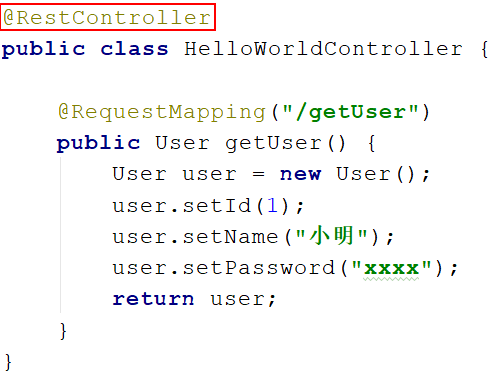
web开发包括常用的json输出、filters、property、log、数据库操作等。

## json接口开发

以往的spring开发的时候需要我们提供json接口的时候需要做哪些配置呢：

1. 添加jackson等相关jar包
2. 配置spring controller扫描
3. 对接的方法添加@ResponseBody

spring boot只需类添加@RestController即可，默认类中的方法都会以json的格式返回。



如果需要使用页面开发只要使用@Controller。

## 自定义Filter

项目中会使用filters用于来调用日志、排除有XSS威胁的字符、执行权限验证等等。

SpringBoot自动添加了OrderedCharacterEncodingFilter和HiddenHttpMethodFilter，并且我们可以自定义Filter。

步骤：

1. 实现Filter接口，重写doFilter方法。

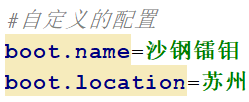
*/\*\*  
 \* 自定义过滤器  
 \* 定义一个拦截访问url的filter 来拦截访问项目的URI和URL  
 \** ***@author*** *nino  
 \** ***@date*** *2019/6/20  
 \*/***public class** MyFilter **implements** Filter {  
  
 **private** Logger **log** = LoggerFactory.*getLogger*(MyFilter.**class**);  
  
 @Override  
 **public void** init(FilterConfig filterConfig) **throws** ServletException {  
  
 }  
  
 @Override  
 **public void** doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain) **throws** IOException, ServletException {  
 HttpServletRequest httpServletRequest = (HttpServletRequest) servletRequest;  
 String requestURI = httpServletRequest.getRequestURI();  
 StringBuffer requestURL = httpServletRequest.getRequestURL();  
 **log**.info(**"requestURI:"**+requestURI+**"requestURL:"**+requestURL);  
 filterChain.doFilter(servletRequest, servletResponse);  
 }  
  
 @Override  
 **public void** destroy() {  
  
 }  
}

1. 添加@Configuration注解，将自定义Filter加入过滤链。

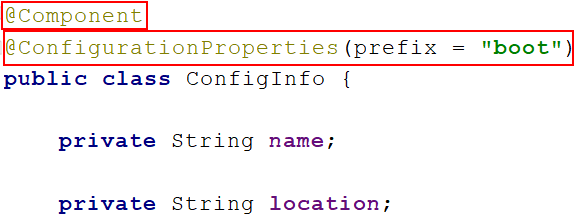
*/\*\*  
 \* 定义springboot过滤链配置类  
 \** ***@author*** *nino  
 \** ***@date*** *2019/6/20  
 \*/*@Configuration  
**public class** FilterConfig {  
  
 @Bean  
 **public** FilterRegistrationBean filterRegistration() {  
 FilterRegistrationBean registration = **new** FilterRegistrationBean();  
 *//设置我们自定义拦截器对象* registration.setFilter(**new** MyFilter());  
 *//设置拦截规则，过滤应用程序中所有资源* registration.addUrlPatterns(**"/\*"**);  
 registration.addInitParameter(**"paramName"**, **"paramValue"**);  
 *//设置拦截器的名称* registration.setName(**"MyFilter"**);  
 *//设置拦截器顺序* registration.setOrder(1);  
 **return** registration;  
 }  
}

## 自定义Property

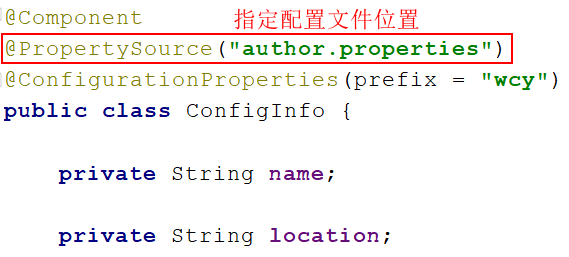
直接写在application.properties中（不建议这么做，如果是全局变量提倡这种方法），当写在此文件中时，不需要指明资源文件路劲，只需要指明前缀即可。



自定义配置类：



自定义配置文件author.properties：





应用场景：

如果只是某个业务中需要获取配置文件中的某项值或者设置具体值，可以使用@Value；

如果一个JavaBean中大量属性值要和配置文件进行映射，可以使用@ConfigurationProperties；

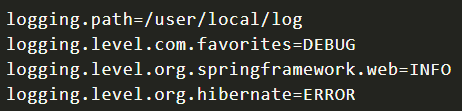
## log配置

logback是springboot默认的日志组件，启动即可看到打印在控制台输出的info级别的日志。springboot的log一般会包含在其他starter里。

TRACE < DEBUG < INFO < WARN < ERROR < FATAL

Spring Boot中默认配置ERROR、WARN和INFO级别的日志输出到控制台。

配置输出地址和输出级别



## 数据库操作

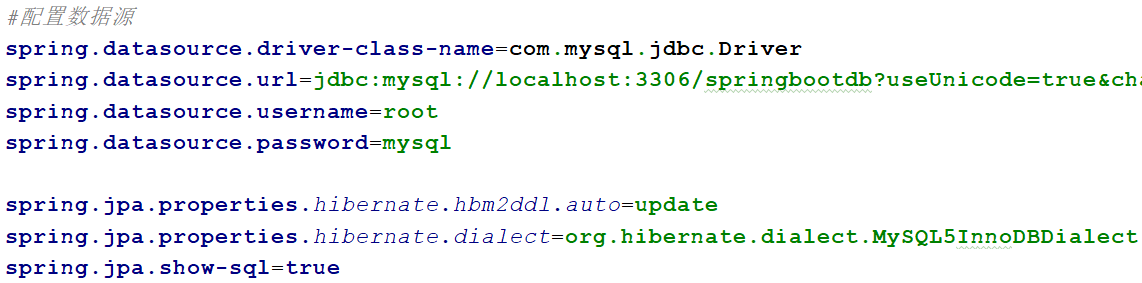
重点是mysql、spring data jpa的使用，其中mysql就不用说了，jpa是利用Hibernate生成各种自动化的sql，如果只是简单的增删改查，基本上不用手写，spring内部已经封装好了。

1）添加相关jar包

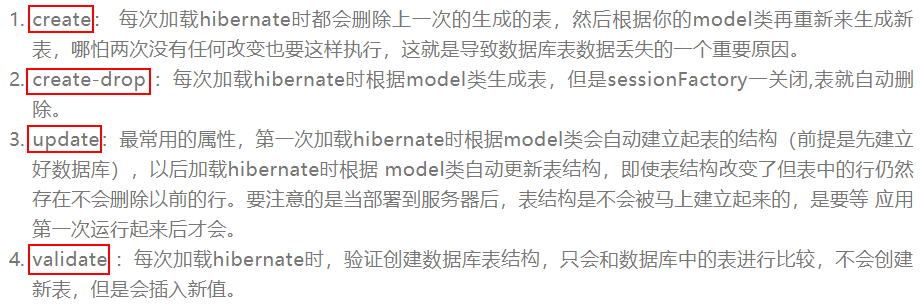
<**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-data-jpa</**artifactId**>  
</**dependency**>

*<!--MYSQL的jdbc驱动包-->*<**dependency**>  
 <**groupId**>mysql</**groupId**>  
 <**artifactId**>mysql-connector-java</**artifactId**>  
</**dependency**>

2）添加配置文件



hibernate.hbm2ddl.auto参数的作用主要用于：自动创建|更新|验证数据库表结构，有四个值：



dialect：主要是指定生成表明的存储引擎为InnoBB。

show-sql：是否打印出自动生成的SQL，方便吊事的时候查看。

3）添加实体类和Dao

@Entity  
@Getter  
@Setter  
@NoArgsConstructor  
**public class** Student **implements** Serializable {  
  
 @Id  
 @GeneratedValue  
 **private** Long **id**;  
  
 @Column(nullable = **false**, unique = **true**)  
 **private** String **studentName**;  
  
 @Column(nullable = **false**)  
 **private** String **password**;  
  
 @Column(nullable = **false**, unique = **true**)  
 **private** String **email**;  
  
 @Column(nullable = **true**, unique = **true**)  
 **private** String **nickName**;  
  
 @Column(nullable = **false**)  
 **private** String **regTime**;  
}

注：Entity中不映射成列的字段得加@Transient 注解，不加注解也会映射成列。

@Id标注用于声明一个实体类的属性映射为数据库的主键列。

@GeneratedValue用于标注主键的生成策略，通过strategy属性指定。默认情况下，JPA自动选择一个最适合底层数据库的主键生成策略：SqlServer对应identity，MySQL对应auto increment。

nullable=false是指这个字段在保存时必须有值，不能还是null值就调用save去保存入库。

unique=true是指这个字段的值在这张表里不能重复，所有记录值都要唯一，就像主键那样。

dao只要继承JpaRepository类就可以，几乎可以不用写方法，可以根据方法名来自动的生产SQL，比如findByUserName 会自动生产一个以 userName 为参数的查询方法，比如 findAlll 自动会查询表里面的所有数据，比如自动分页等等。

**public interface** StudentRepository **extends**

JpaRepository<Student, Long> {  
  
 Student findByStudentName(String studentName);  
  
 Student findByStudentNameOrEmail(String studentName, String email);  
}

4）测试

@RunWith(SpringJUnit4ClassRunner.**class**)  
@SpringBootTest  
**public class** StudentRepositoryTests {  
  
 @Autowired  
 StudentRepository **studentRepository**;  
  
 @Test  
 **public void** test(){  
 Date date = **new** Date();  
 DateFormat dateFormat = DateFormat.*getDateTimeInstance*(DateFormat.***LONG***, DateFormat.***LONG***);  
 String formattedDate = dateFormat.format(date);  
  
 **studentRepository**.save(**new** Student(**"aa"**,**"aa"**,**"aa@126.com"**,**"aa123456"**,formattedDate));  
 **studentRepository**.save(**new** Student(**"bb"**,**"bb"**,**"bb@126.com"**,**"bb123456"**,formattedDate));  
 **studentRepository**.save(**new** Student(**"cc"**,**"cc"**,**"cc@126.com"**,**"cc123456"**,formattedDate));  
  
 Assert.*assertEquals*(3, **studentRepository**.findAll().size());  
 Assert.*assertEquals*(**"bb123456"**, **studentRepository**.findByStudentNameOrEmail(**"bb"**, **"bb@126.com"**).getNickName());  
 **studentRepository**.delete(**studentRepository**.findByStudentName(**"aa"**));  
 }  
}

Spring data jpa还有许多功能，比如封装好的分页，可以自己定义SQL，主从分离等等。

## Thymeleaf

Spring boot推荐使用thymeleaf模板来代替jsp。

1）Thymeleaf介绍：

Thymeleaf是一款用于渲染XML/XHTML/HTML5内容的模板引擎。类似JSP，Velocity，FreeMaker等，它也可以轻易的与Spring MVC等Web框架进行集成作为Web应用的模板引擎。与其它模板引擎相比，Thymeleaf最大的特点是能够直接在浏览器中打开并正确显示模板页面，而不需要启动整个Web应用。

2）Thymeleaf与Velocity、FreeMarker比较：

Thymeleaf是与众不同的，因为它使用了自然的模板技术。这意味着Thymeleaf的模板语法并不会破坏文档的结构，模板依旧是有效的XML文档。模板还可以用作工作原型，Thymeleaf会在运行期替换掉静态值。Velocity与FreeMarker则是连续的文本处理器。

下面的代码示例分别使用Velocity、FreeMarker与Thymeleaf打印出一条消息：

Velocity: <p>$message</p>

FreeMarker: <p>${message}</p>

Thymeleaf: <p th:text="${message}">Hello World!</p>

注意：由于Thymeleaf使用了XML DOM解析器，因此它并不适合于处理大规模的XML文件。

3）URL

URL在Web应用模板中占据着十分重要的地位，需要特别注意的是Thymeleaf对于URL的处理是通过语法@{...}来处理的。Thymeleaf支持绝对路径URL：

<a th:href="@{http://www.thymeleaf.org}">Thymeleaf</a>

4）条件求值

<a th:href="@{/login}" th:unless=${session.user != null}>Login</a>

5）for循环

<tr th:each="prod : ${prods}">

<td th:text="${prod.name}">Onions</td>

<td th:text="${prod.price}">2.41</td>

<td th:text="${prod.inStock}? #{true} : #{false}">yes</td>

</tr>

6）页面即原型

在Web开发过程中一个绕不开的话题就是前端工程师与后端工程师的协作。在传统Java Web开发过程中，前端工程师和后端工程师一样，也需要安装一套完整的开发环境，然后各类Java IDE中修改模板、静态资源文件，启动/重启/重新加载应用服务器，刷新页面查看最终效果。

但实际上前端工程师的职责更多应该关注于页面本身而非后端，使用JSP，Velocity等传统的Java模板引擎很难做到这一点，因为它们必须在应用服务器中渲染完成后才能在浏览器中看到结果，而Thymeleaf从根本上颠覆了这一过程，通过属性进行模板渲染不会引入任何新的浏览器不能识别的标签，例如JSP中的，不会在Tag内部写表达式。整个页面直接作为HTML文件用浏览器打开，几乎就可以看到最终的效果，这大大解放了前端工程师的生产力，他们的最终交付物就是纯的HTML/CSS/JavaScript文件。

## Gradle

spring项目建议使用Gradle进行构建项目，相比maven来讲Gradle更简洁，而且gradle更适合大型复杂项目的构建。gradle吸收了maven和ant的特点而来，不过目前maven仍然是Java界的主流。

一个使用gradle配置的项目：

//脚本

buildscript {

repositories {

maven { url "http://repo.spring.io/libs-snapshot" }

mavenLocal()

}

dependencies {

classpath("org.springframework.boot:spring-boot-gradle-plugin:1.3.6.RELEASE")

}

}

//采用的插件

apply plugin: 'java' //添加 Java 插件, 表明这是一个 Java 项目

apply plugin: 'spring-boot' //添加 Spring-boot支持

apply plugin: 'war' //添加 War 插件, 可以导出 War 包

apply plugin: 'idea' //添加 Eclipse 插件, 添加 Eclipse IDE 支持, Intellij Idea 为 "idea"

war {

baseName = 'favorites'

version = '0.1.0'

}

//指定版本号以及JDK版本

sourceCompatibility = 1.7 //最低兼容版本 JDK1.7

targetCompatibility = 1.7 //目标兼容版本 JDK1.7

//添加Maven仓库

repositories { // Maven 仓库

mavenLocal() //使用本地仓库

mavenCentral() //使用中央仓库

maven { url "http://repo.spring.io/libs-snapshot" } //使用远程仓库

}

//添加依赖

dependencies { // 各种 依赖的jar包

compile("org.springframework.boot:spring-boot-starter-web:1.3.6.RELEASE")

compile("org.springframework.boot:spring-boot-starter-thymeleaf:1.3.6.RELEASE")

compile("org.springframework.boot:spring-boot-starter-data-jpa:1.3.6.RELEASE")

compile group: 'mysql', name: 'mysql-connector-java', version: '5.1.6'

compile group: 'org.apache.commons', name: 'commons-lang3', version: '3.4'

compile("org.springframework.boot:spring-boot-devtools:1.3.6.RELEASE")

compile("org.springframework.boot:spring-boot-starter-test:1.3.6.RELEASE")

compile 'org.webjars.bower:bootstrap:3.3.6'

compile 'org.webjars.bower:jquery:2.2.4'

compile("org.webjars:vue:1.0.24")

compile 'org.webjars.bower:vue-resource:0.7.0'

}

bootRun {

addResources = true

}

## WebJars

WebJars是将客户端（浏览器）资源（JavaScript，CSS等）打成jar包文件，以对资源进行统一依赖管理。WebJars的jar包部署在Maven中央仓库上。

为什么使用：

我们在开发Java web项目的时候会使用像Maven，Gradle等构件工具以实现对jar包版本依赖管理，以及项目的自动化管理，但是对于JavaScript，CSS等前端资源包，我们只能采用拷贝到webapp下的方式，这样做就无法对这些资源进行依赖管理。那么WebJars就提供给我们这些前端资源的jar包形式，我们就可以进行依赖管理。

如何使用：

1）WebJars主官网查找对于的组件，比如Vue.js

<dependency>

<groupId>org.webjars</groupId>

<artifactId>vue</artifactId>

<version>2.6.10</version>

</dependency>

2）页面引入

<link th:href="@{/webjars/bootstrap/3.3.6/dist/css/bootstrap.css}" rel="stylesheet"></link>