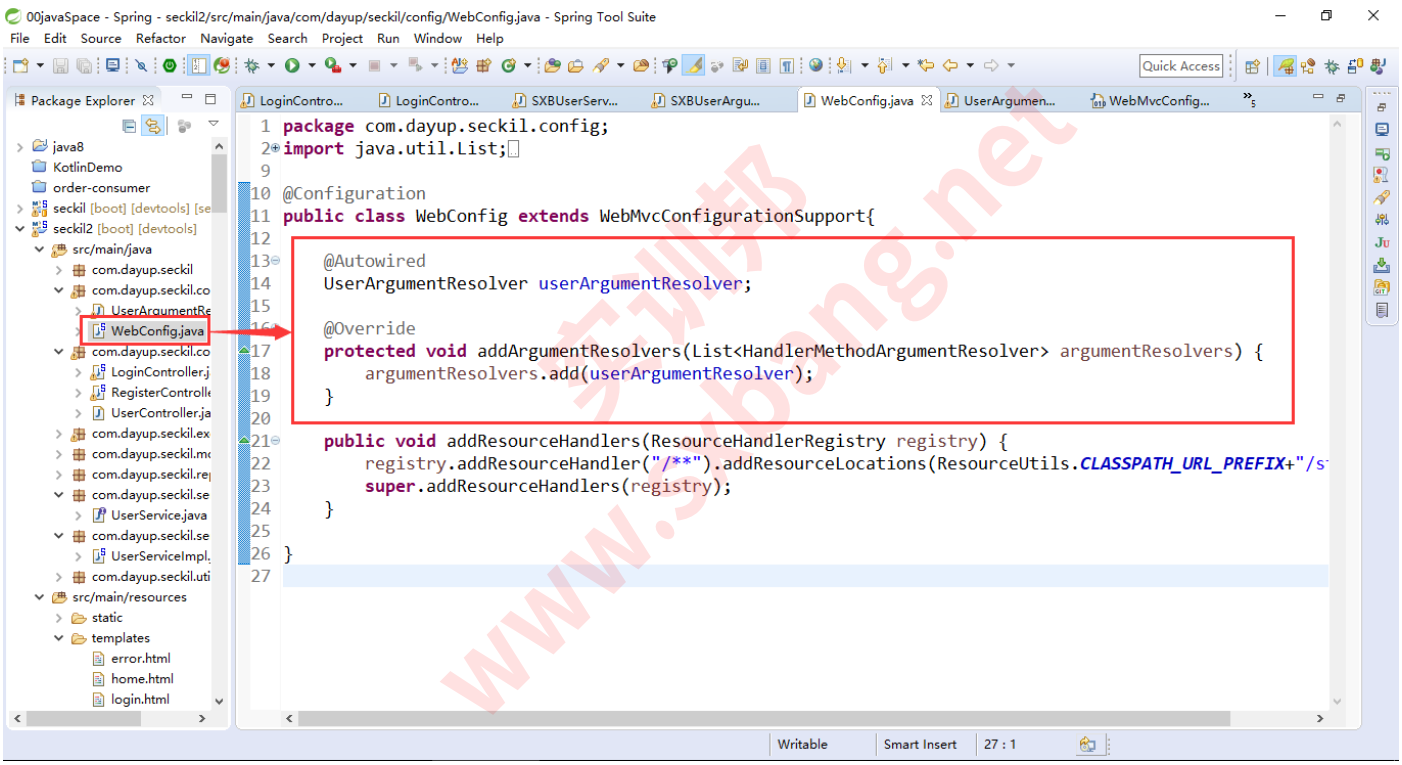
Step1.编写自定义参数解析器 UserArgumentResolver

```
package com.dayup.seckil.config;

import javax.servlet.http.Cookie;

import javax.servlet.http.HttpServletRequest;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.core.MethodParameter;

import org.springframework.stereotype.Service;

import org.springframework.web.bind.support.WebDataBinderFactory;

import org.springframework.web.context.request.NativeWebRequest;

import org.springframework.web.method.support.HandlerMethodArgumentResolver;

import org.springframework.web.method.support.ModelAndViewContainer;

import com.dayup.seckil.model.User;

import com.dayup.seckil.service.UserService;


@Service

public class UserArgumentResolver implements HandlerMethodArgumentResolver{


    @Autowired

    public UserService userService;


    public String getParameterCokies(HttpServletRequest request,String tokenName){

        Cookie[] cookies = request.getCookies();
```

```
        for(Cookie ck : cookies){
            if(ck.getName().equals(tokenName)){
                return ck.getValue();
            }
        }
        return null;
    }

    @Override
    public Object resolveArgument(MethodParameter methodParameter, ModelAndViewContainer modelAndViewContainer,
            NativeWebRequest nativeWebRequest, WebDataBinderFactory webDataBinderFactory) throws Exception {
        HttpServletRequest request = nativeWebRequest.getNativeRequest(HttpServletRequest.class);
        String requestParameter_token = request.getParameter("token");
        String Cokies_token = getParameterCokies(request,"token");
        if(requestParameter_token == null && Cokies_token == null){
            return null;
        }
        return userService.getUserFromRedisByToken((requestParameter_token != null ? requestParameter_token : Cokies_token));
    }
```

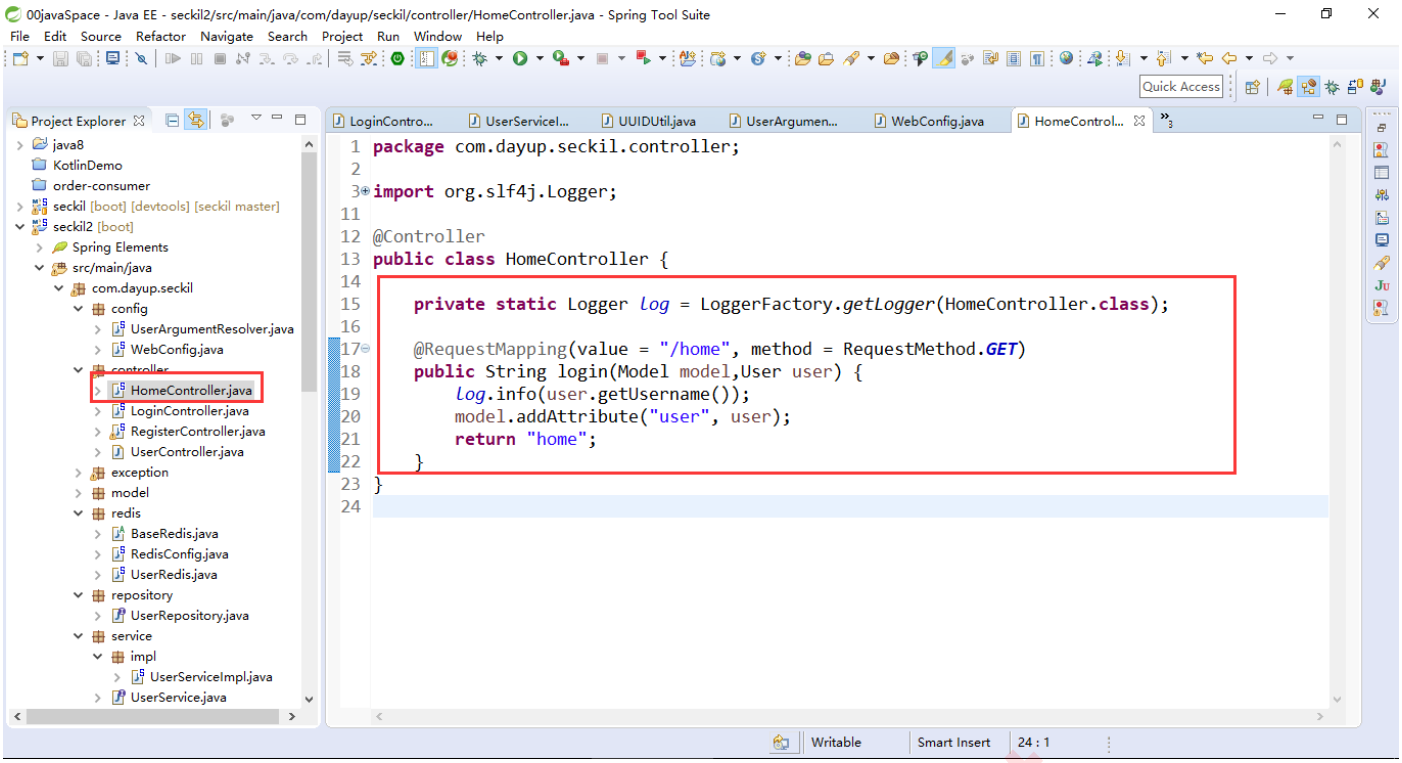
```
    @Override
    public boolean supportsParameter(MethodParameter methodParameter) {
        Class<?> p_class = methodParameter.getParameterType();
        return p_class == User.class;
    }

}
```

Step2.注册自定义参数解析器 UserArgumentResolver




```
    @Autowired
    UserArgumentResolver userArgumentResolver;

    @Override
    protected void addArgumentResolvers(List<HandlerMethodArgumentResolver> argumentResolvers){
        argumentResolvers.add(userArgumentResolver);
    }
```

Step3.使用



源码参考

请从我们提供的码云上获取。

回马枪总结

NA