Spring Boot——开发新一代Spring Java应用 | 天码营

Spring官方网站本身使用Spring框架开发，随着功能以及业务逻辑的日益复杂，应用伴随着大量的XML配置文件以及复杂的Bean依赖关系。随着Spring 3.0的发布，Spring IO团队逐渐开始摆脱XML配置文件，并且在开发过程中大量使用“约定优先配置”（convention over configuration）的思想来摆脱Spring框架中各类繁复纷杂的配置（即时是Java Config）。

Spring Boot正是在这样的一个背景下被抽象出来的开发框架，它本身并不提供Spring框架的核心特性以及扩展功能，只是用于快速、敏捷地开发新一代基于Spring框架的应用程序。也就是说，它并不是用来替代Spring的解决方案，而是和Spring框架紧密结合用于提升Spring开发者体验的工具。同时它集成了大量常用的第三方库配置（例如Jackson, JDBC, Mongo, Redis, Mail等等），Spring Boot应用中这些第三方库几乎可以零配置的开箱即用（out-of-the-box），大部分的Spring Boot应用都只需要非常少量的配置代码，开发者能够更加专注于业务逻辑。

Hello World

传统基于Spring的Java Web应用，需要配置`web.xml`, `applicationContext.xml`，将应用打成war包放入应用服务器(Tomcat, Jetty等)中并运行。如果基于Spring Boot，这一切都将变得简单：

以Maven项目为例，首先引入Spring Boot的开发依赖：

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.2.5.RELEASE</version>
</parent>
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
</dependencies>
```

编写一个类包含处理HTTP请求的方法以及一个`main()`函数：

```
@Controller
@EnableAutoConfiguration
public class SampleController {

    @RequestMapping("/")
    @ResponseBody
    String home() {
```

```
        return "Hello World!";
    }

    public static void main(String[] args) throws Exception {
        SpringApplication.run(SampleController.class, args);
    }
}
```

启动main函数后，在控制台中可以发现启动了一个Tomcat容器，一个基于Spring MVC的应用也同时启动起来，这时访问`http://localhost:8080`就可以看到`Hello World!`出现在浏览器中了。

Spring Boot初探

在Maven依赖中引入了`spring-boot-starter-web`，它包含了Spring Boot预定义的一些Web开发的常用依赖:

- `spring-web`, `spring-webmvc` Spring WebMvc框架
- `tomcat-embed-*` 内嵌Tomcat容器
- `jackson` 处理json数据
- `spring-*` Spring框架
- `spring-boot-autoconfigure` Spring Boot提供的自动配置功能

Java代码中没有任何配置，和传统的Spring应用相比，多了两个我们不认识的符号：

- `@EnableAutoConfiguration`
- `SpringApplication`

它们都是由Spring Boot框架提供。在`SpringApplication.run()`方法执行后，Spring Boot的`autoconfigure`发现这是一个Web应用（根据类路径上的依赖确定），于是在内嵌的Tomcat容器中启动了一个Spring的应用上下文，并且监听默认的tcp端口8080（默认约定）。同时在Spring Context中根据默认的约定配置了Spring WebMvc：

- Servlet容器默认的Context路径是`/`
- `DispatherServlet`匹配的路径(`servlet-mapping`中的`url-patterns`)是`/*`
- `@ComponentScan`路径被默认设置为`SampleController`的同名package，也就是该package下的所有`@Controller`，`@Service`, `@Component`, `@Repository`都会被实例化后并加入Spring Context中。

没有一行配置代码、也没有`web.xml`。基于Spring Boot的应用在大多数情况下都不需要我们去显式地声明各类配置，而是将最常用的默认配置作为约定，在不声明的情况下也能适应大多数的开发场景。

实例：数据库访问

除了最基本的Web框架，另一种非常普遍的开发场景是访问数据库。在传统的Spring应用中，访问数据

库我们需要配置：

- 类路径上添加数据库访问驱动
- 实例化`DataSource`对象，指定数据库`url`, `username`, `password`等信息
- 注入`JdbcTemplate`对象，如果使用`Hibernate`，`Mybatis`等框架，还需要进一步配置框架信息

在Spring Boot中，上述过程会被简化。首先在Maven项目依赖中定义：

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web-jdbc</artifactId>
</dependency>
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
</dependency>
```

`spring-boot-starter-web-jdbc`引入了`spring-jdbc`依赖，`h2`是一个内存关系型数据库。在引入了这些依赖并启动Spring Boot应用程序后，`autoconfigure`发现`spring-jdbc`位于类路径中，于是：

- 根据类路径上的JDBC驱动类型（这里是`h2`，预定义了`derby`, `sqlite`, `mysql`, `oracle`, `sqlserver`等等），创建一个`DataSource`连接池对象，本例中的`h2`是内存数据库，无需任何配置，如果是`mysql`, `oracle`等类型的数据库需要开发者配置相关信息。
- 在Spring Context中创建一个`JdbcTemplate`对象（使用`DataSource`初始化）

接下来开发者的工作就非常简单了，在业务逻辑中直接引入`JdbcTemplate`即可：

```
@Service
public class MyService {

    @Autowired
    JdbcTemplate jdbcTemplate;

}
```

除了`spring-jdbc`，Spring Boot还能够支持JPA，以及各种NoSQL数据库——包括MongoDB，Redis，全文索引工具`elasticsearch`, `solr`等等。

配置

Spring Boot最大的特色是“约定优先配置”，大量的默认配置对开发者十分的友好。但是在实际的应用开发过程中，默认配置不可能满足所有场景，同时用户也需要配置一些必须的配置项——例如数据库连接信息。Spring Boot的配置系统能够让开发者快速的覆盖默认约定，同时支持Properties配置文件和YAML

配置文件两种格式，默认情况下Spring Boot加载类路径上的`application.properties`或`application.yml`文件，例如：

```
spring.datasource.url=jdbc:mysql://localhost/test
spring.datasource.username=dbuser
spring.datasource.password=dbpass
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
```

YAML格式更加简洁：

```
spring:
  datasource:
    url: jdbc:mysql://localhost/test
    username: dbuser
    password: dbpass
    driver-class: com.mysql.jdbc.Driver
```

一旦发现这些信息，Spring Boot就会根据它们创建`DataSource`对象。另一个常见的配置场景是Web应用服务器：

```
# Server settings (ServerProperties)
server:
  port: 8080
  address: 127.0.0.1
  sessionTimeout: 30
  contextPath: /

  # Tomcat specifics
  tomcat:
    accessLogEnabled: false
    protocolHeader: x-forwarded-proto
    remoteIpHeader: x-forwarded-for
    basedir:
    backgroundProcessorDelay: 30 # secs
```

通过`port`和`address`可以修改服务器监听的地址和端口，`sessionTimeout`配置session过期时间（再也不用修改`web.xml`了，因为它根本不存在）。同时如果在生产环境中使用内嵌Tomcat，当然希望能够配置它的日志、线程池等信息，这些现在都可以通过Spring Boot的属性文件配置，而不再需要再对生产环境中的Tomcat实例进行单独的配置管理了。

@EnableAutoCongiguration

从Spring 3.0开始，为了替代繁琐的XML配置，引入了`@Enable...`注解对`@Configuration`类进行修

饰以达到和XML配置相同的效果。想必不少开发者已经使用过类似注解：

- `@EnableTransactionManagement`开启Spring事务管理，相当于XMl中的`<tx:*>`
- `@EnableWebMvc`使用Spring MVC框架的一些默认配置
- `@EnableScheduling`会初始化一个Scheduler用于执行定时任务和异步任务

Spring Boot提供的`@EnableAutoCongiguration`似乎功能更加强大，一旦加上，上述所有的配置似乎都被包含进来而无需开发者显式声明。它究竟是如何做到的呢，先看看它的定义：

```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Import({ EnableAutoConfigurationImportSelector.class,
        AutoConfigurationPackages.Registrar.class })
public @interface EnableAutoConfiguration {

    /**
     * Exclude specific auto-configuration classes such that they will
never be applied.
     */
    Class<?>[] exclude() default {};

}
```

`EnableAutoConfigurationImportSelector`使用的是`spring-core`模块中的`SpringFactoriesLoader#loadFactoryNames()`方法，它的作用是在类路径上扫描`META-INF/spring.factories`文件中定义的类：

```
# Initializers
org.springframework.context.ApplicationContextInitializer=\
org.springframework.boot.autoconfigure.logging.AutoConfigurationReportLogg
ingInitializer

# Auto Configure
org.springframework.boot.autoconfigure.EnableAutoConfiguration=\
org.springframework.boot.autoconfigure.aop.AopAutoConfiguration,\
org.springframework.boot.autoconfigure.amqp.RabbitAutoConfiguration,\
org.springframework.boot.autoconfigure.MessageSourceAutoConfiguration,\
org.springframework.boot.autoconfigure.PropertyPlaceholderAutoConfiguratio
n,\
org.springframework.boot.autoconfigure.batch.BatchAutoConfiguration,\
org.springframework.boot.autoconfigure.data.JpaRepositoriesAutoConfigurati
on,\
org.springframework.boot.autoconfigure.data.MongoRepositoriesAutoConfigura
```

```
tion,\
org.springframework.boot.autoconfigure.redis.RedisAutoConfiguration,\
org.springframework.boot.autoconfigure.jdbc.DataSourceAutoConfiguration,\
org.springframework.boot.autoconfigure.jdbc.DataSourceTransactionManagerAu
toConfiguration,\
org.springframework.boot.autoconfigure.jms.JmsTemplateAutoConfiguration,\
org.springframework.boot.autoconfigure.jmx.JmxAutoConfiguration,\
org.springframework.boot.autoconfigure.mobile.DeviceResolverAutoConfigurat
ion,\
org.springframework.boot.autoconfigure.mongo.MongoAutoConfiguration,\
org.springframework.boot.autoconfigure.mongo.MongoTemplateAutoConfiguratio
n,\
org.springframework.boot.autoconfigure.orm.jpa.HibernateJpaAutoConfigurati
on,\
org.springframework.boot.autoconfigure.reactor.ReactorAutoConfiguration,\
org.springframework.boot.autoconfigure.security.SecurityAutoConfiguration,
\
org.springframework.boot.autoconfigure.security.FallbackWebSecurityAutoCon
figuration,\
org.springframework.boot.autoconfigure.thymeleaf.ThymeleafAutoConfiguratio
n,\
org.springframework.boot.autoconfigure.web.EmbeddedServletContainerAutoCon
figuration,\
org.springframework.boot.autoconfigure.web.DispatcherServletAutoConfigurat
ion,\
org.springframework.boot.autoconfigure.web.ServerPropertiesAutoConfigurati
on,\
org.springframework.boot.autoconfigure.web.MultipartAutoConfiguration,\
org.springframework.boot.autoconfigure.web.HttpMessageConvertersAutoConfig
uration,\
org.springframework.boot.autoconfigure.web.WebMvcAutoConfiguration,\
org.springframework.boot.autoconfigure.websocket.WebSocketAutoConfiguratio
n
```

实际上这就是Spring Boot会自动配置的一些对象，例如前面提到的Web框架由`EmbeddedServletContainerAutoConfiguration`, `DispatcherServletAutoConfiguration`, `ServerPropertiesAutoConfiguration`等配置完成，而`DataSource`的自动配置则是由`DataSourceAutoConfiguration`完成。现在我们以Mongo的配置`MongoAutoConfiguration`为例，来探索Spring Boot是如何完成这些配置的：

```
@Configuration
@ConditionalOnClass(Mongo.class)
@EnableConfigurationProperties(MongoProperties.class)
public class MongoAutoConfiguration {
```

```
    @Autowired
    private MongoProperties properties;

    private Mongo mongo;

    @PreDestroy
    public void close() throws UnknownHostException {
        if (this.mongo != null) {
            this.mongo.close();
        }
    }

    @Bean
    @ConditionalOnMissingBean
    public Mongo mongo() throws UnknownHostException {
        this.mongo = this.properties.createMongoClient();
        return this.mongo;
    }

}
```

首先这是一个Spring的配置`@Configuration`，它定义了我们访问Mongo需要的`@Bean`，如果这个`@Configuration`被Spring Context扫描到，那么Context中自然也就有两个一个`Mongo`对象能够直接为开发者所用。

但是注意到其它几个Spring注解：

- `@ConditionOnClass`表明该`@Configuration`仅仅在一定条件下才会被加载，这里的条件是`Mongo.class`位于类路径上
- `@EnableConfigurationProperties`将Spring Boot的配置文件（`application.properties`）中的`spring.data.mongodb.*`属性映射为`MongoProperties`并注入到`MongoAutoConfiguration`中。
- `@ConditionalOnMissingBean`说明Spring Boot仅仅在当前上下文中不存在`Mongo`对象时，才会实例化一个Bean。这个逻辑也体现了Spring Boot的另外一个特性——自定义的Bean优先于框架的默认配置，我们如果显式的在业务代码中定义了一个`Mongo`对象，那么Spring Boot就不再创建。

接下来看一看`MongoProperties`：

```
@ConfigurationProperties(prefix = "spring.data.mongodb")
public class MongoProperties {

    private String host;
    private int port = DBPort.PORT;
```

```
    private String uri = "mongodb://localhost/test";
    private String database;

    // ... getters/ setters omitted
}
```

显然，它就是以`spring.data.mongodb`作为前缀的属性，然后通过名字直接映射为对象的属性，同时还包含了一些默认值。如果不配置，那么`mongo.uri`就是`mongodb://localhost/test`。

Production特性

从前面的例子可以看出，Spring Boot能够非常快速的做出一些原型应用，但是它同样可以被用于生产环境。为了添加生产环境特性支持，需要在Maven依赖中引入：

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

加入`actuator`依赖后，应用启动后会创建一些基于Web的Endpoint：

- `/autoconfig`，用来查看Spring Boot的框架自动配置信息，哪些被自动配置，哪些没有，原因是什么。
- `/beans`，显示应用上下文的Bean列表
- `/dump`，显示线程dump信息
- `/health`，应用健康状况检查
- `/metrics`
- `/shutdown`, 默认没有打开
- `/trace`

总结

Spring Boot是新一代Spring应用的开发框架，它能够快速的进行应用开发，让人忘记传统的繁琐配置，更加专注于业务逻辑。现在Spring官方文档中所有的Guide中的例子都是使用Spring Boot进行构建，这也是一个学习Spring, Spring Boot非常好的地方。如果想进一步深度学习Spring Boot，可以参考：