springboot

springboot的helloworld

springboot手动的helloworld

子主题1

1.创建Maven工程

2.添加依赖(springboot父工程依赖，web启动依赖)

(1)添加父工程坐标
```
<parent>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-parent</artifactId>
<version>2.2.2.RELEASE</version>
</parent>
```
2)添加web启动依赖
```
<dependencies>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-web</artifactId>
</dependency>
</dependencies>
```
3)配置JDK版本
```
<properties>
<java.version>1.8</java.version>
</properties>
```

提示: 为什么我们这里仅仅配置了以上一个名称，依旧的JDK版本始终改变呢？
因为依赖性已经在父工程中定义好了，那从依赖的java.version中变更值

完整的pom.xml文件
```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>

<parent>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-parent</artifactId>
<version>2.2.2.RELEASE</version>
</parent>

<groupId>com.atguigu</groupId>
<artifactId>springboot_01</artifactId>
<version>1.0-SNAPSHOT</version>

<properties>
<java.version>1.8</java.version>
</properties>

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

</project>
```

3.创建启动类
```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Application {
public static void main(String[] args) {
SpringApplication.run(Application.class,args);
}
}
```

4.编写controller
```
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
@RestController
public class HelloController {
@RequestMapping("/hello")
public String sayHello(){
return "hello spring boot!" ;
}
}
```

5.启动测试　　运行启动类的main方法

springboot自动的helloworld

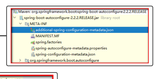
思考

为什么我们就添加动动勤奋的帮不需要在启动器的坐标中指定版本？
答案：因为我们指定了项目的父工程，即spring-boot-starter-parent中已经通过Maven的版本锁定了versiondo版本，所以版本需要再指定了。

为什么我们就添加一个启动器依赖，项目就可以运行起来了，运行项目所需要的jar包从何而来？
答案：因为我们添加了这个启动器的依赖，它没有把自己运行所需要的必要包集成在这个启动器中，通过Maven的依赖传递性，例这些包都依赖到默认的项目里了。

简介

配置文件必须放置在项目的类加载目录下，并且文件名语是application，支持的扩展名有：properties和yaml
现提供了对开：spring-configuration-metadata.json



端口：server.port

additional-spring-configuration-metadata.json
```
server.port
"defaultValue": false
},
{
"name": "server.port",
"defaultValue": 8080
},
```

为什么可以在resources下创建application.properties文件呢？我们查看springboot的启动器：

```
<parent>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-parent</artifactId>
<version>2.2.2.RELEASE</version>
</parent>
```

点击spring-boot-starter-parent

```
<build>
<resources>
<resource>
<filtering>true</filtering>
<directory>${basedir}/src/main/resources</direct
<includes>
<include>**/application*.yml</include>
<include>**/application*.yaml</include>
<include>**/application*.properties</include
</includes>
</resource>
```

配置文件详解

springboot支持二种类型的配置文件properties属性配置文件yaml配置文件

application.properties

将application.properties的属性值读取到java属性中

在 resource 文件夹下面新建 application.properties 配置文件

```
spring.jdbc.datasource.driverClassName=com.mysql.jdbc.driver
spring.jdbc.datasource.url=jdbc:mysql://springboot_01
spring.jdbc.datasource.username=root
spring.jdbc.datasource.password=root
```

新建 properties 包，创建类 DataSourceProperties

```
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;

@Component
public class DataSourceProperties {

@Value("${spring.jdbc.datasource.driverClassName}")
private String driverClassName;
@Value("${spring.jdbc.datasource.url}")
private String url;
@Value("${spring.jdbc.datasource.username}")
private String username;
@Value("${spring.jdbc.datasource.password}")
private String password;

// 省略get set 和toString方法
}
```

在 controller 添加 sayHello() 方法

```
@RestController
public class HelloController {

@Autowired
private DataSourceProperties dataSourceProperties;

@RequestMapping(path = "/hello")
public String sayHello() {
System.out.println(dataSourceProperties);
return "Hello Spring Boot！";
}
}
```

请求地址：http://localhost:8080/hello

控制台打印：
DataSourceProperties{driverClassName='com.mysql.jdbc.Driver', url='jdbc:mysql://springboot_01', username='root', password='root'}

YAML配置文件

语法：
1.数据结构明确和结构紧密相，通过缩进来表示层级。
2.连续的项目连过通每个 " - " 来表示
3.键值对用英语的key:value对用每个 " : " 来分隔。
4.YAML配置文件的扩展名为yml 或 yml
在 resource 文件夹下面新建 application.yml 配置文件，修改 application.properties 配置文件名字为 application.properties.bak

```
spring:
jdbc:
datasource:
driverClassName: com.mysql.jdbc.Driver
url: jdbc:mysql://springboot_01
username: root
password: root
```

运行项目，重新请求 http://localhost:8080/hello
yml配置文件的好处：
1.很好实现结构层次配置信息。
2.配置文件相比属性文件的优势部分行，空间他。
3.配置项目所有属性，那么所要在 之后的一般再写配置语句。

yaml与properties配置文件除了展示形式不相同以外，其它功能和作用都是一样的

多环境profile切换配置

介绍

我们测试时以及与Spring Boot项目中配置文件的名称只能是"**application**"，如果我们把所有的配置全都写在一个配置文件中就有配置项目冲突，配置文件就会显得比较复杂和臃肿！不利于后期的项目维护和开发

解决方案：使用profile来分配置

spring boot项目中允许使用多个YAML配置文件。
这些文件名称必须为application-*** yml，并且在application.yml中激活。

提示：并项目中的开发环境配置和生产环境配置进行拆分，可以根据需求模块化设置

步骤：
1. 编写开发环境配置文件
2. 编写生产环境配置文件
3. 编写主配置文件，指定要使用的配置文件

即建application-dev.yml文件如下：
配置数据库连接信息，开发环境
```
spring:
jdbc:
datasource:
driverClassName: com.mysql.jdbc.Driver
url: jdbc:mysql://springboot
username: root
password: root
```
即建application-pro.yml文件如下：
配置数据库连接信息，上线环境
```
spring:
jdbc:
datasource:
driverClassName: com.mysql.jdbc.Driver
url: jdbc:mysql://business
username: business
password: business
```
在 application.yml 文件中添加如下配置：
```
spring:
profiles:
active: dev
```
启动运行测试：http://localhost:8080/hello
打印结果：
DataSourceProperties{driverClassName='com.mysql.jdbc.Driver', url='jdbc:mysql://springboot', username='root', password='root'}

修改 application.yml 配置文件：
```
# 激活配置文件
spring:
profiles:
active: pro
```
打印结果：
DataSourceProperties{driverClassName='com.mysql.jdbc.Driver', url='jdbc:mysql://business', username='business', password='business'}

如果properties的yml文件都存在，本存在spring.profiles.active设置，如果有重叠属性，默认以properties优先。
如果设置了spring.profiles.active，并且设置多属性，以active设置优先。
可以在其终文件中分别指定server.port属性指定不同的端口，即动项目查看运行端口号进行测试。

@ConfigurationProperties注解

@ConfigurationProperties把springBoot的的需要注解，给可以将一些配置属性"批量"注入到bean对象。

```
application.yml配置文件
spring:
jdbc:
datasource:
driverClassName: com.mysql.jdbc.driver
url: jdbc:mysql://springboot_01
username: root
password: root
```

```
DataSourceProperties.java
public class DataSourceProperties {

private String driverClassName;
private String url;
private String username;
private String password;

// 编写getter和setter.....
```

方式一：使用@Value一个个注入

这种注入方式，如果属性特别多，一个一个注入太麻烦Mvy__v0n
```
@Component
public class DataSourceProperties {

@Value("${spring.jdbc.datasource.driverClassName}")
private String driverClassName;
@Value("${spring.jdbc.datasource.url}")
private String url;
@Value("${spring.jdbc.datasource.username}")
private String username;
@Value("${spring.jdbc.datasource.password}")
private String password;

// 编写getter和setter.....
}
```

方式二：使用@ConfigurationProperties批量注入

这种注入方式，属性再多，只要按照规则就可以一次性自动注入。方便的原理以^-^y~
```
package com.atguigu.properties;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.stereotype.Component;
@Component
@ConfigurationProperties(prefix = "spring.jdbc.datasource")
public class DataSourceProperties2 {
private String driverClassName;
private String url;
private String username;
private String password;
// 编写getter和setter.....
}
```

在类上通过@ConfigurationProperties注解声明该类要读取属性配置
prefix="spring.jdbc.datasource"读取属性文件中前缀为
spring.jdbc.datasource的属，即能匹配到在的配置文件中的key的属值得到一致才可以注入成功
Spring Boot默认读取application.properties属性文件

开启@ConfigurationProperties注解使用
```
@Controller
private DataSourceProperties2.dataSourceProperties2.class)
public class HelloController {

@Autowired
private DataSourceProperties2.dataSourceProperties2;

@RequestMapping(path = "/hello")
@ResponseBody
public String sayHello(){
System.out.println(dataSourceProperties2);
return "hello spring boot";
}
}
```

使用@EnableConfigurationProperties(DataSourceProperties2.class)，开启DataSourceProperties2类上的@ConfigurationProperties注解，他就会生成了，并会帮助我们注入数据了。
请求地址：http://localhost:8080/hello
打印结果：
DataSourceProperties2{driverClassName='com.mysql.jdbc.Driver', url='jdbc:mysql://springboot_01', username='root', password='root'}

如果有报错提示，请在pom文件添加配置信息

```
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot configuration-processor</artifactId>
<optional>true</optional>
</dependency>
```