Spring boot学习笔记

## 搭建第一个Spring boot项目（Maven）

### 创建Maven项目spring.boot，修改pom.xml

### 添加spring.boot启动支持

方式一：继承 **spring-boot-starter-parent**

<parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>1.3.2.RELEASE</version>  
 <relativePath/>

</parent>

方式二：添加 **spring-boot-dependencies**

<dependencyManagement>  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.data</groupId>  
 <artifactId>spring-data-releasetrain</artifactId>  
 <version>Fowler-SR2</version>  
 <scope>import</scope>  
 <type>pom</type>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-dependencies</artifactId>  
 <version>1.4.0.RELEASE</version>  
 <type>pom</type>  
 <scope>import</scope>  
 </dependency>  
 </dependencies>  
</dependencyManagement>

**注：推荐方式一，方式二可能遇到一些问题！用方式二学习redis时，就遇到奇怪的问题，换成方式一，问题就没有了，为了不麻烦，还是使用方式一吧！**

### Spring.boot启动器列表

| **Name** | **Description** | **Pom** |
| --- | --- | --- |
| spring-boot-starter-thymeleaf | Starter for building MVC web applications using Thymeleaf views | [Pom](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-starters/spring-boot-starter-thymeleaf/pom.xml) |
| spring-boot-starter-ws | Starter for using Spring Web Services. Deprecated as of 1.4 in favor of[spring-boot-starter-web-services](http://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/#spring-boot-starter-web-services) | [Pom](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-starters/spring-boot-starter-ws/pom.xml) |
| spring-boot-starter-data-couchbase | Starter for using Couchbase document-oriented database and Spring Data Couchbase | [Pom](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-starters/spring-boot-starter-data-couchbase/pom.xml) |
| spring-boot-starter-artemis | Starter for JMS messaging using Apache Artemis | [Pom](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-starters/spring-boot-starter-artemis/pom.xml) |
| spring-boot-starter-web-services | Starter for using Spring Web Services | [Pom](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-starters/spring-boot-starter-web-services/pom.xml) |
| spring-boot-starter-mail | Starter for using Java Mail and Spring Framework’s email sending support | [Pom](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-starters/spring-boot-starter-mail/pom.xml) |
| spring-boot-starter-data-redis | Starter for using Redis key-value data store with Spring Data Redis and the Jedis client | [Pom](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-starters/spring-boot-starter-data-redis/pom.xml) |
| spring-boot-starter-web | Starter for building web, including RESTful, applications using Spring MVC. Uses Tomcat as the default embedded container | [Pom](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-starters/spring-boot-starter-web/pom.xml) |
| spring-boot-starter-data-gemfire | Starter for using GemFire distributed data store and Spring Data GemFire | [Pom](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-starters/spring-boot-starter-data-gemfire/pom.xml) |
| spring-boot-starter-activemq | Starter for JMS messaging using Apache ActiveMQ | [Pom](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-starters/spring-boot-starter-activemq/pom.xml) |
| spring-boot-starter-data-elasticsearch | Starter for using Elasticsearch search and analytics engine and Spring Data Elasticsearch | [Pom](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-starters/spring-boot-starter-data-elasticsearch/pom.xml) |
| spring-boot-starter-integration | Starter for using Spring Integration | [Pom](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-starters/spring-boot-starter-integration/pom.xml) |
| spring-boot-starter-test | Starter for testing Spring Boot applications with libraries including JUnit, Hamcrest and Mockito | [Pom](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-starters/spring-boot-starter-test/pom.xml) |
| spring-boot-starter-hornetq | Starter for JMS messaging using HornetQ. Deprecated as of 1.4 in favor of[spring-boot-starter-artemis](http://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/#spring-boot-starter-artemis) | [Pom](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-starters/spring-boot-starter-hornetq/pom.xml) |
| spring-boot-starter-jdbc | Starter for using JDBC with the Tomcat JDBC connection pool | [Pom](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-starters/spring-boot-starter-jdbc/pom.xml) |
| spring-boot-starter-mobile | Starter for building web applications using Spring Mobile | [Pom](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-starters/spring-boot-starter-mobile/pom.xml) |
| spring-boot-starter-validation | Starter for using Java Bean Validation with Hibernate Validator | [Pom](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-starters/spring-boot-starter-validation/pom.xml) |
| spring-boot-starter-hateoas | Starter for building hypermedia-based RESTful web application with Spring MVC and Spring HATEOAS | [Pom](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-starters/spring-boot-starter-hateoas/pom.xml) |
| spring-boot-starter-jersey | Starter for building RESTful web applications using JAX-RS and Jersey. An alternative to[spring-boot-starter-web](http://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/#spring-boot-starter-web) | [Pom](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-starters/spring-boot-starter-jersey/pom.xml) |
| spring-boot-starter-data-neo4j | Starter for using Neo4j graph database and Spring Data Neo4j | [Pom](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-starters/spring-boot-starter-data-neo4j/pom.xml) |
| spring-boot-starter-websocket | Starter for building WebSocket applications using Spring Framework’s WebSocket support | [Pom](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-starters/spring-boot-starter-websocket/pom.xml) |
| spring-boot-starter-aop | Starter for aspect-oriented programming with Spring AOP and AspectJ | [Pom](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-starters/spring-boot-starter-aop/pom.xml) |
| spring-boot-starter-amqp | Starter for using Spring AMQP and Rabbit MQ | [Pom](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-starters/spring-boot-starter-amqp/pom.xml) |
| spring-boot-starter-data-cassandra | Starter for using Cassandra distributed database and Spring Data Cassandra | [Pom](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-starters/spring-boot-starter-data-cassandra/pom.xml) |
| spring-boot-starter-social-facebook | Starter for using Spring Social Facebook | [Pom](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-starters/spring-boot-starter-social-facebook/pom.xml) |
| spring-boot-starter-jta-atomikos | Starter for JTA transactions using Atomikos | [Pom](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-starters/spring-boot-starter-jta-atomikos/pom.xml) |
| spring-boot-starter-security | Starter for using Spring Security | [Pom](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-starters/spring-boot-starter-security/pom.xml) |
| spring-boot-starter-mustache | Starter for building MVC web applications using Mustache views | [Pom](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-starters/spring-boot-starter-mustache/pom.xml) |
| spring-boot-starter-data-jpa | Starter for using Spring Data JPA with Hibernate | [Pom](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-starters/spring-boot-starter-data-jpa/pom.xml) |
| spring-boot-starter | Core starter, including auto-configuration support, logging and YAML | [Pom](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-starters/spring-boot-starter/pom.xml) |
| spring-boot-starter-velocity | Starter for building MVC web applications using Velocity views. Deprecated since 1.4 | [Pom](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-starters/spring-boot-starter-velocity/pom.xml) |
| spring-boot-starter-groovy-templates | Starter for building MVC web applications using Groovy Templates views | [Pom](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-starters/spring-boot-starter-groovy-templates/pom.xml) |
| spring-boot-starter-freemarker | Starter for building MVC web applications using FreeMarker views | [Pom](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-starters/spring-boot-starter-freemarker/pom.xml) |
| spring-boot-starter-batch | Starter for using Spring Batch | [Pom](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-starters/spring-boot-starter-batch/pom.xml) |
| spring-boot-starter-redis | Starter for using Redis key-value data store with Spring Data Redis and the Jedis client. Deprecated as of 1.4 in favor of [spring-boot-starter-data-redis](http://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/#spring-boot-starter-data-redis) | [Pom](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-starters/spring-boot-starter-redis/pom.xml) |
| spring-boot-starter-social-linkedin | Stater for using Spring Social LinkedIn | [Pom](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-starters/spring-boot-starter-social-linkedin/pom.xml) |
| spring-boot-starter-cache | Starter for using Spring Framework’s caching support | [Pom](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-starters/spring-boot-starter-cache/pom.xml) |
| spring-boot-starter-data-solr | Starter for using the Apache Solr search platform with Spring Data Solr | [Pom](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-starters/spring-boot-starter-data-solr/pom.xml) |
| spring-boot-starter-data-mongodb | Starter for using MongoDB document-oriented database and Spring Data MongoDB | [Pom](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-starters/spring-boot-starter-data-mongodb/pom.xml) |
| spring-boot-starter-jooq | Starter for using jOOQ to access SQL databases. An alternative to[spring-boot-starter-data-jpa](http://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/#spring-boot-starter-data-jpa) or [spring-boot-starter-jdbc](http://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/#spring-boot-starter-jdbc) | [Pom](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-starters/spring-boot-starter-jooq/pom.xml) |
| spring-boot-starter-jta-narayana | Spring Boot Narayana JTA Starter | [Pom](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-starters/spring-boot-starter-jta-narayana/pom.xml) |
| spring-boot-starter-cloud-connectors | Starter for using Spring Cloud Connectors which simplifies connecting to services in cloud platforms like Cloud Foundry and Heroku | [Pom](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-starters/spring-boot-starter-cloud-connectors/pom.xml) |
| spring-boot-starter-jta-bitronix | Starter for JTA transactions using Bitronix | [Pom](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-starters/spring-boot-starter-jta-bitronix/pom.xml) |
| spring-boot-starter-social-twitter | Starter for using Spring Social Twitter | [Pom](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-starters/spring-boot-starter-social-twitter/pom.xml) |
| spring-boot-starter-data-rest | Starter for exposing Spring Data repositories over REST using Spring Data REST | [Pom](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-starters/spring-boot-starter-data-rest/pom.xml) |
| spring-boot-starter-actuator | Starter for using Spring Boot’s Actuator which provides production ready features to help you monitor and manage your application | [Pom](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-starters/spring-boot-starter-actuator/pom.xml) |
| spring-boot-starter-remote-shell | Starter for using the CRaSH remote shell to monitor and manage your application over SSH | [Pom](https://github.com/spring-projects/spring-boot/tree/master/spring-boot-starters/spring-boot-starter-remote-shell/pom.xml) |

详细可参阅：<http://blog.csdn.net/chszs/article/details/50610474>

### 添加基本启动器

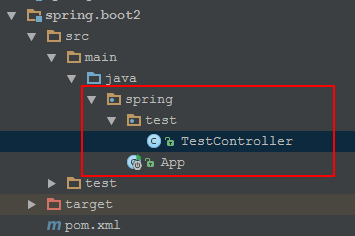
<dependencies>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
  
</dependencies>

* spring-boot-starter：基本启动器
* spring-boot-starter-web：支持web方式的启动器
* spring-boot-starter-test：支持单元测试启动器

注：经测试，如果添加了spring-boot-starter-web，即使没有spring-boot-starter，也是可以正常启动，不会有问题！这两者的关系应该类似与javaee与javase，前者范围更大，后者只包含了核心的功能！

### 添加启动与测试代码

* 代码结构，如图：



* App.java是启动类，启动类不能直接在java下，不然会报错，错误信息：**\*\* WARNING \*\* : Your ApplicationContext is unlikely to start due to a @ComponentScan of the default package.**

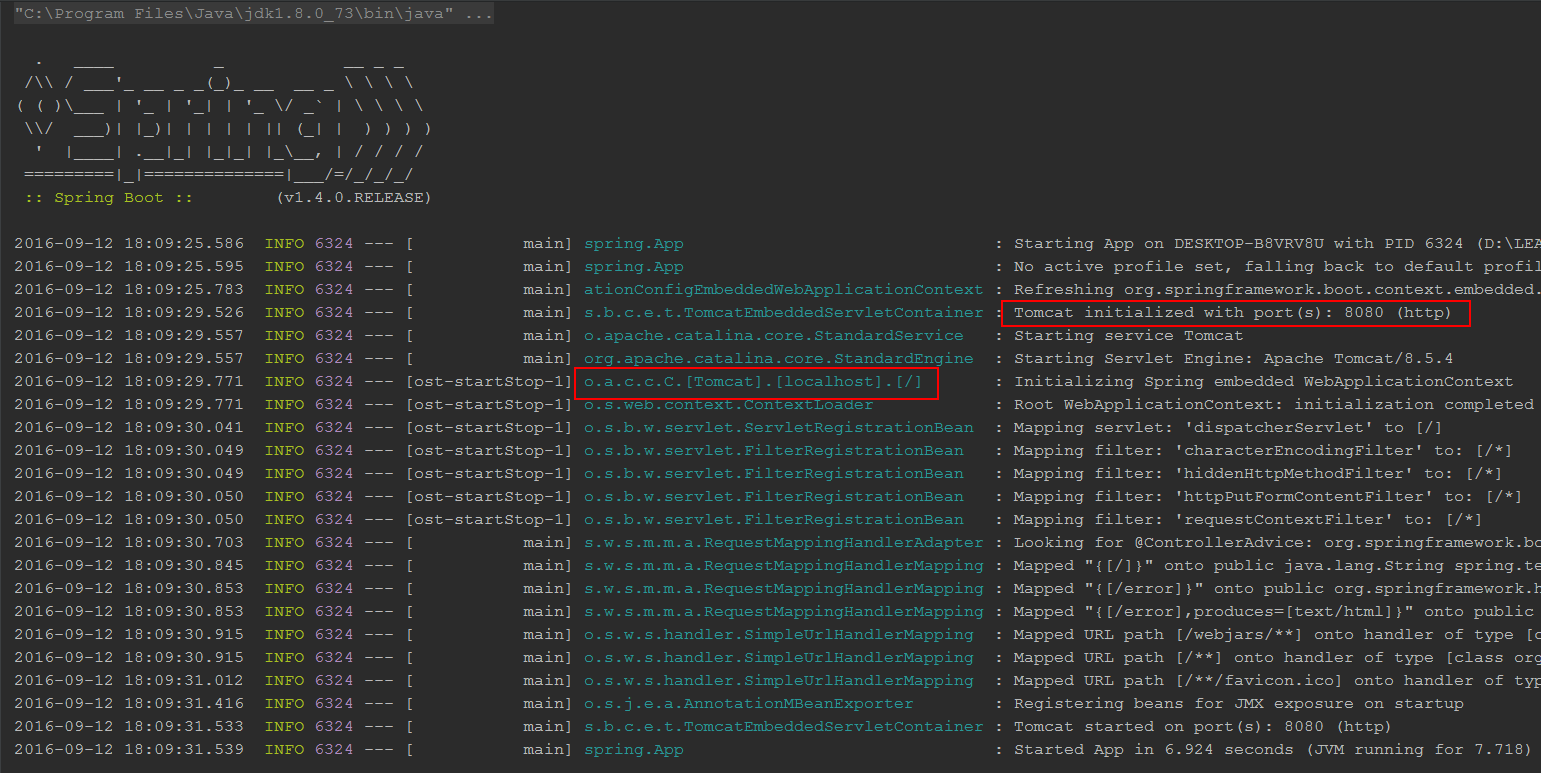
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
  
@SpringBootApplication  
public class App {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(App.class, args);  
 }  
  
}

* TestController.java为测试Controller

import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
@RestController  
public class TestController {  
  
 @RequestMapping("/")  
 public String say(){  
 return "hello world...";  
 }  
  
}

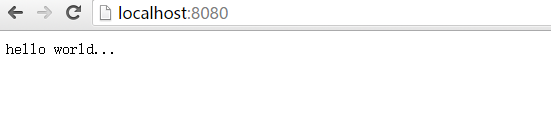
### 启动第一个Spring.boot项目

* 其实，就是执行App.java中的main方法，启动日志：



不难看出，Spring.boot内置了tomcat，实质上还是启动的tomcat！

* 访问：<http://localhost:8080/>，如图：



OK，第一个Spring.boot项目已经搭建完成！

## Spring.boot高级配置

### 禁用tomcat，切换jetty服务器

修改pom.xml，如图：

<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 <version>${spring.boot.version}</version>  
 <exclusions>  
 <exclusion>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-tomcat</artifactId>  
 </exclusion>  
 </exclusions>  
</dependency>  
<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-jetty</artifactId>  
 <version>${spring.boot.version}</version>  
</dependency>

### 说说 @SpringBootApplication

*@SpringBootApplication* *// same as @Configuration @EnableAutoConfiguration @ComponentScan*

这是Spring.boot官方文档上的描述，实事也是如此！

### Spring.boot热部署

这个功能在开发时还是非常有用的，具体实现：

* pom.xml中添加配置

<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-devtools</artifactId>  
 <optional>true</optional>  
</dependency>

<build>  
 <plugins>  
 <plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 <configuration>  
 <fork>true</fork>  
 </configuration>  
 </plugin>  
 </plugins>  
</build>

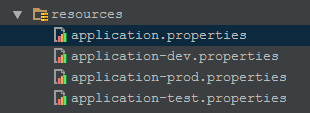
* intellj idea 修改classpath下文件后，执行make，即会自动激活restart
* eclipse 修改classpth下文件后，执行build即可，如果设置了下去编译，那就会自动激活restart
* 静态资源不需要restart，可以使用：

spring.devtools.restart.exclude=static/\*\*,public/\*\*

### 切换配置

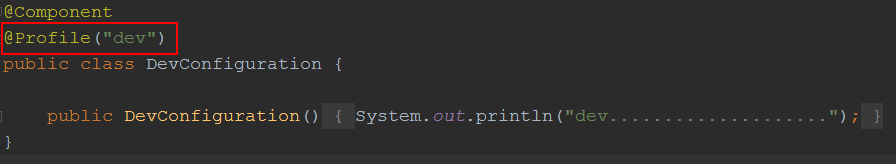
Spring.boot会根据application.properties中属性：spring.profiles.active，激活相应的配置，当 spring.profiles.active=dev时：

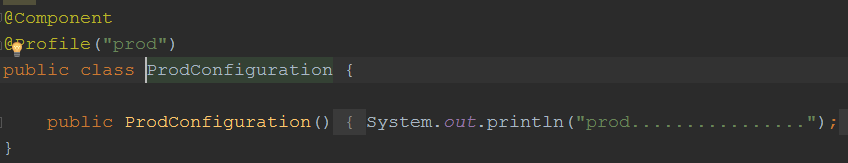
* 激活配置文件，如：



会自动加载 application-dev.properties文件

* 激活Profile配置，如：





会加载@Profile(“dev”)修饰的类：**DevConfiguration.java**

### 自定义初始化

If you need to run some specific code once the SpringApplication has started, you can implement the ApplicationRunner or CommandLineRunner interfaces. Both interfaces work in the same way and offer a single run method which will be called just before SpringApplication.run(…​) completes.

ApplicationRunner or CommandLineRunner都会在容器初始化完成之前执行，对于一些特殊的操作，这个还是很有用的！

### 修改容器启动端口

可以通过设置application.properties中属性 server.port 修改端口号！

### 使用YAML配置

* 在resources目录下创建文件application.yml（此配置默认会加载），添加如下内容：

**my:  
 servers:** - dev.bar.com  
 - foo.bar.com  
 **dev:  
 url:** http://dev.bar.com  
 **name:** Developer Setup  
 **prod:  
 url:** http://foo.bar.com  
 **name:** My Cool App

* Bean中读取这些属性，代码如下：

@Configuration  
@ConfigurationProperties(prefix = "my")  
public class PropertiesBean {

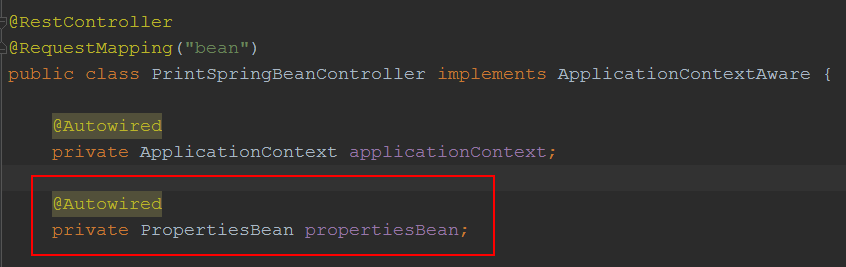
// 读取list  
 private List<String> servers = new ArrayList<String>();

// 读取map  
 private Map<String, String> dev = new HashMap<String, String>();

// 读取map  
 private Map<String, String> prod = new HashMap<String, String>();

// 略去getter/setter

* 使用，像使用普通spring bean一样，注入即可，详细如下：



### 使用@ControllerAdvice

* 使用场景：如当抛出XXXXException时，统一响应处理！

@ControllerAdvice  
public class DefaultExceptionHandler {

// 当抛出 Exception 时，使用的响应处理器  
 @ExceptionHandler(value = Exception.class)  
 public ModelAndView defaultErrorHandler(HttpServletRequest req, Exception e) throws Exception {  
 ModelAndView mav = new ModelAndView();  
 mav.addObject("exception", e);  
 mav.addObject("url", req.getRequestURL());  
 mav.setViewName("error");  
 return mav;  
 }

// 当抛出 NullPointerException时，使用的响应处理器  
 @ExceptionHandler(value = NullPointerException.class)  
 @ResponseBody  
 public Map<String, Object> jsonErrorHandler(HttpServletRequest req, Exception e) throws Exception {  
 Map<String, Object> error = new HashMap<String, Object>();  
 error.put("Exception", e);  
 error.put("code", 0000);  
 error.put("url", req.getRequestURL().toString());  
 return error;  
 }  
  
}

### 使用freemarker作为View渲染对象

* 在pom.xml中添加start：

<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-freemarker</artifactId>  
 <optional>true</optional>  
</dependency>

* 在application.properties中添加前/后缀：

spring.freemarker.template-loader-path=classpath:/pages/  
spring.freemarker.suffix=.html

### 配置mysql数据源（默认）

* 在pom.xml中添加：

<dependency>  
 <groupId>mysql</groupId>  
 <artifactId>mysql-connector-java</artifactId>  
 <version>5.1.39</version>  
</dependency>

* 在application.properties中添加：

spring.datasource.url=jdbc:mysql://10.199.197.148:3306/log  
spring.datasource.username=root  
spring.datasource.password=123456  
spring.datasource.driver-class-name=com.mysql.jdbc.Driver

### 使用JdbcTemplate实现持久化

* 在pom.xml中添加starter：

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

* 使用：

@Repository  
public class BaseDao<T> implements IDao<T> {  
  
 @Autowired  
 private NamedParameterJdbcTemplate namedParameterJdbcTemplate;  
  
 public T findById(final Long id, String sql, Class<T> mappedClass) {  
 Map<String, Object> params = new HashMap<String, Object>() {{  
 put("id", id);  
 }};  
 return namedParameterJdbcTemplate.queryForObject(sql, params, new BeanPropertyRowMapper<T>(mappedClass));  
 }  
  
}

### 使用jpa实现持久化

* 在pom.xml中添加startter：

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

* application.properties中添加：

spring.jpa.hibernate.ddl-auto=update

* 使用：

public interface UserRepository extends JpaRepository<User, Long> {  
  
 @Query("from User u where u.name = :name")  
 List<User> findByName(@Param("name") String name);  
  
}

### 使用mybatis实现持久化

* 在pom.xml中添加starter：

<dependency>  
 <groupId>org.mybatis.spring.boot</groupId>  
 <artifactId>mybatis-spring-boot-starter</artifactId>  
 <version>1.1.1</version>  
</dependency>

* 使用：

@Mapper  
public interface UserMapper {  
  
 @Select("select \* from user where id = #{id}")  
 User findById(@Param("id") Long id);  
  
 @Select("select \* from user where name = #{name}")  
 List<User> findByName(@Param("name") String name);  
}

### 多数据源配置

* 在application.properties中添加：

spring.datasource.primary.url=jdbc:mysql://localhost:3306/test1  
spring.datasource.primary.username=root  
spring.datasource.primary.password=123456  
spring.datasource.primary.driver-class-name=com.mysql.jdbc.Driver  
  
spring.datasource.secondary.url=jdbc:mysql://localhost:3306/test2  
spring.datasource.secondary.username=root  
spring.datasource.secondary.password=123456  
spring.datasource.secondary.driver-class-name=com.mysql.jdbc.Driver

* 创建dataSource配置bean：

@Configuration  
public class DataSourceBean {  
  
 @Bean(name = "primaryDataSource")  
 @Qualifier("primaryDataSource")  
 @Primary  
 @ConfigurationProperties(prefix = "spring.datasource.primary")  
 public DataSource primaryDataSource(){  
 return DataSourceBuilder.*create*().build();  
 }  
  
 @Bean(name = "secondaryDataSource")  
 @Qualifier("secondaryDataSource")  
 @ConfigurationProperties(prefix = "spring.datasource.secondary")  
 public DataSource secondaryDataSource(){  
 return DataSourceBuilder.*create*().build();  
 }  
  
}

说明：注意标红的几个点，要相对就，不然就会报错！

* 配置jdbc模板类，创建JdbcTemplateBean：

@Configuration  
public class JdbcTemplateBean {  
  
 @Autowired  
 @Qualifier("primaryDataSource")  
 private DataSource primaryDataSource;  
  
 @Autowired  
 @Qualifier("secondaryDataSource")  
 private DataSource secondaryDataSource;  
  
  
 @Bean(name = "primaryJdbcTemplate")  
 @Qualifier("primaryJdbcTemplate")  
 public NamedParameterJdbcTemplate primaryJdbcTemplate(){  
 return new NamedParameterJdbcTemplate(primaryDataSource);  
 }  
  
 @Bean(name = "secondaryJdbcTemplate")  
 @Qualifier("secondaryJdbcTemplate")  
 public NamedParameterJdbcTemplate secondaryJdbcTemplate(){  
 return new NamedParameterJdbcTemplate(secondaryDataSource);  
 }  
  
}

* 使用：

@Service("userService2")  
public class UserService2Impl implements UserService2 {  
  
 @Resource(name = "primaryJdbcTemplate")  
 private NamedParameterJdbcTemplate primaryJdbcTemplate;  
  
 @Resource(name = "secondaryJdbcTemplate")  
 private NamedParameterJdbcTemplate secondaryJdbcTemplate;  
  
 public User findByIdPri(final long id) {  
 return findById(id, primaryJdbcTemplate);  
 }  
  
 public User findByIdSec(final long id) {  
 return findById(id, secondaryJdbcTemplate);  
 }  
  
 private User findById(final long id, NamedParameterJdbcTemplate template) {  
 Map<String, Object> params = new HashMap<String, Object>() {{  
 put("id", id);  
 }};  
 String sql = "select \* from user where id = :id";  
 return template.queryForObject(sql, params, new BeanPropertyRowMapper<User>(User.class));  
 }  
  
}

* 注意点：
  + 当配置自定义的 DataSource 后，spring boot 将不会再产生默认 dataSource；
  + 当配置自定义**NamedParameterJdbcTemplate**后，spring boot将不会再产生默认的 **namedParameterJdbcTemplate**；

### 使用 log4jdbc+logback输出sql和参数

* 在pom.xml中添加：

<dependency>  
 <groupId>com.googlecode.log4jdbc</groupId>  
 <artifactId>log4jdbc</artifactId>  
 <version>1.2</version>  
</dependency>

* 修改application.properties中的jdbc.url与jdbc.driver：

#spring.datasource.url=jdbc:mysql://localhost:3306/spring\_boot\_f  
spring.datasource.url=jdbc:log4jdbc:mysql://localhost:3306/spring\_boot\_f  
spring.datasource.username=root  
spring.datasource.password=123456  
#spring.datasource.driver-class-name=com.mysql.jdbc.Driver  
spring.datasource.driver-class-name=net.sf.log4jdbc.DriverSpy

* 在resources目录中添加logback-spring.xml文件，添加：

<logger name="jdbc.sqlonly" level="warn"/>  
<logger name="jdbc.sqltiming" level="info"/>  
<logger name="jdbc.resultset" level="warn"/>  
<logger name="jdbc.connection" level="info"/>  
<logger name="jdbc.audit" level="warn"/>

* 使用效果：

spring.boot2 2016-09-24 11:06:04.115 [13482] [main] INFO jdbc.sqltiming - insert into user (age, name, remark, sex) values (28, 'name', 'hihi...', 'M')

{executed in 1 msec}

spring.boot2 2016-09-24 11:06:04.202 [13569] [main] INFO jdbc.sqltiming - select \* from user where id = 1

{executed in 3 msec}

spring.boot2 2016-09-24 11:06:04.465 [13832] [main] INFO jdbc.sqltiming - select \* from user where id in ( 1,2,3 )

{executed in 3 msec}

### 使用 connectionLifecycleInterceptors 监控事务（mysql）

* 参考：<https://dev.mysql.com/doc/connector-j/5.1/en/connector-j-reference-configuration-properties.html>
* 重写事务监听器，实现接口：**ConnectionLifecycleInterceptor**，实现其中的方法：

public interface ConnectionLifecycleInterceptor extends Extension {  
 void close() throws SQLException;  
  
 boolean commit() throws SQLException;  
  
 boolean rollback() throws SQLException;  
  
 boolean rollback(Savepoint var1) throws SQLException;  
  
 boolean setAutoCommit(boolean var1) throws SQLException;  
  
 boolean setCatalog(String var1) throws SQLException;  
  
 boolean transactionBegun() throws SQLException;  
  
 boolean transactionCompleted() throws SQLException;  
}

实现类：

public class ConnectionLifecycleInterceptorImpl implements ConnectionLifecycleInterceptor

* 在jdbc.url中添加参数：

spring.datasource.url=jdbc:log4jdbc:mysql://localhost:3306/test?connectionLifecycleInterceptors=x.y.ConnectionLifecycleInterceptorImpl

* 此时，即会在日志中转出：

spring.boot2 2016-09-24 14:10:34.693 [14666] [main] DEBUG s.m.m.ConnectionLifecycleInterceptorImpl - conn com.mysql.jdbc.JDBC4Connection@78422efb:setAutoCommit=false

spring.boot2 2016-09-24 14:10:34.875 [14848] [main] DEBUG s.m.m.ConnectionLifecycleInterceptorImpl - conn com.mysql.jdbc.JDBC4Connection@78422efb:transactionBegun

spring.boot2 2016-09-24 14:10:34.877 [14850] [main] INFO jdbc.sqltiming - insert into user (age, name, remark, sex) values (28, 'name', 'hihi...', 'M')

{executed in 1 msec}

spring.boot2 2016-09-24 14:10:34.904 [14877] [main] DEBUG s.m.m.ConnectionLifecycleInterceptorImpl - conn com.mysql.jdbc.JDBC4Connection@78422efb:commit

spring.boot2 2016-09-24 14:10:34.917 [14890] [main] DEBUG s.m.m.ConnectionLifecycleInterceptorImpl - conn com.mysql.jdbc.JDBC4Connection@78422efb:transactionCompleted

spring.boot2 2016-09-24 14:10:34.917 [14890] [main] DEBUG s.m.m.ConnectionLifecycleInterceptorImpl - conn com.mysql.jdbc.JDBC4Connection@78422efb:setAutoCommit=true

### 使用redis

* 在pom.xml中添加：

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

* 在application.properties中添加redis配置：

# REDIS (RedisProperties)  
# Redis数据库索引（默认为0）  
spring.redis.database=0  
# Redis服务器地址  
spring.redis.host=localhost  
# Redis服务器连接端口  
spring.redis.port=6379  
# Redis服务器连接密码（默认为空）  
spring.redis.password=  
# 连接池最大连接数（使用负值表示没有限制）  
spring.redis.pool.max-active=8  
# 连接池最大阻塞等待时间（使用负值表示没有限制）  
spring.redis.pool.max-wait=-1  
# 连接池中的最大空闲连接  
spring.redis.pool.max-idle=8  
# 连接池中的最小空闲连接  
spring.redis.pool.min-idle=0  
# 连接超时时间（毫秒）  
spring.redis.timeout=30000

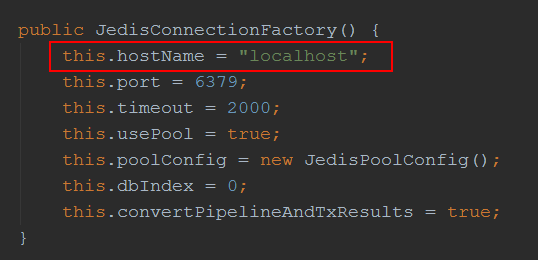
* 配置redis：

@Configuration  
public class RedisBean {  
  
 @Bean  
 JedisConnectionFactory jedisConnectionFactory() {  
 return new JedisConnectionFactory();  
 }  
  
 @Bean  
 public RedisTemplate<String, Object> redisTemplate(RedisConnectionFactory factory) {  
 RedisTemplate<String, Object> template = new RedisTemplate<String, Object>();  
 template.setConnectionFactory(factory);  
 template.setKeySerializer(new StringRedisSerializer());  
 template.setValueSerializer(new Jackson2JsonRedisSerializer(Object.class));  
 return template;  
 }  
  
}

* 使用：

@Autowired  
private RedisTemplate<String, Object> redisTemplate;

* Spring redis默认 host 配置：



注：默认配置是 localhost，此时修改application.properties中的 **spring.redis.host** 是无济于是的，需要使用**自定义的factory**

* 使用自定义的 host：
  + 修改 application.properties：

spring.redis.host=10.199.199.224

spring.redis.port=6379

* + 实现自定义的Factory：

@Configuration  
@ConfigurationProperties(prefix = "spring.redis")  
public class RedisBean {  
  
 private String host;  
 private int port;  
  
 @Bean  
 JedisPoolConfig jedisPoolConfig(){  
 return new JedisPoolConfig();  
 }  
  
  
 @Bean  
 JedisConnectionFactory jedisConnectionFactory() {  
 JedisConnectionFactory factory = new JedisConnectionFactory(jedisPoolConfig());  
 factory.setHostName(host);  
 factory.setPort(port);  
 return factory;  
 }  
  
 @Bean  
 public RedisTemplate<String, Object> redisTemplate(RedisConnectionFactory factory) {  
 RedisTemplate<String, Object> template = new RedisTemplate<String, Object>();  
 template.setConnectionFactory(jedisConnectionFactory());  
 template.setKeySerializer(new StringRedisSerializer());  
 template.setValueSerializer(new Jackson2JsonRedisSerializer(Object.class));  
 return template;  
 }  
  
  
 public void setHost(String host) {  
 this.host = host;  
 }  
  
 public void setPort(int port) {  
 this.port = port;  
 }  
}

### 使用异常方法（@Async）

* Spring boot 启动器上加注解，启用异常方法功能：

@EnableAsync

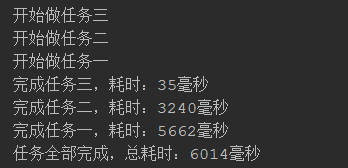
* 使用：

@Component  
public class AsyncTask {  
  
 public static Random *random* =new Random();  
  
 @Async  
 public Future<String> doTaskOne() throws Exception {  
 System.*out*.println("开始做任务一");  
 long start = System.*currentTimeMillis*();  
 Thread.*sleep*(*random*.nextInt(10000));  
 long end = System.*currentTimeMillis*();  
 System.*out*.println("完成任务一，耗时：" + (end - start) + "毫秒");  
 return new AsyncResult<>("任务一完成");  
 }  
  
 @Async  
 public Future<String> doTaskTwo() throws Exception {  
 System.*out*.println("开始做任务二");  
 long start = System.*currentTimeMillis*();  
 Thread.*sleep*(*random*.nextInt(10000));  
 long end = System.*currentTimeMillis*();  
 System.*out*.println("完成任务二，耗时：" + (end - start) + "毫秒");  
 return new AsyncResult<>("任务二完成");  
 }  
  
 @Async  
 public Future<String> doTaskThree() throws Exception {  
 System.*out*.println("开始做任务三");  
 long start = System.*currentTimeMillis*();  
 Thread.*sleep*(*random*.nextInt(10000));  
 long end = System.*currentTimeMillis*();  
 System.*out*.println("完成任务三，耗时：" + (end - start) + "毫秒");  
 return new AsyncResult<>("任务三完成");  
 }  
  
}

* 测试：

@RunWith(SpringJUnit4ClassRunner.class)  
@SpringBootTest  
public class AsyncTaskTest {  
  
 @Autowired  
 private AsyncTask asyncTask;  
  
 @Test  
 public void test() throws Exception {  
  
 long start = System.*currentTimeMillis*();  
  
 Future<String> task1 = asyncTask.doTaskOne();  
 Future<String> task2 = asyncTask.doTaskTwo();  
 Future<String> task3 = asyncTask.doTaskThree();  
  
 while(true) {  
 if(task1.isDone() && task2.isDone() && task3.isDone()) {  
 // 三个任务都调用完成，退出循环等待  
 break;  
 }  
 Thread.*sleep*(1000);  
 }  
  
 long end = System.*currentTimeMillis*();  
  
 System.*out*.println("任务全部完成，总耗时：" + (end - start) + "毫秒");  
  
 }  
  
}

* 控制台显示：



### 使用spring batch

* 修改pom.xml

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

* 修改application.properties

spring.batch.initializer.enabled=true  
spring.batch.job.enabled=false #应用启动后不启动job  
spring.batch.job.names=noJob #应用启动后不启动job，noJob指无需要启动的job

* 具体使用参照：
* 
* Reader参数传递与Job多次执行支持：

@RequestMapping("job")  
public String job(@RequestParam("mod") Long mod) throws Exception {  
 JobParameters jobParameters = new JobParametersBuilder()  
 // 传递参数到reader  
 .addLong("mod", mod)  
 // job可以被多次执行  
 .addLong("mills", System.*currentTimeMillis*())  
 .toJobParameters();  
 jobLauncher.run(testJob, jobParameters);  
 return "index";  
}

@StepScope  
@Bean  
public ItemReader<User> itemReader(PagingQueryProvider pagingQueryProvider,  
 @Value("#{jobParameters['mod']}") Long mod)  
 throws Exception {  
 JdbcPagingItemReader<User> reader = new JdbcPagingItemReader<>();  
 reader.setDataSource(primaryDataSource);  
 reader.setQueryProvider(pagingQueryProvider);  
 // reader 参数传递  
 reader.setParameterValues(new HashMap(){{  
 put("mod", mod);  
 }});  
 reader.setRowMapper(new RowMapper<User>() {  
 @Override  
 public User mapRow(ResultSet resultSet, int i) throws SQLException {  
 User user = new User();  
 user.setId(resultSet.getLong("id"));  
 user.setName(resultSet.getString("name"));  
 user.setAge(resultSet.getInt("age"));  
 user.setRemark(resultSet.getString("remark"));  
 user.setSex(resultSet.getString("sex"));  
 return user;  
 }  
 });  
 reader.setPageSize(300);  
 return reader;  
}