链滴

spring-security-jwt Restful API Token 安全认证

作者：xjtushilei

原文链接：https://ld246.com/article/1496982153867

来源网站：链滴

介绍

自己尝试在基于spring-boot的restful API的安全认证上做的调研与测试。

微服务架构是未来的趋势，但是在API安全认证方面的文章不多，大多数是基于 MVC 和 session 的
但是微服务时代的API都是无状态的，遵循 Restful(附一篇比较好的博客介绍restful 理解RESTful架构
约束。不能满足需求，所以我就推动一下，将文章弄新一点。

适合读者

- Spring-boot 的 Restful API 安全认证，无状态，无session
- spring-data-jpa 进行用户权限存储管理，基于mysql，不是网上一大堆教程基于内存的用户认真
- jwt 协议的 token

为什么不采用 httpBasic

由于基于 httpBasic 认证的很简单，网上教程一大堆，不做过多介绍。这里我采用的是基于jwt协议的
PI安全认证。

用户每次都在传输过程中传输用户名和密码，太不安全了

用户每次请求一个url都会访问数据库，会导致数据库的访问压力！太大了。

为什么不采用 OAuth2

OAuth2协议也是十分优秀的，但是我还是个菜鸟，虽然理解了OAuth2 ，实现感觉也不难，但是还
有看懂 spring-boot 的框架源码，所以在集成方面不是很顺手。

其实还有一点就是 OAuth2 相对 jwt 等来说还是相对复杂的。如果我们不对外提供提供授权服务，n
me使用 OAuth 个人感觉还是相当费劲的，传输效率和模式上，都没有 jwt 等token来的方便。

其实，基于token的认证，大多数自己实现也是十分方便的，加密解密，存在redis里，然后自动过期
一切都很简单，大家自己设计适合自己的认证才是最关键的。

实现

基于mysql的用户认证

网上代码一大堆，其实就是继承 UserDetailsService 就行了，需要自己设计jpa的表，我的代码里有
子，可以随意扩展。

```
@Service
public class UserService implements UserDetailsService {
    @Autowired
    SysUserRepository sysUserRepository;
```

然后在 WebSecurityConfigurerAdapter 中进行配置自己的 UserDetailsService，很简单。

```
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

    @Bean
    UserDetailsService customUserService() {
        return new UserService();
    }

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(customUserService());
    }
```

jwt 的token生成

就是jwt协议的 加密和解密。需要设置加密算法，加密秘钥，过期时间等。

加密的时候，把username加进去，讲道理应该是userid。这样我们前端就可以使用该id来进行请求他服务。

解密的时候，需要进行过期校验，秘钥校验等

```
class TokenAuthenticationService {
    static final long EXPIRATIONTIME = 1000*60*60*24*1; // 1 days
    static final String SECRET = "ThisIsASecret";
    static final String TOKEN_PREFIX = "Bearer";
    static final String HEADER_STRING = "Authorization";

    static void addAuthentication(HttpServletResponse res, String username) {
        String JWT = Jwts.builder()
                .setSubject(username)
                .setExpiration(new Date(System.currentTimeMillis() + EXPIRATIONTIME))
                .signWith(SignatureAlgorithm.HS512, SECRET)
                .compact();
        res.addHeader(HEADER_STRING, TOKEN_PREFIX + " " + JWT);
    }

    static Authentication getAuthentication(HttpServletRequest request) {
        String token = request.getHeader(HEADER_STRING);
        if (token != null) {
            // parse the token.
            String user = Jwts.parser()
                    .setSigningKey(SECRET)
                    .parseClaimsJws(token.replace(TOKEN_PREFIX, ""))
                    .getBody()
                    .getSubject();

            return user != null ?
                    new UsernamePasswordAuthenticationToken(user, null, emptyList()) :
                    null;
        }
```

```
        return null;
    }
}
```

jwt 的 filter 进行 rul 认证

这里设置 登录 只能从post进行，并设置了两个filter分别对login和其他url进行拦截。

```
@Override
    protected void configure(HttpSecurity http) throws Exception {
        http.csrf().disable().authorizeRequests()
                .antMatchers("/").permitAll()
                .antMatchers(HttpMethod.POST, "/login").permitAll()
                .anyRequest().authenticated()
                .and()
                // We filter the api/login requests
                .addFilterBefore(new JWTLoginFilter("/login", authenticationManager()),
                        UsernamePasswordAuthenticationFilter.class)
                // And filter other requests to check the presence of JWT in header
                .addFilterBefore(new JWTAuthenticationFilter(),
                        UsernamePasswordAuthenticationFilter.class);
    }
```

如何使用

使用之前

由于我集成了spring-data-jpa，所以在使用之前需要配置数据库，并插入一些数据。

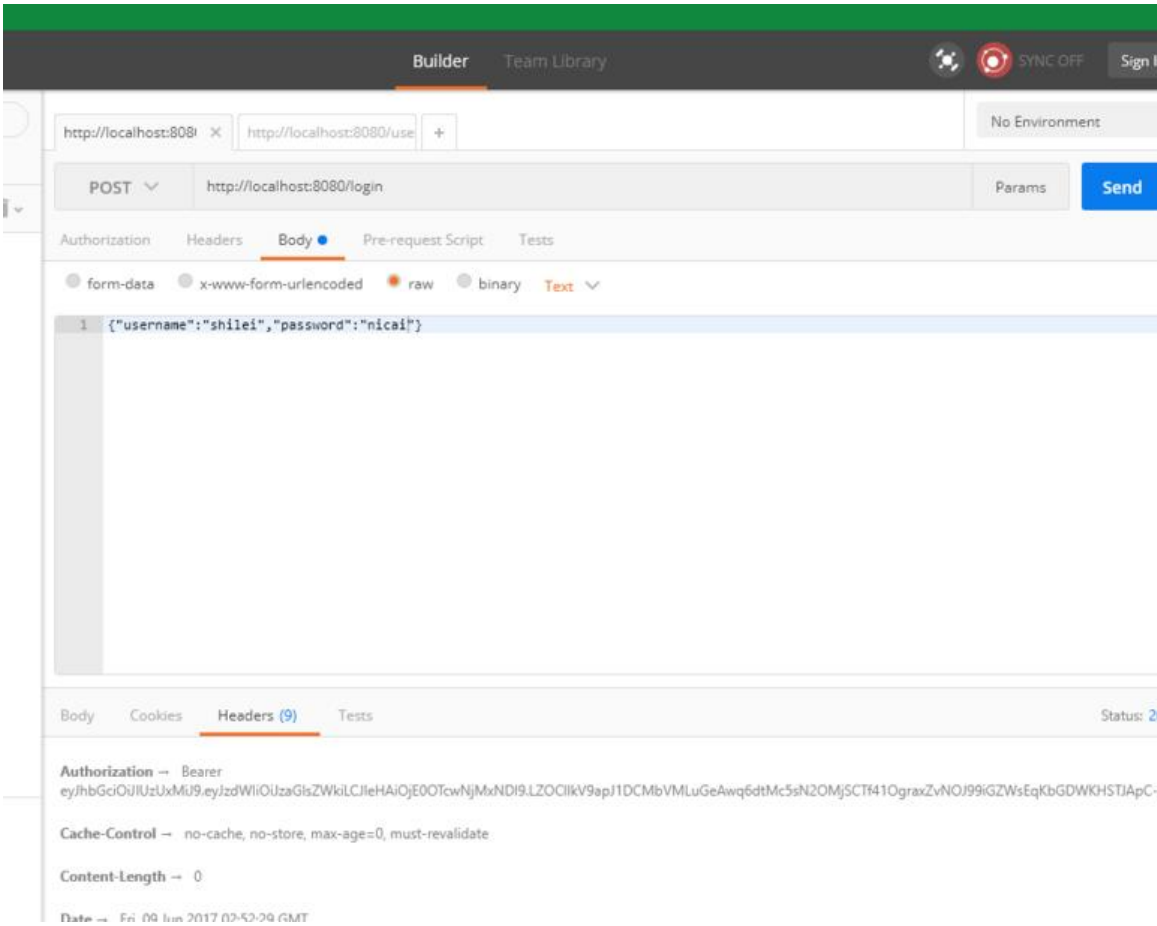
还好，我在初始化工程的时候替你们做了。我在init目录中进行了初始化设置，所以会自动插入两个户名。

```
@Override
    public void run(String... args) throws Exception {
        logger.info(">>>>>>>>>>>>>>>>服务启动检查1，开始检查用户系统<<<<<<<<<<<<<
);

        if (sysUserRepository.count() != 0) {
            logger.info(">>>>>>>>>>>>>>>>用户已经初始化<<<<<<<<<<<<<");
        } else {
            logger.info(">>>>>>>>>>>>>>>不存在初始用户，开始创建<<<<<<<<<<<<<");
            SysRole sysRole1 = new SysRole(1L, "ROLE_ADMIN");
            SysRole sysRole2 = new SysRole(2L, "ROLE_USER");
            sysRoleRepository.save(Arrays.asList(sysRole1, sysRole2));

            List<SysRole> roles1 = Arrays.asList(sysRole1);
            SysUser sysUser1 = new SysUser(1L, "root", "nibudong");
            sysUser1.setRoles(roles1);

            List<SysRole> roles2 = Arrays.asList(sysRole2);
```

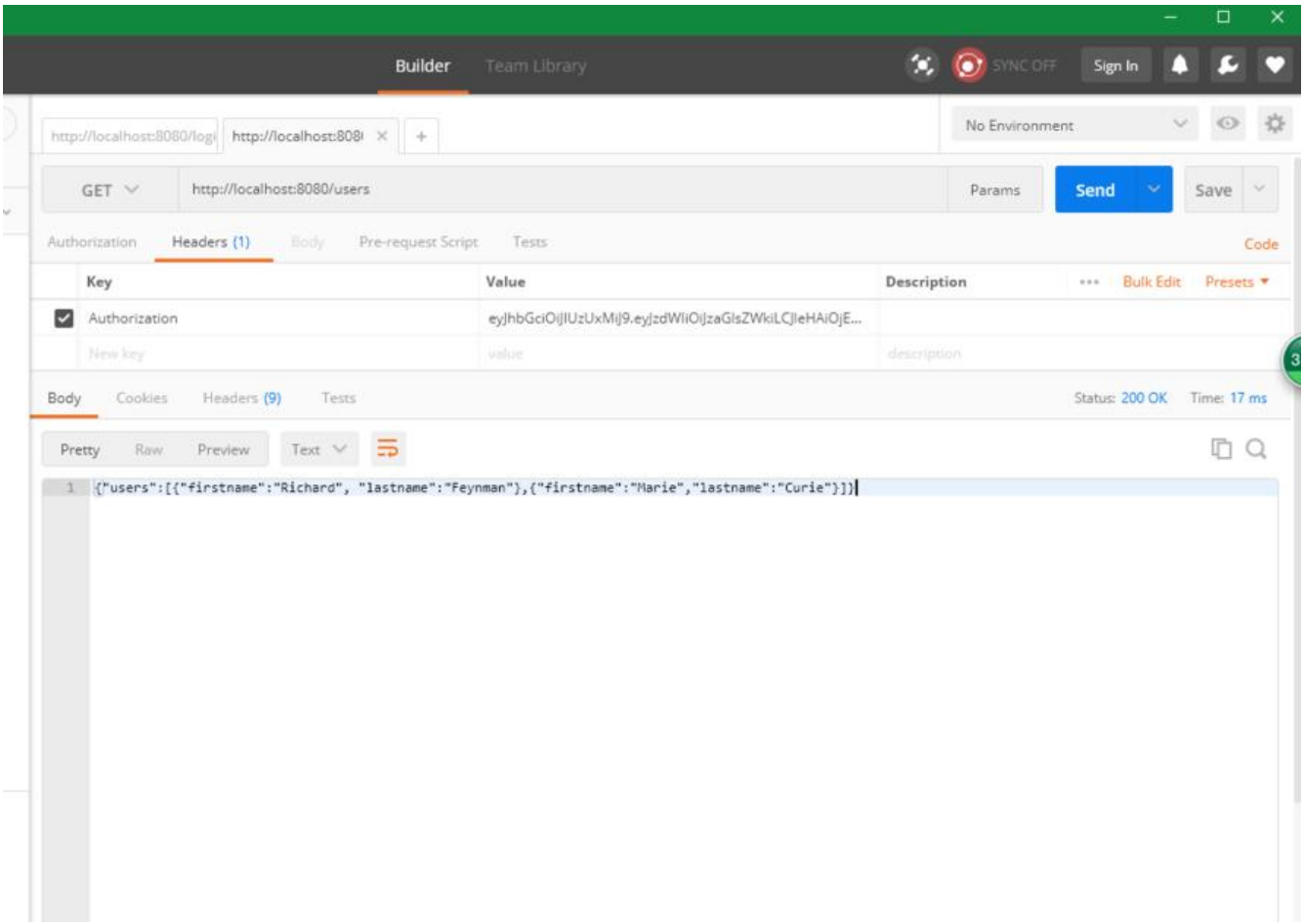
```
            SysUser sysUser2 = new SysUser(2L, "shilei", "nicai");
            sysUser2.setRoles(roles2);

            sysUserRepository.save(Arrays.asList(sysUser1, sysUser2));
            logger.info(">>>>>>>>>>>>>>>初始用户创建结束<<<<<<<<<<<<<");
        }

        logger.info(">>>>>>>>>>>>>>>检查用户系统结束<<<<<<<<<<<<<");

    }
```

如何使用

1. 先获得jwt的token



2. 再请求url



代码

代码在的github里，https://github.com/xjtushilei/spring-boot-security

最后

登录请求一定要使用 HTTPS，否则无论 Token 做的安全性多好密码泄露了也是白搭