853769620

一、demo程序中的知识说明

1、spring-boot-starter-parent的作用

（1）默认的编译级别是JDK 1.6

（2）指定了UTF-8为source coding

（3）有一个<dependencyManagement>声明，对spring boot可以集成的大部分流行第三方依赖，都指定了对应的可以互相兼容的版本

（4）开启了资源过滤器，支持对应资源文件中占位符的替换

（5）指定了多个插件：exec、surefire、git commit id、shade，用来支持运行、测试、版本控制、打包

2、不直接继承spring-boot-starter-parent

如果要继承自己的parent工程，而不是直接继承spring-boot-starter-parent，那么可以考虑使用import方式，将spring-boot-start-parent中的dependency management引入进来。但是这样的话，就没法用到plugin management了

<dependencyManagement>

    <dependencies>

       <dependency>

           <!-- Import dependency management from Spring Boot -->

           <groupId>org.springframework.boot</groupId>

           <artifactId>spring-boot-dependencies</artifactId>

           <version>1.5.9.RELEASE</version>

           <type>pom</type>

           <scope>import</scope>

       </dependency>

   </dependencies>

</dependencyManagement>

3、自己指定JDK版本

<properties>

   <java.version>1.8</java.version>

</properties>

4、使用spring boot的打包插件

spring-boot-starter-parent指定了pluginManagement，配置好了一系列的插件，其中一个就是用于打成可执行jar包的shade插件。如果我们要使用的话，需要自己手动声明这个插件，然后通过mvn pakcage命令就可以将工程打包成可以执行的jar包了。

<build>

   <plugins>

       <plugin>

           <groupId>org.springframework.boot</groupId>

           <artifactId>spring-boot-maven-plugin</artifactId>

       </plugin>

   </plugins>

</build>

5、spring-boot-starter-\*

spring-boot提供了一些列的starter类依赖，跟流行的常用项目进行整合，比如mybatis、redis、mongodb、elasticsearch，等等。我们只要声明对应的spring-boot-starter-\*，就可以直接使用对应版本的依赖，多个依赖的版本都是兼容的。可能就是通过application.properties，注解，少量的配置，就可以快速整合进来一个技术开始使用

6、基于spring boot进行开发需要遵守的约定规则

一般建议基于spring boot来开发的时候，按照spring boot约定的规则来设置我们的包结构、代码布局，比如下面这样，这样spring boot在按照约定搜索各种注解的时候，一般不会出现什么问题

com

+- zhss

    +- springboot

        +- Application.java

        |

        +- domain

        |   +- User.java

        +- dao

        |   +- UserDAO.java

        |

        +- service

        |   +- UserService.java

        |

        +- web

            +- UserController.java

用于启动的类一般会按照下面这样来写

package com.zhss.springboot-demo;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.EnableAutoConfiguration;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.context.annotation.Configuration;

@Configuration

@EnableAutoConfiguration

@ComponentScan

public class Application {

   public static void main(String[] args) {

       SpringApplication.run(Application.class, args);

  }

}

7、Configuration类

spring boot的一个核心思想，就是尽量消除掉各种复杂的xml配置文件，所以一般建议如果要进行一些配置，可以采用Configuration类，在Java类中做一些配置

而且通常比较好的实践是，就将用来放main方法的Application类，作为Configuration类，所以一般会给Application类加@Configuration注解

如果不想将所有的配置都放在Application类中，也可以使用@Import注解将其他的Configuration类引入进来，或者是依靠@ComponentScan注解自动扫描其他的Configuration类

即使一定要使用一个xml配置文件，建议是用@ImportResource来导入一个xml配置文件

8、Auto Configuration

（1）什么是Auto Configuration

Auto Configuration是spring boot中非常核心的一个功能，因为spring boot的一个核心思想，就是尽可能减少我们要配置的东西，尽量才有约定规则自动完成一些配置。这个Auto Configuration功能，就会根据我们引入的依赖，来推测我们做什么事情，然后自动给我们完成需要的一些配置。举个例子，如果依赖了HSQLDB，那么spring boot会基于HSQLDB自动配置和创建一个内存数据库。

（2）如何启用Auto Configuration

我们给@Configuration类，增加了@EnableAutoConfiguration注解之后，就会开启Auto Configuration功能，而且一般推荐使用使用@EnableAutoConfiguration开启Auto Configuration功能

9、与Spring核心框架进行整合使用

spring boot可以无缝与spring框架进行整合使用，一般就是在Application类上加@ComponentScan注解，启用自动扫描所有的spring bean，同时搭配好@Autowired注解来进行自动装配。只要按照上面的那个约定，将Application类放在最顶层的包结构中，那么使用了@ComponentScan之后，就可以自动扫描和搜索到所有的spring bean，包括了：@Component、@Contorller、@Service、@Repository

10、@SpringBootApplication

大多数的spring boot项目，都会在main类上标注：@Configuration、@EnableAutoConfiguration、@ComponantScan，因为这个实在是太过于常见了，所以spring boot提供了一个@SpringBootApplication注解，这个注解就相当于是上面3个注解的综合。所以实际上在开发的时候，一般就会在main类上加一个@SpringBootApplication注解

11、启动spring boot程序

spring boot的另外一个核心思想，就是尽量简化各种应用的开发。

比如最常见的web应用程序，如果使用传统的方式来开发，需要安装和配置tomcat/jetty服务器，然后还需要将web程序打包后放入tomcat中才可以运行。或者是需要将eclipse等ide工具配置支持tomcat，才能在eclipse中直接启动web程序。

而在spring boot中，直接基于了内置的tomcat/jetty容器，可以在写好代码之后，一键启动web程序，然后就可以在浏览器中访问了。对于本地开发、测试以及debug个调试，都是非常的方便的

（1）在IDE中启动spring boot程序

直接运行main方法即可，默认会绑定在8080端口上

（2）在命令行执行jar包来启动spring boot程序

使用mvn package将应用程序打成一个可以执行的jar包之后，就可以在命令行使用jar -jar命令来启动了，scp命令，自行百度，拷贝到线上的服务器

$ java -jar target/myproject-0.0.1-SNAPSHOT.jar

同样在启动spring boot应用程序时可以传递一些参数

$ java -Xdebug -Xrunjdwp:server=y,transport=dt\_socket,address=8082,suspend=n \

      -jar target/springboot-demo-1.0.0.jar

（3）在命令行直接启动spring boot程序

也可以使用spring-boot插件的run goal来直接在命令行基于代码启动spring boot程序，几乎很少用

$ mvn spring-boot:run

二、devtools工具

12、devtools工具介绍

spring boot为了贯彻简化开发过程的原则，提供了一个devtools工具，基于这个工具，可以在开发的各个环节，尽可能地简化工作

<dependencies>

   <dependency>

       <groupId>org.springframework.boot</groupId>

       <artifactId>spring-boot-devtools</artifactId>

       <optional>true</optional>

   </dependency>

</dependencies>

如果我们启动一个打好包的spring boot应用程序，这个应用程序会被认为是生产应用，那么此时是不会让devtools生效的。而且一般推荐将optional设置为true，这样如果我们的项目被别人依赖，devtools是不会传递过去的

13、让系统在代码变化的时候自动重启（常见于本地开发和调试）

如果我们的应用程序使用了devtools，此时如果在IDE中直接启动，则会认为是开发调试过程。则此时如果classpath中的任何一个文件变化了，就会自动重启应用程序。在IDE中启动程序的时候，这个功能时很有用的，因为我们启动了一个web程序之后，会随时调试和修改代码，代码修改之后，不需要手动重启，web程序自己就重启了

但是上面介绍中也说了，打成一个jar包，java -jar启动一个jar包，对于执行jar包的情况下，spring boot会认为这是生产环境，则不会启用devtools的任何功能，包括这个自动重启功能

devtools使用了两个classloader来加载类，一个是base classloader来加载第三方依赖的类；一个是restart classloader来加载我们自己编写的文件。如果我们自己编写的文件改动了，则此时会重新创建一个restart classloader来重新加载我们的类，这个速度是很快的，因为base classloader是始终保持的，不需要重新加载第三方依赖的类

如果要排除掉某些资源文件的变动不要自动重启，那么可以进行如下的配置，application.properties

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

如果要彻底禁用自动重启功能，需要做如下配置，application.properties

spring.devtools.restart.enabled=false

14、将本地修改的代码同步到远程服务器上部署的应用

devtools也可以用来管理远程机器部署的系统，如果要启用远程系统管理支持，需要进行如下配置，下面的配置可以确保说打包后的程序是包含devtools的。但是要记住的一点是，使用devtools管理远程部署的系统，是比较危险的，尤其在生产环境中，一般强烈建议不要基于devtools来管理

你在eclipse里面修改代码，直接就可以将修改后的代码，热同步到远程服务器上启动的spring boot应用中去。绝对绝对不要用于生产环境。如果你要用于测试环境，假设你要修复bug，速度要快一些，可以用这个功能，直接将本地修复好bug的代码同步到测试服务器上的spring boot应用中去。

Git工作流，集成测试，develop在本地复现代码，commit提交，push到远程gitlab上的分支上去跟别人的代码做集成，接着再次部署这个develop分支的代码到咱们的测试服务器上去。也不支持去用这个功能

<dependencies>

   <dependency>

       <groupId>org.springframework.boot</groupId>

       <artifactId>spring-boot-devtools</artifactId>

   </dependency>

</dependencies>

<build>

   <plugins>

       <plugin>

           <groupId>org.springframework.boot</groupId>

           <artifactId>spring-boot-maven-plugin</artifactId>

           <configuration>

               <excludeDevtools>false</excludeDevtools>

           </configuration>

       </plugin>

   </plugins>

</build>

然后还需要设置一个参数

spring.devtools.remote.secret=123456

做了上面的配置之后，如果我们将这个程序打包后，在我们的某台机器上启动，此时会基于devtools暴露出来一个可以连接的端口，如果我们设置了上面的那个参数之后，这个远程服务器上的服务组件，就会自动启用

然后我们需要在本地手动运行一个客户端组件，一般就是在IDE中启动即可，在IDE中需要使用run configurations，对我们的应用程序，使用org.springframework.boot.devtools.RemoteSpringApplication，作为main class。同时将http://localhost:8080/作为启动参数传递过去

此时连接上以后，一个好处就是，如果在本地写代码，会直接将更新后的代码文件传输到远程机器上然后自动进行重启

这个功能其实怎么说呢，一般是不建议使用的，因为较为鸡肋【没有合入别人代码，集成测试等等步骤】

三、SpringApplication高阶功能介绍

15、SpringApplication介绍

通常我们开发spring boot应用程序，非常重要的一个地方，就是在Application类的main方法中，使用SpringApplication类来启动应用程序

package com.zhss.springboot;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.annotation.Import;

import com.zhss.springboot.config.DruidDBConfig;

@SpringBootApplication

@Import(DruidDBConfig.class)

public class Application {

   public static void main(String[] args) {

       SpringApplication.run(Application.class, args);

  }

}

16、启动失败的处理

如果启动失败了，那么失败的异常会交给spring boot预先注册好的某个FailureAnalyzer来处理，这个FailureAnalyzer就会打印出完整的失败原因以及解决的办法，比如说，如果启动要使用的8080端口被占用了，会显示下面的内容：

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

APPLICATION FAILED TO START

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Description:

Embedded servlet container failed to start. Port 8080 was already in use.

Action:

Identify and stop the process that's listening on port 8080 or configure this application to listen on another port.

spring boot内置了多个FailureAnalyzer，分别用于处理不同的启动失败问题

此外，我们还可以在启动应用程序的时候，打开debug开关，这样，即使没有一个FailureAnalyzer可以处理启动失败，也会打印出完整的auto configuration信息，供我们参考，比如下面这样

$ java -jar myproject-0.0.1-SNAPSHOT.jar --debug

17、传递应用启动参数

如果要在系统中其他组件处要获取传递给应用的启动参数，可以用下面的方式来实现。给组件注入一个ApplicationArguments bean，在里面可以获取到所有的参数

import org.springframework.boot.\*

import org.springframework.beans.factory.annotation.\*

import org.springframework.stereotype.\*

@Component

public class MyBean {

   @Autowired

   public MyBean(ApplicationArguments args) {

       boolean debug = args.containsOption("debug");

       List<String> files = args.getNonOptionArgs();

       // if run with "--debug logfile.txt" debug=true, files=["logfile.txt"]

  }

}

18、ApplicationRunner / CommandLineRunner

如果要在SpringApplication.run()开始运行，但是完成应用启动之前，同时并行运行某些代码，比如一些系统初始化的工作，那么可以用ApplicationRunner / CommandLineRunner，两者唯一的区别，就是ApplicationRunner会传递进来ApplicationArguments，CommandLineRunner会传递进来数组

import org.springframework.boot.\*

import org.springframework.stereotype.\*

@Component

public class MyBean implements CommandLineRunner {

   public void run(String... args) {

       // Do something...

  }

}