1. LoginAction.java

package com.glitter.spring.boot.web.action;  
  
@RestController  
@RequestMapping("/login")  
public class LoginAction extends BaseAction{  
  
 private static final Logger *logger* = LoggerFactory.*getLogger*(LoginAction.class);  
  
 @Autowired  
 private IUserInfoService userInfoService;  
  
 @Autowired  
 private IRsaService rsaService;  
  
 */\*\*  
 \* 获取公钥  
 \** ***@return*** *\*/* @RequestMapping(value = "getPublicKey", method = RequestMethod.*GET*)  
 public ResponseResult<String> getPublicKey() {  
 String publicKey = rsaService.getPublicKey();  
 return ResponseResult.*success*(publicKey);  
 }  
  
 */\*\*  
 \* 获取图形验证码  
 \** ***@return*** *\** ***@throws*** *IOException  
 \*/* @RequestMapping(value = "/getLoginGraphCaptcha", method = RequestMethod.*GET*)  
 public ResponseResult<String> getLoginGraphCaptcha() throws IOException {  
 // 1.设置验证码属性  
 int width = 110;  
 int height = 35;  
 CaptchaUtils captchaUtils = CaptchaUtils.*getInstance*();  
 captchaUtils.set(width, height);  
  
 // 2.生成验证码值并存入Session  
 String graphCaptcha = captchaUtils.generateCaptchaCode();  
 sessionHandler.getSession().setAttribute("loginGraphCaptcha",graphCaptcha);  
   
 // 3.将验证码图片输入到客户端  
 ByteArrayOutputStream out = new ByteArrayOutputStream();  
 ImageIO.*write*(captchaUtils.generateCaptchaImg(graphCaptcha), "jpg", out);  
 return ResponseResult.*success*(new String(Base64.*encodeBase64*(out.toByteArray())));  
 }  
  
 */\*\*  
 \* 登陆  
 \** ***@return*** *\*/* @RequestMapping(value = "login", method = RequestMethod.*POST*)  
 public ResponseResult<String> login(@RequestBody LoginInfo paramBean) throws Exception {  
 // 1.参数基本验证  
 if (null == paramBean) { throw new BusinessException("-1","请输入账号和密码"); }  
 if (null == paramBean.getAccount()) { throw new BusinessException("-1","请输入账号"); }  
 if (null == paramBean.getPassword()) { throw new BusinessException("-1","请输入密码"); }  
 if (null == paramBean.getGraphCaptcha()) { throw new BusinessException("-1","请输入图形验证码"); }  
  
 // 2.验证图形验证码  
 // 如果客户端凭证jsessionId值为空或非法,则会得到一个全新的session对象,其中的验证码值自然是空的。  
 // 如果客户端凭证jsessionId值在服务器存在,则会取出其中的验证码值进行比较(当然你如果提前没有往session中放入验证码值的话,验证码的值自然也会是空的)  
 String loginGraphCaptcha = (String)sessionHandler.getSession().getAttribute(GlitterConstants.*SESSION\_LOGIN\_GRAPHCAPTCHA*);  
 if(!paramBean.getGraphCaptcha().equals(loginGraphCaptcha)){  
 sessionHandler.getSession().removeAttribute(GlitterConstants.*SESSION\_LOGIN\_GRAPHCAPTCHA*);  
 throw new BusinessException("-2","图形验证码输入错误");  
 }  
  
 // 3.密码解密  
 String pwd = null;  
 try {  
 pwd = rsaService.decrypt(paramBean.getPassword());  
 } catch (Exception e) {  
 *logger*.error("密码解密失败");  
 throw new BusinessException("-1","密码错误");  
 }  
  
 // 4.验证用户名密码  
 UserInfo userInfo = userInfoService.getByAccount(paramBean.getAccount());  
 if(null == userInfo){ throw new BusinessException("-1","用户名或密码错误"); }  
 if (!pwd.equals(userInfo.getPassword())) { throw new BusinessException("-1", "用户名或密码错误"); }  
  
 // 5.记录用户登录会话信息  
 sessionHandler.getSession().setAttribute(GlitterConstants.*SESSION\_USER*, userInfo);  
 // 6.限制多端同时登陆逻辑  
 commonCache.add(cacheKeyManager.getLimitMultiLoginKey(String.*valueOf*(userInfo.getId())), sessionHandler.getSession().getId(), cacheKeyManager.getLimitMultiLoginKeyExpireTime());  
  
 return ResponseResult.*success*(null);  
 }  
  
 */\*\*  
 \* 退出  
 \** ***@return*** *\** ***@throws*** *Exception  
 \*/* @RequestMapping(value = "logout", method = RequestMethod.*POST*)  
 public ResponseResult logout() throws Exception {  
 // 1.获取session会话对象  
 ISession session = sessionHandler.getSession();  
  
 // 2.如果session会话对象中不存在用户信息,说明已经退出登录,返回成功即可(容错处理,这种情况不可能发生).  
 UserInfo userInfo;  
 if (null == (userInfo = (UserInfo)session.getAttribute(GlitterConstants.*SESSION\_USER*))){  
 return ResponseResult.*success*(null);  
 }  
  
 // 3.注销session会话  
 session.invalidate();  
  
 // 4.注销限制多端同时登陆的相关代码逻辑  
 commonCache.del(cacheKeyManager.getLimitMultiLoginKey(String.*valueOf*(userInfo.getId())));  
 return ResponseResult.*success*(null);  
 }  
  
}

2. UserInfoAction.java

package com.glitter.spring.boot.web.action;  
  
import com.glitter.spring.boot.bean.UserInfo;  
import com.glitter.spring.boot.common.ResponseResult;  
import com.glitter.spring.boot.constant.CoreConstants;  
import com.glitter.spring.boot.constant.GlitterConstants;  
import com.glitter.spring.boot.exception.BusinessException;  
import com.glitter.spring.boot.service.ISessionHandler;  
import com.glitter.spring.boot.service.IUserInfoService;  
import com.pagehelper.plugin.PageInfo;  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.web.bind.annotation.\*;  
  
@RestController  
@RequestMapping("/userInfo")  
public class UserInfoAction{  
  
 private static final Logger *logger* = LoggerFactory.*getLogger*(UserInfoAction.class);  
  
 @Autowired  
 private IUserInfoService userInfoService;  
  
 @Autowired  
 private ISessionHandler sessionHandler;  
  
 @RequestMapping(value = "getUserInfoById", method = RequestMethod.*GET*)  
 public ResponseResult<UserInfo> getUserInfoById(@RequestParam Long id) {  
 UserInfo result = null;  
 if(null == id) {  
 return ResponseResult.*success*(result);  
 }  
 result = userInfoService.getUserInfoById(id);  
 return ResponseResult.*success*(result);  
 }  
  
 @RequestMapping(value = "getLoginUserInfo", method = RequestMethod.*GET*)  
 public ResponseResult<UserInfo> getUserInfoById() {  
 UserInfo result = (UserInfo) sessionHandler.getSession().getAttribute(GlitterConstants.*SESSION\_USER*);  
 return ResponseResult.*success*(result);  
 }  
  
}

3. GlitterConstants.java

package com.glitter.spring.boot.constant;  
  
  
public class GlitterConstants {  
  
 public static final String *SESSION\_LOGIN\_GRAPHCAPTCHA* = "loginGraphCaptcha";  
  
 public static final String *SESSION\_USER* = "userInfo";  
  
 public static final String *JSESSIONID* = "jsessionId";  
  
}

4. ISessionHandler.java

package com.glitter.spring.boot.service;  
  
public interface ISessionHandler {  
  
 ISession getSession();  
  
}

5. SessionHandler.java，核心关键。

package com.glitter.spring.boot.service.impl;  
  
@Service  
public class SessionHandler implements ISessionHandler {  
  
 private static final Logger *logger* = LoggerFactory.*getLogger*(UserInfoAction.class);  
  
 @Autowired  
 ICommonCache commonCache;  
  
 @Autowired  
 ICacheKeyManager cacheKeyManager;  
  
  
 */\*\*  
 \* 整个项目中获取session对象的唯一方式  
 \* 并且方法内部也是整个项目中创建session的唯一地方  
  
 \* 每一个线程中多处调用该方法得到的session对象都是同一个,  
 \* 除非某处在调用该方法时session在redis中过期或认为销毁,则该方法会返回一个全新的session对象,这是完成正常的逻辑.  
 \** ***@return*** *\*/* @Override  
 public ISession getSession() {  
 ISession session = SessionContext.*get*();  
  
 // 如果当前线程中的session不为空,并且在缓存中依然存在,则表明session未过期,可以继续直接使用该线程中的session  
 if(null != session && commonCache.isExists(cacheKeyManager.getSessionKey(session.getId()))){  
 commonCache.renewal(cacheKeyManager.getSessionKey(session.getId()), cacheKeyManager.getSessionKeyExpireTime());  
 SpringContextUtil.*getBean*(SessionRenewalPublisher.class).publishEvent(session);  
 return session;  
 }  
  
 // 某线程中第一次调用该方法时,要么执行此逻辑  
 if (StringUtils.*isBlank*(JsessionIdCookieContext.*get*()) || null == (session = commonCache.get(cacheKeyManager.getSessionKey(JsessionIdCookieContext.*get*()))) ) {  
 session = new Session(UUID.*randomUUID*().toString());  
  
 Cookie cookie = new Cookie(GlitterConstants.*JSESSIONID*, session.getId());  
 cookie.setPath("/");  
 CookieUtils.*updateCookie*(ResponseContext.*get*(), cookie);  
  
 SessionContext.*set*(session);  
 return session;  
 }  
  
 // 某线程中第一次调用该方法时,要么执行此逻辑  
 commonCache.renewal(cacheKeyManager.getSessionKey(session.getId()), cacheKeyManager.getSessionKeyExpireTime());  
 SpringContextUtil.*getBean*(SessionRenewalPublisher.class).publishEvent(session);  
 SessionContext.*set*(session);  
 return session;  
 }  
}

6.Isession.java，核心。

package com.glitter.spring.boot.service;  
  
import java.util.List;  
  
public interface ISession {  
  
 Long getCreationTime();  
  
 String getId();  
  
 Object getAttribute(String var1);  
  
 List<String> getAttributeNames();  
  
 void setAttribute(String var1, Object var2);  
  
 void removeAttribute(String var1);  
  
 void invalidate();  
  
}

7.Session.java，核心。

package com.glitter.spring.boot.service.impl;  
  
public class Session implements ISession,Serializable {  
  
 private static final Logger *logger* = LoggerFactory.*getLogger*(UserInfoAction.class);  
  
 private String id;  
 private ConcurrentMap<String, Object> attributes;  
 private List<String> attributeNames;  
 private Long creationTime;  
  
 protected Session(){  
  
 }  
  
 protected Session(String id){  
 Long now = System.*currentTimeMillis*();  
 this.id = id;  
 this.attributes = new ConcurrentHashMap();  
 this.creationTime = now;  
 SpringContextUtil.*getBean*(SessionCreatePublisher.class).publishEvent(this);  
 }  
  
 public void setId(String id) {  
 this.id = id;  
 }  
  
 public ConcurrentMap<String, Object> getAttributes() {  
 return attributes;  
 }  
  
 public void setAttributes(ConcurrentMap<String, Object> attributes) {  
 this.attributes = attributes;  
 }  
  
 public void setCreationTime(Long creationTime) {  
 this.creationTime = creationTime;  
 }  
  
 @Override  
 public Long getCreationTime() {  
 return this.creationTime;  
 }  
  
 @Override  
 public String getId() {  
 return this.id;  
 }  
  
 @Override  
 public Object getAttribute(String key) {  
 return this.attributes.get(key);  
 }  
  
 @Override  
 public List<String> getAttributeNames() {  
 List<String> result = null;  
 if (null == this.getAttributes() || this.getAttributes().size() <= 0) { return result; }  
 result = new ArrayList<>();  
 for (Map.Entry<String, Object> entry : this.getAttributes().entrySet()) {  
 result.add(entry.getKey());  
 }  
 return result;  
 }  
  
 @Override  
 public void setAttribute(String key, Object value) {  
 this.attributes.put(key, value);  
 SpringContextUtil.*getBean*(SessionUpdatePublisher.class).publishEvent(this);  
 }  
  
 @Override  
 public void removeAttribute(String key) {  
 this.attributes.remove(key);  
 SpringContextUtil.*getBean*(SessionUpdatePublisher.class).publishEvent(this);  
 }  
  
 @Override  
 public void invalidate() {  
 ISession session = SessionContext.*get*();  
 if(null != session ){  
 if(this == session){  
 SessionContext.*remove*();  
 }  
 }  
  
 SpringContextUtil.*getBean*(SessionDeletePublisher.class).publishEvent(this);  
}

8. 观察者模式事件监听器，事件发布和事件类这里不再描述。

@Component  
public class SessionCreateListener implements ApplicationListener<SessionCreateEvent> {  
  
 private static final Logger *logger* = LoggerFactory.*getLogger*(UserInfoAction.class);  
  
 @Autowired  
 private ICommonCache commonCache;  
  
 @Autowired  
 private ICacheKeyManager cacheKeyManager;  
  
 @Override  
 public void onApplicationEvent(SessionCreateEvent applicationEvent) {  
 *logger*.info("SessionCreateListener.onApplicationEvent,输入参数:{}", applicationEvent);  
 ISession session = applicationEvent.getContent();  
 *logger*.info("SessionCreateListener.onApplicationEvent,sessionId:{}", session.getId());  
 commonCache.add(cacheKeyManager.getSessionKey(session.getId()),session,cacheKeyManager.getSessionKeyExpireTime());  
 return;  
 }  
  
}

@Component  
public class SessionUpdateListener implements ApplicationListener<SessionUpdateEvent> {  
  
 private static final Logger *logger* = LoggerFactory.*getLogger*(UserInfoAction.class);  
  
 @Autowired  
 private ICommonCache commonCache;  
  
 @Autowired  
 private ICacheKeyManager cacheKeyManager;  
  
 @Override  
 public void onApplicationEvent(SessionUpdateEvent applicationEvent) {  
 *logger*.info("SessionUpdateListener.onApplicationEvent,输入参数:{}", applicationEvent);  
 ISession session = applicationEvent.getContent();  
 *logger*.info("SessionUpdateListener.onApplicationEvent,sessionId:{}", session.getId());  
 commonCache.add(cacheKeyManager.getSessionKey(session.getId()),session,cacheKeyManager.getSessionKeyExpireTime());  
 return;  
 }  
   
}

@Component  
public class SessionDeleteListener implements ApplicationListener<SessionDeleteEvent> {  
  
 private static final Logger *logger* = LoggerFactory.*getLogger*(UserInfoAction.class);  
  
 @Autowired  
 private ICommonCache commonCache;  
  
 @Autowired  
 private ICacheKeyManager cacheKeyManager;  
  
 @Override  
 public void onApplicationEvent(SessionDeleteEvent applicationEvent) {  
 *logger*.info("SessionDeleteListener.onApplicationEvent,输入参数:{}", applicationEvent);  
 ISession session = applicationEvent.getContent();  
 *logger*.info("SessionDeleteListener.onApplicationEvent,sessionId:{}", session.getId());  
 commonCache.del(cacheKeyManager.getSessionKey(session.getId()));  
 return;  
 }  
  
}

@Component  
public class SessionRenewalListener implements ApplicationListener<SessionRenewalEvent> {  
  
 private static final Logger *logger* = LoggerFactory.*getLogger*(UserInfoAction.class);  
  
 @Autowired  
 private ICommonCache commonCache;  
  
 @Autowired  
 private ICacheKeyManager cacheKeyManager;  
  
 @Override  
 public void onApplicationEvent(SessionRenewalEvent applicationEvent) {  
 *logger*.info("SessionRenewalListener.onApplicationEvent,输入参数:{}", applicationEvent);  
 ISession session = applicationEvent.getContent();  
 *logger*.info("SessionRenewalListener.onApplicationEvent,sessionId:{}", session.getId());  
 UserInfo userInfo = null;  
 if(null == (userInfo = (UserInfo)session.getAttribute(GlitterConstants.*SESSION\_USER*))){ return; }  
  
 // 对限制单端登陆进行续期  
 if (commonCache.isExists(cacheKeyManager.getLimitMultiLoginKey(String.*valueOf*(userInfo.getId())))) {  
 commonCache.renewal(cacheKeyManager.getLimitMultiLoginKey(String.*valueOf*(userInfo.getId())), cacheKeyManager.getLimitMultiLoginKeyExpireTime());  
 }  
 return;  
 }  
  
}

观察者模式主要由三部分组成：事件，事件发布者，事件监听者。

理解起来主要有两种：

一.本身代码做了自己逻辑该做的事情，然后发布一个对应的事件，说我做了这件事情，凡是监听该事件的监听者，你们监听到后都可以执行你们自己的逻辑了。

二.本身代码做了自己逻辑该做的事情，然后发布一个我想发布的事件，这个事件可能跟我之前的代码逻辑没太大关系，但是我就是发布一个我想发布的事件，然后凡是监听该事件的监听者，你们监听到后都可以执行你们自己的逻辑了。说白了，就是利用监听者模式，

通过事件发布的方式，继续完成代码后面需要完成的事情。

个人认为，上面两种情况都可以使用观察者模式，没什么对错，但是个人强烈建议在一个项目中，观察者模式最好都是统一理解的，要么都是第一种理解，要么都是第二种理解，否则对于后续的开发者，甚至是当前的开发者本人都会产生逻辑上的混乱，为什么要给自己找麻烦呢？

并且个人强烈推荐使用第一种理解方式，不要使用第二种方式。因为第一种是你只要通过事件发布告诉外界你做了什么就好了，然后外部监听到后想做什么做什么就好，自然都会做好后续的事情。而第二种是你在发号施令，可能这件事情你自己什么都没做呢，你通过发号施令的方式

发布了一个事件，那么监听者就比较困惑了，我是应该做发布者让我做的这件事情呢，还是发布者已经把这件事情做完了，我继续做后面的事情呢？还是那句话，不给自己找麻烦，不给自己找歧义，一个大型项目的各种逻辑有多复杂就不说了，凡事要理清楚，都不找歧义，自然项目也就清晰了。

简而言之：我代码已经做了什么事情，我就发布什么事件，让监听者根据自己的意愿去做自己相应的响应。而不是我发号施令一个事件，让监听者去针对这个事件去做相应的实现，即把事情的实现推后到监听者去做，这失去了观察者模式的本意，虽然代码运行这样写。

9. ICommonCache.java

package com.glitter.spring.boot.persistence.cache;  
  
*/\*\*  
 \* 简单通用缓存  
 \* 如果有特殊处理的缓存,可以单独写  
 \*/*public interface ICommonCache {  
  
 */\*\*  
 \* 新增  
 \** ***@param*** *o  
 \*/* void add(String key, Object o);  
  
 */\*\*  
 \* 新增  
 \** ***@param*** *o  
 \** ***@param*** *expirtTime  
 \*/* void add(String key, Object o, Integer expirtTime);  
  
 */\*\*  
 \* 删除  
 \** ***@param*** *key  
 \*/* void del(String key);  
  
 */\*\*  
 \* 续期  
 \** ***@param*** *key  
 \*/* void renewal(String key, Integer expirtTime);  
  
 */\*\*  
 \* 获取  
 \** ***@param*** *key  
 \** ***@param*** <*T*>  
 *\** ***@return*** *\*/* <T> T get(String key);  
  
 */\*\*  
 \* 验证键是否存在  
 \** ***@param*** *key  
 \** ***@return*** *\*/* boolean isExists(String key);  
  
}

10. ICacheKeyManager.java

package com.glitter.spring.boot.persistence.cache;  
  
import com.glitter.spring.boot.SpringBootStartApplication;  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
  
*/\*\*  
 \* 缓存key的统一管理中心  
 \* 所有业务相关的key必须全部在这里生成,保证项目中所有地方使用到的key值的统一性  
 \* 方法命名规则:  
 \* 以值的所属对象名称+key,如getSessionKey,  
 \* 如果值是字符串类型,则可以根据其业务含义命名,如getSessionKeyKey,即获取SessionKey的key*  
 *\*/*public interface ICacheKeyManager {  
  
 Logger *logger* = LoggerFactory.*getLogger*(SpringBootStartApplication.class);  
  
 String getSessionKey(String uuid);  
  
 Integer getSessionKeyExpireTime();  
  
 String getLimitMultiLoginKey(String userId);  
  
 Integer getLimitMultiLoginKeyExpireTime();  
  
}

11. CommonCacheImpl.java

package com.glitter.spring.boot.persistence.cache.redis;  
  
import com.glitter.spring.boot.persistence.cache.ICommonCache;  
import org.apache.commons.lang3.StringUtils;  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.data.redis.core.RedisTemplate;  
import org.springframework.stereotype.Service;  
  
import java.util.concurrent.TimeUnit;  
  
@Service  
public class CommonCacheImpl implements ICommonCache {  
  
 private static final Logger *logger* = LoggerFactory.*getLogger*(CommonCacheImpl.class);  
  
 @Autowired  
 private RedisTemplate redisTemplate;  
  
 */\*\*  
 \* 新增  
 \** ***@param*** *key  
 \** ***@param*** *o  
 \*/* @Override  
 public void add(String key, Object o) {  
 this.add(key, o, null);  
 }  
  
 */\*\*  
 \* 新增  
 \** ***@param*** *key  
 \** ***@param*** *o  
 \** ***@param*** *expirtTime  
 \*/* @Override  
 public void add(String key, Object o, Integer expirtTime) {  
 if (expirtTime == null) {  
 redisTemplate.opsForValue().set(key, o, expirtTime);  
 }else {  
 redisTemplate.opsForValue().set(key, o, expirtTime, TimeUnit.*SECONDS*);  
 }  
 }  
  
 */\*\*  
 \* 删除  
 \*  
 \** ***@param*** *key  
 \*/* @Override  
 public void del(String key) {  
 if(StringUtils.*isBlank*(key)) { return; }  
 redisTemplate.delete(key);  
 }  
  
 */\*\*  
 \* 续期  
 \*  
 \** ***@param*** *key  
 \*/* @Override  
 public void renewal(String key, Integer expirtTime) {  
 redisTemplate.expire(key, expirtTime, TimeUnit.*SECONDS*);  
 }  
  
 @Override  
 public <T> T get(String key) {  
 T o = (T)redisTemplate.opsForValue().get(key);  
 return o;  
 }  
  
 */\*\*  
 \* 验证键是否存在  
 \*  
 \** ***@param*** *key  
 \** ***@return*** *\*/* @Override  
 public boolean isExists(String key) {  
 return redisTemplate.hasKey(key);  
 }  
  
}

12. CacheKeyManagerImpl.java

package com.glitter.spring.boot.persistence.cache.redis;  
  
import com.glitter.spring.boot.exception.BusinessException;  
import com.glitter.spring.boot.persistence.cache.ICacheKeyManager;  
import org.apache.commons.lang3.StringUtils;  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
import org.springframework.stereotype.Service;  
  
@Service  
public class CacheKeyManagerImpl implements ICacheKeyManager {  
  
 private static final Logger *logger* = LoggerFactory.*getLogger*(CacheKeyManagerImpl.class);  
  
 */\*\* 1分钟 \*/* private static final int *DEFAULT\_EXPIRE\_TIME* = 60 \* 2;  
  
 */\*\*  
 \* 接口变量默认修饰符public、static、final可省  
 \*/* private static final String *KEY\_PREFIX* = "gliter";  
  
 static String getKey(String... str) {  
 String result = *KEY\_PREFIX*;  
 if (null == str || str.length <= 0) {  
 *logger*.error("系统运行异常,ICommonCache接口getKey方法输入参数为空");  
 throw new BusinessException("-1", "系统运行异常");  
 }  
 for (int i = 0; i < str.length; i++) {  
 if (StringUtils.*isBlank*(str[i])) {  
 continue;  
 }  
 result += ":" + str[i];  
 }  
 return result;  
 }  
  
 @Override  
 public String getSessionKey(String uuid) {  
 return CacheKeyManagerImpl.*getKey*("session", uuid);  
 }  
  
 @Override  
 public Integer getSessionKeyExpireTime() {  
 return *DEFAULT\_EXPIRE\_TIME*;  
 }  
  
  
 @Override  
 public Integer getLimitMultiLoginKeyExpireTime() {  
 return *DEFAULT\_EXPIRE\_TIME*;  
 }  
  
 */\*\*  
 \* 限制多端同时登陆时,getSessionKeyKey  
 \** ***@param*** *userId  
 \** ***@return*** *\*/* @Override  
 public String getLimitMultiLoginKey(String userId) {  
 return CacheKeyManagerImpl.*getKey*("session", "limitMultiLogin", "userId", userId);  
 }  
  
}

13. WebMvcConfiguration.java

@Configuration  
public class WebMvcConfiguration implements WebMvcConfigurer {  
 @Bean  
 RequestInterceptor requestInterceptor(){  
 return new RequestInterceptor();  
 }  
  
 @Bean  
 LoginInterceptor loginInterceptor(){  
 return new LoginInterceptor();  
 }  
  
 @Override  
 public void addInterceptors(InterceptorRegistry registry) {  
 List<String> requestInterceptorAddPathPatterns = new ArrayList<>();  
 requestInterceptorAddPathPatterns.add("/\*\*");  
  
 List<String> loginInterceptorAddPathPatterns = new ArrayList<>();  
 loginInterceptorAddPathPatterns.add("/userInfo/\*");  
 loginInterceptorAddPathPatterns.add("/login/logout");  
  
 registry.addInterceptor(requestInterceptor()).addPathPatterns(requestInterceptorAddPathPatterns);  
 registry.addInterceptor(loginInterceptor()).addPathPatterns(loginInterceptorAddPathPatterns);  
 }  
}

14. RequestInterceptor.java

public class RequestInterceptor implements HandlerInterceptor {  
 private final Logger logger = LoggerFactory.*getLogger*(this.getClass());  
  
 @Override  
 public boolean preHandle(HttpServletRequest httpServletRequest, HttpServletResponse httpServletResponse, Object handler) throws Exception {  
 String jsessionIdCookie = CookieUtils.*getCookieValueByName*(httpServletRequest, GlitterConstants.*JSESSIONID*);  
 JsessionIdCookieContext.*set*(jsessionIdCookie);  
 ResponseContext.*set*(httpServletResponse);  
 RequestContext.*set*(httpServletRequest);  
 return true;  
 }  
  
 @Override  
 public void postHandle(HttpServletRequest httpServletRequest, HttpServletResponse httpServletResponse, Object o, ModelAndView modelAndView) throws Exception {  
  
 }  
  
 @Override  
 public void afterCompletion(HttpServletRequest httpServletRequest, HttpServletResponse httpServletResponse, Object o, Exception e) throws Exception {

// 这一行极端重要,线程执行完毕后,要释放掉绑定在线程中的信息,因为线程池在下一个请求到来时为了提升性能,可能使用的还是之前的线程对象。  
 ContextManager.*removeAllContext*();  
 }  
  
}

15. LoginInterceptor.java

public class LoginInterceptor implements HandlerInterceptor {  
 private final Logger logger = LoggerFactory.*getLogger*(this.getClass());  
  
 @Autowired  
 ISessionHandler sessionHandler;  
  
 @Autowired  
 ICommonCache commonCache;  
  
 @Autowired  
 ICacheKeyManager cacheKeyManager;  
  
 @Override  
 public boolean preHandle(HttpServletRequest httpServletRequest, HttpServletResponse httpServletResponse, Object handler) throws Exception {  
 String jsessionIdCookie = JsessionIdCookieContext.*get*();  
 if(StringUtils.*isBlank*(jsessionIdCookie)){ throw new BusinessException("-2", "用户未登陆"); }  
 ISession session = sessionHandler.getSession();  
 if (!session.getId().equals(jsessionIdCookie)){ throw new BusinessException("-2", "未登录或会话超时，请重新登陆。"); }

UserInfo userInfo;  
 if (null == (userInfo = (UserInfo)session.getAttribute(GlitterConstants.*SESSION\_USER*))){ throw new BusinessException("-2", "用户未登陆"); }  
 String jsessionIdEffective = commonCache.get(cacheKeyManager.getLimitMultiLoginKey(String.*valueOf*(userInfo.getId())));  
 if(StringUtils.*isBlank*(jsessionIdEffective)) { throw new BusinessException("-2", "未登录或会话超时，请重新登陆。"); }  
 if(!jsessionIdCookie.equals(jsessionIdEffective)){  
 session.invalidate();  
 throw new BusinessException("-2", "您的账号已在其它地方登陆，若不是本人操作，请注意账号安全！");  
 }  
 logger.info("LoginInterceptor.preHandle验证成功,jsessionId:{},userId:{},fullName:{}",jsessionIdCookie,userInfo.getId(),userInfo.getFullName());  
 return true;  
 }  
  
 @Override  
 public void postHandle(HttpServletRequest httpServletRequest, HttpServletResponse httpServletResponse, Object o, ModelAndView modelAndView) throws Exception {  
  
 }  
  
 @Override  
 public void afterCompletion(HttpServletRequest httpServletRequest, HttpServletResponse httpServletResponse, Object o, Exception e) throws Exception {

// 这一行极端重要,线程执行完毕后,要释放掉绑定在线程中的信息,因为线程池在下一个请求到来时为了提升性能,可能使用的还是之前的线程对象。  
 ContextManager.*removeAllContext*();  
 }  
  
}

16. ContextManager.java

public class ContextManager {  
  
 public static void removeAllContext() {  
 RequestContext.*remove*();  
 ResponseContext.*remove*();  
 JsessionIdCookieContext.*remove*();  
 SessionContext.*remove*();  
 }  
  
}

17. GlobalExceptionHandler.java

@RestControllerAdvice  
public class GlobalExceptionHandler {  
  
 private static final Logger *logger* = LoggerFactory.*getLogger*(GlobalExceptionHandler.class);  
  
 @ExceptionHandler(BusinessException.class)  
 public ResponseResult handleBusinessException(BusinessException e) {  
 try {  
 // 如果有需要的话,这里也可以将参数打印出来RequestLogInfoContext.get()  
 *logger*.error("handleBusinessException捕获业务异常信息:{}", JSONObject.*toJSONString*(e));  
 // 这一行极端重要,线程执行完毕后,要释放掉绑定在线程中的信息,因为线程池在下一个请求到来时为了提升性能,可能使用的还是之前的线程对象。  
 ContextManager.*removeAllContext*();  
 return new ResponseResult(e.getCode(), e.getMessage());  
 } catch (Exception ex) {  
 *logger*.error(TemplateUtil.*getExceptionLogMsg*(this.getClass().getName(), Thread.*currentThread*().getStackTrace()[1].getMethodName(), ex));  
 return new ResponseResult(e.getCode(), e.getMessage());  
 }  
 }  
  
 @ExceptionHandler(Exception.class)  
 public ResponseResult handleException(Exception e) {  
 try {  
 *logger*.error("handleException捕获运行异常信息:{}", JSONObject.*toJSONString*(e));  
 // 这一行极端重要,线程执行完毕后,要释放掉绑定在线程中的信息,因为线程池在下一个请求到来时为了提升性能,可能使用的还是之前的线程对象。  
 ContextManager.*removeAllContext*();  
 return new ResponseResult("-1", "系统异常");  
 } catch (Exception ex) {  
 *logger*.error(TemplateUtil.*getExceptionLogMsg*(this.getClass().getName(), Thread.*currentThread*().getStackTrace()[1].getMethodName(), ex));  
 return new ResponseResult("-1", "系统异常");  
 }  
 }  
  
}

由上可以看到，在拦截器和全局异常处理中都有ContextManager.*removeAllContext*();

这行代码是极其重要的，一定要保证线程用完之后，线程中绑定数据的解绑，其中拦截器的这行代码保证了代码运行正常情况下的解绑，而全局异常则保证了代码异常情况下的解绑，这样无论代码执行正常还是异常，线程执行完都会进行数据解绑，确保万无一失。

另外，关于拦截器，其实只要在最外层的拦截器有这行代码就可以了，第二第三第n拦截器可以不用写也行，因为正常情况下，一定会运行到这个拦截器的，异常情况下就去全局异常处理了。

18.login.html

<!DOCTYPE HTML>  
<html>  
<head>  
 <script type="text/javascript" src="js/jquery-3.3.1.min.js"></script>  
 <script type="text/javascript" src="js/jsencrypt.min.js"></script>  
  
 <script type="text/javascript">  
 $(**function**(){  
 // var domain = "http://localhost:8081";  
 **var** domain = "http://limengjun.com/demo";  
  
 getLoginGraphCaptcha();  
  
 **function** getLoginGraphCaptcha() {  
 $.ajax({  
 type: "GET",  
 url: domain + "/login/getLoginGraphCaptcha",  
 data: "id=1",  
 async: **false**,  
 success: **function**(data){  
 captcha = "data:image/png;base64,"+data.data;  
 $("#captcha").val()  
 $("#captcha").attr("src",captcha);  
 }  
 });  
 }  
  
 $("#submitBtn").click(**function**(){  
 $.ajax({  
 type: "POST",  
 url: domain + "/login/login",  
 contentType: "application/json; charset=utf-8",  
 data: getLoginFromData(),  
 dataType: "json",  
 success: **function** (resultObj) {  
 **if**(resultObj.code=='0'){  
 window.location.href=domain + "/index.html";  
 } **else** {  
 **if**(resultObj.code=='-2'){  
 alert(resultObj.message);  
 }  
 **if**(resultObj.code=='-1'){  
 alert(resultObj.message);  
 }  
 getLoginGraphCaptcha();  
 }  
 },  
 error: **function** (message) {  
 alert(message);  
 }  
 });  
 });  
  
 **function** getLoginFromData() {  
 **var** encrypt = **new** JSEncrypt();  
 encrypt.setPublicKey(getPublicKey());  
 **var** encryptData = encrypt.encrypt($("#password").val());  
 **var** pwd = encodeURI(encryptData);  
 // +号的处理：因为get请求,数据在网络上传输时，非字母数字字符都将被替换成百分号（%）后跟两位十六进制数，  
 // 而base64编码在传输到后端的时候，+会变成空格，因此先替换掉。后端再替换回来  
 // var pwd = encodeURI(encryptData).replace(/\+/g, '%2B');  
 **var** json = {  
 "account": $("#account").val(),  
 "password": pwd,  
 "graphCaptcha": $("#graphCaptcha").val()  
 };  
 **return** JSON.stringify(json);  
 }  
  
 **function** getPublicKey() {  
 **var** resultStr = "";  
 $.ajax({  
 type: "GET",  
 url: domain + "/login/getPublicKey",  
 data: "id=1",  
 async: **false**,  
 success: **function**(data){  
 resultStr = data.data;  
 }  
 });  
 **return** resultStr;  
 }

});  
 </script>  
</head>  
<body>  
<div id="myDiv"></div>  
<form name="loginFrom">  
 帐号:

<input type="text" name="account" id="account" size="18" height="28px" value="" /><br>  
 密码:

<input type="password" name="password" id="password" size="18" height="28px" value="" /><br>  
 验证码:

<input type="text" name="graphCaptcha" id="graphCaptcha" size="18" height="28px" value="" /><img id="captcha" width="100px" src="" >  
 <input type="button" name="submitBtn" id="submitBtn" value="登陆" />  
</form>  
</body>  
</html>

19.index.html

<!DOCTYPE HTML>  
<html>  
<head>  
 <script type="text/javascript" src="js/jquery-3.3.1.min.js"></script>  
 <script type="text/javascript" src="js/jsencrypt.min.js"></script>  
  
 <script type="text/javascript">  
 $(**function**(){  
 // **var** domain = "http://localhost:8081";  
 var domain = "http://limengjun.com/demo";  
  
 getUserInfo();  
  
 **function** getUserInfo() {  
 $.ajax({  
 type: "GET",  
 url: domain + "/userInfo/getLoginUserInfo",  
 data: "id=1",  
 async: **false**,  
 success: **function**(data){  
 **if**(data.code == 0){  
 $("#userNameSpan").html(","+data.data.fullName);  
 } **else if** (data.code == -2){  
 alert(data.message);  
 window.location.href=domain + "/login.html";  
 } **else** {  
 alert(data.message);  
 }  
 }  
 });  
 }  
  
  
 $("#logoutBtn").click(**function**(){  
 $.ajax({  
 type: "POST",  
 url: domain + "/login/logout",  
 async: **false**,  
 success: **function**(data){  
 **if**(data.code == 0){  
 window.location.href=domain + "/login.html";  
 } **else**{  
 alert(data.message);  
 }  
 }  
 });  
 });  
  
 });  
 </script>  
</head>  
<body>  
欢迎你来到首页<span id="userNameSpan"></span>! <br>  
<input type="button" name="logoutBtn" id="logoutBtn" value="退出" />  
</body>  
</html>