My-store 体系结构

客户端功能

登陆注册接口

```
export async function login(thePhone: string,thePassword: string) {
  const loginInfo: LoginInfo = {
    phone: thePhone,
    password: thePassword,
  }
  return await session.post(`${USERS}/login`, loginInfo)
}
@Concurrent
export async function register(user:User) {
  return await session.post(`${'users'}/register`,user);
}
```

多线程开发

在删除地址和获得地址两个接口实现了多线程开发，通过@Concurrent注释实现

```
@Concurrent
export async function deleteAddressInfo(addressInfoId: number) {
  return await session.delete(`${'users'}/deleteAddressInfo/${addressInfoId}`);
}
```

服务端结构

Product

```
export interface Product {
    productId?: number
    productName?: string
    productOriginalPrice?: number
    productNowPrice?: number
    productImages?: string[]
    productCategory?: string
```

```
    productDescription?: string
    productScore?: number
    productScoreCount?: number // 评分人数
    productCommentList?: Comment[],
}
```

ProductOption

```
export interface ProductOption {
    productOptionId?: number;
    productId?: number;
    productOptionName?: string; // 选项名，如"颜色"
}

export interface ProductOptionValue {
    productOptionValueId?: number;
    productId?: number;
    productOptionName?: string;
    value?: string; // 选项值，如"银色"
}
```

Comment 评论

```
export interface Comment {
    commentId?: number,
    userId?: number,
    content?: string,
    date?: Date,
    // ......
}
```

CartItem 购物车item

```
class CartItem {
    Integer cartItemId;
    Integer userId;
    Product product;
    Integer quantity; // 商品数量
    Date date; // 创建时间，用于排序
}
```

AddressInfo 地址信息

```ts
export interface AddressInfo {
    addressInfoId?: number,
    userId: number,
    areaAddress: Array<string>, // 省市区
    detailAddress: string, // 详细地址
    phone: string,
    receiver: string, // 收件人
    isDefault: boolean, // 是否为默认地址
}
```

需要的接口

```ts
// '/users/getShoppingCart/{userId}'
export async function getShoppingCart(userId: number): Promise<AxiosResponseData<CartItem[]>> {
    return await userService.get(`/getShoppingCart/${userId}`)
}

// '/users/addToShoppingCart' (RequestBody CartItem cartItem)
export async function addToShoppingCart(userId: number, product: Product, quantity: number): Promise<AxiosResponseData<CartItem>> {
    const cartItem: CartItem = {
        userId: userId,
        product: product,
        quantity: quantity,
        date: new Date()
    }
    return await userService.post(`/addToShoppingCart`, cartItem)
}

/**
 * 获取用户的所有收货地址
 * 如果用户没有收货地址，返回空 list
 * 如果有默认地址，返回时需将默认地址排在最前面
 */
export async function getAllAddressInfo(userId: number): Promise<AxiosResponseData<AddressInfo[]>> {
    return await userService.get(`/getAllAddressInfo/${userId}`)
}

```

```ts
/**
 * 添加收货地址
 * 如果 addressInfo.isDefault 为 true 且原来已有默认地址，还需将原来的默认地址设为非默认
 * @return 返回新添加的地址
 */
export async function addAddressInfo(addressInfo: AddressInfo): Promise<AxiosResponseData<AddressInfo>> {
    return await userService.post(`/addAddressInfo`, addressInfo)
}

/**
 * 更新收获地址
 * 如果 addressInfo.isDefault 为 true，还需将原来的默认地址设为非默认
 * @return 返回更新后的地址
 */
export async function updateAddressInfo(addressInfo: AddressInfo): Promise<AxiosResponseData<AddressInfo>> {
    return await userService.post(`/updateAddressInfo`, addressInfo)
}

export async function deleteAddressInfo(addressInfoId: number): Promise<AxiosResponseData<boolean>> {
    return await userService.delete(`/deleteAddressInfo/${addressInfoId}`)
}

/**
 * 更新购物车项(中的商品数量)
 */
export async function updateCartItem(cartItem: CartItem): Promise<AxiosResponseData<CartItem>> {
    return await userService.post(`/updateCartItem`, cartItem)
}

/**
 * 删除购物车项
 */
export async function deleteCartItem(cartItemId: number): Promise<AxiosResponseData<boolean>> {
    return await userService.delete(`/deleteCartItem/${cartItemId}`)
}
```

OrderInfo

```ts
public class OrderInfo {
```

```
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Id
    private Integer orderInfoId;

    private Integer userId;

    @OneToMany(cascade = CascadeType.ALL, orphanRemoval = true)
    @JoinColumn(name = "orderInfo_id")
    private List<CartItem> products;

    private Integer addressInfoId;

    @Enumerated(EnumType.STRING)
    private OrderStatusEnum orderStatus;

    private double totalPrice;

    Date createDate;
    }
```

实现的接口

```
@GetMapping("/getAllOrders/{userId}")
public ResultVO<List<OrderInfoVO>> getAllOrders(@PathVariable("userId") Integer userId) {
    return ResultVO.buildSuccess(orderService.getAllOrders(userId));
}

@PostMapping("/createOrder")
public ResultVO<Boolean> createOrder(@RequestBody OrderInfoVO orderInfoVO) {
    return ResultVO.buildSuccess(orderService.createOrder(orderInfoVO));
}

@PostMapping("/deleteOrder")
public ResultVO<Boolean> deleteOrder(@RequestParam ("orderId") Integer orderId) {
    return ResultVO.buildSuccess(orderService.deleteOrder(orderId));
}

@PostMapping("/deleteProduct")
public ResultVO<Boolean> deleteProduct(@RequestParam("orderId") Integer orderId, @RequestParam ("productId") Integer productId) {
    return ResultVO.buildSuccess(orderService.deleteProduct(orderId, productId));
```

```
}

@PostMapping("/updateProduct")
public ResultVO<Boolean> updateProduct(@RequestParam("orderId") Integer orderId, @RequestParam ("productId") Integer productId, @RequestParam("quantity") Integer quantity) {
    return ResultVO.buildSuccess(orderService.updateProduct(orderId, productId, quantity));
}
```

接口鉴权

web拦截器

```
@Configuration
public class MyWebMvcConfig implements WebMvcConfigurer {

    @Autowired
    AccessInterceptor accessInterceptor;

    @Override
    public void addInterceptors(InterceptorRegistry registry) {
        registry.addInterceptor(accessInterceptor)
                .addPathPatterns("/api/users/getAllOrders/{userId}")
                .addPathPatterns("/api/users/logout/{userId}")
                .addPathPatterns("/api/users/getShoppingCart/{userId}")
                .addPathPatterns("/api/users/getAllAddressInfo/{userId}");

    }
}


public class AccessInterceptor implements HandlerInterceptor {

    @Autowired
    private StringRedisTemplate stringRedisTemplate;


    @Override
    public boolean preHandle(@NonNull HttpServletRequest request, @NonNull HttpServletResponse response, @NonNull Object handler) {
        if (handler instanceof HandlerMethod) {
            HandlerMethod handlerMethod = (HandlerMethod) handler;
            AccessUtil accessutil = handlerMethod.getMethodAnnotation(AccessUtil.class);
```

```
            if (accessutil == null) {
                accessutil = handlerMethod.getBeanType().getAnnotation(AccessUtil.class);
            }
            if (accessutil != null) {
                String uri = request.getRequestURI();
                String userId  = uri.substring(uri.lastIndexOf("/") + 1);

                String key = "UserRole" + userId;
                String userRole = stringRedisTemplate.opsForValue().get(key);
                if (userRole == null || !Arrays.asList(accessutil.roles()).contains(UserRoleEnum.valueOf(userRole))) {
                    response.setStatus(HttpServletResponse.SC_FORBIDDEN);
                    throw MyStoreException.accessDenied();
                }
            }
        }
        return true;
    }
}
```

接口权限限制

```
@AccessUtil(roles = {UserRoleEnum.CHILD, UserRoleEnum.PARENT})
@PostMapping("/logout/{userId}")
public ResultVO<Boolean> logout(@PathVariable("userId") Integer userId) {
    return ResultVO.buildSuccess(userService.logout(userId));
}

@AccessUtil(roles = {UserRoleEnum.CHILD, UserRoleEnum.PARENT})
@GetMapping("/getShoppingCart/{userId}")
public  ResultVO<List<CartItemVO>> getShoppingCart(@PathVariable("userId") Integer userId) {
    return ResultVO.buildSuccess(userService.getCartItems(userId));
}

@AccessUtil(roles = {UserRoleEnum.CHILD, UserRoleEnum.PARENT})
@GetMapping("/getAllAddressInfo/{userId}")
public ResultVO<List<AddressInfoVO>> getAllAddressInfo(@PathVariable("userId") Integer userId) {
    return ResultVO.buildSuccess(userService.getAddressInfo(userId));
}

@AccessUtil(roles = {UserRoleEnum.CHILD, UserRoleEnum.PARENT})
@GetMapping("/getAllOrders/{userId}")
```

```
21 public ResultVO<List<OrderInfoVO>> getAllOrders(@PathVariable("userId")
   Integer userId) {
22     return ResultVO.buildSuccess(orderService.getAllOrders(userId));
23 }
```

缓存

redis实现用户登入后角色的缓存

```
1  @Configuration
2  public class RedisConfig {
3
4      /**
5       * 配置缓存管理器
6       *
7       * @param factory Redis 线程安全连接工厂
8       * @return 缓存管理器
9       */
10     @Bean
11     public CacheManager cacheManager(RedisConnectionFactory factory) {
12         // 生成两套默认配置，通过 Config 对象即可对缓存进行自定义配置
13         RedisCacheConfiguration cacheConfig =
   RedisCacheConfiguration.defaultCacheConfig()
14                 // 设置过期时间 10 分钟
15                 .entryTtl(Duration.ofMinutes(10))
16                 // 设置缓存前缀
17                 .prefixKeysWith("cache:user:")
18                 // 禁止缓存 null 值
19                 .disableCachingNullValues()
20                 // 设置 key 序列化
21                 .serializeKeysWith(keyPair())
22                 // 设置 value 序列化
23                 .serializeValuesWith(valuePair());
24         // 返回 Redis 缓存管理器
25         return RedisCacheManager.builder(factory)
26                 .withCacheConfiguration("user", cacheConfig).build();
27     }
28
29     private RedisSerializationContext.SerializationPair<String> keyPair() {
30         return RedisSerializationContext.SerializationPair.fromSerializer(new
   StringRedisSerializer());
31     }
32
33     private RedisSerializationContext.SerializationPair<Object> valuePair() {
```

```
        return RedisSerializationContext.SerializationPair.fromSerializer(new GenericJackson2JsonRedisSerializer());
    }


    @Bean
    public RedisTemplate<String, Object> redisTemplate(RedisConnectionFactory redisConnectionFactory) {
        RedisTemplate<String, Object> redisTemplate = new RedisTemplate<>();
        redisTemplate.setConnectionFactory(redisConnectionFactory);
        //指定key序列化策略为String序列化，Value为JDK自带的序列化策略
        redisTemplate.setKeySerializer(new StringRedisSerializer());
        redisTemplate.setValueSerializer(new JdkSerializationRedisSerializer());
        return redisTemplate;
    }

    @Bean
    public StringRedisTemplate stringRedisTemplate(RedisConnectionFactory redisConnectionFactory) {
        StringRedisTemplate stringRedisTemplate = new StringRedisTemplate();
        stringRedisTemplate.setConnectionFactory(redisConnectionFactory);
        return stringRedisTemplate;
    }
```

redis使用

在用户登录之后持续持有user_role的缓存，在登出后释放

```
@Override
public User login(UserVO userVO) {
    User user = userRepository.findByPhoneAndPassword(userVO.getPhone(), userVO.getPassword());
    if (user == null) {
        throw MyStoreException.phoneOrPasswordError();
    }
    String key = "User" + user.getId();
    stringRedisTemplate.opsForValue().set(key, String.valueOf(user.getRole()));
    return user;
}

@Override
public Boolean logout(Integer userId) {
    String key = "UserRole" + userId;
    stringRedisTemplate.delete(key);
    return true;
```

```
17 }
```