1、Spring Boot的基础配置853769620

在application.properties中即可完成spring boot的配置

（1）监听端口配置

默认的监听端口是8080，但是可以用如下三种方式来修改监听的端口

在application.properties中：server.port=9090

启动系统的时候：java -jar target\springboot-demo-1.0.0.jar --server.port=9090

启动系统的时候：java -Dserver.port=9090 -jar target\springboot-demo-1.0.0.jar

1.2 web上下文配置【url访问路径增加前缀】

默认的web上下文是：/，可以通过属性来修改web上下文

server.context-path=/springboot-demo

1.3 使用其他web服务器

默认的web服务器是用的内嵌的tomcat，可以使用jetty或者是undertow

比如使用jetty作为web服务器

<dependency>

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

   <artifactId>spring-boot-starter-jetty</artifactId>

</dependency>

<dependency>

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

   <artifactId>spring-boot-starter-web</artifactId>

   <exclusions>

       <exclusion>

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

           <artifactId>spring-boot-starter-tomcat</artifactId>

       </exclusion>

   </exclusions>

</dependency>

但是一般来说，都是用tomcat作为web容器即可，较为重要的配置参数如下

# 打开tomcat访问日志

server.tomcat.accesslog.enabled=true

# 访问日志所在的目录

server.tomcat.accesslog.directory=logs

# 允许HTTP请求缓存到请求队列的最大个数，默认是不限制的

server.tomcat.accept-count=

# 最大连接数，默认是不限制的，如果连接数达到了上限，那么剩下的连接就会保存到请求缓存队列里，也就是上面参数指定的个数

server.tomcat.max-connections=

# 最大工作线程数

server.tomcat.max-threads=

# HTTP POST内容最大长度，默认不限制

server.tomcat.max-http-post-size=

2、spring组件加载外部配置文件

在实际开发中，将核心配置放在外部配置文件，是很重要的。因为通常我们都是对不同的环境使用一套完全不同的配置，但是保证我们的代码必须是相同的。

比如我们有一个类，其中要从外部配置文件中注入一个name属性

import org.springframework.stereotype.\*

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

// 无论用什么spring组件都可以被注入外部配置

@Component

public class MyBean {

   // 用@Value注解就可以将外部配置注入到类中来

   // 对配置项的名称用${}形式来表达

   @Value("${name}")

   private String name;

   // ...

}

然后通常，我们会在src/main/resources中，放置一个application.properties，其中可以给name属性一个值。也可以在命令行中传递这个参数值来替代：java -jar app.jar --name="Spring"。

而且spring boot还提供了一个随机属性值生成器，供我们使用，可以对某个属性是用的随机生成的一个值：

my.secret=${random.value}

my.number=${random.int}

my.bignumber=${random.long}

my.uuid=${random.uuid}

my.number.less.than.ten=${random.int(10)}

my.number.in.range=${random.int[1024,65536]}

当然，也可以用application.yml来替代application.properties，用yaml格式来编写配置文件，但是这两种方式都可以，一般没必要搞的那么复杂，用application.properties也就可以了

3、spring boot多环境支持

spring boot支持使用@Profile注解来标志，在哪个环境profile下，可以激活使用某个@Configuration类

@Configuration

@Profile("production")

public class ProductionConfiguration {

   // ...

}

接着可以在启动的时候，命令行中，使用如下参数来指定某个环境profile：--spring.profiles.active=dev,hsqldb

但是实际上如果基于spring boot的profile支持来做，一般是没法完全满足我们的期望的，所以多环境profile支持，通常还是基于彻底的maven profile来使用，不同的profile直接对应不同的文件夹，然后mvn pakcage打包的时候，指定对应的profile来打包，将对应环境的配置文件，全部放到src/main/resources下面去

这种方式是最彻底的，而且足够灵活

4、系统读取应用的配置

（1）Environment类

spring boot提供了一个Environment类，可以读取applicaiton.properties、命令行参数、系统参数、操作系统环境变量，等等，各种参数和配置。可以通过spring bean注入的方式获取到这个Environment，很多时候，可能你并不想直接使用@Value将某些参数读取到系统里面来，但是你可以注入一个Environment，从这个里面去获取需要的一些参数

@Configuration

public class EnvironmentConfig {

   @Autowired

   private Environment env;

   public int getServerPort() {

       return env.getProperty("server.port", Integer.class);

  }

}

（2）@Value

也可以通过@Value直接将外部的配置参数注入到配置管理类中

这块可以参考我们之前写的那个DruidDbConfig类，就是基于@Value将外部的druid配置参数注入进来，然后基于配置参数初始化了一个Druid连接池bean

（3）@ConfigurationProperties

如果有一组相关联的配置属性，比如说下面这样的：

server.port=9090

server.context-path=/springboot-demo

都是server打头的，那么可以一次性将一组配置属性读取到一个配置管理类中

@ConfigurationProperties("server")

@Configuration

public class ServerConfig {

   private int port; 【对应server/port参数的读取】

   private String contextPath;

   // port和contextPath的getter和setter方法

}

可以使用这个@ConfigurationProperties来重构dDruidDbConfig类

5、spring boot的自动装配

4.1 @Configuration和@Bean

spring boot的核心思想，就是用@Configuration来标注一个类，从外部配置文件中，基于@Value或者是@ConfigurationProperties来加载各种配置信息进来，然后基于这些配置信息，实例化对应的bean

比如Druid连接池就是一个经典的例子