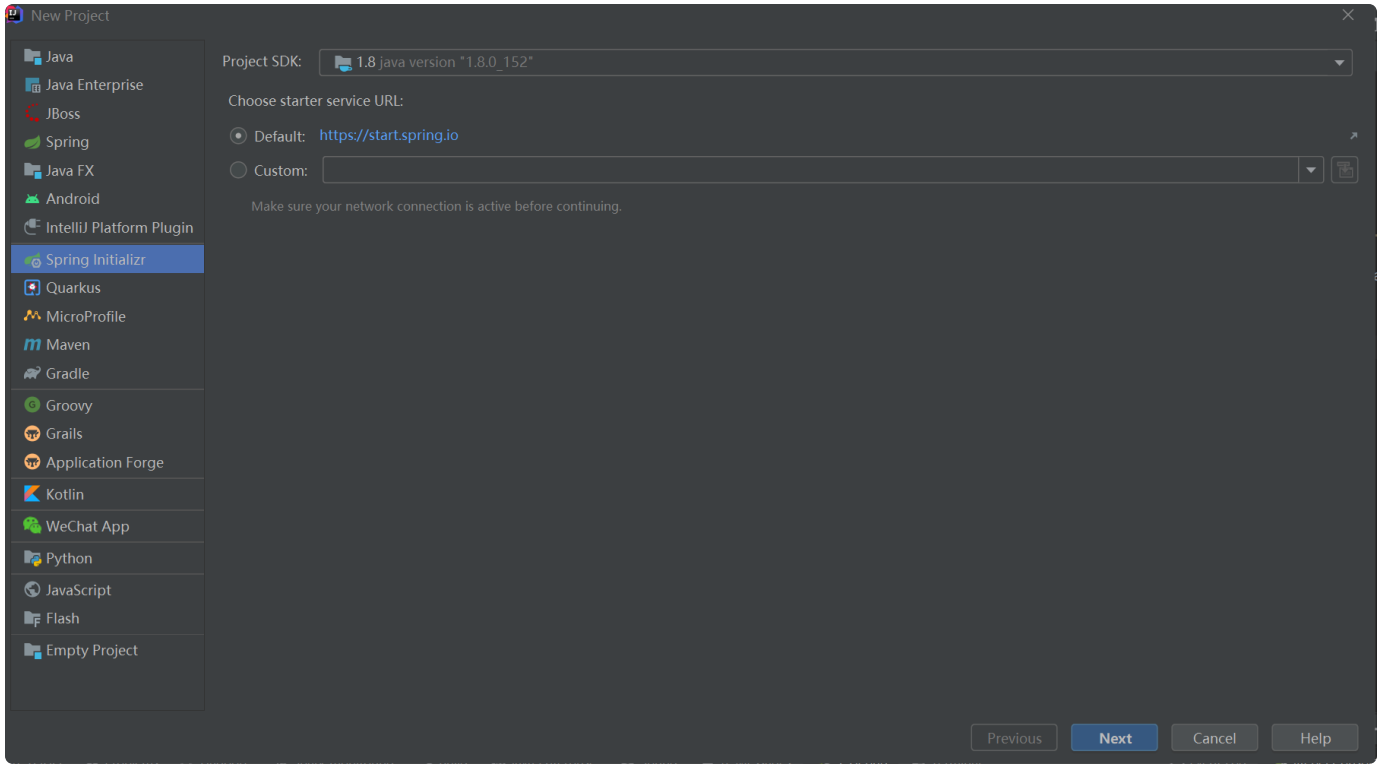
Spring Security快速入门

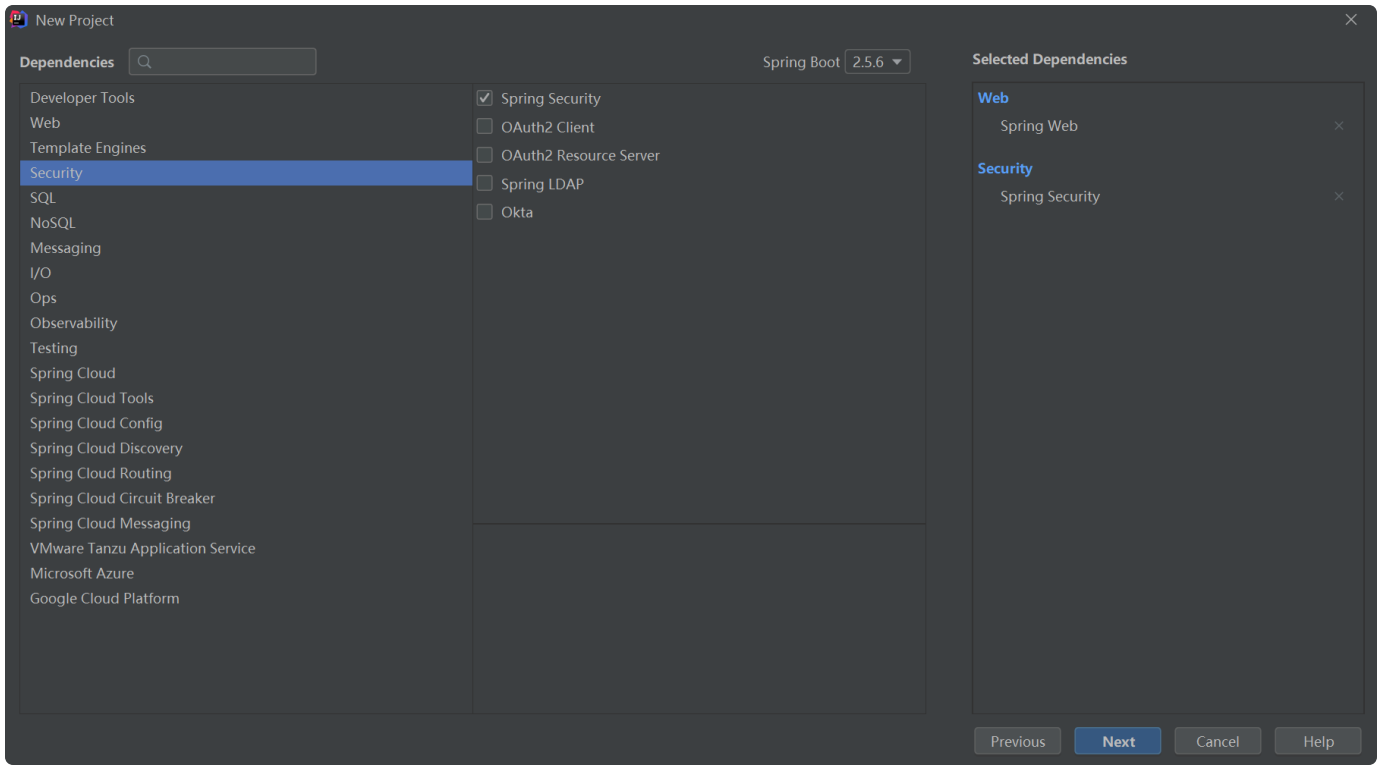
1.什么是Spring security

Spring security是Spring Security是一个能够为基于Spring的企业应用系统提供声明式的安全访问控制解决方案的安全框架。

2.第一个Spring security应用

创建一个Spring boot项目，之后设置项目名、Artificial id 等，只需要导入 **Spring Web** 和 **Spring Security** 两个依赖即可。





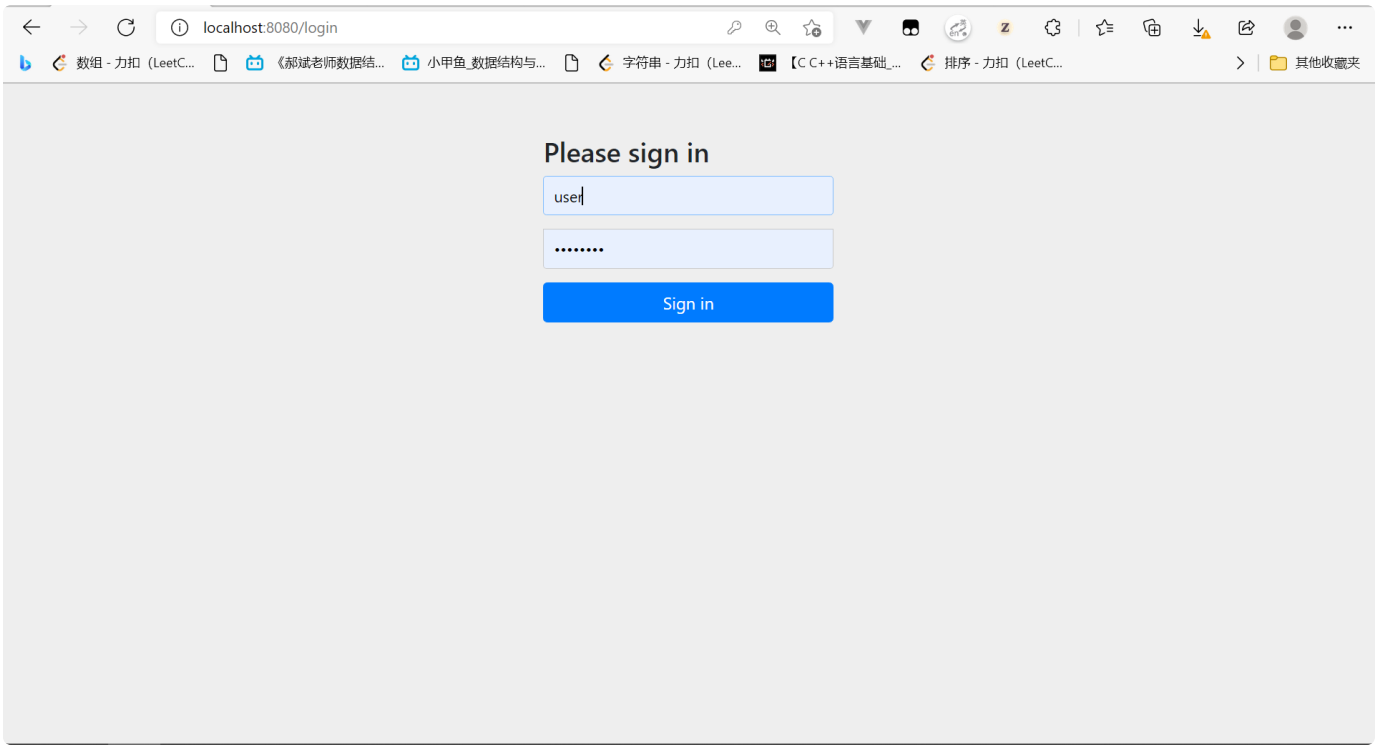
之后创建一个Controller

```java
@RestController
public class HomeController {

    @GetMapping(value = {"/","/home"})
    public String home(){
        return "Hello Spring Security !";
    }

}
```

接下来直接启动项目，我们访问 http://localhost:8080/home ，就会被重定向到Spring Security页面。



非常简单，只需要简单的导入依赖，我们就能达成对所有接口进行保护的目的。

我们没有自定义账号和密码，那么应该怎么登陆呢?
在没有设置密码的情况下，项目会在控制台打印自动生成的密码（如下图），且默认登陆名为user。

```
Using generated security password: 96373168-766c-44cd-a4a3-8e8ca3eae18f
```

使用提供的账号密码登陆成功后，我们就能访问设置的"/home"页面了。这就是第一个Spring security第一个应用。

那么密码是在哪里生成的呢？与用户相关的配置在 **UserDetailsServiceAutoConfiguration** 类中，进入该类，有以下的代码：

```java
        String password = user.getPassword();
        if (user.isPasswordGenerated()) {
            logger.info(String.format("%n%nUsing generated security password: %s%n", user.getPassword()));
        }
```

再进入 **getPassword（）**方法所处的**SecurityProperties** 类中，再User类中，可以发现默认的用户名为"user"且默认会自动密码，且密码是随机生成的UUID码（Universally Unique Identifier，通用唯一标识码）

```java
        private String name = "user";
        private String password = UUID.randomUUID().toString();
        private List<String> roles = new ArrayList();
        private boolean passwordGenerated = true;
```

3.自定义配置

3.1配置文件

使用默认的生成密码的方式，在每次重启项目时，密码都会改变，这样不太方便。我们可以通过以下方式自定义帐号号密码。

对用户信息的配置是在前面提到 **SecurityProperties** 类中完成的，该类配置@ConfigurationProperties配制的前缀为 "spring.security" 。

```java
@ConfigurationProperties(
    prefix = "spring.security"
)
public class SecurityProperties {
```

这样就比较清晰了，我们只需要以"spring.security" 就能在application.yml配置文件中对用户的信息进行配置了

```
spring:
  security:
    user:
      name: user
      password: 123456
```

对于用户的密码是通过set方法注入的，可以了解到如果用户设置了密码，该方法就会设置passwordGenerated 为false，并注入用户自定义的密码。

重启项目，就能用自定义的账号密码登陆。

```
            public void setPassword(String password) {
              if (StringUtils.hasLength(password)) {
                  this.passwordGenerated = false;
                  this.password = password;
              }
          }
```

3.2配置类

如果需要自定义一些配置，则需要编写一个类继承 **WebSecurityConfigurerAdapter** 类并重写其中的方法。例如，我们可以通过以下方式创建用户。

```
@Configuration
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth
            .inMemoryAuthentication()
                .withUser("foo")
                .password("12345")
                .roles("admin");
        //                .and()
        //                .withUser("bar") ...
}
```

1. 首先我们定义了一个 **SecurityConfig** 类继承 **WebSecurityConfigurerAdapter** ，重写里边的configure方法

2. 通过 **inMemoryAuthentication** 在内存中定义用户，withUser 是用户名 ，password 是密码 ， roles 是用户角色。
3. 如果需要配置多个角色，用 and 相连。

此时，通过密码登陆会控制台会报以下异常，原因是Spring boot 2.0引用的security 依赖是 spring security 5.X版本，此版本需要提供一个 **PasswordEncoder** 的实例，否则后台汇报错误。**PasswordEncoder** 中方法的功能是加密密码和解密密码及校验。

Plain Text 复制代码

```
java.lang.IllegalArgumentException: There is no PasswordEncoder mapped for the id "null"
```

PasswordEncorder 是一个接口类，该接口只有三个方法

Java 复制代码

```java
publicinterface PasswordEncoder {
  String encode(CharSequence rawPassword);
  boolean matches(CharSequence rawPassword, String encodedPassword);
  default boolean upgradeEncoding(String encodedPassword) {
    return false;
  }
}
```

1. encode 方法对明文密码进行加密，返回加密后的密文
2. matches 方法用于校验密码是否正确，以用户输入的明文密码和数据库中保存的密文作为参数，通过返回的布尔值判断密码是否正确
3. upgradeEncoding 是否再次加密，默认是false，一般不使用

Spring Security 官方提供了许多多种加密方案，其中推荐使用 **BCryptPasswordEncoder** ，**BCrypt** 是一种加盐的单向hash，不可逆的加密算法，同一种明文每次加密生成的密文都不同，且不可反向破解生成密文。相较常见的**MD5** 算法，**BCrypt** 具有以下特点:

1. BCrypt 生成生成的密文是60位的，MD5 生成的密文是32位
2. BCrypt比MD5更安全
3. Bcrypt是种慢哈希算法，执行时间较长，暴力破解的时间成本更高

为了使用官方提供的BCrypt加密算法，我们只需要在上面的 **SecurityConfig** 类中加入以下代码。

```java
    @Bean
    public PasswordEncoder passwordEncoder(){
        return new BCryptPasswordEncoder();
    }
```

需要注意到，我们不应该使用 **NoOpPasswordEncoder** ，因为该编码器并未对密码进行加密，其中的 encode方法只是将字符序列转化为字符串，也就是说仍然是以明文存储密码，这样与框架的设计理念不符。在Spring Security5.0之前，它是作为未主动设置密码编码器时的默认选择，现在该类已被弃用。

此外，由于我们设置了解码器为 **BCryptPasswordEncoder**，我们不能以明文的方式设置密码，而应该使用编码器中的 encode 方法对密码进行加密。故 **SecurityConfig** 的完整代码如下

```java
@Configuration
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Bean
    public PasswordEncoder passwordEncoder(){
        return new BCryptPasswordEncoder();
    }

    @Autowired
    PasswordEncoder passwordEncoder;

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth
            .inMemoryAuthentication()
                .withUser("foo")
                .password(passwordEncoder.encode("12345"))
                .roles("admin");

//                .and()
//                .withUser("bar") ...

    }
}
```

配置完成后，重启项目即可以设置的账号密码登陆。

4.自定义表单登陆页

这里使用Thymeleft模板创建登陆页面，导入以下依赖即可。

Plain Text 复制代码

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

4.1服务器端设置

我们继续在前面的 SecurityConfig 类中完善代码

Java 复制代码

```java
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.authorizeRequests()
                .anyRequest().authenticated()
                .and()
                .formLogin()
                .loginPage("/login.html")
                .permitAll()
                .and()
                .csrf().disable();
    }
```