* **参考书籍**
* 请看《深入实践SpringBoot》 陈韶健著。本文主要参考该书。
* **参考网址**

要快速搭建最简单项目，请看 <http://www.cnblogs.com/larryzeal/p/5765945.html>

要看看原理、优缺点 <https://www.cnblogs.com/larryzeal/p/5799195.html>

* **SpringBoot项目特征优点**

1. 内嵌了servlet容器（tomcat），独立运行jar包，以普通的java application方式启动。免部署。
2. 使用更多的注解，符合spring4的推荐。
3. IDE方面用官方提供的开发工具Spring Tool Suite比较方便。
4. 配置所写的位置不同了

尽量用 注解+纯java代码配置 的方式。

1. 配置：有更强的“自动配置”、“预设配置”

“习惯优于配置”的理念：大量的东西已经配置好了，最多我们只需要改少量设置。

1. 对“自动配置”的属性进行修改

以视图路径的指定为例，在全局属性文件中，用下面的语句即可设置视图的前缀、后缀

spring.mvc.view.prefix指定mvc视图的前缀。

spring.mvc.view.suffix指定mvc视图的后缀。

* **第一个SpringBoot项目**
* 在pom文件中编写依赖等

（必须）导入parent：这样，maven会将必须的包全部引入（并会做自动配置），

|  |
| --- |
| <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>1.5.9.RELEASE</version>  <relativePath/>  </parent> |

这个parent包含了大量的默认配置，大大简化了我们的开发。

（一般必须）导入依赖的web支持。因我们一般是web项目，需要web支持。

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

注意，没有写版本号，因为parent就已经为它以及其它依赖的组件都指定了版本号。

（根据需要选择）导入依赖的其它组件。比如redis，如果打算用的话。

这些组件的也是属于spring的管理下（因<groupId>是org.springframework.boot）。示例代码如下

|  |
| --- |
| <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-redis</artifactId>  </dependency> |

（可选）添加spring-boot-maven-plugin插件：这样就可以使用插件来运行项目（没有这个步骤也可以）。

|  |
| --- |
| <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  </build> |

* 编写启动类，具体说明如下

1.命名一般为xxxApplication

2.它必须被@SpringBootAppliation标记，这样这个类才能被按springboot应用的方式运行起来

3.如果它有@Controller（或@RestController）注解，说明它是控制器类。

4.如果它有@Configuration注解，说明它是一个“纯java代码配置类”。

运行main方法，就运行起来了。（也可以用maven插件运行它，但是反而操作要麻烦一点）

|  |
| --- |
| @RestController  @SpringBootApplication  @Configuration //这句也可以不写，原因详见本文  **public** **class** DemoApplication {  @RequestMapping("/")  **public** String home() {  **return** "hello";  }    **public** **static** **void** main(String[] args) {  SpringApplication.*run*(DemoApplication.**class**, args);  }  } |

注：

|  |
| --- |
| @RestController相当于@Controller并且将所有的方法置成了@ResponseBody。这样也就省了很多的工作量。毕竟很多微服务的项目都不是直接返回页面而是返回json、xml这样的数据。当然，如果需要使用页面的情况下，也可以换用@Controller，并在需要的方法上标记@ResponseBody即可。 |

* 在eclipse中启动项目：法一：java application方式。 法二：用插件调spring-boot:run的方式。
* 打包部署到tomcat上，参见http://blog.csdn.net/james\_wade63/article/details/51009423。
* **将以往的项目改变成SpringBoot项目所需的步骤**

主要是使用注解使配置有效。由于springboot的配置是全新的写法，所以这个步骤也是把以往的springWeb项目转化为springboot项目的关键步骤。

|  |
| --- |
| **沿用以往项目的部分**   * 直接可以从以往项目拷贝过来的部分：   POM依赖文件：但需要修改，添加<parent>  Java代码  properties文件  视图jsp页面  可以不需要的部分是：Web.xml  **后面是改写的部分**   * 应用入口类必备注解是@SpringBootApplication * 用@ComponentScan修饰应用入口类 * 用@ComponentScan设置扫描包，用basePackages="。。。" * 用@ComponentScan应将我们的配置类（即标注了@Configuration的类）所在包涵盖进来。   注：1、入口类所在包本就涵盖进来了（因@SpringBootAppliation被@ComponentScan修饰了）  2、默认配置也已生效（因@SpringBootAppliation被@EnableAutoConfiguration修饰了）   * 用@ComponentScan应将我们的controller、service类所在包涵盖进来（此意味着自动扫描注入）。 * 对于DAO类，我们一般是在配置类中，用@bean修饰以便能够被注入给service类用（此意味着手工指定注入而不是扫描）。DAO不使用扫描注入是由于（1）自动扫描会拖慢性能；（2）实现某个功能的DAO可能有多个，而自动扫描有唯一性限制）。   注：当然，如果service的实现类有多个时，也只能用手工指定注入了。   * 用@PropertySource加载properties文件，用value={,,,} * 在application.properties里面需要覆盖的属性。   比如端口设置server.port=xxxx   * 数据源设置   1、配置基本数据源，在某个配置类（可以是入口类，有@Configuration修饰的）中写一个方法dataSource()，它的返回值类型是一个DataSource类型对象（可以根据你自己的选择来选用是否带连接池的），此方法用@bean修饰。具体写法可以参见相关DataSource类的文档  2、配置SqlSessionFactoryBean，可新开一个配置类文件（有@Configuration修饰的）。具体写法可以参见相关SqlSessionFactoryBean类的文档  3、配置MapperScannerConfigurer，可新开一个配置类文件（有@Configuration修饰的）。具体写法可以参见相关MapperScannerConfigurer类的文档 |
| **总结核心规律：**  **1、controller用自动扫描找到，而@Autowired指定的要注入的东西（service、dao）可自动扫描注入或配置类中手工指定注入。**  **2、系统默认的那些配置如果要改的话就在全局application.properties里面改。** |

* **Spring核心注解 @SpringBootApplication的说明等等**

此@SpringBootApplication修饰应用的入口类，如下所示

|  |
| --- |
| @SpringBootApplication  @RestController或@Controller  //也有的人会用上@Configuration  **public** **class** DemoApplication {  **public** **static** **void** main(String[] args) {  SpringApplication.*run*(DemoApplication.**class**, args);  }  ……  } |

此@SpringBootApplication注解的定义如下。可以看出它是组合注解

|  |
| --- |
| @Target(ElementType.***TYPE***)  @Retention(RetentionPolicy.***RUNTIME***)  @Documented  @Inherited  @SpringBootConfiguration  @EnableAutoConfiguration  @ComponentScan(excludeFilters = {  @Filter(type = FilterType.***CUSTOM***, classes = TypeExcludeFilter.**class**),  @Filter(type = FilterType.***CUSTOM***, classes = AutoConfigurationExcludeFilter.**class**) })  **public** **@interface** SpringBootApplication {  } |
| 说明：   * 因包含了@SpringBootConfiguration,而@SpringBootConfiguration又是从@Configuration继承过来的，因此@SpringBootApplication本身就是一个配置。所以入口类可以不需要写@Configuration也仍然是一个配置类，在里面书写@Bean标记是有效果的。   当然,我们一般也可以经常用@SpringBootConfiguration来替代@Configuration。   * 注意到@EnableAutoConfiguration，开启**自动配置**。该注解会使SpringBoot根据项目中依赖的jar包自动配置项目的配置项。parent中继承过来的组件都会被配置上，其它的组件，只要程序员纳入<dependency>依赖，就会被spring配置上。也就是所谓的starter-pom，   比如如下的redis会被配置上：   |  | | --- | | <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-redis</artifactId>  </dependency>  //spring一旦看见了此spring-boot-starter-redis，就会尝试进行相关配置。 |  * 注意到@ComponentScan，那么@SpringBootApplication标记的类的同级目录以及它的子目录就定然会被扫描。 * 如果要排除对某个类进行自动配置，就一般是这样   @SpringBootApplication(exclude={xxxAutoConfiguration.class}) |

* **全局属性文件**

SpringBoot项目使用一个全局的配置文件application.properties或者是application.yml，它放在resources目录下或者类路径下的/config下，一般我们放到resources下。此文件最初是空的，由我们程序员来加入内容。所作的设置可以覆盖springboot默认属性值。

官方有直接的文档告诉我们属性文件内容可以设置哪些值。我们常常只需要在这一个文件中设置就好了。

参见 <https://my.oschina.net/wangnian/blog/666641> ，它里面有一些常用的。

* **示例:改tomcat的启动端口**

比如，在/resources/application.properties中，写上如 server.port=8877 即可。

* **设置日志级别**

比如，在/resources/application.properties中，写上如

logging.level.org.springframework=xxx

其中，xxx是DEBUG、INFO、ERROR等等

* **springboot的starter pom**

Starter pom主要用来简化依赖。比如我们之前做MVC时要引入日志组件，那么需要去找到log4j的版本，然后引入，现在有了Starter之后，直接用这个之后，log4j就自动引入了，也不用关心版本这些问题。

实际上，我们仅需指定spring-boot-starter-parent的版本，其它的组件版本就被自动选择了。

相关的starter可以参见以下文档

<http://www.importnew.com/27101.html> 【有原理说明】

https://www.cnblogs.com/EasonJim/p/7615801.html

<http://blog.csdn.net/chszs/article/details/50610474>

* **springboot的自动配置原理**

我们应用程序的入口类是被框架自带的SpringApplication类启动起来的，通过跟踪SpringApplication类的实例化时调用的initialize()方法，一直跟踪下去，可以跟进到SpringFactoriesLoader类并看到里面放置了下面这句

|  |
| --- |
| public static final String FACTORIES\_RESOURCE\_LOCATION = "META-INF/spring.factories"; //指定了文件路径 |

这句指明了spring初始化容器要用的文件，不过系统里能够发现有两个文件都满足这个路径写法

（1）\spring-boot-1.5.9.RELEASE.jar里的/META-INF/spring.factories文件。它里面都是写的“类全名”，即是说Spring会把这些类创建出一个个的对象。

（2）\spring-boot-autoconfigure-1.5.9.RELEASE.jar里的/META-INF/spring.factories文件，指定的xxxAutoConfiguration类全名，也即是说Spring会把这些类创建出一个个的对象。

以redis组件为例进行说明，

(2)的spring.factories中指定“org.springframework.boot.autoconfigure.data.redis.RedisAutoConfiguration,\”，它指定了RedisAutoConfiguration类，其形式是

|  |
| --- |
| @Configuration  @ConditionalOnClass({ JedisConnection.**class**, RedisOperations.**class**, Jedis.**class** }) //有条件加载  @EnableConfigurationProperties(RedisProperties.**class**) //指定属性文件  **public** **class** RedisAutoConfiguration {  /\*\*  \* Redis connection configuration.  \*/  @Configuration  @ConditionalOnClass(GenericObjectPool.**class**)  **protected** **static** **class** RedisConnectionConfiguration { //内部类  **private** **final** RedisProperties properties;  **private** **final** RedisSentinelConfiguration sentinelConfiguration;  **private** **final** RedisClusterConfiguration clusterConfiguration; |

可以看到，@EnableConfigurationProperties指定了属性类RedisProperties.class，此类指定了各项最基本的设置，其中代码形如

|  |
| --- |
| @ConfigurationProperties(prefix = "spring.redis")  **public** **class** RedisProperties {  **private** **int** database = 0;  **private** String url;  **private** String host = "localhost";  **private** String password;  **private** **int** port = 6379;  **private** **boolean** ssl; |

当欲覆盖这些设置的时候，程序员应在全局属性文件/resources/application.properties中写上相应的设置即可。比如

Spring.redis.port=6666

（注，@ConfigurationProperties标记的prefix指定了前缀，而成员名是后缀。）

问：如果要自行开发组件并自动配置上又怎么办？请参阅相关文档。（一般现成的就很好也足够我们用啦！）

* **SpringBoot的web开发**

我们最关注的就是web开发。其中最关注的是Mvc的自动配置。

这些配置是写在org.springframework.boot.autoconfigure.web.WebMvcAutoConfiguration.class类文件中的。

* **配置MVC视图解析**

视图解析器是InternalResourceViewResolver类，在WebMvcAutoConfiguration.class中有

|  |
| --- |
| @Bean  @ConditionalOnMissingBean  **public** InternalResourceViewResolver defaultViewResolver() {  InternalResourceViewResolver resolver = **new** InternalResourceViewResolver();  resolver.setPrefix(**this**.mvcProperties.getView().getPrefix());  resolver.setSuffix(**this**.mvcProperties.getView().getSuffix());  **return** resolver;  } |

此方法标记了@Bean，说明它会被容器创建出来并管理。此方法里面配合了其它的一些类共同参与工作（可跟踪进去看，细节不详说）。

回顾以前：我们设置视图路径前后缀是用下面的xml语句

|  |
| --- |
| <bean id="viewResolver"  class="org.springframework.web.servlet.view.InternalResourceViewResolver">  <property name="prefix" value="/WEB-INF/jsp/"></property>  <property name="suffix" value=".jsp"></property>  </bean> |

假如我们没有写上述xml，因InternalResourceViewResolver是框架运行必须要用的，那其实框架会使用一个预置的InternalResourceViewResolver类对象。现在我们在xml文件中指定了这个bean，那么框架就会用我们自己的设置。

不过到了springBoot时代，等价于上面的做法是在全局属性文件application.properties里进行设置，如下

|  |
| --- |
| spring.mvc.view.prefix=/WEB-INF/jsp/  spring.mvc.view.suffix=.jsp |

* 配置静态资源

SpringBoot默认是可以访问任意目录的静态资源的，这恰恰与以往的spring框架的默认设置恰恰相反。

如果要进行排除，则是显式的使用一次spring.resources.static-location={……} ，这样其它的任何目录就都无法按静态资源目录方式访问到了。

其它知识点

* **制作Banner**
* 默认banner文件名是banner.txt，放在resource目录下即可。

可以利用这个网站来制作banner。<http://patorjk.com/software/taag>。

* 关闭banner的方式。

需要将启动代码写成如下形式

|  |
| --- |
| **public** **static** **void** main(String[] args) {  SpringApplication app = **new** SpringApplication(DemoApplication.**class**);  app.setBannerMode(Mode.***OFF***);  app.run(args);  } |

* **当不得已而必须使用xml文件作配置时的写法**

用 @importResource({"classpath:xxx.xml","classpath:yyy.xml","classpath:zzz.xml",......})

* **消息转发器**

是指将响应中的对象如何转化为输出用的字符串。比如（1）将字符串对象按照某个编码输出（2）将对象以json格式字符串输出。

* **MVC拦截器**

法一：

参见：http://blog.sina.com.cn/s/blog\_c90ce4e0010328r9.html

第一步：自己的拦截类需要实现接口HandleInterceptor。

第二步：然后扩展类WebMvcConfiAdapter，改写override它的addInterceptros()方法，在其中把自己的拦截类用registry.addInterceptor(new 自己的拦截类())。

1. 该类必须用@configuration注解
2. 该类必须被@ComponentScan所指定的包 涵盖。（如果@ComponentScan未指定，则扫描的包是@ComponentScan所修饰的那个注解（比如@SpringBootApplication）所修饰的类所在的包）。

(注:SpringBoot之前时代的这种拦截器需要在mvc的xml配置文件中用<interceptors>节点中声明)。

法二：aspectj方式

* **条件注解**

比较重要的有@ConditionalOnBean、@ConditionalOnClass、@ConditionalOnExpression等等。