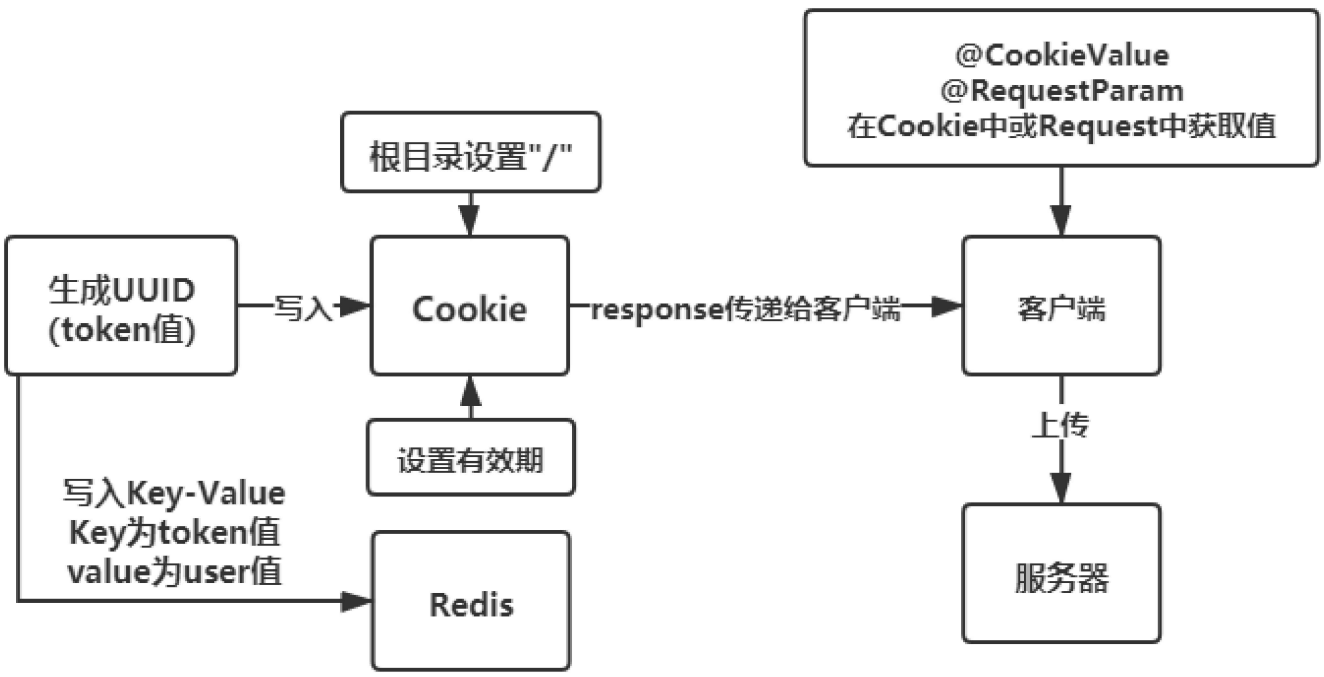
目录

---

1. 实现分布式Session

1.1 原理图解



- **作用**：用 Redis存储Session值 ， 在Redis中 通过token值来获取用户信息

1.2 每次登陆，将Session的过期时间进行修正

- 怎么说呢？我们的Session值固定过期时间为30min，要在每次登陆的时候， 以当前时间继续顺延30分钟
- 我们的 解决方法 就是，每次登陆时，重新再添加一次Cookie，则能够完成时间延长

以下是 封装addCookie() 的方法

```
    private void addCookie(HttpServletResponse response, MiaoShaUser user, String token)
        //首次登陆的时候，需要将Cookie存入Redis
        redisService.set(MiaoShaUserKey.getTokenPrefix,token,user);
        Cookie cookie = new Cookie(COOKIE_NAME_TOKEN, token);
        cookie.setMaxAge(MiaoShaUserKey.getTokenPrefix.expireSeconds());
        //设置为根目录，则可以在整个应用范围内使用cookie
        cookie.setPath("/");
        response.addCookie(cookie);
    }
```

1.3 Cookie有什么用?

在我们这个项目中，Cookie中存储的是token值。而这个token值是和用户信息是一一绑定的，将会存储在Redis中。我们从Cookie中获取到token，从而就可以获取到用户，下面简化代码的过程，便是对这一过程的演示。

1.4 分布式Session的理解

服务器中的原生session是无法满足需求的，因为用户的请求有可能随机落入到不同的服务器中，这样的结果将会导致用户的session丢失，传统做法中有解决方案，是进行session同步，将一个服务器上的session进行同步到另一个服务器上，在一个集群中无论你访问哪个服务器都可以共享，但是这种方法有个明显缺陷，就是性能问题，传输有时延问题，其次这样每台服务器的session重复拥有，这样其内存必然受到影响，如果只有几台服务器还好，如果是十台，二十台服务器呢？这种恐怖的场景会是什么样的体验呢，我就无法得知了。

那么我们应该如何有效的解决这样的问题呢，我们可以使用传说中的token来解决，简单明了的说就是用户每次登陆的时候生成一个类似sessionId的东西（也就是所谓的token，这将是全局的唯一标识，如UUID，作用类似于（sessionId）），将其写到cookie当中传送给客户端，客户端对数据库访问过程中不断上传这个token，而我们服务端拿到这个token就可以获取用户的信息，这个道理其实在很多地方是相通的，比如我们容器中实现原生session，也是将生成的id写入cookie当中。

---

2. 解决注解获取参数造成的代码冗余

我们看一下，如下代码

```
    @RequestMapping("/to_list")
    public String toList(Model model,
                         @CookieValue(value = MiaoShaUserService.COOKIE_NAME_TOKEN,requir
                         @RequestParam(value = MiaoShaUserService.COOKIE_NAME_TOKEN,requi
                         ){
        if(StringUtils.isEmpty(cookieToken) && StringUtils.isEmpty(paramToken)){
            return "login";
        }

        String token = StringUtils.isEmpty(paramToken) ? cookieToken : paramToken;
        MiaoShaUser user = miaoShaUserService.getByToken(response,token);
        model.addAttribute("user",user);

        return "goods_list";
    }
```

- `@CookieValue`： 这个注解能够根据参数value在Cookie中获取值
- `@RequestParam`： 该注解让我们在Request中能获取参数，解决的主要是，移动手机端不使用Cookie存值的问题

我们在如上代码中，可以发现，注解标记获取参数，使得代码很厚重，若我们每次想从Cookie中获取token值时，都需要复现如上代码，所以我们要把它剖离出来

2.1 WebMvcConfigurerAdapter

在这个项目中，我们采用的是继承 `WebMvcConfigurerAdapter` ， 重写其中 `addArgumentResolvers()方法` ，该方法实现的是 参数解析的功能

```
@Configuration
public class WebConfig extends WebMvcConfigurerAdapter{

    @Autowired
    UserArgumentResolver userArgumentResolver;

    @Override
    public void addArgumentResolvers(List<HandlerMethodArgumentResolver> argumentResolver
        super.addArgumentResolvers(argumentResolvers);
        argumentResolvers.add(userArgumentResolver);
    }
}
```

2.1.1 该方法在Spring5.0之后就过时了

- **现用方式**

1. 实现WebMvcConfigurer

```
@Configuration
public class WebMvcConfg implements WebMvcConfigurer {
        //TODO
}
```

2. 继承WebMVCConfigurationSupport

```
@Configuration
public class WebMvcConfg extends WebMvcConfigurationSupport {
        //TODO
}
```

2.2 在argumentResolvers中添加我们的参数解析逻辑

- 首先，我们应该搞清楚，我们想要的参数是什么？回看代码冗余的问题，最终我们想获取的是 `MiaoShaUser` ，这下我们进行代码的编写

```
@Service
public class UserArgumentResolver implements HandlerMethodArgumentResolver {

    @Autowired
    MiaoShaUserService miaoShaUserService;

    @Override
    public boolean supportsParameter(MethodParameter methodParameter) {
        //这个方法判断参数类型是否支持
        Class<?> clazz = methodParameter.getParameterType();
        return clazz == MiaoShaUser.class;
    }

    @Override
    public Object resolveArgument(MethodParameter methodParameter, ModelAndViewContainer
                                  NativeWebRequest nativeWebRequest, WebDataBinderFactory
        //这个方法实现对参数的处理
        HttpServletRequest request = nativeWebRequest.getNativeRequest(HttpServletRequest
        HttpServletResponse response = nativeWebRequest.getNativeResponse(HttpServletResp
```

```
        String paramToken = request.getParameter(miaoShaUserService.COOKIE_NAME_TOKEN);
        String cookieToken = getCookieValue(request, miaoShaUserService.COOKIE_NAME_TOKEN
        if(StringUtils.isEmpty(paramToken) && StringUtils.isEmpty(cookieToken)){
            return null;
        }
        String token = StringUtils.isEmpty(paramToken) ? cookieToken : paramToken;

        return miaoShaUserService.getByToken(response,token);
    }

    private String getCookieValue(HttpServletRequest request,String cookieName){
        Cookie[] cookies = request.getCookies();

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

- 实现 `HandlerMethodArgumentResolver接口` ，必须重写其中的两个方法， `supportsParameter()` 和 `resolveArgument()`
- 前者是对我们要进行解析的 `参数类型` 进行判断，符合才执行后者
- 后者是我们对 `参数的处理逻辑` ，两种情况，一是从request中获取token值，二是从cookie中拿取token值，根据token值来获取到对应的user

以上就将我们需要的参数的处理逻辑实现了，在Mvc配置中，用 `argumentResolvers.add(userArgumentResolver)` 方法进行添加即可，这样我们再想获取user的时候就简单多了，如下

2.3 如此清爽的代码

```
    @RequestMapping("/to_list")
    public String toList(Model model,MiaoShaUser user){
        model.addAttribute("user",user);
        return "goods_list";
    }
```

省去了@CookieValue和@RequestParam注解的冗余，而且我们对user的获取也方便多了