爆破专栏丨Spring Security系列教程之基于默认数据库模型实现授权

原创 一一哥 Java架构栈 9月24日

前言

在上一个章节中，**一一哥**给大家讲解了如何基于内存模型来实现授权，在这种模型里，用户的信息是保存在内存中的。你知道，保存在内存中的信息，是无法持久化的，也就是程序一旦关闭，或者断电等情况发生，内存中的信息就丢失了，所以这种方式并不适用于生产环境！

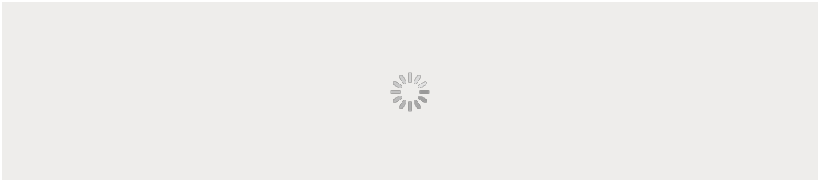
那么我们肯定要把用户信息持久化，**但是持久化到哪里去呢？那就是数据库呗！**数据库是我们做程序员必会必熟的知识点，尤其是做后端开发，开发时常用的数据库有MySQL和Oracle，本案例中我们采用的是MySQL数据库。

一. JdbcUserDetailsManager类介绍

在进行编码实现之前，**壹哥**先和各位一起来看看Spring Security给我们提供的持久化API都有哪些，以及这些API之间的关系结构。

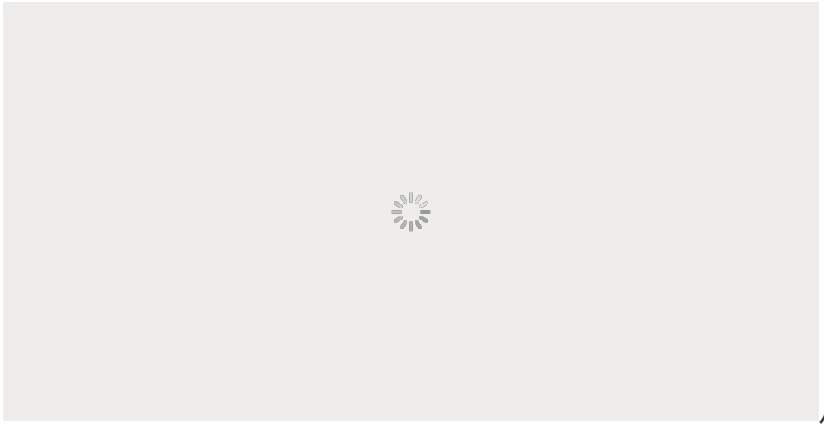
1. UserDetailsService接口

Spring Security 支持MySQL、Oracle等多种不同的数据源，这些不同的数据源最终都由 UserDetailsService 这个接口的子类来负责进行操作，我们先来看看 UserDetailsService 接口都有哪些实现类:



2. JdbcUserDetailsManager实现类

在上一章节中，我给大家介绍了如何在内存中创建并保存用户及角色信息，这种实现方式主要是利用InMemoryUserDetailsManager这个子类来实现的。如果我们想在数据库中创建并保存用户及角色信息，Spring Security也很贴心，它还给我们提供了另一个UserDetailsService的实现子类，也就是JdbcUserDetailsManager。其中JdbcUserDetailsManager类的关系结构如下图所示：



从上图中，我们可以看到JdbcUserDetailsManager的直接父类是JDBCDaoSuport，利用JdbcUserDetailsManager可以帮助我们以JDBC的方式对接数据库进行增删改查等操作。它内部设定了一个默认的数据库模型，只要遵从这个模型，我们就可以很方便的实现在数据库中创建用户名和密码、角色等信息，但是灵活性不足。

二. 项目实现

了解完上面的相关API之后，咱们闲言少叙，直接动起手来撸码吧。

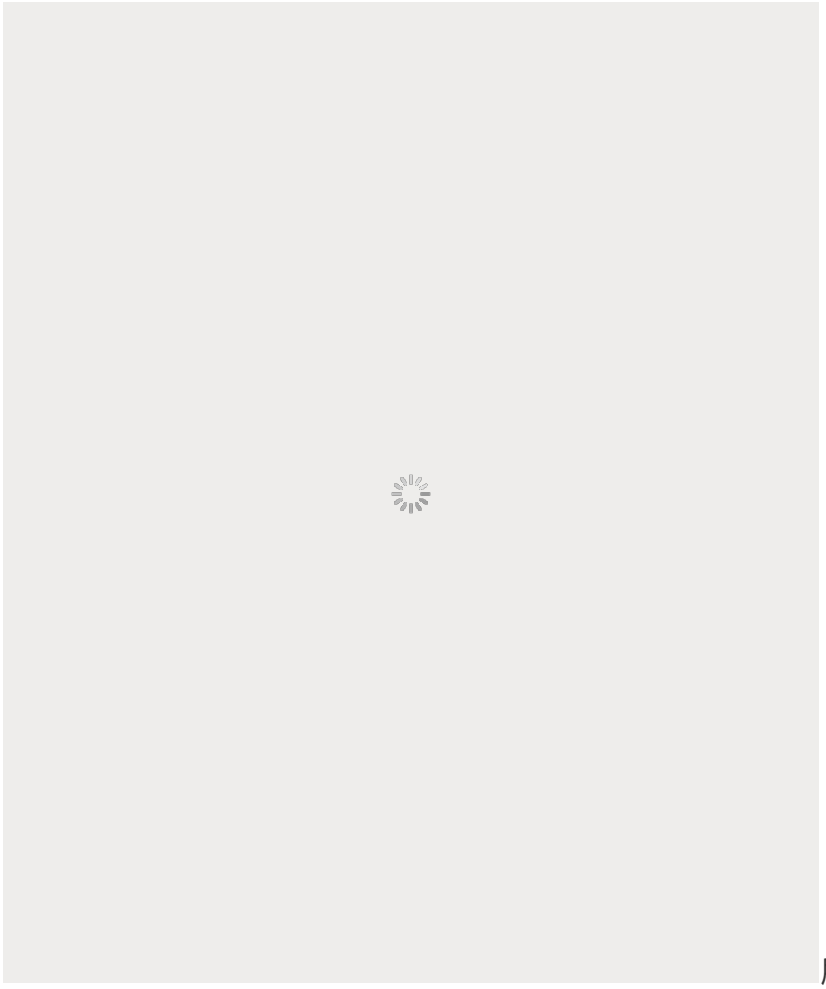
1. 准备测试接口

接下来我们就在上一章节创建的项目基础之上，创建一个新的model，并对项目进行改造。我们还是跟之前一样，先创建3个测试接口，具体过程请参考之前章节。

2. 准备数据库

既然我们要操作数据库，那肯定得先建库建表啊，那么数据库和表结构是什么样呢？我们怎么创建出来呢？其实JdbcUserDetailsManager本身就给我们提供了对应的数据库脚本模型，这个数据库脚本模型保存在如下位置：

```
org/springframework/security/core/userdetails/jdbc/users.dd
```



所以我们直接去这个对应的位置下，找到这个数据库脚本文件打开即可。

3. users数据库脚本

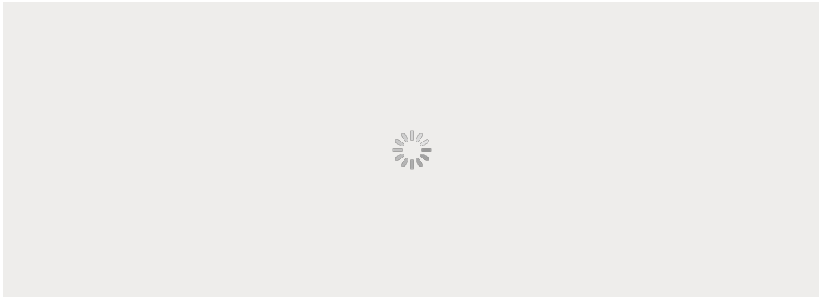
当我们打开这个users.ddl文件，可以看到如下内容，会发现其中有2个建表语句，分别是创建了users表和authorities表，并且在authorities表中创建了唯一索引。

```
create table users(username varchar_ignorecase(50) not null

create table authorities (username varchar_ignorecase(50) n

create unique index ix_auth_username on authorities (userna
```

这时候我们只需要先自己创建一个数据库，编码格式就采用UTF-8，然后把上面的建表语句，复制粘贴并执行，创建出2个表即可。users表用来存放用户名、用户密码以及账户是否可用，authorities表用来存放用户名及其对应权限，authorities 和 users 会通过 username 字段关联起来。

如下图所示： 

注意：上面的建表脚本中，有一种数据类型 varchar_ignorecase，这个是针对 HSQLDB 数据库创建的，但是我们使用的 MySQL 数据库并不支持这种数据类型，所以这里需要我们手动将这个数据类型改为 varchar。

4. 添加数据库依赖

接下来我们在pom.xml文件中，除了已有的spring-web和spring-security的依赖之外，还要再新增关于数据库的依赖包。

```
create table users(username varchar_ignorecase(50) not null

create table authorities (username varchar_ignorecase(50) n

create unique index ix_aut<dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactI
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-security</arti
        </dependency>

        <dependency>
            <groupId>org.mybatis.spring.boot</groupId>
            <artifactId>mybatis-spring-boot-starter</artifa
            <version>1.3.1</version>
        </dependency>

        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <scope>runtime</scope>
        </dependency>
```

```
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artif
            </plugin>
        </plugins>
    </build><dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-security</artif
        </dependency>

        <dependency>
            <groupId>org.mybatis.spring.boot</groupId>
            <artifactId>mybatis-spring-boot-starter</artifa
            <version>1.3.1</version>
        </dependency>

        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <scope>runtime</scope>
        </dependency>

        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
```

```
                <artifactId>spring-boot-maven-plugin</artif
            </plugin>
        </plugins>
    </build>h_username on authorities (username,authority);
```

5. 添加数据库配置文件

添加完依赖包之后，我们在resource目录下创建一个application.yml文件，并在其中添加关于数据库的配置信息。

```
spring:
  datasource:
    url: jdbc:mysql://localhost:3306/db-security02?useUnic
    username: root
    password: syc
```

注意： url中数据库的名字，需要改成自己的数据库名称，用户名和密码也要改成自己mysql的用户信息。

6. 编写编码类

然后我们编写一个SecurityConfig类，在其中配置对资源的访问控制，创建一个UserDetailsService实例，并在其中配置，生成存储用户和角色信息。

```
@EnableWebSecurity(debug = true)
public class SecurityConfig extends WebSecurityConfigurerAd

    @Override
    protected void configure(HttpSecurity http) throws Exce
        http.authorizeRequests()
                .antMatchers("/admin/**")
                .hasRole("ADMIN")
                .antMatchers("/user/**")
                .hasRole("USER")
                .antMatchers("/app/**")
                .permitAll()
                .anyRequest()
                .authenticated()
                .and()
```

```
                .formLogin()
                .permitAll();
    }

    /**********************************在默认的数据库中创

    @Autowired
    private DataSource dataSource;

    @Bean
    public UserDetailsService createUserDetailService() {
        JdbcUserDetailsManager manager = new JdbcUserDetail
        manager.setDataSource(dataSource);

        if (!manager.userExists("user")) {
            manager.createUser(User.withUsername("user").pa
        }

        if (!manager.userExists("admin")) {
            manager.createUser(User.withUsername("admin").p
        }

        return manager;
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        return NoOpPasswordEncoder.getInstance();
    }

}
```

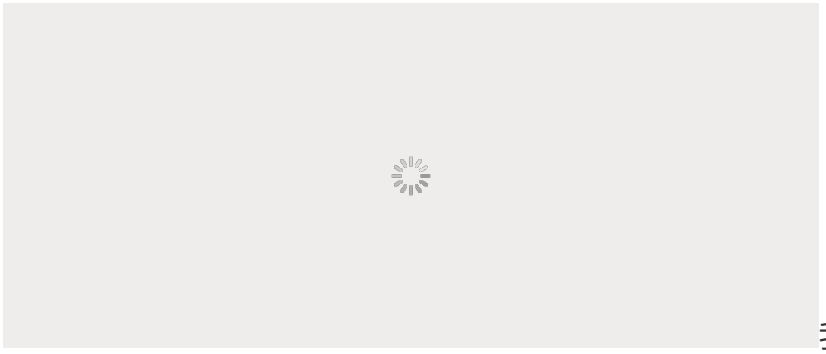
因为我们利用的是SpringBoot环境，只要我们配置好了数据库依赖和环境，就可以直接在这个SecurityConfig配置类中引用DataSource对象。

7. 核心代码释义

对上面SpringSecurity类中的代码，我主要针对createUserDetailService()方法进行解释。

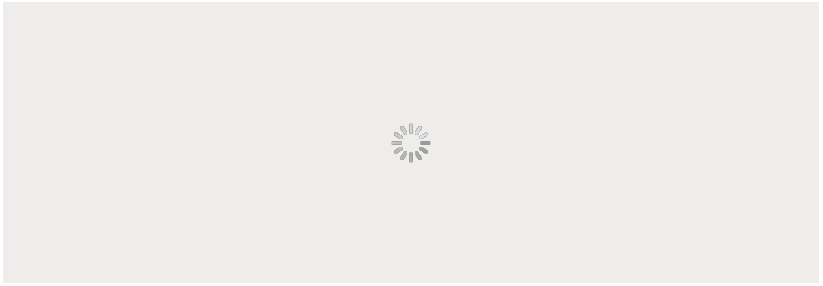
- 在这个方法中，我们首先构建一个 JdbcUserDetailsManager 实例对象，并给 JdbcUserDetailsManager 实例添加一个 DataSource 对象。
- 接下来调用 userExists()方法 判断用户是否存在，如果不存在，就创建一个新的用户出来(因为每次项目启动时这段代码都会执行，所以加一个判断，避免重复创建用户)。
- 用户的创建方法和我们之前 InMemoryUserDetailsManager 中的创建用户的方法基本一致。

以上就是我们基于默认的数据库模型实现的对用户及角色的操作，之所以可以实现，就是因为在UserDetailsManager这个父接口中，定义了如下方法，这些方法在子类中都有具体的实现。



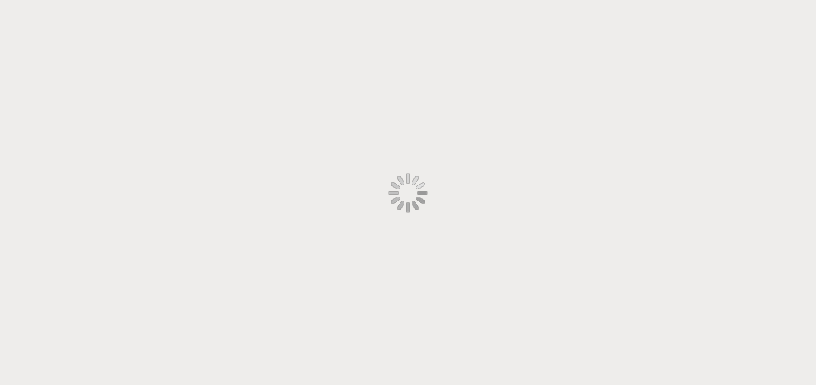
我们可以看到UserDetailsManager类中，提供了创建、修改、删除、判断用户是否存在的方法，和修改用户密码的方法。

在JdbcUserDetailsManager这个实现子类中，已经定义好了users与authorities表对应的CRUD语句，所以我们直接调用相关方法即可实现对用户及角色的管理。另外我们在特殊情况下，也可以自定义这些SQL语句，如有需要，调用对应的setXxxSQL()方法即可。



8. 项目结构

以上案例的完整代码结构如下图所示，请参考创建。



三. 测试运行

1. 启动项目测试

接下来我们把项目启动起来进行测试，这时候效果跟基于内存模型的授权实现效果是一样的，具体测试界面**壹哥**这里就不再展示了。

2. 小结

当我们使用Spring Security默认的数据库模型来操作实现用户授权时，存在灵活性不足的问题，因为我们必须按照源码规定的方式去建库建表。而我们真正开发时，用户角色等表肯定是根据自己的项目需求来单独设计的，所以真正开发时，我们有必要进行用户及角色表的自定义设计。

接下来**壹哥**就会带各位学习更灵活的基于自定义数据库模型的开发实现方式，敬请期待哦。对于本篇不明白的地方，请在评论区留言！

往期精彩

Spring Security系列教程之创建项目(1)

Spring Security系列教程之实现HTTP基本认证(2)

Spring Security系列教程之实现Form表单认证(3)

Spring Security系列之实现HTTP摘要认证(4)

Spring Security系列之前后端分离时的安全处理方案(5)

Spring Security系列教程之基于内存模型实现授权（6）

文末福利

留言评论，今天的内容你学会了吗?

点赞最多的同学获得水杯一个！

收录于话题 #Spring Security爆破专栏·16个 >

上一篇
爆破专栏丨Spring Security系列教之基于自定义数据库模型实现授权

下一篇
爆破专栏丨Spring Security系列教程之基于内存模型实现授权

阅读原文