

重拾后端之Spring Boot（四）：使用JWT和Spring Security保护REST API

2017 年 03 月 10 日 接灰的电子产品

[重拾后端之Spring Boot（一）：REST API的搭建可以这样简单](#)

[重拾后端之Spring Boot（二）：MongoDb的无缝集成](#)

[重拾后端之Spring Boot（三）：找回熟悉的Controller，Service](#)

[重拾后端之Spring Boot（四）：使用 JWT 和 Spring Security 保护 REST API](#)

通常情况下，把API直接暴露出去是风险很大的，不说别的，直接被机器攻击就喝一壶的。那么一般来说，对API要划分出一定的权限级别，然后做一个用户的鉴权，依据鉴权结果给予用户开放对应的API。目前，比较主流的方案有几种：

1. 用户名和密码鉴权，使用Session保存用户鉴权结果。
2. 使用OAuth进行鉴权（其实OAuth也是一种基于Token的鉴权，只是没有规定Token的生成方式）
3. 自行采用Token进行鉴权

第一种就不介绍了，由于依赖Session来维护状态，也不太适合移动时代，新的项目就不要采用了。第二种OAuth的方案和JWT都是基于Token的，但OAuth其实对于不做开放平台的公司有些过于复杂。我们主要介绍第三种：JWT。

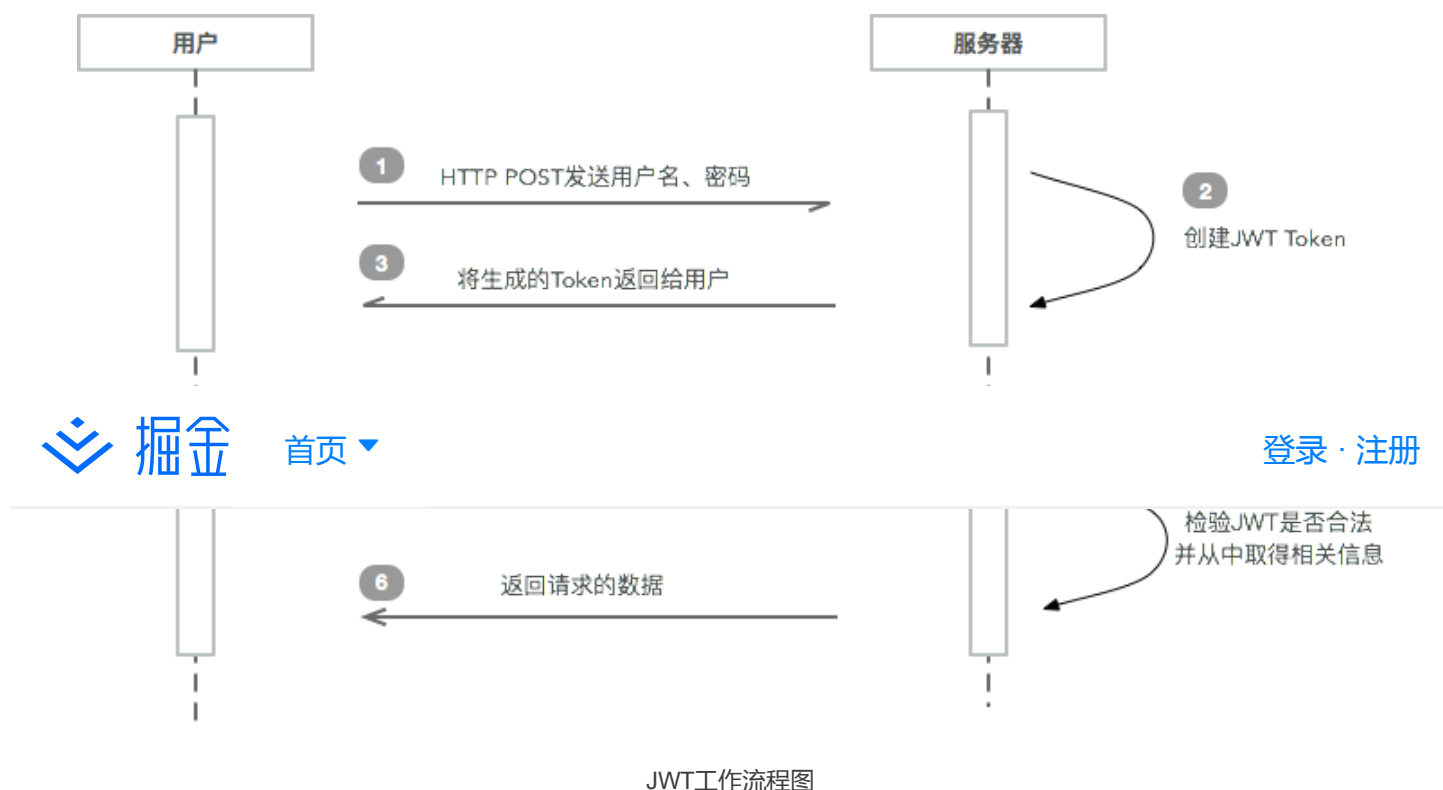
什么是JWT？

JWT是 `Json Web Token` 的缩写。它是基于 [RFC 7519](#) 标准定义的一种可以安全传输的 **小巧** 和 **自包含** 的JSON对象。由于数据是使用数字签名的，所以是可信任的和安全的。JWT可以使用HMAC算法对secret进行加密或者使用RSA的公钥私钥对来进行签名。

JWT的工作流程

下面是一个JWT的工作流程图。模拟一下实际的流程是这样的（假设受保护的API在 `/protected` 中）

1. 用户导航到登录页，输入用户名、密码，进行登录
2. 服务器验证登录鉴权，如果改用户合法，根据用户的信息和服务器的规则生成JWT Token
3. 服务器将该token以json形式返回（不一定要json形式，这里说的是一种常见的做法）
4. 用户得到token，存在localStorage、cookie或其它数据存储形式中。
5. 以后用户请求 `/protected` 中的API时，在请求的header中加入 `Authorization: Bearer xxxx(token)`。此处注意token之前有一个7字符长度的 `Bearer`
6. 服务器端对此token进行检验，如果合法就解析其中内容，根据其拥有的权限和自己的业务逻辑给出对应的响应结果。
7. 用户取得结果



JWT工作流程图

为了更好的理解这个token是什么，我们先来看一个token生成后的样子，下面那坨乱糟糟的就是了。

```
eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJ3YW5nIiwiaWY3JlYXRlZCI6MTQ4OTA3OTk4MTM5MywiZXhwIjoxNDg5Njg0NzgxfQ.f
```

但仔细看到的话还是可以看到这个token分成了三部分，每部分用 `.` 分隔，每段都是用 [Base64](https://www.base64decode.org/) 编码的。如果我们用一个Base64的解码器的话（<https://www.base64decode.org/>），可以看到第一部分 `eyJhbGciOiJIUzUxMiJ9` 被解析成了：

```
{
  "alg": "HS512"
}
```

这是告诉我们HMAC采用HS512算法对JWT进行的签名。

第二部分 `eyJzdWIiOiJ3YW5nIiwiaWY3JlYXRlZCI6MTQ4OTA3OTk4MTM5MywiZXhwIjoxNDg5Njg0NzgxfQ` 被解码之后是

```
{
  "sub": "wang",
  "created": 1489079981393,
  "exp": 1489684781
}
```

这段告诉我们这个Token中含有的数据声明（Claim），这个例子里面有三个声明：`sub`，`created` 和 `exp`。在我们这个例子中，分别代表着用户名、创建时间和过期时间，当然你可以把任意数据声明在这里。

看到这里，你可能会想这是个什么鬼token，所有信息都透明啊，安全怎么保障？别急，我们看看token的第三段 `RC-BYCe UZ2URtWddUpWXIp4NMsoeq206UF-8tVplqXY1-CI9u1-a-`

最后一段其实是签名，这个签名必须知道密钥才能计算。这个也是JWT的安全保障。这里提一点注意事项，由于数据声明（Claim）是公开的，千万不要把密码等敏感字段放进去，否则就等于是公开给别人了。

也就是说JWT是由三段组成的，按官方的叫法分别是header（头）、payload（负载）和signature（签名）：

```
header.payload.signature
```

头中的数据通常包含两部分：一个是我们刚刚看到的 `alg`，这个词是 `algorithm` 的缩写，就是指明算法。另一个可以添加的字段是token的类型(按RFC 7519实现的token机制不只JWT一种)，但如果我们采用的是JWT的话，指定这个就多余了。

```
{  
  "alg": "HS512",  
  "typ": "JWT"  
}
```

payload中可以放置三类数据：系统保留的、公共的和私有的：

- 系统保留的声明（Reserved claims）：这类声明不是必须的，但是是建议使用的，包括：`iss` (签发者), `exp` (过期时间), `sub` (主题), `aud` (目标受众)等。这里我们发现都用的缩写的三个字符，这是由于JWT的目标就是尽可能小巧。
- 公共声明：这类声明需要在 [IANA JSON Web Token Registry](#) 中定义或者提供一个URI，因为要避免重名等冲突。
- 私有声明：这个就是你根据业务需要自己定义的数据了。

签名的过程是这样的：采用header中声明的算法，接受三个参数：base64编码的header、base64编码的payload和密钥（secret）进行运算。签名这一部分如果你愿意的话，可以采用RSASHA256的方式进行公钥、私钥对的方式进行，如果安全性要求的高的话。

```
HMACSHA256(  
  base64UrlEncode(header) + "." +  
  base64UrlEncode(payload),  
  secret)
```



为了简化我们的工作，这里引入一个比较成熟的JWT类库，叫 `jjwt` (<https://github.com/jwt/jjwt>)。这个类库可以用于Java和Android的JWT token的生成和验证。

JWT的生成可以使用下面这样的代码完成：

```
String generateToken(Map<String, Object> claims) {  
    return Jwts.builder()  
        .setClaims(claims)  
        .setExpiration(generateExpirationDate())
```

```
.signWith(SignatureAlgorithm.HS512, secret) //采用什么算法是可以自己选择的，不一定非要采用  
.compact();  
}
```

数据声明（Claim）其实就是一个Map，比如我们想放入用户名，可以简单的创建一个Map然后put进去就可以了。

```
Map<String, Object> claims = new HashMap<>();  
claims.put(CLAIM_KEY_USERNAME, username());
```

解析也很简单，利用 `jjwt` 提供的parser传入秘钥，然后就可以解析token了。

```
Claims getClaimsFromToken(String token) {  
    Claims claims;  
    try {  
        claims = Jwts.parser()  
            .setSigningKey(secret)  
            .parseClaimsJws(token)  
            .getBody();  
    } catch (Exception e) {  
        claims = null;  
    }  
    return claims;  
}
```

JWT本身没啥难度，但安全整体是一个比较复杂的事情，JWT只不过提供了一种基于token的请求验证机制。但我们的用户权限，对于API的权限划分、资源的权限划分，用户的验证等等都不是JWT负责的。也就是说，请求验证后，你是否有权限看对应的内容是由你的用户角色决定的。所以我们这里要利用Spring的一个子项目Spring Security来简化我们的工作。

Spring Security

[登录](#) · [注册](#)

展开讲这个框架，而是如何利用Spring Security和JWT一起来完成API保护。所以关于Spring Security的基础内容或展开内容，请自行去官网学习（<http://projects.spring.io/spring-security/>）。

简单的背景知识

如果你的系统有用户的概念的话，一般来说，你应该有一个用户表，最简单的用户表，应该有三列：Id，Username和Password，类似下表这种

ID	USERNAME	PASSWORD
10	wang	abcdefg

而且不是所有用户都是一种角色，比如网站管理员、供应商、财务等等，这些角色和网站的直接用户需要的权限可能是不一样的。那么我们就需要一个角色表：

ID	ROLE
10	USER
20	ADMIN

当然我们还需要一个可以将用户和角色关联起来建立映射关系的表。

USER_ID	ROLE_ID
10	10
20	20

这是典型的一个关系型数据库的用户角色的设计，由于我们要使用的MongoDB是一个文档型数据库，所以让我们重新审视一下这个结构。

这个数据结构的优点在于它避免了数据的冗余，每个表负责自己的数据，通过关联表进行关系的描述，同时也保证的数据的完整性：比如当你修改角色名称后，没有脏数据的产生。

但是这种事情在用户权限这个领域发生的频率到底有多少呢？有多少人每天不停的改的角色。

我们可以将其简化为

```
{
  _id: <id_generated>
  username: 'user',
  password: 'pass',
  roles: ['USER', 'ADMIN']
}
```

基于以上考虑，我们重构一下 `User` 类，

```
@Data
public class User {
    @Id
    private String id;

    @Indexed(unique=true, direction= IndexDirection.DESENDING, dropDups=true)
    private String username;

    private String password;
    private String email;
    private Date lastPasswordResetDate;
    private List<String> roles;
}
```

当然你可能发现这个类有点怪，只有一些field，这个简化的能力是一个叫 `lombok` 类库提供的，这个很多开发过Android的童鞋应该熟悉，是用来简化POJO的创建的一个类库。简单说一下，采用 `lombok` 提供的 `@Data` 修饰符后可以简写成，原来的一坨getter和setter以及constructor等都不需要写了。类似的 `Todo` 可以改写成：

```
@Data
public class Todo {
    @Id private String id;
    private String desc;
    private boolean completed;
    private User user;
}
```

增加这个类库只需在 `build.gradle` 中增加下面这行



}

引入Spring Security

要在Spring Boot中引入Spring Security非常简单，修改 `build.gradle`，增加一个引用

`org.springframework.boot:spring-boot-starter-security`：

```
dependencies {
    compile("org.springframework.boot:spring-boot-starter-data-rest")
}
```

```

compile("org.springframework.boot:spring-boot-starter-data-mongodb")
compile("org.springframework.boot:spring-boot-starter-security")
compile("io.jsonwebtoken:jjwt:${jjwtVersion}")
compile("org.projectlombok:lombok:${lombokVersion}")
testCompile("org.springframework.boot:spring-boot-starter-test")
}

```

你可能发现了，我们不只增加了对Spring Security的编译依赖，还增加 `jjwt` 的依赖。

Spring Security需要我们实现几个东西，第一个是UserDetails：这个接口中规定了用户的几个必须要有的方法，所以我们创建一个JwtUser类来实现这个接口。为什么不直接使用User类？因为这个UserDetails完全是为了安全服务的，它和我们的领域类可能有部分属性重叠，但很多的接口其实是安全定制的，所以最好新建一个类：

```

public class JwtUser implements UserDetails {
    private final String id;
    private final String username;
    private final String password;
    private final String email;
    private final Collection<? extends GrantedAuthority> authorities;
    private final Date lastPasswordResetDate;

    public JwtUser(
        String id,
        String username,
        String password,
        String email,
        Collection<? extends GrantedAuthority> authorities,
        Date lastPasswordResetDate) {
        this.id = id;
        this.username = username;
        this.password = password;

```



```

}
//返回分配给用户的角色列表
@Override
public Collection<? extends GrantedAuthority> getAuthorities() {
    return authorities;
}

@JsonIgnore
public String getId() {

```



```
        return id;
    }

    @JsonIgnore
    @Override
    public String getPassword() {
        return password;
    }

    @Override
    public String getUsername() {
        return username;
    }
    // 账户是否未过期
    @JsonIgnore
    @Override
    public boolean isAccountNonExpired() {
        return true;
    }
    // 账户是否未锁定
    @JsonIgnore
    @Override
    public boolean isAccountNonLocked() {
        return true;
    }
    // 密码是否未过期
    @JsonIgnore
    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }
    // 账户是否激活
    @JsonIgnore
```

```
        return true;
    }
    // 这个自定义的，返回上次密码重置日期
    @JsonIgnore
    public Date getLastPasswordResetDate() {
        return lastPasswordResetDate;
    }
}
```


这个接口中规定的很多方法我们都简单粗暴的设成直接返回某个值了，这是为了简单起见，你在实际开发环境中还是要根据具体业务调整。当然由于两个类还是有一定关系的，为了写起来简单，我们写一个工厂类来由领域对象创建 `JwtUser`，这个工厂就叫 `JwtUserFactory` 吧：

```
public final class JwtUserFactory {

    private JwtUserFactory() {
    }

    public static JwtUser create(User user) {
        return new JwtUser(
            user.getId(),
            user.getUsername(),
            user.getPassword(),
            user.getEmail(),
            mapToGrantedAuthorities(user.getRoles()),
            user.getLastPasswordResetDate()
        );
    }

    private static List<GrantedAuthority> mapToGrantedAuthorities(List<String> authorities) {
        return authorities.stream()
            .map(SimpleGrantedAuthority::new)
            .collect(Collectors.toList());
    }
}
```

第二个要实现的是 `UserDetailsService`，这个接口只定义了一个方法 `loadUserByUsername`，顾名思义，就是提供一种从用户名可以查到用户并返回的方法。注意，不一定是数据库哦， 文本文件、xml文件等等都可能成为数据源，这也是为什么Spring提供这样一个接口的原因：保

[首页](#) ▼[登录](#) · [注册](#)

```
@Service
public class JwtUserDetailsService implements UserDetailsService {

    @Autowired
    private UserRepository userRepository;

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        User user = userRepository.findByUsername(username);
```

```
if (user == null) {
    throw new UsernameNotFoundException(String.format("No user found with username '%s'."));
} else {
    return JwtUserFactory.create(user);
}
}
```

为了让Spring可以知道我们想怎样控制安全性，我们还需要建立一个安全配置类

WebSecurityConfig：

```
@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class WebSecurityConfig extends WebSecurityConfigurerAdapter{

    // Spring会自动寻找同样类型的具体类注入，这里就是JwtUserDetailsServiceImpl了
    @Autowired
    private UserDetailsService userDetailsService;

    @Autowired
    public void configureAuthentication(AuthenticationManagerBuilder authenticationManagerBuilder
        authenticationManagerBuilder
        // 设置UserDetailsService
        .userDetailsService(this.userDetailsService)
        // 使用BCrypt进行密码的hash
        .passwordEncoder(passwordEncoder());
    }
    // 装载BCrypt密码编码器
    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
}
```



首页 ▾

登录 · 注册

API

程序员

数据库

Spring



196

分享到



...

参加掘金征文赢取千元奖金

本期的征文主题是：聊聊你的最佳实践，程序员在开发过程中，随着经验的积累，往往会形成对某类难题的最佳解决方式，这种方式通常比其他解决方式都更有效，甚至成为了某种问题的标准解决方案。所以本期主题，我们希望大家来分享一下自己的最佳实践。

相关文章

炮灰都不如 ---我们如何看待大学计算机教育

大漠穷秋 ❤️ 94 🗨️ 90

典型数据库架构设计与实践

彷徨 ❤️ 10

破解前端面试系列（3）：如何搞定纸上...

王仕军 ❤️ 120 🗨️ 19

沸点：你小时候有哪些愿望？

稀土君 ❤️ 4 🗨️ 58

git 干货系列：（四）我要连接远程仓库...

嘟嘟MD ❤️ 14

评论

说说你的看法

接灰的电子产品 大叔级码农 @ 终身学习

回复[接灰的电子产品](#)：把 branch 的名字也改正了，还是 chap04

▲ 0 回复 2月前



首页 ▾

登录 · 注册

chap03 的代码吧

▲ 0 回复 2月前

接灰的电子产品 大叔级码农 @ 终身学习

回复[Dog-Lee](#)：不全？不会吧

▲ 0 回复 2月前

Dog-Lee

回复[Dog-Lee](#)：求楼主补全这章的代码

▲ 0 回复 2月前

Dog-Lee

哭。github 代码补全，不对套路啊

▲ 0 回复 2月前

阿卡林

回复[接灰的电子产品](#)：没，只是当时做大作业，前后端分离，前端用 React，后端 Spring Data REST。认证部分 JWT 什么资料都查不到，就没做了 Ծ_Ծ

▲ 0 回复 2月前

接灰的电子产品 大叔级码农 @ 终身学习

回复[阿卡林](#)：有什么故事吗？

▲ 0 回复 2月前

接灰的电子产品 大叔级码农 @ 终身学习

回复[总是要增肥的大男人](#)：写一个类似 ExceptionTranslationFilter 的自定义 filter，但注意这个 filter 的顺序要放在最后

▲ 0 回复 2月前

阿卡林

这篇文章如果能早一学年看到多好。

▲ 0 回复 2月前

总是要增肥的大男人 Android @ tiger

我知道错误处理能这么处理，但是我用 Spring Security 根本无法抛出自定义的异常

▲ 0 回复 2月前

接灰的电子产品 大叔级码农 @ 终身学习

一年前自己搭建的 Spring Boot 项目，现在可以部署到生产环境了，分享给大家，希望对大家有所帮助。



首页 ▾

登录 · 注册

// }

▲ 0 回复 2月前

总是要增肥的大男人 Android @ tiger

你好，我在处理 Spring Security 出错的时候，跳到了 403 页面，但是我怎么获取处理出错的不同错误类型，这样我才能给移动端返回不同状态码，因为我的检验都是用 Spring Security 完成的

▲ 0 回复 2月前

Geekbing

就是对这块比较模糊，看后觉得豁然开朗

▲ 0 回复 2月前

Geekbing

写得太棒了

▲ 0 回复 2月前

接灰的电子产品 大叔级码农 @ 终身学习

回复yxc2016：这个得看你自己的策略是什么样的，比如可以在临近过期时刷新 token，或者过期后重新登录等。但这个就不是 jwt 或 spring 的问题了，而是你要制定什么策略

▲ 0 回复 2月前

yxc2016

客户端对 token 的过期怎么处理

▲ 0 回复 2月前



高质量的技术分享社区

开发者 · 设计师 · 产品经理

立即下载



首页 ▼

登录 · 注册