

SpringBoot在项目构建过程中，会自动将tomcat需要的组件内建到当前的项目中，启动SpringBoot，tomcat便会提供服务。

可以将jar包（多个SpringBoot服务）批量上传至服务器，完成自动化上线部署。

**习惯优于配置**

**极简的组件依赖，自动发现、自动装配**

**提供运行时的应用监控**

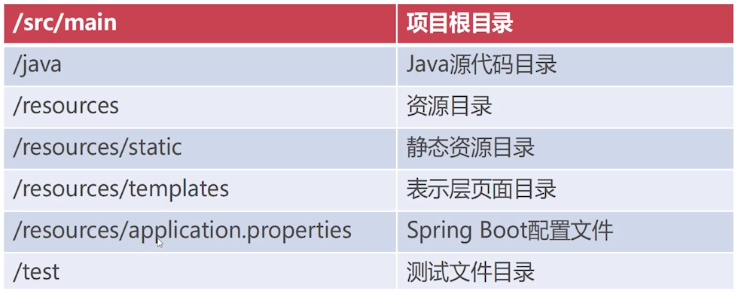
**与分布式架构和云计算的天然集成**

**使用maven进行项目搭建**

在web.xml中配置

<**parent**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-parent</**artifactId**>  
 <**version**>2.0.1.RELEASE</**version**>  
</**parent**>  
  
<**dependencies**>  
 <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-web</**artifactId**>  
 </**dependency**>  
</**dependencies**>  
  
<**build**>  
 <**plugins**>  
 <**plugin**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-maven-plugin</**artifactId**>  
 </**plugin**>  
 </**plugins**>  
</**build**>

目录结构



新建一个类，名称约定为工程名+Application，

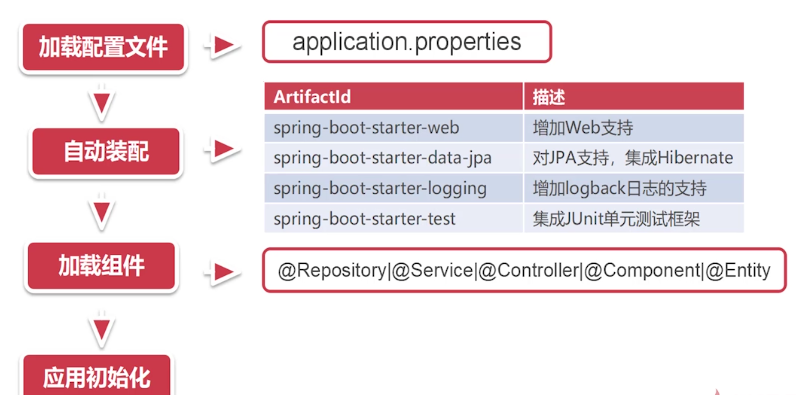
@SpringBootApplication  
**public class** MyspringbootApplication {  
 **public static void** main(String[] args) {  
 SpringApplication.*run*(MyspringbootApplication.**class**);  
 }  
}

SpringApplication.run(..)即启动SpringBoot应用。

配置Controller的注解和SpringMVC相同。

**Spring Initializr可以直接把工程的目录、依赖、启动入口类都创建好**

Spring Boot启动流程



常用配置

server.port=8080 设置端口号

server.servlet.context-path=/myspringboot 访问应用的上下文

logging.file=e:/myspringboot.log 生成日志文件

logging.level.root=error 提高日志输出的最低级别

（五个级别，由低到高：debug-info-warn-error-fatal）

debug=true 设置日志输入级别为debug

spring.datasource.driver-class-name=com.mysql.jdbc.Driver

spring.datasource.url=jdbc:mysql://localhost:3306/test

spring.datasource.username=root

spring.datasource.password=root

YAML是一种简洁的非标记语言。YAML以数据为中心，使用空白，缩进，分行组织数据，从而使得表示更加简洁易读。

key:(空格)

使用空格代表层级关系

spring:

datasource:

driver-class-name:XXX

<url:XXX>

username:XXX

password:XXX

自定义配置项，如网站描述、官网地址、首页热卖数量、是否显示广告

mall:

config:

name: XX商城

description: 这是一家化妆品特卖网站

hot-sales: 20

show-advert: true

程序中读取自定义配置项的方法，在属性上方加注解

@Value(“${mall.config.name}”)

**注意，:后面一定要有空格再填写值**

Spring Boot可针对不同的环境提供不同的Profile文件，Profile文件的默认命名格式为application-{env}.yml，在application.yml中使用spring.profiles.active选项来指定不同的profile。

打包与运行

利用Maven的package命令，生成可独立运行的Jar包。

利用java -jar xxx.jar命令启动Spring Boot应用。

Jar包可自动加载同目录的application配置文件。

**devtools热部署**

页面即时刷新，配置spring.thymeleaf.cache=false

配置devtools生效

spring.devtools.restart.enable=true

spring.devtools.restart.additional-paths=src/main/java

上面这一句配置了监听哪个目录下文件的改变。热部署，即只重启必要的改动，其他的第三方依赖不需要重启

**资源文件属性配置**

添加依赖 **spring-boot-configuration-processor**

在resources中添加文件**resource.properties**，并配置属性，如

**com.example.opensource.name=James**

**com.example.opensource.website=www.baidu.com**

**com.example.opensource.language=java**

在java文件中，配置实体类，添加注解

@Configuration

@ConfigurationProperties(**prefix=”com.example.opensource**”)

@PropertySource(**value=”classpath:resource.properties”**)

**服务端配置**

server.port=8088

server.context-path=/oa

server.session-timeout=60

**tomcat配置**

server.tomcat.max-threads=100

server.tomcat.uri-encoding=UTF-8

**引入freemarker模板引擎**

添加依赖spring-boot-starter-freemarker

在application.properties中配置路径、缓存、编码、扩展名等信息

spring.freemarker.template-loader-path=classpath:/templates

spring.freemarker.cache=false

spring.freemarker.charset=UTF-8

spring.freemarker.check-template-location=true

spring.freemarker.content-type=text/html

spring.freemarker.expose-request-attributes=true

spring.freemarker.expose-session-attributes=true

spring.freemarker.request-context-attribute=request

spring.freemarker.suffix=.ftl

在resources中添加模板文件，以ftl结尾，通过EL表达式注入信息

在Controller中直接映射到响应的文件即可

**引入thymeleaf模板引擎**

步骤类似freemarker，更推荐使用，因为其文件后缀为.html

spring.thymeleaf.prefix=classpath:/templates/

spring.thymeleaf.suffix=.html

spring.thymeleaf.mode=HTML5

spring.thymeleaf.encoding=UTF-8

spring.thymeleaf.content-type=text/html

spring.thymeleaf.cache=false

**SpringBoot配置全局的异常捕获**

新建一个异常捕获类，添加类注解@ControllerAdvice

实现方法errorHandler，[并添加注解@ExceptionHandler(value=Exception.class)](mailto:并添加注解@ExceptionHandler(value=Exception.class))，在方法中可以捕获到异常信息，配置ModelAndView，指定界面和参数。

**如何判断是否是Ajax请求**

(httpRequest.getHeader(**"X-Requested-With"**) != **null** && **"XMLHttpRequest"** .equals( httpRequest.getHeader(**"X-Requested-With"**).toString()) )

**整合mybatis**

**mybatis**:  
 **type-aliases-package**: com.hao.springboot1.pojo  
 **mapper-locations**: classpath:mapper/\*.xml

使用通用Mapper，参考https://github.com/abel533/MyBatis-Spring-Boot