Spring-Boot的相关最佳实践

(1). 不建议使用java中的默认包 package ----- 如果这样使用 对注解@ComponentScan, @EntityScan和@SpringBootApplication的使用会产生一些问题 =🡺 这是因为每一个jar的每一个class都会被扫描到



通常建议把main application class定位到其他类的root package

@EnableAutoConfiguration通常被标注在你的main class上面 并且对一定的items就会定义一个基本的搜索路径 ---- search package

比如 如果你正在构建一个JPA的application 那么 @EnableAutoConfiguration这个注解标注的类的package就会被用来search @Entity标注的class

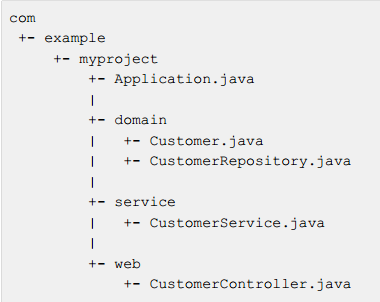
【从字面意思上面看 @EnableAutoConfiguration含义就是启用自动配置 那么 自动配置的东西就应该在这个标注的类所在的包下面(实际上 包在文件系统中不就是路径么)】

----- 对于 default package的使用建议

使用root package运行使用不指定basePackage属性的@ComponentScan这个标注

如果你的main class是在root package中的话，那么**@SpringApplication**(含义就是Spring 应用)是可以被使用

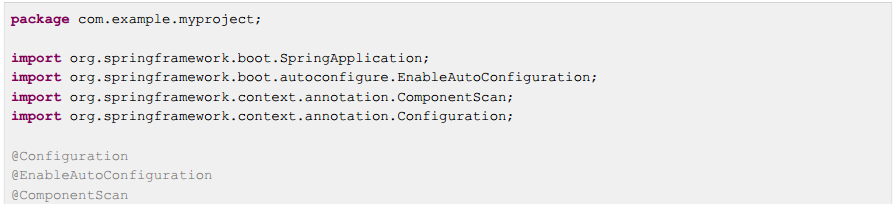
这里有一个典型的文件布局



可以看出 Application.java就在所有的其他类的最上面

这样根据刚才的说明 这个Application为root目录的所有子目录下面的class就是基本的search path

基于这样的一个文件结构 配置的主类main class的文件写法如下



-------------------------------------------

15. Configuration Classes ----- 类文件的配置

可以使用SpringApplication.run()来启动应用 但是推荐primary class是被@Configuration标注的类

=🡺通常的做法：定义了main方法的类也是一个**主要的@Configuration**标注的类

小技巧

尽量使用等价的基于Java的配置来替代XML配置 ----- 做到这一点可以从尝试查找enable开头的注解作为开始

15.1 Importing additional configuration classes(也就是引入@Configuration标注的Class)

这个和配置xml的思路是一致的

如果一个大项目可以通过xml拆分成多个小的xml 然后可以在最外面的xml使用<import>标签进行整合

这样 我们同样使用@Import进行多个@Configuration的整个【因为一个@Configuration代表一个配置文件】

@Import注解可以用来引入额外的@Configuration标注的class

<==> 也可以使用@ComponentScan来自动引入所有的Spring component ---- 这样包括引入@Configuration标注的class

15.2 Importing XML configuration

【这个意思是一样的 就是 @Configuration实际上就等价于XML的配置文件】

就是 如果系统非要使用xml进行配置 ---- 也建议使用一个@Configuration标记的类进行统一的整合 这时候使用对应的@ImportResource注解进行整合 ---- 这个用来加载xml文件

-----------

16. Auto-Configuration 自动配置

Spring Boot的auto-configuration尝试你已经添加的jar的依赖来自动配置你的Spring应用

比如 如果HSQLDB位于你的classpath下面 并且 你并没有手动配置任何数据库连接的beans ---- 那么我们就会自动配置一个内存数据库

【实际上就是 你的classpath有相关的jar 如果使用了auto-configuration 那么即使是没有手动配置对应的<dependency> 我们也会按照在classpath路径中的jar 来进行配置】

------ 如何开启自动配置呢？

可以在其中的一个@Configuration标注的类上面打上注解@EnableAutoConfiguration或者

@SpringBootApplication这个注解

【自动的意思就是 你不去手动配置 也会根据类路径中的jar进行相关的配置】

--- Tips 应该仅仅使用一个@EnableAutoConfiguration这个注解 通常推荐这个注解添加在你的primary @Configuration的class上面

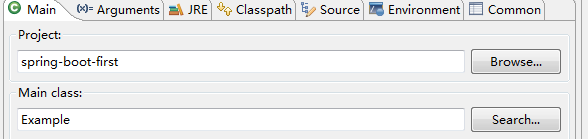
16.1 逐渐替换auto-configuration

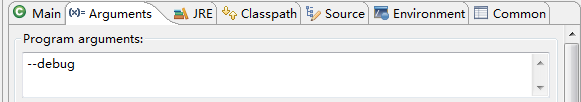
----- 因为如果使用auto-configuration 这样很多配置是不可控的 所以 渐进逐步替换

比如 你添加了自己的DataSource的Bean 那么默认的嵌入的database的支持就会被去掉

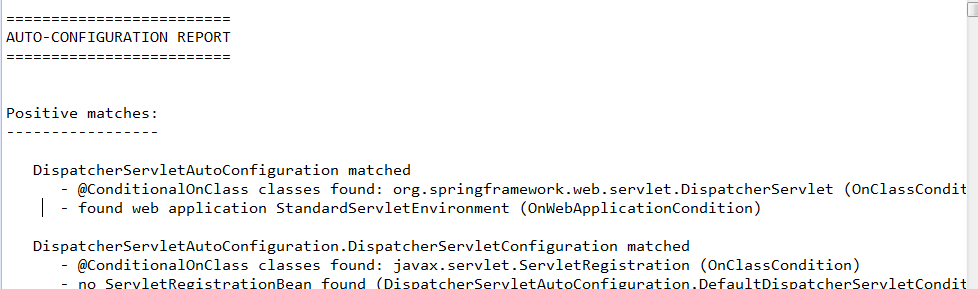
可以在启动你的java应用 + --debug来查看正在使用auto-configuration配置出来的自动配置有哪些

这样使用--debug启动 会把这种auto-configuration配置出来的信息记录到控制台上面



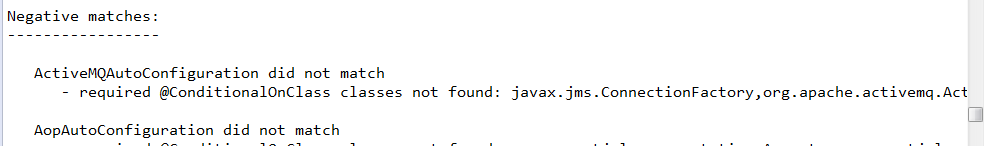


这样启动之后



其中的这个 Positive matches就是在我规定的这些auto配置条目中，现有的工程有哪些jar可以匹配上

还有一个Negative matches

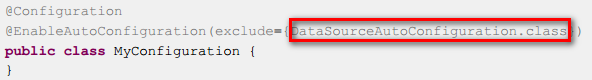


这个就是规定有的 但是 你的classpath中没有这个jar 这样 就会被列在这个negative matches这个report中

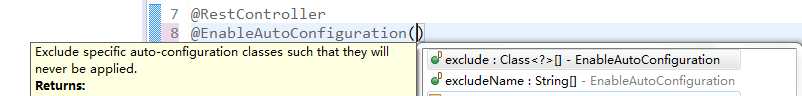
16.2 禁掉指定的auto-configuration

在找到了有哪些auto-configuration的items之后 可以对你并不想要的item进行禁用

比如我们找到的DataSourceAutoConfiguration正在被使用 --- 此时我们想要替代掉这个自动配置出来的选项 我们可以在@EnableAutoConfiguration中使用exclude这个选项进行过滤 保证对应的Configuration不被自动配置



如果这个class没有在你的classpath下面 ----> 你可以使用excludeName这个属性并且制定全类名来替代上面exclude属性+class



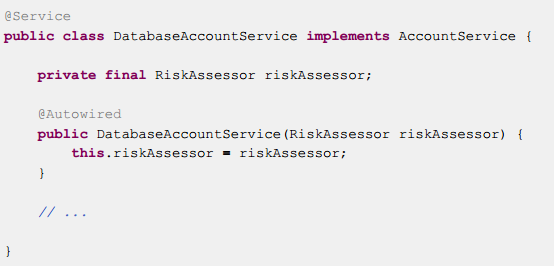
【Auto-Configuration的启动时机：是在用户定义的bean被注册成功之后 才开始起作用 =🡺 这样就能保证我定义的模块可以覆盖掉对应的auto-configuration模块】

17. Spring Beans和依赖注入

通常为了简便，通常使用@ComponentScan注解来查找你定义的beans

并且配合使用@Autowired进行依赖注入

如果你的代码结果可以在一个root package中被定位出来，那么你的@ComponentScan就可以不使用任何参数 ------ 这样所有@Component，@Service，@Repository和@Controller等等的application Components都将自动配置为Spring的beans

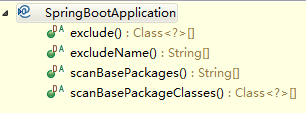


这个是使用@Autowired进行构造注入

18. @SpringBootApplication注解的使用

通常@Configuration + @ComponentScan + @EnableAutoConfiguration是联合使用的 ---- 这是前面讲的最佳实践

===> 这三个注解的默认参数和他们注解本身合到一起 可以简写成一个@SpringBootApplication

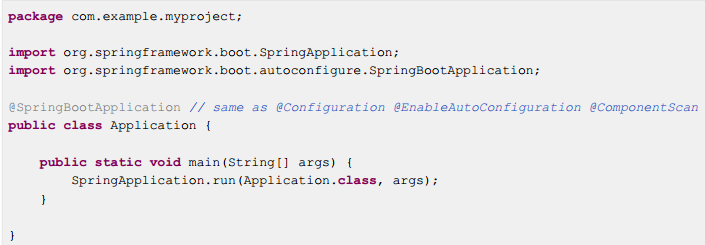


其中 exclude()+ excludeName()这两个属性是@EnableAutoConfiguration注解的属性

scanBasePackages() + scanBasePackageClasses()是@ComponentScan的属性

还有就是@Configuration没有自己的属性 所以 这样@SpringBootApplication只有四个属性

我们可以直接调解@SpringBootApplication的属性来调节相应的@ComponentScan和@EnableAutoConfiguration 其中@EnableAutoConfiguration就是spring的boot模块的注解



19 运行你的Application

19.1 脱离IDE来运行你的Application

你可以仅仅以一个简单的Java应用来运行你的Spring Boot应用 ---- 这样做的前提就是你需要正确导入你的maven工程

19.2 以一个打包的application形式进行运行

如果使用Spring Maven或者Gradle生成了一个可执行的jar包 我们可以以打包的方式来运行这个应用



也可以开启远程调试



使用Maven 插件生成可执行的jar 并进行启动

Spring Boot Maven插件包含了一个run ---- 这个工具可以快速编译并且运行你的应用

Applications可以按照IDE中那种分散的组织形式来运行

热替换 ---- hot swapping ---- 可以使用JRebel 或者Spring Loaded项目

这个spring-boot-devtools模块也包含了对应用快速重启的支持

----- 第20章会详细介绍这个问题

20. Developer Tools ---- 开发者工具

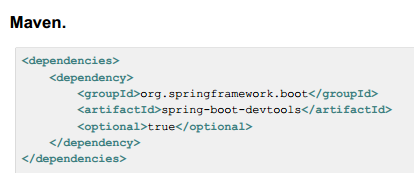
------ Spring Boot中的SpringApplication这个类的用途 不要混淆了

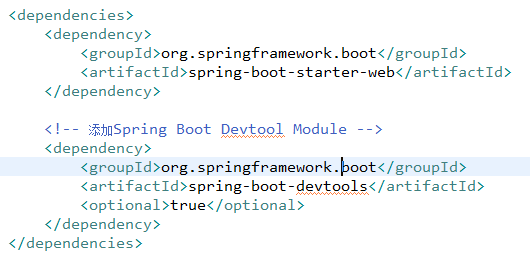
Boot中的SpringApplication.run是启动的Web项目 不是普通的应用

Spring Boot有一套可以使用应用开发体验更好的工具

这个spring-boot-devtools模块可以被嵌入到任何项目中来提供额外的开发支持

为了包含这个devtools支持 ---- 简单吧下面的模块依赖添加到你的项目中

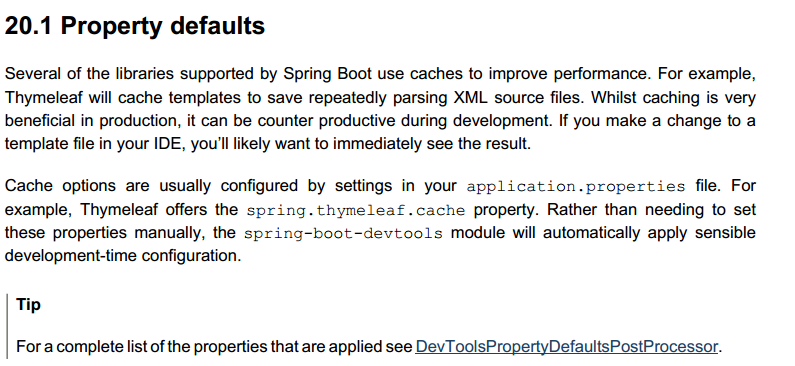




这里面这个<option>标签的作用

如果你的应用是使用java –jar启动或者使用了一个特殊的classLoader那么这个应用就被认定为生产应用 ---- production application

把这个dependency标记为optional --- 也就是可选的dependency ----> 这是一种最佳实践 ---- 可以防止这个devtools被引用了你的module的project也引用到了这个devtools



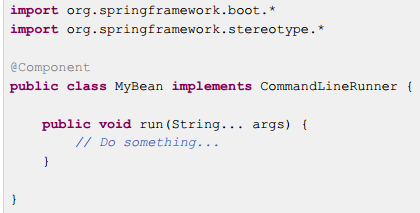
20这一块没看完 先不看了

23.7 Using the ApplicationRunner or CommondLineRunner 使用ApplicationRunner或者CommandLineRunner

如果你需要在SpringApplication已经运行的时候，运行指定的代码，你可以实现ApplicationRunner或者CommandLineRunner接口

这两个接口都运行在一种方式并且提供了一个单一的run方法 ----- 这个run方法会在SpringApplication.run()方法被调用之前调用。

尽管ApplicationRunner使用了ApplicationArguments接口，但是这个CommandLineRunner以简单的字符串数组的形式来访问应用的参数



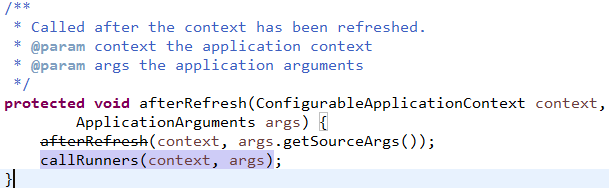
你可以额外实现org.springframework.core.Orderd接口 或者

使用org.springframewporkl.core.annotation.Orderd注解 ----- 这是如果有急几个CommandLineRunner或者ApplicationRunner 这样的beans被定义 并且 这些被定义的beans必须按照一定的顺序进行调用。

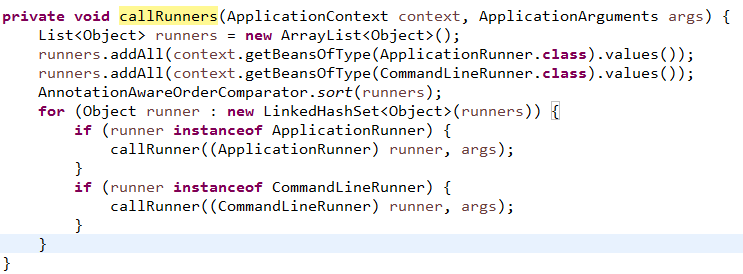
【查看了一下源码

在org.springframework.boot.SpringApplication这个类中

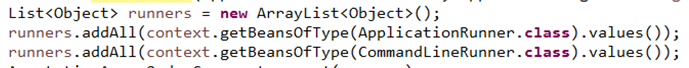
重写了afterRefresh这个方法



在这个callRunner(context, args)的私有方法中 调用了 CommandLineRunner的实现类 作为参数

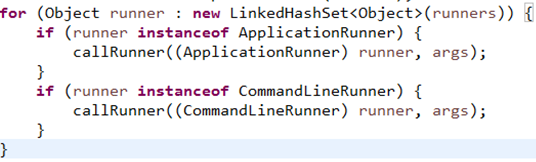


-----可以看到 这个有一个for循环 是 runners



这个里面 通过context通过getBeansOfType找到ApplicationRunner和CommandLineRunner的实现类 放到这runners list中

然后对这个list进行遍历



这样 就调用callRunner((Xxxx) runner, args)

哪一种类型的接口实现类 就调用哪一种对应的runner

注意

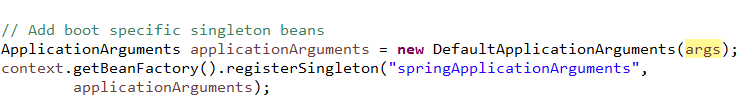
这个args的类型就是上面的



这个args的类型是ApplicationArguments类型

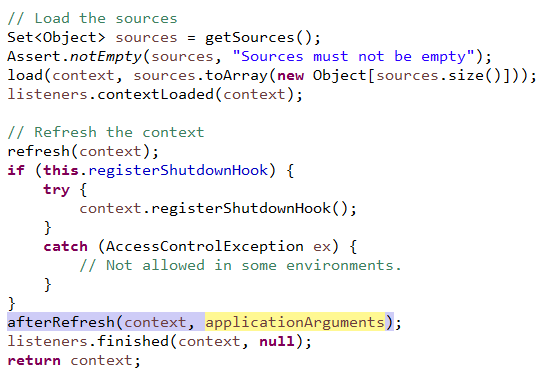
这个参数是在SpringApplication#doRun(SpringApplicationRunListeners listeners, String… args)

这里面进行了调用

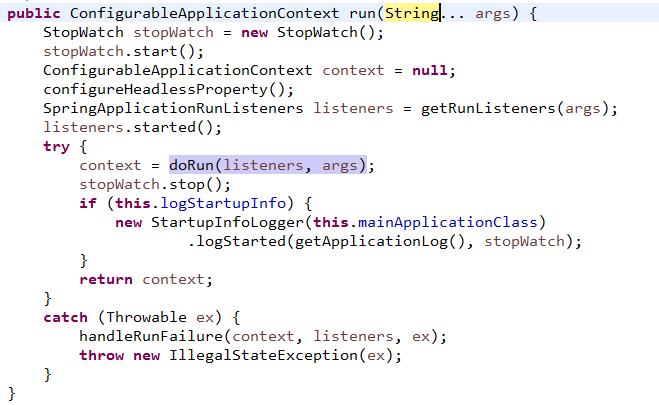


这样就把字符串类型的args变成了ApplicationArguments类型的参数

最后 在afterRefresh(context, applicationArguments)中调用了前面的字符串类型的参数



-------------- 这个doRun是在哪里调用的呢？----- 就是在SpringApplication.run()这个静态方法中调用的



所以这个args就是你的main方法中的String[]… 字符串数组类型的参数】

24. 外部化的配置

Spring boot 允许你把你的配置暴露在外面 这样 你可以在不同的环境里面运行相同的应用代码。你可以使用properties文件，YAML文件，环境变量和命令行参数来外部化你的配置。 Property 的值通过@Value可以被直接注入到你的beans中 ---- 可以通过Spring的环境抽象进行访问 或者通过@CofigurationProperties绑定结构对象

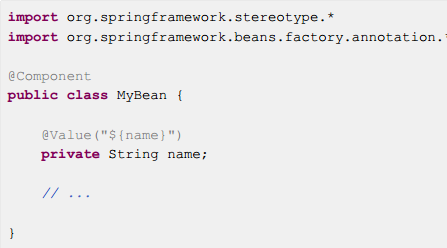
Spring Boot使用一个非常特别的PropertySource顺序 --- 这个顺序是用来运行合理的值的覆盖。

Properties会按照下面的顺序进行覆盖：

命令行参数-> 来自SPRING\_APPLICATION\_JSON的Properties（嵌入在一个环境变量或者嗣同属性的内置JSON）->来自java:comp/dev下面的JNDI属性 -> Java系统属性(System.getProperties()) -> OS环境变量 -> 一个在random.\*.中仅仅有properties的RandomValuePropertySource –> 位于jar包外部的指定的Profile应用properties – 比如 application-(profile).properties和YAML变量 -> 位于jar内部的 xxx -> 位于你的jar外部的Application属性（application.properties和YAML变量）-> 位于你的jar内部的Application属性 -> 位于@Configuration标注的类的@PropertySource注解 -> 指定的使用SpringApplication.setDefaultProperties指定的默认属性

下面是一个具体一点的例子

假设你开发了一个@Component注解标注的类 这个类有一个name属性

 ---- 这个类使用了@Value来直接获取对应的属性值

在你的应用的类路径或者你的jar内部 你可以有一个application.properties文件 这个文件为name提供了一个合理的默认值。当在一个新的环境运行的时候，一个application.properties文件可以在你的jar的外面提供name 进行覆盖。。。。 为了一次性的测试，亦可以运行=命令行 比如 java –jar app.jar –name=”Spring” ==== 前面说过了 properties的参数首先就来自命令行 所以 命令行是提供properties的一个途径。

26. Logging

Spring Boot内部日志使用Commons Logging组件， 但是对潜在的log实现进行了开放预留。

对java util Logging, Log4j, Log4j2和Logback提供了默认的配置。

在每一种情况中，logger都已经预先进行了配置来使用命令行进行输出，也可选文件输出进行备选。

**默认情况，如果你使用了”Starter POMs”, logback将被用来作为日志记录的工具。**

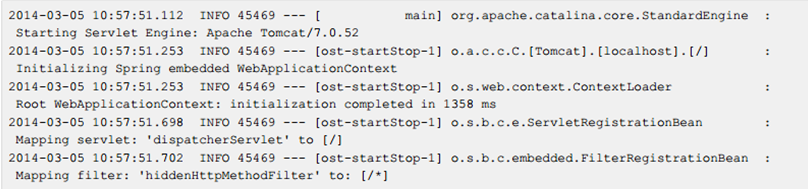
其当的logback路由也可以包含进来 这样确保独立的使用了java util logging Commons Logging Log4j或者SLF4j的独立的lib可以全部正常工作。【logback没有全部看完 但是 logback貌似可以做到这一点】

【技巧】

对于Java 有很多可用的日志框架。如果上面的框架列表让你感觉到困惑，其实没事。通常情况下，你不必修改你的logging依赖并且Spring Boot的默认会很好的进行工作。

26.1 日志形式

来自Spring boot的默认日志输出形式如下：





日期时间 + 日志级别（INFO ERROR DEBUG WARN FATAL）+ process id + 分隔符（用来标记真正日志起始位置+ 线程名字（使用[]进行扩住）+ log信息来源的类名（叫做Logger Name --- 因为这个类输出了日志 所以交Logger Name）+ 真正的日志信息

**[注意]**

**Logback没有FATAL级别 ---- FATAL被ERROR取代**

-----------------

26.2 控制板输出

默认的log配置就会在信息被记录的同时也输出到控制台上。默认情况ERROR, WARN和INFO级别的日志被记录。你可以通过使用—debug这个表示来开启debug模式



----------------------

另一种开启debug的方式就是在application.properties中添加

debug=true这个配置

当这个debug模式被打开的时候，一系列的核心logger（嵌入的web容器，Hibernate和Spring）都会被配置输出更多的信息。启动debug模式并不会配置你的应用日志魏DEBUG Level。

26.3 颜色是醋户

如果你的终端支持ANSI，带颜色的输出可以用来增加日志的可读性。你可以设置spring.out.ansi.enabled为一个支持的值来覆盖自动的检测值。

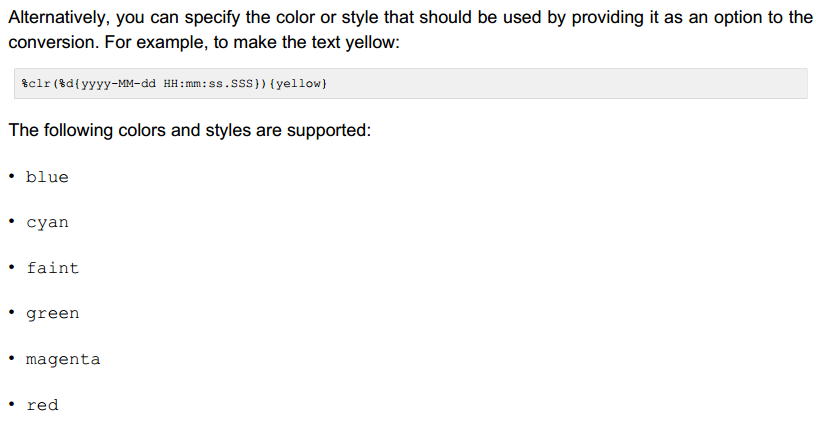
带颜色的coding的配置使用%clr这个词。 在最简单的的形式中这个转换器会按照log level来为输出上颜色。比如



Log level和颜色的对应关系如下



作为可选，你可以指定color或者样式通过



===

26.3 文件输出

默认情况下，Spring Boot仅仅把日志打印到控制台 但是并不会写日志文件。出了输出到控制台以外，你还想要写日志文件，你需要设置一个logging.file或者logging.path属性（在application.properties文件中）

下面的表回给出logging.\*属性如果被联合使用

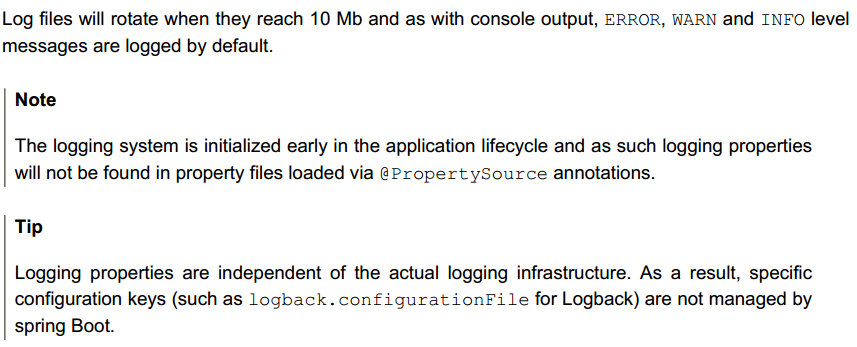
日志属性：

Logging.file logging.path Example Decription

(none) (none) 仅仅输出到控制台

指定文件 (none) my.log 会把日志写入到指定日志文件。文件名可以是一个绝对精确的路径或者一个相对于当前文件夹的相对路径

(none) 指定路径 /var/log 在指定的目录中写一个spring.log的日志文件。要求同上



注意

这个日志系统会在早期的应用的生命周期被初始化并且 这样的日志属性不会在属性文件（@PropertySource这个注解加载）中找到

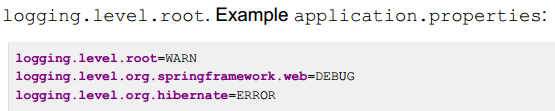
提示

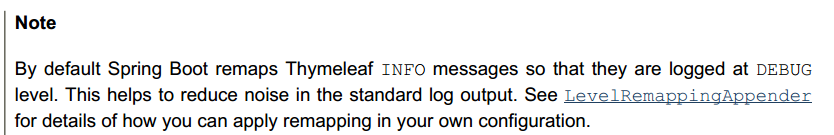
日志属性是和真正的日志框架是相互独立的。结果，指定的配置key（比如 Logback的logback.configurationFile）并不是被Spring Boot所管理。

===== 26.4日志级别

所有支持的日志系统可以拥有设置在Spring 环境【比如 就是在application.properties】中使用了logging.level.\*=LEVEL 设置的日志级别。 Level是TRACE DEBUG INFO WARN ERROR FATAL OFF之一。Root logger可以使用logging.level.root来进行配置。

比如





====

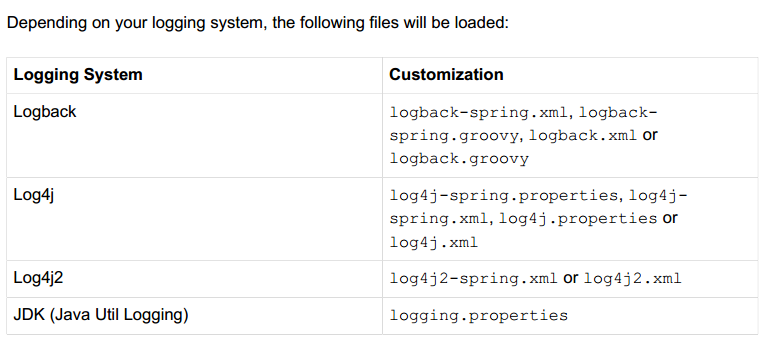
26.5定制自己的log配置

各种日阿志系统可通过在类路径中包含一个恰当的lib和更进一步通过在classpath的根路径提供一个恰当的配置文件来激活 ---- 或者也可以通过在Spring 环境【就是前面支出的application.properties中设置】指定的属性logging.config来激活。

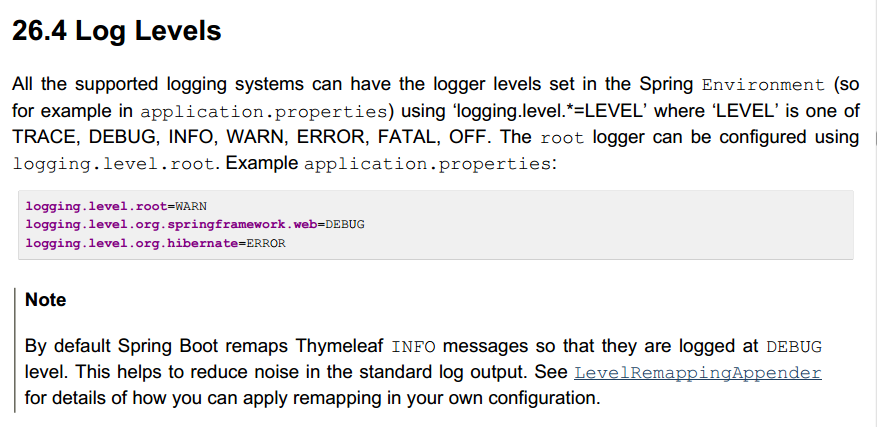
注意

由于logging是在ApplicationContext创建之前就被初始化了，所以不能通过在Spring的@Configuration文件中的@PropertySources来控制日志信息。

系统属性和传统的Spring Boot外部配置文件都可以很好的工作(application.properties)



==== 后面差一点就不看了



27. Developing web applications ---- 开发Web应用

27.1 The Spring Web MVC framework

Spring MVC让你创建简单的@Controller或者@RestController的beans来处理过来的HTTP请求 ---- 你的Controller中的方法通过@RequestMapping注解被映射成HTTP

------ Spring MVC的 auto-configuration

Spring Boot为Spring MVC提供了auto-configuration

这个auto-configuration为Spring的默认添加了下面的特性

(1). 包含ContentNegotiatingViewResolver和BeanNameViewResolver

(2). 支持为静态资源（含有WebJars）的支持

(3). 自动为Converter, GenericVonverter, Formatter的Beans进行注册

(4). 支持HttpMessageConverters【和@RequestBody， @ResponseBody相关】

(5). 支持自动注册MessageCodesResolver

(6). 静态的index.html支持

(7). 客户化的Favicon的支持

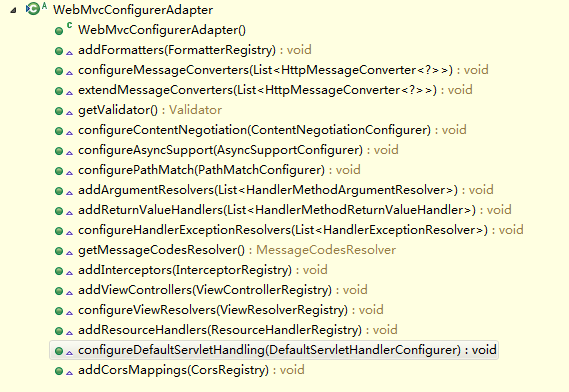
(8). ConfigurableWebBindingInitializer的Beans的自动使用

如果你想完全控制Spring MVC ----- 你可以添加你自己的@Configuration---- 这个@Configuration被@EnableWebMVC标注过 ----- 【enablexxx 表示启用xxx 这样就是要求自己来配置 不要auto-configuration】

如果想保持Spring Boot MVC的特点 并且仅仅添加额外的MVC配置(interceptors, formatters, view controllers等等) ---- **你可以添加WebMvcConfigurerAdapter标注的@Bean --- 但是不要使用@EnableWebMvc**

这两个注解到网上搜 就能搜索到如何配置 ---- 这个注解是Spring3.1开始的

http://blog.csdn.net/giveyousool/article/details/7177906

[http://cache.baiducontent.com/c?m=9d78d513d98616ff07a9c52f504a833d4c13c0347c8486463d92c40884652b563761f4ca21351174c4b37d7070ae5e2cefe74578234460e99492ce0c9fac935b3295776a2d499141658258f9975125b67cc71cf4fe47bafdaf658efd929584050d9d4e532dc6a3d0044109cc65f6153ae5a692&p=987dc64ad4934eaf5be8c02557498d&newp=9c769a479fd006b708e2947d085c9e231610db2151d7d31636c7d450&user=baidu&fm=sc&query=webmvcconfigureradapter%B5%C4%D3%C3%B7%A8&qid=f9c700200003e3e5&p1=2](http://cache.baiducontent.com/c?m=9d78d513d98616ff07a9c52f504a833d4c13c0347c8486463d92c40884652b563761f4ca21351174c4b37d7070ae5e2cefe74578234460e99492ce0c9fac935b3295776a2d499141658258f9975125b67cc71cf4fe47bafdaf658efd929584050d9d4e532dc6a3d0044109cc65f6153ae5a692&p=987dc64ad4934eaf5be8c02557498d&newp=9c769a479fd006b708e2947d085c9e231610db2151d7d31636c7d450&user=baidu&fm=sc&query=webmvcconfigureradapter%B5%C4%D3%C3%B7%A8&qid=f9c700200003e3e5&p1=2)

看一下这两个类 一个注解 一个这个的java源码

**HttpMessageConverters**

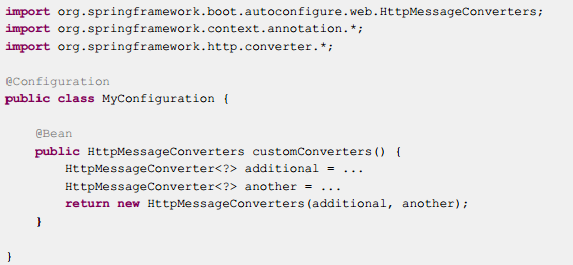
SpringMVC使用HttpMessageConverter接口来转换Http请求和响应。

合理的默认值都是立即可用的 --- 比如 对象可以自动转换成JSON

（使用的是Jackson library）或者XML（使用Jackson XML扩展 如果不行 就使用JAXB）

如果你需要添加或者自己定制converters ---- 你需要使用Spring Boot的

HttpMessageConverter*s*这个类[多了一个s]



任何位于这个context中的HttpMessageConverter都会被添加到相应的list列表中

你也可以使用这种方式来覆盖默认的转换器

29. 和SQL数据库协同工作

Spring框架为和SQL数据库协同工作提供了非常多的支持。从使用JdbcTemplate来直接操作JDBC到完全的ORM技术，比如 Hibernate。Spring Data提供了提供了额外的一层功能性，能够直接从Repository接口创建实现并且使用约定来从你的方法名生成查询语句。

29.1 配置DataSource数据源

Java的javax.sql.DataSource接口提供了一个和数据库链接工作的标准接口。传统的一个数据源使用一个带有一些验证的信息（用来建立一个数据库链接）的URL。

嵌入的数据库支持

使用一个内存嵌入的数据库来开发应用通常是很方便的。很显然，内存数据库不提供持久化的存储 ---- 当你的开始并且准备在你的应用结束的时候抛出数据的时候考虑迁移你的数据库。

【populate --- 移植于 移民于】

**【技巧】第74部分告诉你如何初始化一个数据库**

Spring Boot可以自动配置嵌入式的Hibernate, HSQL和Derby数据库。你不必提供任何URL --- 仅仅包含一个你想食用的<dependency>即可。

举例：典型的POM依赖如下



【先配置spring data jpa + 内存数据库的依赖】

不管因为什么原因，你确实要为一个嵌入式的数据库配置链接的URL的时候，你需要确保数据库的自动关闭是禁掉的。如果你正在使用H2，你应该使用DB\_CLOSE\_EXIT=FALSE来达到上面的目的。如果你使用的是HSLDB，你应该确保shutdown=true没有被使用。禁掉数据库的自动关闭功能就能允许在数据库被关掉的时候，Spring Boot可以控制得到 --- 所以 确保这种禁掉自动关闭的功能发生一次 --- 这样访问数据库就不再需要了。

【注意】

你需要为一个嵌入式的数据库在spring-jdbc上面添加一个依赖 以达到能够自动配置。 这个例子中，这个依赖就通过spring-boot-starter-data-jpa被传递的过来。

生产数据库的连接

使用一个池化的DataSource可以使得一个生产数据库被自动配置。下面是选择一个指定的数据库实现的算法：

\* 由于性能和并发性的影响，我们更喜欢使用Tomcat池化的DataSource ---- 所以 如果我们如果已经选择了 这个tomcat池化的DataSource就是可用的。

\* 如果HihariCP可用，我们就使用这个

\* 如果Commons DBCP可用 就用这个 但是 我们不推荐这个数据源应用到生产

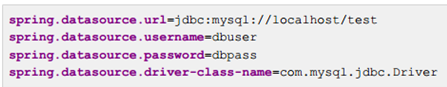
\* 最后 如果是DBCP2 可用 我们就使用这个

如果你是用了带有spring-boot-starter-jdbc或者spring-boot-starter-data-jpa的starter POM 那么你将自动获取到tomcat-jdbc的依赖。

注意

你完全可以略过上面的算法并且通过spring.datasource.type属性指定连接池。同时，额外的连接池也可以经常被手动配置。如果你定义了你自己的DataSource Bean---- 自动配置就不会发生。

数据源的配置是通过定义在外部配置属性中的spring.datasource.\*控制的。举个例子，你可以在application.properties中定义了下面的片段：



看一下DataSourceProperties获得更多的支持选项。注意到可以通过spring.datasource.\*来配置任何一种指定的数据源的实现。

【提示】

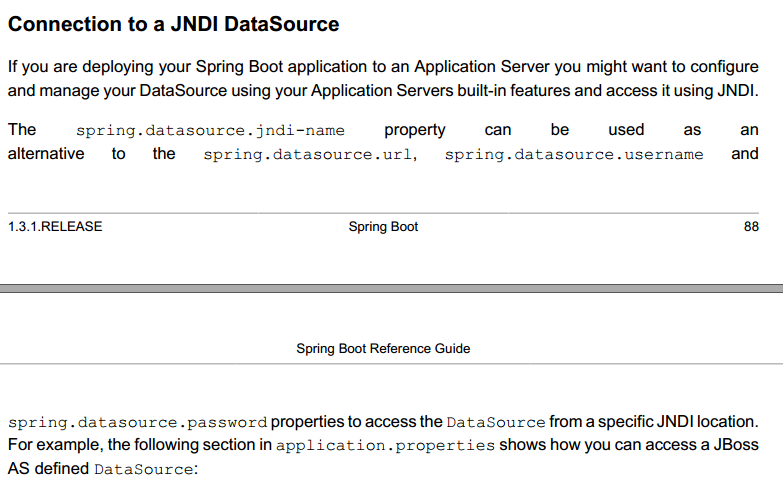
由于spring Boot可以从大多数额数据库的url上面推断出driver-class-name，所以可以不必指定这个url。

【注意】

对于要创建的一个池化的dataSource，我们需要能够验证一个有效的Driver是否可用，所以我们可以在做事情之前进行检查 --- 也就是 如果你设置了 spring.datasource.driverClassName=com.mysql.jdbc.Driver 那么这个类就会被加载进来。

JNDI数据源的链接

如果你正在把一个Spring Boot的应用部署到一个应用服务器 --- 你想使用这个应用服务器的内置特点来配置和管理你的数据源 并且使用JNDI来访问这个数据源。

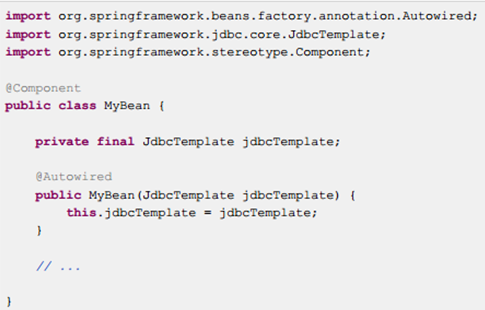


这个spring.datasource.jndi-name属性可以作为spring.datasource.url, spring.datasource.username, Spring.datasource.username和spring.datasource.password的替代品 ---- 用来访问来自一个指定JNDI位置。

举个例子：下面的application.properties中的片段可以访问一个作为数据源的JBoss



29.2 使用JdbcTemplate  
 Spring的JdbcTemplate和NamedParameterJdbcTemplate这两个类都能自动配置并且可以在你的Beans中直接使用@Autowired进行标注



------------------

29.3 JPA和Spring Data

JPA是一个允许你把对象映射到关系型数据的标准技术。Spring-boot-starter-data-jpa POM提供了一种快捷的方式来开始。提供了下面关键的依赖

\* Hibernate – 一种最著名的JPA实现

\* Spring Data JPA --- 实现基于JPA的数据库访问更加容易

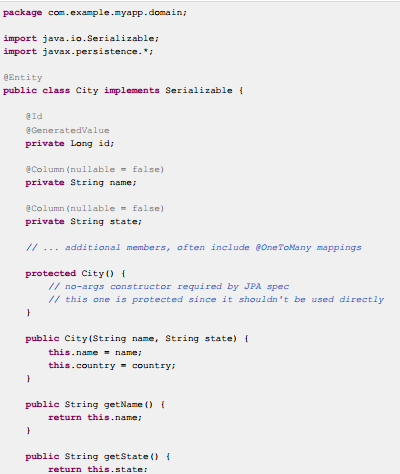
\* Spring ORMs --- 来自Spring Framework的核心ORM支持

**Entity Classes**

传统的，JPA Entity 类都会制定在persistentce.xml这个文件中。在Spring Boot中，这个文件不是必须的并且使用Entity Scanning进行替代。默认情况下，位于你的主配置类下面的所有的包都会被检索到。

---- 主配置文件(就是使用了@EnableAutoConfiguration或者@SpringBootApplication标记过的类)

任何使用了@Entity, @Embeddable或者@MappedSuperClass标记过的类都会被考虑到。一个典型的实体类就会看起来像下面一样：



【技巧】

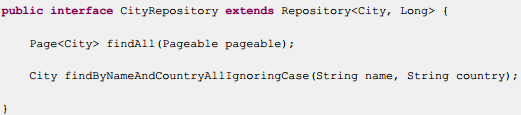
你可以使用@EntityScan这个注解来个性化定制你要扫描的位置。

**Spring Data JPA Repositpories**

Spring Data JPA 都是你可以定义访问数据的接口。JPA的查询都会从你的方法名上面自动创建出来。举例 一个CityRepository接口也许声明了一个findAllByState(String state)方法来找到所有给定state的城市。

Spring Data接口通常会继承Repository或者CrudRepository接口。如果你使用了自动配置，repositoies就会从含有你的主配置类所在的包中搜索到。

这是一个典型的Spring Data的Repository



=============

**创建和关闭JPA数据库**

默认情况下，仅仅如果你使用了一个嵌入的数据库（H2， HSQL 或者 Derby）JPA的数据库，JPA数据库就会呗自动建立。你可以使用spring.jpa.\*属性显式配置JPA。举个例子，为了创建和丢弃数据库表，你可以在application.properties中添加下面的属性



【注意】

Hibernate自己的内的属性名是 hibernate.hbm2ddl.auto.。你可以和其他的Hibernate本地的属性一起进行设置 ---- 配置方式是spring.jpa.properties.\*进行配置 ---- 这些属性在送给Entitymanager之前会把前缀去掉

比如



就会把spring.jpa.properties.hibernate.globally\_quoted\_identifiers传递给EntityManager

默认情况下，DDL的执行或者验证会推迟到直到ApplicationContext已经启动才进行。这也有一个spring.jpa.generate-ddl标志 ---- 但是这个配置没有被使用 ---这是因为如果Hibernate的自动配置激活的话 ddl-auto配置更加细致。

74. DataSource初始化

依据你的堆栈信息，一个SQL数据库可以有多种初始化的方式。当然了，只要数据库是一个单独的过程，数据库你也可以手动初始化数据库。

74.1 使用JPA初始化一个数据库

JPA在DDL的生成方面有一些特点。并且这些特点可以依据数据库在一开始启动的时候就建立起来。这是通过下面两个额外的属性控制的：

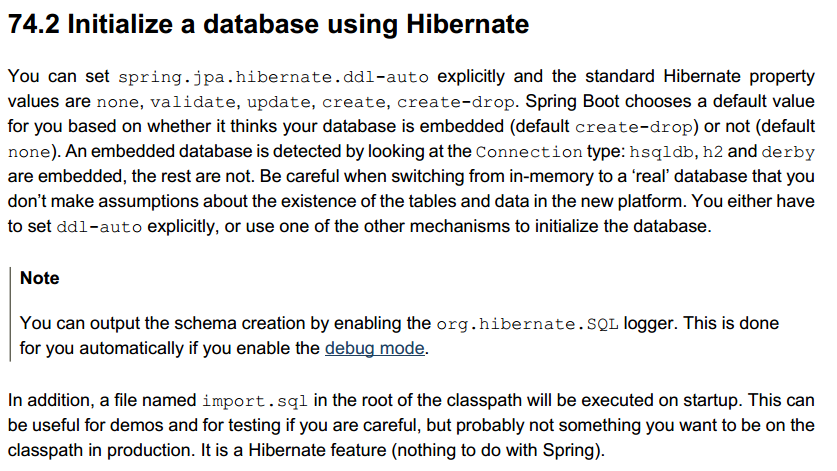
\* spring.jpa.generate-ddl(boolean)控制着这个开关的开和闭 ---- 并且和数据库运营商没有关系。

\* spring.jpa.hibernate.ddl-auto(enum) 是一个可以在更细的粒度层面控制行为的Hibernate特征。

74.2 使用Hibernate初始化数据库

你可以显式设置spring.jpa.hibernate.ddl-auto并且这个标准的Hibernate属性值是none, validate, update, create, create-drop。Spring Boot会基于是否你的数据库是嵌入的（默认是 create-drop）或者不是（默认是 none）。一个嵌入式的数据库可以通过查看Connection类型 是否是hsqldb, h2和derby来确定是不是嵌入式数据库。当从内存数据库切换到真实的数据库你时候 --- 一定要当心在新的平台你没有做表和数据存在的假设。要么直接显式设置ddl-auto 要么使用其他的机制来初始化数据库【我在我的emailSms中就是显式指定这个auto-ddl属性来完成Dao测试的 否则会出问题 ---- 数据库无法初始化 但是 zanghongmin使用的是Flyw这个东西】

74.3



=============== 项目中使用的Log4j配置 =======================

# config Logback --logging system to a specified path

# in the product enviroment, should be assigned to a linux path...

#debug=true

logging.path=E:/slog/

logging.level.root=WARN