快速打造企业级声明式安全认证授权解决方案之SpringBoot+SpringSecurity

# 项目目标

spring boot的介绍

spring boot的入门

spring boot属性配置

springboot 集成mybatis

spring security介绍

spring security 入门

spring security 自定义登录认证、自定义用户认证、加密处理

# spring boot介绍

## spring boot是什么?

Spring Boot 是由 Pivotal 团队提供的全新框架，其设计目的是用来简化Spring应用的初始搭建以及开发过程。

Spring Boot可以轻松创建独立的，生产级的基于Spring的应用程序。

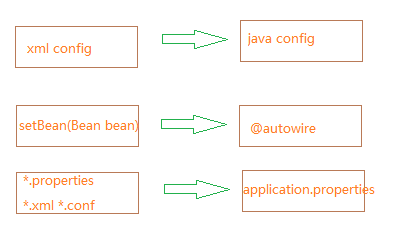
## spring boot解决了哪些问题？

* spring boot 使编码变的更简单
* spring boot 使配置变的更简单
* spring boot 使部署变的更简单
* 。。。。

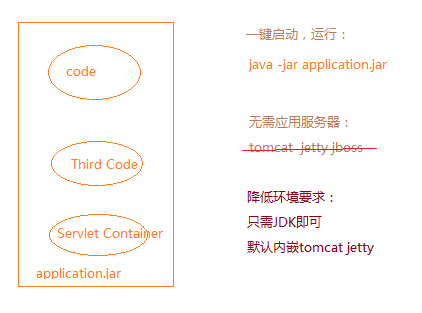
### 使编码变的简单



### 使得配置变的简单



### 使得部署变的简单



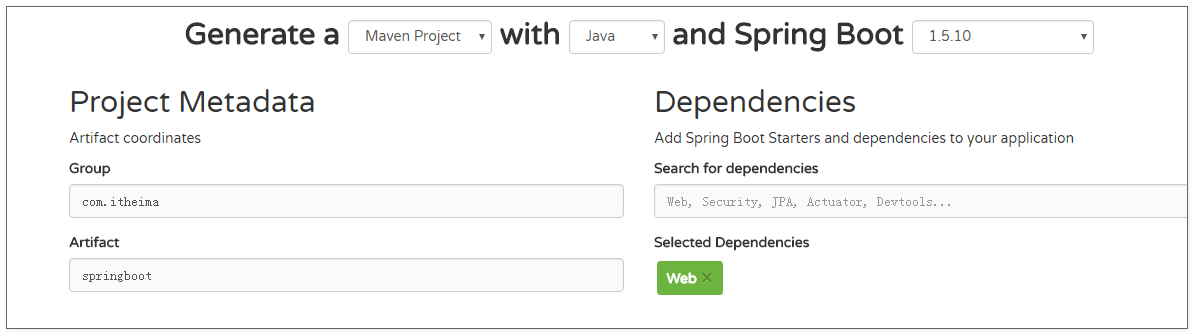
# Spring Boot入门

## Spring boot环境搭建

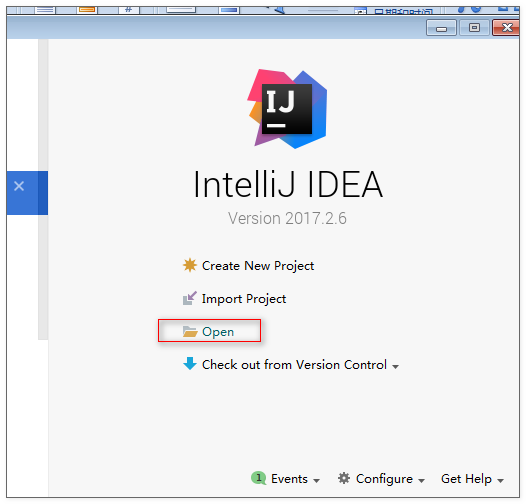
**环境说明：Idea + maven+jdk1.8+mysql**

搭建步骤：

一：浏览器输入：start.spring.io



二：拷贝生成的项目，并使用Idea打开即可



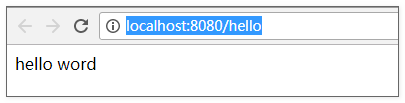
## Hello World

第一个：创建Controller

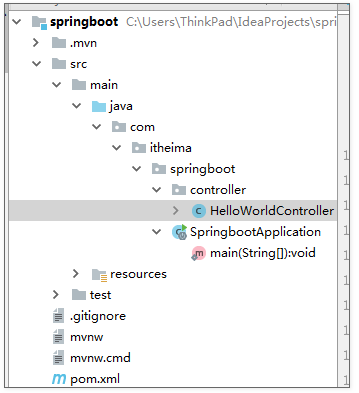
|  |
| --- |
| @Controller **public class** HelloWorldController {   @RequestMapping(**"/hello"**)  @ResponseBody  **public** String showHelloWolrd(){  **return "hello word"**;  } } |

第二个：运行main方法

第三个：浏览器输入地址：http://localhost:8080/hello



项目结构如下：



总结：

1.创建一个maven工程，然后添加依赖即可（添加parent ,添加相关的启动器，例如：添加web）

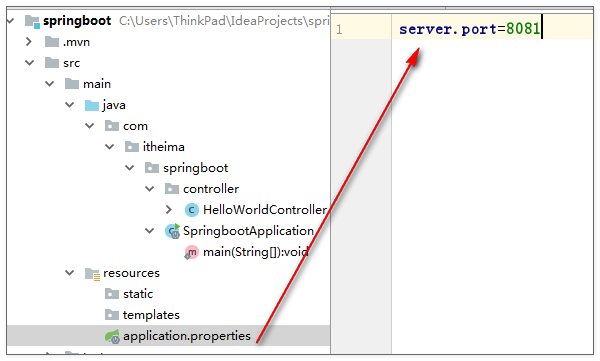
2.创建一个引导类（SpringBootAplication.class）

3.创建Controller即可。

# Spring Boot的属性配置

## application.properties

默认的情况下，springboot 所有的属性配置都在application.properties中进行配置，我们可以在里面添加必要的属性及对应的值，比如：默认的情况下 启动项目，使用端口8080,可以设置端口为其他。例如：



访问时即可 使用8081.

其他的属性配置，参考如下文档：

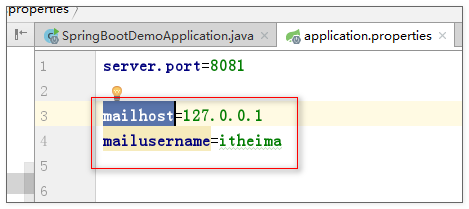
https://docs.spring.io/spring-boot/docs/current/reference/html/common-application-properties.html

## 自定义属性值及获取

有时需要自定义配置属性文件，设置值，并在代码中使用它，可以在application.properties中设置。

例如：

* 定义值：



* 获取值：

直接通过@value注解获取即可。

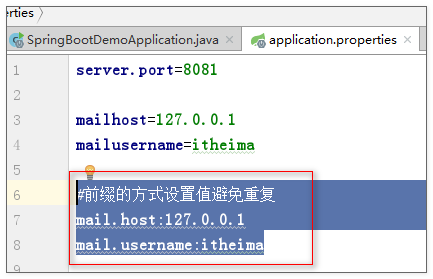


## java config获取属性值

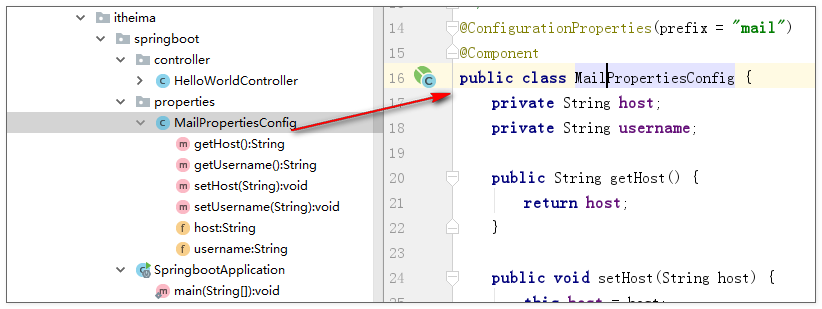
由于如果属性值比较多，一个一个通过@value注解的方式来获取，比较繁琐，可以通过java类进行映射，将属性值映射到java类中，并交给spring管理，使用时直接注入即可，比较方便。

例如：

第一个：定义属性：



第二个：定义映射属性的类： 并设置getter和setter方法



|  |
| --- |
| @ConfigurationProperties(prefix = **"mail"**) @Component **public class** MailPropertiesConfig {  **private** String **host**;  **private** String **username**;   **public** String getHost() {  **return host**;  }   **public void** setHost(String host) {  **this**.**host** = host;  }   **public** String getUsername() {  **return username**;  }   **public void** setUsername(String username) {  **this**.**username** = username;  } } |

第三个：使用属性文件中的值：直接注入即可。



# mybatis的集成

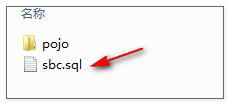
在正式的项目中，需要将数据从数据库中查询出来展示到前台相关的业务逻辑中。所有持久层的技术必不可少，这里我们介绍现阶段比较流行的持久层框架mybatis集成到springboot中。

## mybatis集成的环境搭建

第一个：添加mybatis、mysql的依赖坐标：

|  |
| --- |
| <**dependency**>  <**groupId**>org.mybatis.spring.boot</**groupId**>  <**artifactId**>mybatis-spring-boot-starter</**artifactId**>  <**version**>1.3.1</**version**> </**dependency**>  <**dependency**>  <**groupId**>mysql</**groupId**>  <**artifactId**>mysql-connector-java</**artifactId**>  <**scope**>runtime</**scope**> </**dependency**> |

第二个：创建数据库实例，并配置连接信息



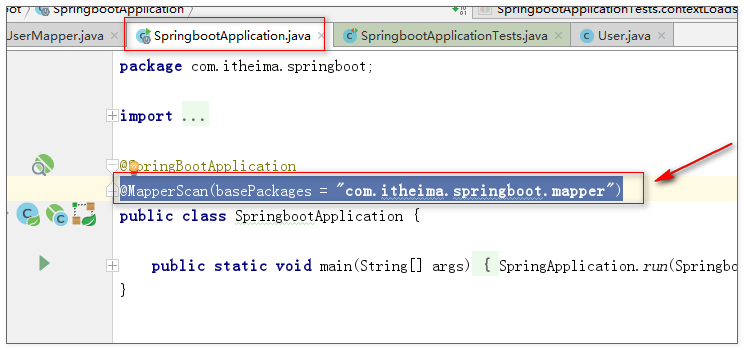
创建数据库实例，并导入如上图所示的sql。

|  |
| --- |
| **spring.datasource.driverClassName**=**com.mysql.jdbc.Driver spring.datasource.url**=**jdbc:mysql://127.0.0.1:3306/sbc spring.datasource.username**=**root spring.datasource.password**=**itcast** |

第三个:创建mapper(dao)接口

|  |
| --- |
| **public interface** UserMapper {   @Select(value = **"select \* from user where username=#{username}"**)  **public** User selectByUsername(String username); } |

第四个：扫描mapper接口所在的包，将由spring创建这些接口的代理对象



第五：测试，在test目录下实现测试

|  |
| --- |
| @RunWith(SpringRunner.**class**) @SpringBootTest **public class** SpringbootApplicationTests {   @Autowired  **private** UserMapper **mapper**;  @Test  **public void** contextLoads() {  System.***out***.println(**mapper**.selectByUsername(**"zhangsan"**));  }  } |

测试OK

集成mybatis

总结：

1.添加mybaits的启动器的依赖，

2.添加mysql的依赖

3.配置连接池的信息（application.properties配置连接数据库的用户名和密码以及数据库驱动，URL）

4.编写mapper接口（相当于是Dao）

5.测试

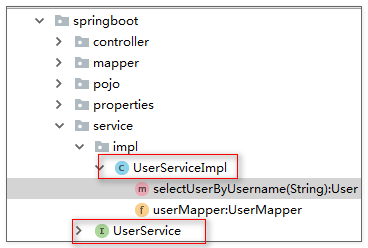
## 页面展示用户相关信息

### 编写service 接口及实现类

|  |
| --- |
| **public interface** UserService {  **public** User selectUserByUsername(String username); } |

|  |
| --- |
| @Service **public class** UserServiceImpl **implements** UserService {  @Autowired  **private** UserMapper **userMapper**;  @Override  **public** User selectUserByUsername(String username) {  **return userMapper**.selectByUsername(username);  } } |

结构如下图：



### 编写controller

|  |
| --- |
| @Controller **public class** UserController {  @Autowired  **private** UserService **userService**;   @RequestMapping(**"/user/info"**)  @ResponseBody  **public** User info(){  **return userService**.selectUserByUsername(**"zhangsan"**);  } } |

其他的CURD与此类似，此处不再讨论。

# spring security介绍

随着互联网的发展，当我们在淘宝或者京东上很方便的购物时，是否会想到我们的这种便利，如果不能保证我们的账号安全的话，可想而知会出现什么样的后果。所以，系统的安全性显得尤为重要。当用户购买下订单时，系统必须要验证此用户的身份和相应的权限。而如果使用springmvc的拦截器，验证过程显得非常繁琐。利用spring boot 和spring security将大大简化我们的配置项和提高网站的安全性。

Spring Security是一个基于spring专注于为Java应用程序提供身份验证（Authentication）和授权(Authorization)的框架。同时在 Web 请求级和方法调用级处理身份确认和授权。

# spring security入门

springsecurity 使用4.2.4.RELEASE版本。

第一个：添加依赖：

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

第二个：测试，访问之前写好的hello world

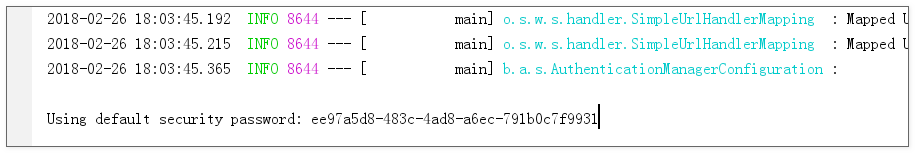
启动项目 并在浏览器中输入：http://localhost:8081/user/info

如图：



默认的情况下：spring boot 和spring security已经集成 并拦截所有的请求，需要验证通过才能放行。弹出框的方式 是默认的spring security的认证方式即（http basic的方式）

默认的情况下：用户名为user 密码为每次启动项目是所生成的的随机字符串，如图：



# 表单登录认证

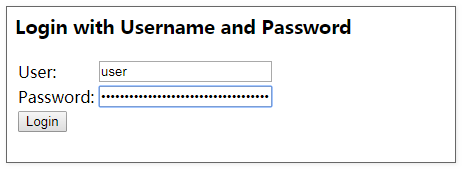
正常的情况下，目前的验证的方式不是我们想要的，更多的时候，需要有一个表单的登录界面来进行用户的身份确认（认证）。所以此时不能满足我们的需要，我们修改成表单的方式来登录。

第一个：创建配置类继承WebSecurityConfigurerAdapter，进行配置。

|  |
| --- |
| @EnableWebSecurity **public class** MySecurityConfig **extends** WebSecurityConfigurerAdapter {   **protected void** configure(HttpSecurity http) **throws** Exception {  http  .authorizeRequests()  .anyRequest().authenticated()*//授权所有的请求 都需要通过认证才能访问* .and()*//并且* .formLogin();*//采用表单提交的方式进行认证登录* } } |

第二个：启动项目，测试；输入地址：http://localhost:8081/user/info

输入用户名和密码即可。



# 自定义用户认证

认证包括三个方面：第一个 获取用户的信息，第二个 校验用户 第三个 密码加密

用户的信息获取主要使用mybatis获取用户信息，将获取到的用户的信息和页面传递过来的用户的信息进行匹配，如果成功，则放行，如果失败，则返回错误的信息。匹配的过程是由spring security自己本身来实现的。

用户的信息获取需要通过实现一个接口：UserDetailsService 来实现这个功能。

|  |
| --- |
| @Component **public class** MyUserDetailsService **implements** UserDetailsService {  @Override  **public** UserDetails loadUserByUsername(String username) **throws** UsernameNotFoundException {  *//从数据库中获取到用户的信息* **return new** User(username,**"123456"**, AuthorityUtils.*commaSeparatedStringToAuthorityList*(**"admin"**));  } } |

此时任何的用户只要输入了密码为123即可登录成功。

因此需要从数据库中查询数据，并校验数据的密码是否正确

|  |
| --- |
| @Component **public class** MyUserDetailsService **implements** UserDetailsService {   @Autowired  **private** UserService **userService**;  @Override  **public** UserDetails loadUserByUsername(String username) **throws** UsernameNotFoundException {  *//从数据库中获取到用户的信息* com.itheima.springboot.pojo.User user = **userService**.selectUserByUsername(username);  **if**(user==**null**){  **return null**;*//表示不存在* }    **return new** User(username,user.getPassword(), AuthorityUtils.*commaSeparatedStringToAuthorityList*(**"admin"**));  } } |

此时如果还有其他的需求，比如：当用户名和密码都正确但是有时这个用户已经被冻结了，此时也不能登录成功，如果要实现这个需求，那么需要对UserDetails接口说明一下。代码需要重新构造：

|  |
| --- |
| @Component **public class** MyUserDetailsService **implements** UserDetailsService {   @Autowired  **private** UserService **userService**;  @Override  **public** UserDetails loadUserByUsername(String username) **throws** UsernameNotFoundException {  *//从数据库中获取到用户的信息* com.itheima.springboot.pojo.User user = **userService**.selectUserByUsername(username);  **if**(user==**null**){  **return null**;*//表示不存在* }   *//校验用户是否已经被冻结 用户已经被冻结* **boolean** flagNonlock = **false**;*//已经被冻结  // return new User(username,user.getPassword(), AuthorityUtils.commaSeparatedStringToAuthorityList("admin"));* **return new** User(username,user.getPassword(),**true**,**true**,**true**,flagNonlock, AuthorityUtils.*commaSeparatedStringToAuthorityList*(**"admin"**));  } } |

此时,用户查询的数据的数据是明文的，在企业开发中，密码需要加密进行处理，不能以明文展示。此时需要加密。在spring security 中，处理加密的是一个passwordEncoder接口。

只需配置一个passwordEncoder接口的实现类即可。

首先配置passwordEncoder:

|  |
| --- |
| @EnableWebSecurity **public class** MySecurityConfig **extends** WebSecurityConfigurerAdapter {   @Bean  **public** PasswordEncoder createPasswordEncoder(){  **return new** BCryptPasswordEncoder();  }   **protected void** configure(HttpSecurity http) **throws** Exception {  http  .authorizeRequests()  .anyRequest().authenticated()*//授权所有的请求 都需要通过认证才能访问* .and()*//并且* .formLogin();*//采用表单提交的方式进行认证登录* } } |

修改校验逻辑：

|  |
| --- |
| @Component **public class** MyUserDetailsService **implements** UserDetailsService {   @Autowired  **private** UserService **userService**;   @Autowired  **private** PasswordEncoder **passwordEncoder**;   @Override  **public** UserDetails loadUserByUsername(String username) **throws** UsernameNotFoundException {  *//从数据库中获取到用户的信息* com.itheima.springboot.pojo.User user = **userService**.selectUserByUsername(username);  **if**(user==**null**){  **return null**;*//表示不存在* }  *//进行加密；注意此处不应该这么写，此处的方法应当在用户注册的时候调用此方法进行加密处理，验证身份时直接获取数据库中加过密的信息即可。* String encode = **passwordEncoder**.encode(user.getPassword());  System.***out***.println(**"密码》》》"**+encode);   *//校验用户是否已经被冻结 用户已经被冻结 // boolean flagNonlock = true;* **return new** User(username,encode, AuthorityUtils.*commaSeparatedStringToAuthorityList*(**"admin"**)); *// return new User(username,encode,true,true,true,flagNonlock, AuthorityUtils.commaSeparatedStringToAuthorityList("admin"));* } } |

自定义用户认证

总结：

1.获取用户的信息的校验（UserDetailsService）

2.处理用户的校验的业务逻辑处理（UserDetails）

3.密码的加密（PasswordEncoder）---BCryptPasswordEncoder