JWT

1、什么是JSON WEB TOKEN?

JSON Web令牌（JWT）是一个开放标准（RFC 7519），它定义了一种紧凑而独立的方法，用于在各方之间安全地将信息作为JSON对象传输。由于此信息是经过数字签名的，因此可以被验证和信任。可以使用秘密（使用HMAC算法）或使用RSA或ECDSA的公用/专用密钥对对JWT进行签名。尽管可以对JWT进行加密以提供双方之间的保密性，但我们将重点关注已签名的令牌。签名的令牌可以验证其中包含的声明的完整性，而加密的令牌则将这些声明隐藏在其他方的面前。当使用公钥/私钥对对令牌进行签名时，签名还证明只有持有私钥的一方才是对其进行签名的一方。

2、什么时候我们应该使用JWT?

- **授权**：这是使用JWT的最常见方案。一旦用户登录，每个后续请求将包括JWT，从而允许用户访问该令牌允许的路由，服务和资源。单点登录是当今广泛使用JWT的一项功能，因为它的开销很小并且可以在不同的域中轻松使用。
- **信息交换**：JSON Web令牌是在各方之间安全传输信息的一种好方法。因为可以对JWT进行签名（例如，使用公钥/私钥对），所以您可以确定发件人是他们所说的人。此外，由于签名是使用标头和有效负载计算的，因此您还可以验证内容是否遭到篡改。

3、JWT的结构组成

- Header 头部
- Payload 负载
- Signature 签名

组成：xxxxx.yyyyy.zzzzz

3.1 Header

Header通常由两部分组成：令牌的类型（即JWT）和所使用的签名算法，例如HMAC SHA256或RSA。

```
//for example
{
  "alg": "HS256",       //签名算法
  "typ": "JWT"          //令牌的类型
}
//这个header一般去改变它
```

然后，此JSON被Base64Url编码以形成JWT的第一部分.

3.2 Payload

令牌的第二部分是有效负载，其中包含声明。声明是有关实体（通常是用户）和其他数据的声明。共有三种类型的声明：`registered`， `public`， and `private` claims.。

3.2.1 registered claims:

这些是一组非强制性的但建议使用的预定义声明，以提供一组有用的可互操作的声明。其中一些是：iss（issuer：发行方），exp（expiration time：到期时间），sub（subject：主题），aud（audience：受众群体）等

请注意，声明名称仅是三个字符，因为JWT是紧凑的。

3.2.2 public claims

这些可以由使用JWT的人员随意定义。但为避免冲突，应在IANA JSON Web令牌注册表中定义它们，或将其定义为包含抗冲突名称空间的URI。

3.2.3 private claims

这些是自定义声明，旨在在同意使用它们的各方之间共享信息，并且既不是注册声明也不是公共声明。

```
//for example
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

然后对有效负载进行Base64Url编码，以形成JSON Web令牌的第二部分。

> 请注意，对于已签名的令牌，此信息尽管可以防止篡改，但任何人都可以读取。除非加密，否则不要将机密信息放入JWT的有效负载或报头元素中

3.3 Signature

要创建签名部分，您必须获取编码的报头、编码的有效负载、一个秘密、在报头中指定的算法，并对其进行签名。

例如，如果要使用HMAC SHA256算法，将按以下方式创建签名：

```
//使用HMAC SHA256算法进行加密签名
HMACSHA256(
  base64UrlEncode(header)       //对header进行base64加密
  + "." +
  base64UrlEncode(payload),     //对payload进行base64加密
  secret    //秘密，随机盐，这个不能给别人知道
)
```

签名用于验证消息在整个过程中没有被更改，并且，在使用私钥签名的令牌的情况下，它还可以验证JWT的发送者是它所说的人。

输出是三个base64url字符串，用点分隔，可以在HTML和HTTP环境中轻松地传递这些字符串，但与基于XML的标准（如SAML）相比，它更紧凑。

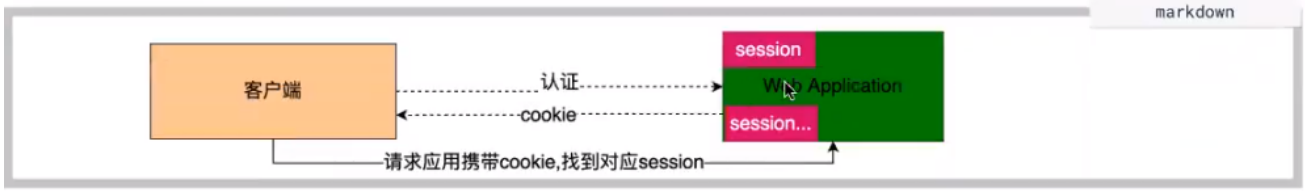
下面显示了一个JWT，它对前一个头和有效负载进行了编码，并用一个秘密进行了签名。

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4
gRG9lIiwiaXNTb2NpYWwiOnRydWV9.
4pcPyMD09olPSyXnrXCjTwXyr4BsezdI1AVTmud2fU4
```

4、基于传统的Session认证

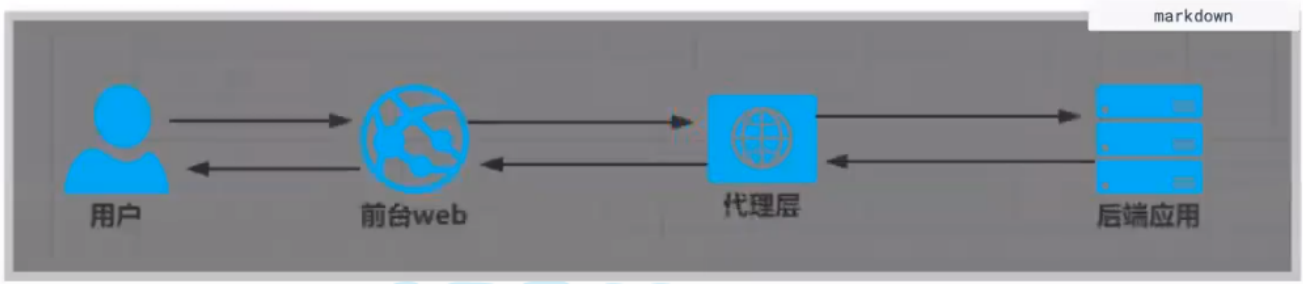
```
# 1.认证方式
- 我们知道，http协议本身是一种无状态的协议，而这就意味着如果用户向我们的应用提供了用户名和密码来进行用户认证，那么下一次请求时，用户还要再一次进行用户认证才行，因为根据http协议，我们并不能知道是哪个用户发出的请求，所以为了让我们的应用能识别是哪个用户发出的请求，我们只能在服务器存储一份用户登录的信息，这份登录信息会在响应时传递给浏览器，告诉其保存为cookie,以便下次请求时发送给我们的应用，这样我们的应用就能识别请求来自哪个用户了,这就是传统的基于session认证。

# 2.认证流程
```

markdown

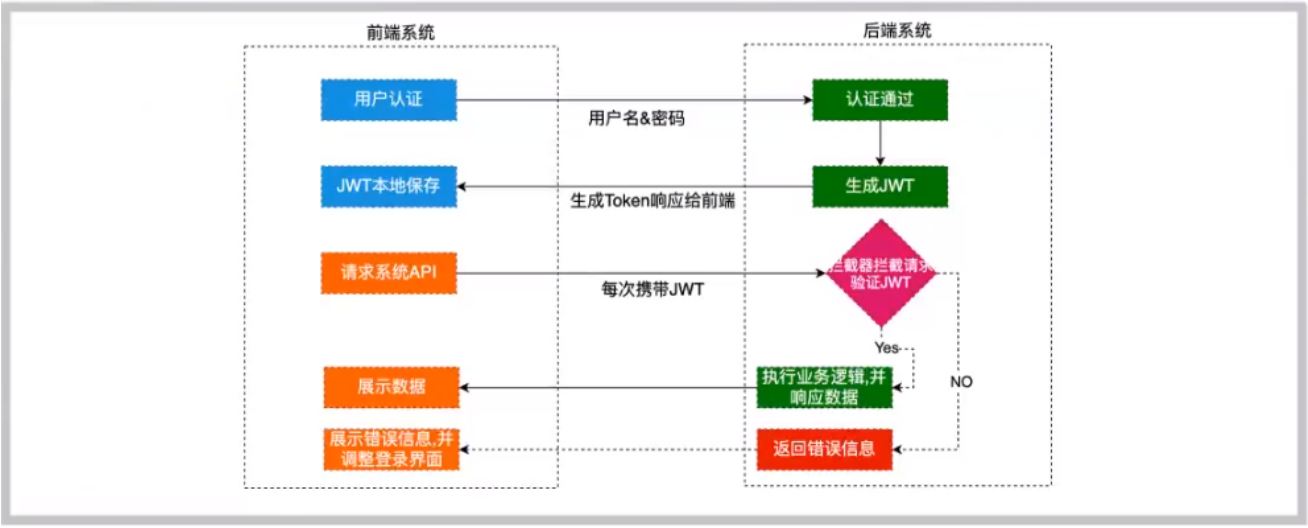



```
# 3.暴露问题
- 1.每个用户经过我们的应用认证之后，我们的应用都要在服务端做一次记录，以方便用户下次请求的鉴别，通常而言session都是保存在内存中，而随着认证用户的增多，服务端的开销会明显增大

- 2.用户认证之后，服务端做认证记录，如果认证的记录被保存在内存中的话，这意味着用户下次请求还必须要请求在这台服务器上,这样才能拿到授权的资源，这样在分布式的应用上，相应的限制了负载均衡器的能力。这也意味着限制了应用的扩展能力。

- 3.因为是基于cookie来进行用户识别的， cookie如果被截获，用户就会很容易受到跨站请求伪造的攻击。

- 4.在前后端分离系统中就更加痛苦:如下图所示
  也就是说前后端分离在应用解耦后增加了部署的复杂性。通常用户一次请求就要转发多次。如果用session 每次携带sessionid 到服务器，服务器还要查询用户信息。同时如果用户很多。这些信息存储在服务器内存中，给服务器增加负担。还有就是CSRF（跨站伪造请求攻击）攻击，session是基于cookie进行用户识别的， cookie如果被截获，用户就会很容易受到跨站请求伪造的攻击。还有就是sessionid就是一个特征值，表达的信息不够丰富。不容易扩展。而且如果你后端应用是多节点部署。那么就需要实现session共享机制。不方便集群应用。
```

markdown



5、基于JWT的认证




```
# 1.认证流程
- 首先，前端通过Web表单将自己的用户名和密码发送到后端的接口。这一过程一般是一个HTTP POST请求。建议的方式是通过SSL加密的传输（https协议），从而避免敏感信息被嗅探。
- 后端核对用户名和密码成功后，将用户的id等其他信息作为JWT Payload（负载），将其与头部分别进行Base64编码拼接后签名，形成一个JWT(Token)。形成的JWT就是一个形同lll.zzz.xxx的字符串。 token head.payload.singurater

- 后端将JWT字符串作为登录成功的返回结果返回给前端。前端可以将返回的结果保存在localStorage或sessionStorage上，退出登录时前端删除保存的JWT即可。

- 前端在每次请求时将JWT放入HTTP Header中的Authorization位。(解决XSS和XSRF问题) HEADER

- 后端检查是否存在，如存在验证JWT的有效性。例如，检查签名是否正确；检查Token是否过期；检查Token的接收方是否是自己（可选）。
- 验证通过后后端使用JWT中包含的用户信息进行其他逻辑操作，返回相应结果。

# 2.jwt优势

- 简洁(Compact)：可以通过URL，POST参数或者在HTTP header发送，因为数据量小，传输速度也很快

- 自包含(Self-contained)：负载中包含了所有用户所需要的信息，避免了多次查询数据库

- 因为Token是以JSON加密的形式保存在客户端的，所以JWT是跨语言的，原则上任何web形式都支持。

- 不需要在服务端保存会话信息，特别适用于分布式微服务。
```

6、使用JWT

6.1 导入依赖

```
<dependency>
    <groupId>com.auth0</groupId>
    <artifactId>java-jwt</artifactId>
    <version>3.11.0</version>
</dependency>
```

6.2 构建token

```
public void test(){
    //生成token
    String token = JWT.create()
        .withHeader(null)   //设置header，一般采用默认值
        .withClaim("name", "zhangsan")  //设置payload
        .withClaim("age", 18)
        .withClaim("sex", "girl")
        .sign(Algorithm.HMAC256("!@#QSDF@#@%￥……%"));     //设置签名，采用HMAC256算法
    System.out.println(token);
}
```

构建token抛出的异常：

- JWTCreationException //构建token失败时，抛出的异常，非法签名结构，无法转换Claims

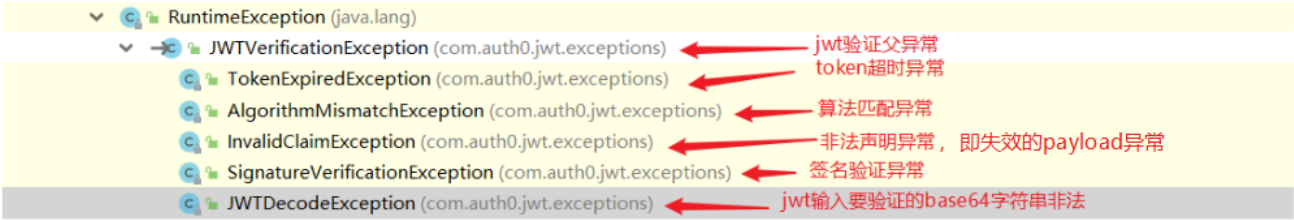
6.3 验证token

```
public void test2(){
    //验证token
    JWTVerifier build = JWT.require(Algorithm.HMAC256("!@#QSDF@#@%￥……%")).build();
    //验证token，验证失败会抛出异常
    DecodedJWT verify =
build.verify("yJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzZXgiOiJnaXJsIiwibmFtZSI6InpoYW5nc2FuIiwiYWdlIjoxOH0.Cne13wAqFmRgmxgWOKgykGXVSfba5XjS-c4soh83QqQ");

    System.out.println(verify.getClaim("name").asString());
    System.out.println(verify.getClaim("age").asInt());
    System.out.println(verify.getClaim("sex").asString());
}
```

随机盐：!@#QSDF@#@%￥……% 随机设置，不能被别人知道，加密和解密保持一致

加密算法：HMAC256 加密和解密保持一致

验证token抛出的异常：



先验签，后验token。

7、JWT工具类

```
/**
 * JWT工具类
```

```
 */
public class JWTUtils {

    //随机盐，随机设置
    private final static String secret = "$%#$%%^FSDFDS@#$@ASDdfg";

    /**
     * 根据提供的k-v进行生成token
     * @param payload k-v信息
     * @param expireTime 过期时间，单位：秒
     * @return token
     */
    public static String createToken(Map<String,String> payload, Integer expireTime) {
        Calendar instance = Calendar.getInstance();
        instance.add(Calendar.SECOND,expireTime);
        String token = null;
        try{
            //创建Builder对象
            JWTCreator.Builder builder = JWT.create().withHeader(null); //header
            //payload
            payload.forEach((k,v)->{
                builder.withClaim(k,v);
            });
            builder.withExpiresAt(instance.getTime());      //过期时间
            token = builder.sign(Algorithm.HMAC256(secret));    //签名
        }catch (JWTCreationException e){
            System.out.println("构建token异常，请检查签名是否合法...");
        }
        return token;
    }

    /**
     * 验证token是否合法
     * @param token
     * @return false:验证失败  true：验证合法
     */
    public static boolean verifyJWT(String token){
        try{
            JWT.require(Algorithm.HMAC256(secret)).build().verify(token);
        }catch (JWTVerificationException e){
            return false;
        }
        return true;
    }


    /**
     * 获取token的数据
     * @return map
     */
    public static Map<String,String> decodeJWT(String token){
        Map<String,String> map = new HashMap<>();

        try{
```

```
            DecodedJWT verify = JWT.require(Algorithm.HMAC256(secret)).build().verify(token);
            Map<String, Claim> claims = verify.getClaims();
            claims.forEach((k,v)->{
                map.put(k,v.asString());
            });
            return map;
        }catch (JWTVerificationException e){
            return null;
        }
    }
}
```

8、JWT在WEB中的应用

1、创建springboot项目，导入依赖

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>

<dependency>
    <groupId>com.auth0</groupId>
    <artifactId>java-jwt</artifactId>
    <version>3.11.0</version>
</dependency>
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.15</version>
</dependency>
<dependency>
    <groupId>com.baomidou</groupId>
    <artifactId>mybatis-plus-boot-starter</artifactId>
    <version>3.4.1</version>
</dependency>

<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <version>1.18.12</version>
</dependency>
<dependency>
    <groupId>org.codehaus.jackson</groupId>
    <artifactId>jackson-mapper-asl</artifactId>

    <version>1.9.13</version>
```

```
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>

<dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-lang3</artifactId>
    <version>3.7</version>
</dependency>
```

2、配置application.yml

```
mybatis-plus:
  type-aliases-package: top.linging.jwt.pojo
  mapper-locations: classpath:top/linging/jwt/mapper/*.xml
  configuration:
    map-underscore-to-camel-case: true
spring:
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://localhost:3306/student?characterEncoding=UTF-
8&useSSL=false&allowPublicKeyRetrieval=true&serverTimezone=UTC
    username: root
    password: 123456
  thymeleaf:
    prefix: classpath:/templates/
    suffix: .html
    cache: false
server:
  port: 8082
```

3、编写登录拦截器

```
public class LoginInterceptor implements HandlerInterceptor {

    /**
     * 获取requestHeader中的token
     * @param request
     * @param response
     * @param handler
     * @return
     * @throws Exception
     */
    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object
handler) throws Exception {
        System.out.println("拦截诶器中.............");
        //获取requestHeader中的token

        String token = CookieUtils.getCookieValue(request,"userToken");
```

```
        //返回结果
        Map<String, Object> map = new HashMap<>();
        //验证token失败
        if(token == null || !JWTUtils.verifyJWT(token)) {
            map.put("status",false);
            map.put("msg","login fail");
            response.setContentType("application/json;charset=utf-8");
            response.getWriter().println(JsonUtil.mapToJson(map));
            return false;
        }
        //放行
        System.out.println("成功.....");
        return true;
    }
}
```

4、注册拦截器

```
@Configuration
public class WebConfig implements WebMvcConfigurer {

    //拦截器对请求进行拦截与放行
    @Override
    public void addInterceptors(InterceptorRegistry registry) {
        registry.addInterceptor(new LoginInterceptor())
                .addPathPatterns("/user/**")
                .excludePathPatterns("/user/login")
                .excludePathPatterns("/router/toLogin");
    }

    private CorsConfiguration corsConfig() {
        CorsConfiguration corsConfiguration = new CorsConfiguration();
        // 请求常用的三种配置，*代表允许所有，也可以自定义属性（比如 header 只能带什么，只能是 post 方
式等）
        corsConfiguration.addAllowedOrigin("*");
        corsConfiguration.addAllowedHeader("*");
        corsConfiguration.addAllowedMethod("*");
        corsConfiguration.setAllowCredentials(true);
        corsConfiguration.setMaxAge(3600L);
        return corsConfiguration;
    }

    //解决跨域
    @Bean
    public CorsFilter corsFilter() {
        UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
        source.registerCorsConfiguration("/**", corsConfig());
        return new CorsFilter(source);
    }

}
```

5、编写mapper层

```
@Repository
public interface UserMapper extends BaseMapper<User> {
}
```

6、编写service层

```
@Service
@Transactional
public class UserServiceImpl implements UserService {

    @Autowired
    private UserMapper userMapper;

    /**
     * 登录
     * @param name
     * @return
     */
    @Override
    public User findUserByName(String name) {
        QueryWrapper<User> queryWrapper = new QueryWrapper<>();
        queryWrapper.eq("name",name);
        return userMapper.selectOne(queryWrapper);
    }

    /**
     * 需要登录才能访问的api
     * @return
     */
    @Override
    public List<User> findAllUser() {
        return userMapper.selectList(null);
    }
}
```

7、编写controller层

```
//@CrossOrigin(value = "*")
@RestController
@RequestMapping("/user")
public class UserController {

    @Autowired
    private UserService userService;

    //登录的方法
    @PostMapping("/login")
    public Map<String,Object> login(@RequestParam("name") String username,
                                    HttpServletRequest request,
```

```
                                    HttpServletResponse response){
        int expireTime = 10;    //过期时间，10s
        System.out.println(username);
        Map<String, String> payload = new HashMap<>();
        payload.put("name",username);
        String token = JWTUtils.createToken(payload,expireTime);
        Map<String, Object> map = new HashMap<>();
        if(token == null){
            map.put("status",false);
            map.put("msg","login fail");
            map.put("token","");
            return map;
        }
        CookieUtils.setCookie(request,response,"userToken",token,expireTime,"",true);
        map.put("status",true);
        map.put("msg","login ok");
        map.put("token",token);
        return map;
    }

    //需要登录之后才能访问的方法
    @GetMapping("/list")
    public List<User> findAllUser(){
        return userService.findAllUser();
    }
}
```

用到的工具类：

```
/**
 * @ClassName JsonUtil
 * @Description json工具类
 * @Author AlphaJunS
 * @Date 2020/3/27 18:59
 * @Version 1.0
 */
public class JsonUtil {

    /**
     * @Author AlphaJunS
     * @Date 19:06 2020/3/27
     * @Description json转map
     * @param json
     * @return java.util.Map<java.lang.String,java.lang.Object>
     */
    public static Map<String,Object> jsonToMap(String json) throws Exception{
        ObjectMapper objectMapper = new ObjectMapper();
        Map<String,Object> map;
        try {
            map = (Map<String,Object>)objectMapper.readValue(json, Map.class);
        } catch (JsonParseException e) {
```

```
            // TODO Auto-generated catch block
            e.printStackTrace();
            throw e;
        } catch (JsonMappingException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
            throw e;
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
            throw e;
        }
        return map;
    }

    /**
     * @Author AlphaJunS
     * @Date 19:07 2020/3/27
     * @Description json转List<Map<String,?>>
     * @param json
     * @return java.util.List<java.util.Map<java.lang.String,?>>
     */
    public static List<Map<String,?>> jsonToMapList(String json) throws Exception{
        ObjectMapper objectMapper = new ObjectMapper();
        JavaType javaType = objectMapper.getTypeFactory().constructParametricType(List.class,
Map.class);
        List<Map<String, ?>> mapList;
        try {
            mapList = (List<Map<String,?>>)objectMapper.readValue(json, javaType);
        } catch (JsonParseException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
            throw e;
        } catch (JsonMappingException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
            throw e;
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
            throw e;
        }
        return mapList;
    }

    /**
     * @Author AlphaJunS
     * @Date 19:09 2020/3/27
     * @Description map转json
     * @param map
     * @return java.lang.String
     */

    public static String mapToJson(Map<String,Object> map) throws Exception{
```

```
        ObjectMapper objectMapper = new ObjectMapper();
        String jsonString = objectMapper.writeValueAsString(map);
        return jsonString;
    }

    /**
     * @Author AlphaJunS
     * @Date 19:10 2020/3/27
     * @Description 对象转json
     * @param object
     * @return java.lang.String
     */
    public static String objectToJson(Object object) throws Exception{
        ObjectMapper objectMapper = new ObjectMapper();
        String jsonList;
        try {
            jsonList = objectMapper.writeValueAsString(object);
        } catch (JsonParseException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
            throw e;
        } catch (JsonMappingException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
            throw e;
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
            throw e;
        }
        return jsonList;
    }

    /**
     * @Author AlphaJunS
     * @Date 20:52 2020/3/27
     * @Description List<Map<String,?>>转json
     * @param cardInfoList
     * @return java.lang.String
     */
    public static String mapListToJson(List<Map<String, Object>> cardInfoList) throws Exception{
        ObjectMapper objectMapper = new ObjectMapper();
        String jsonList;
        try {
            jsonList = objectMapper.writeValueAsString(cardInfoList);
        } catch (JsonParseException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
            throw e;
        } catch (JsonMappingException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();

            throw e;
```

```
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
            throw e;
        }
        return jsonList;
    }

    /**
     * @Author AlphaJunS
     * @Date 20:57 2020/3/27
     * @Description 数组转json
     * @param args
     * @return java.lang.String
     */
    public static String arrayToJson(String[] args) throws Exception{
        // 先讲数组转化为map，然后map转json
        Map<String,String> map = new HashMap<String,String>();
        for(int i=0; i<args.length; i++){
            map.put(i+"", args[i]);
        }
        String json = JsonUtil.objectToJson(map);
        return json;
    }

}
```

```
/**
 *
 * Cookie 工具类
 *
 */
public final class CookieUtils {

    static final Logger logger = LoggerFactory.getLogger(CookieUtils.class);

    /**
     * 得到Cookie的值，不编码
     *
     * @param request
     * @param cookieName
     * @return
     */
    public static String getCookieValue(HttpServletRequest request, String cookieName) {
        return getCookieValue(request, cookieName, false);
    }

    /**
     * 得到Cookie的值，
```

```
     *
     * @param request
     * @param cookieName
     * @return
     */
    public static String getCookieValue(HttpServletRequest request, String cookieName, boolean
isDecoder) {
        Cookie[] cookieList = request.getCookies();
        if (cookieList == null || cookieName == null){
            return null;
        }
        String retValue = null;
        try {
            for (int i = 0; i < cookieList.length; i++) {
                if (cookieList[i].getName().equals(cookieName)) {
                    if (isDecoder) {
                        retValue = URLDecoder.decode(cookieList[i].getValue(), "UTF-8");
                    } else {
                        retValue = cookieList[i].getValue();
                    }
                    break;
                }
            }
        } catch (UnsupportedEncodingException e) {
            logger.error("Cookie Decode Error.", e);
        }
        return retValue;
    }

    /**
     * 得到Cookie的值,
     *
     * @param request
     * @param cookieName
     * @return
     */
    public static String getCookieValue(HttpServletRequest request, String cookieName, String
encodeString) {
        Cookie[] cookieList = request.getCookies();
        if (cookieList == null || cookieName == null){
            return null;
        }
        String retValue = null;
        try {
            for (int i = 0; i < cookieList.length; i++) {
                if (cookieList[i].getName().equals(cookieName)) {
                    retValue = URLDecoder.decode(cookieList[i].getValue(), encodeString);
                    break;
                }
            }
        } catch (UnsupportedEncodingException e) {
            logger.error("Cookie Decode Error.", e);

        }
```

```
        return retValue;
    }

    /**
     * 生成cookie，并指定编码
     * @param request 请求
     * @param response 响应
     * @param cookieName name
     * @param cookieValue value
     * @param encodeString 编码
     */
    public static final void setCookie(HttpServletRequest request, HttpServletResponse response,
String cookieName, String cookieValue, String encodeString) {
        setCookie(request,response,cookieName,cookieValue,null,encodeString, null);
    }

    /**
     * 生成cookie，并指定生存时间
     * @param request 请求
     * @param response 响应
     * @param cookieName name
     * @param cookieValue value
     * @param cookieMaxAge 生存时间
     */
    public static final void setCookie(HttpServletRequest request, HttpServletResponse response,
String cookieName, String cookieValue, Integer cookieMaxAge) {
        setCookie(request,response,cookieName,cookieValue,cookieMaxAge,null, null);
    }

    /**
     * 设置cookie，不指定httpOnly属性
     */
    public static final void setCookie(HttpServletRequest request, HttpServletResponse response,
String cookieName, String cookieValue, Integer cookieMaxAge, String encodeString) {
        setCookie(request,response,cookieName,cookieValue,cookieMaxAge,encodeString, null);
    }

    /**
     * 设置Cookie的值，并使其在指定时间内生效
     *
     * @param cookieMaxAge
     *            cookie生效的最大秒数
     */
    public static final void setCookie(HttpServletRequest request, HttpServletResponse response,
String cookieName, String cookieValue, Integer cookieMaxAge, String encodeString, Boolean
httpOnly) {
        try {
            if(StringUtils.isBlank(encodeString)) {
                encodeString = "utf-8";
            }

            if (cookieValue == null) {

                cookieValue = "";
```

```
            } else {
                cookieValue = URLEncoder.encode(cookieValue, encodeString);
            }
            Cookie cookie = new Cookie(cookieName, cookieValue);
            if (cookieMaxAge != null && cookieMaxAge > 0)
                cookie.setMaxAge(cookieMaxAge);
            if (null != request)// 设置域名的cookie
                cookie.setDomain(getDomainName(request));
            cookie.setPath("/");

            if(httpOnly != null) {
                cookie.setHttpOnly(httpOnly);
            }
            response.addCookie(cookie);
        } catch (Exception e) {
            logger.error("Cookie Encode Error.", e);
        }
    }

    /**
     * 得到cookie的域名
     */
    private static final String getDomainName(HttpServletRequest request) {
        String domainName = null;

        String serverName = request.getRequestURL().toString();
        if (serverName == null || serverName.equals("")) {
            domainName = "";
        } else {
            serverName = serverName.toLowerCase();
            serverName = serverName.substring(7);
            final int end = serverName.indexOf("/");
            serverName = serverName.substring(0, end);
            final String[] domains = serverName.split("\\.");
            int len = domains.length;
            if (len > 3) {
                // www.xxx.com.cn
                domainName = domains[len - 3] + "." + domains[len - 2] + "." + domains[len - 1];
            } else if (len <= 3 && len > 1) {
                // xxx.com or xxx.cn
                domainName = domains[len - 2] + "." + domains[len - 1];
            } else {
                domainName = serverName;
            }
        }

        if (domainName != null && domainName.indexOf(":") > 0) {
            String[] ary = domainName.split("\\:");
            domainName = ary[0];
        }
        return domainName;
    }
```

```
}
```

html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <script src="https://cdn.bootcdn.net/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
</head>
<body>

    <div id="form_data">
        <input type="text" name="name" id="name"><br>
        <input type="password" name="password" id="password"><br>
        <button id="sub_ajax">login in</button>
    </div>

    <div id="res"></div>

    <button id="getData">获取数据</button>
    <hr>
    <table border="1" id="dataTable">
        <tr>
            <td>id</td>
            <td>name</td>
        </tr>
    </table>

    <script>
        $("#sub_ajax").click(function () {
            $.ajax({
                url:'/user/login',
                type:'post',
                dataType:'json',
                data:{name:$("#name").val()},
                success:function (res) {
                    $("#res").html(res.token);
                },
                error:function (res) {
                    console.log(res);
                }
            })
        });

        $("#getData").click(function () {
            $.ajax({

                url:'/user/list',
```

```
                type:'get',
                dataType:'json',
                data:{},
                success:function (res) {
                    var tb = '<tr>\n' +
                        '            <td>id</td>\n' +
                        '            <td>name</td>\n' +
                        '        </tr>';
                    for(var i = 0; i < res.length; i++){
                        var tr = '<tr>\n' +
                            '            <td>'+res[i].id+'</td>\n' +
                            '            <td>'+res[i].name+'</td>\n' +
                            '        </tr>';
                        tb += tr;
                    }
                    $("#dataTable").html(tb);
                },
                error:function (res) {
                    console.log(res);
                }
            })
        });
    </script>

</body>
</html>
```