Spring Boot 研究和应用

王永和，张劲松，邓安明，周智勋
（云南云电同方科技有限公司，云南 昆明 650218）

摘要：Spring Boot 是一个用于简化、加速 Spring 开发的框架，文章对 Spring Boot 的开发做了简要介绍，之后对整合传统 Spring 的开发做了探索性研究。

关键词：Spring Boot；开发；整合传统 Spring
中图分类号：TN943.6 文献标识码：A 文章编号：1673-1131（2016）10-0091-04

1 Spring Boot 简介

Spring Boot 是在 2013 年推出的新项目，主要用来简化 Spring 开发框架的开发、配置、调试、部署工作，同时在项目内集成了大量易于使用且实用的基础框架。

在 Spring Boot 中集成的基础框架，是在开发中经常需要使用的框架，如内嵌容器（Tomcat、Jettty、Undertow）、日志框架、JMS 框架、持久化框架、流行的 NOSQL 数据库（Cassandra、MongoDB）、缓存框架等。

在传统的 Spring 框架中开发，用户需要自行编写 XML 文件，并在 Maven、Gradle 中加入相应的依赖包，在响应的代码中添加功能代码，才能使用。

同时对于引入多个依赖包时，包之间的版本调整也是个困难的问题，经常需要对依赖包的版本冲突进行处理。调整依赖包的版本以及冲突问题是一个费时费力、重复、繁琐的工作。

如果使用 Spring Boot 开发，仅需在 Maven、Gradle 配置文件中加上少量的配置，即可在代码中使用所需的框架，让原本的配置简化到几乎是零代码、零 XML 配置，同时依赖包的版本问题也由 Spring Boot 轻松解决。

Spring Boot 对目前的的主流构建工具 Maven、Gradle 都提供了良好的支持，对其他构建工具也提供了支持，如 Ant，但是相对 Maven、Gradle 来说，支持的力度相对要少些，所以需要尽量使用 Maven、Gradle 来构建 Spring Boot 程序。

本文的构建环境是 Maven，使用 Gradle 可以参考 Spring Boot 的官方 Reference 文档。

2 简单示例 Hello World 开发对比

2.1 使用传统 Spring 开发 Hello World 程序

2.1.6 重连编组位置识别

由于所有节点的应用程序相同，为了让显示器能正确读取到编组的数据并进行编组区分，通过综合判断各个编组的编组编号、编组索引之间的关联关系，分别定义编组 1、编组 2、编组 3，并将 3 个编组的数据分别发送至本编组固定的 MVB 端口，由显示器读取并显示到正确的位置。各个编组的 MVB 主节点根据确定的重连拓扑与本节点的编组编号，节点索引以及本编组的拓扑进行分析，将编组号映射到显示器上显示的车辆编号。

2.2 重连列车行驶方向

2.2.1 全车行驶方向

全车的行驶方向应由主控车司机室的方向命令唯一确定。司机通过输入向前或是向后命令，控制全车向主控车的头车或者尾车方向行驶。该方向命令传输到 WTB 过程数据的全车控制信息，通过 WTB 网络控制编组的运动方向。

2.2.2 非主控编组牵引方向

列车的牵引逆变器通过正相序或反相序输出交流电来控制牵引电机和车辆轮对的正转或者反转，从而实现司机所期望的列车“向前”或者“向后”运行。所有编组在收到主控车的方向命令后，根据自编组与主控编组拓扑关系判断本编组的牵引方向，并逐个向每一个牵引逆变器单独发送“正向”或“反向”指令。

2.3 WTB 协议与整车控制

整车控制信息和关键状态信息数据量小，实时性要求高；用于显示和记录的状态和故障信息数据量大，实时性要求较低。WTB 端口大小为 128 字节，为了最大限度利用总线带宽，同时降低总线负荷，对实时性要求不高的状态数据进行分页分周期发送，通过写入页码的方法识别每个周期发送的设备状态，将节点的 128 字节平均分为控制信息、关键状态、设备状态和故障信息 4 个部分，控制信息和关键状态部分的所有信息在每个周期内都实时发送，各编组的设备状态与故障信息分页分周期发送。在协议的第 95 和第 127 字节分别加入页码用于识别当前周期发送的状态和故障页。

3 实现效果

实际测试结果表明：实时性要求较高的控制数据与关键状态的发送周期为 16ms，实时性要求相对较低的全车设备状态单页发送周期为 80ms，总发送周期为 1040ms，重连后控制信息能够准确达到所有编组，所有编组的拓扑结构与设备状态都能够实时准确地显示。

4 结语

本文在原有的基础上重新设计动车组重连控制逻辑，有效解决了可变编组与编组方式的多编组自动重连逻辑，保证了重连拓扑与多编组设备状态显示的实时性和准确性。

参考文献：

[1] 李国平. 列车通信网络 WTB_MVB 与 LonWorks 的技术比较与应用[J].铁道车辆,2004(1):27-30.
[2] 姜娜. WTB 底层协议的研究与实现 [D]. 北京交通大学, 2007.
[3] 蔡国强,贾利民,刘春煌,李熙.绞线式列车总线初运行算法分析[J].铁路计算机应用,2007(5): 8-10.

作者简介：张卫杰（1989-），男，湖南人，工程师，主要从事动车网络研发工作。

使用传统 Spring 开发一个 Hello World 的 Web 程序，构建开发工程是一个比较繁琐的过程，典型的工程，需要修改、配置一下 4 个文件：Hello.java、Application.xml、servlet-hello.xml、web.xml，即使在 IDE 运行、调试，还是需要配置好 J2EE 容器才能进行调试工作。

如果要进行运维、部署工作，还需要用 IDE 将应用打包为 war 文件，之后还要进行 J2EE 容器的配置、部署，再将 war 文件部署到容器中运行，才能看到运行效果。

2.2 使用 Spring Boot 开发 Hello World 程序

使用 Spring Boot 开发 Hello World 程序，所需的工作量将大大减少。仅仅需要修改、配置 2 个文件：Hello.java、pom.xml，借助 Spring Boot 框架内的内嵌容器，在 IDE 即可轻松启动 Web 程序。

pom.xml 核心内容如下：

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.3.5.RELEASE</version>
</parent>
<dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
</dependencies>
```

Hello.java 代码：

```
package hello;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure. SpringBootApplication;
import org.springframework.web.bind.annotation. RequestMapping;
import org.springframework.web.bind.annotation. ResponseBody;
    @SpringBootApplication
    public class Hello {
    @RequestMapping("/test/v1")
    @ResponseBody
    String home() {
        return "Hello World";
}
public static void main(String[] args) throws Exception {
        SpringApplication.run(Hello.class, args);
    }
}
```

从 pom.xml 和 Hello.java 的代码可见，使用 Spring Boot，代码、配置均得到了极大的简化，仅仅十多行代码加上 Maven 配置，即可完成一个 Web 程序。

同时这个 web 程序可以轻松打包为单一的可运行 jar 文件 hello.jar，在命令行使用如下命令：

```
java -jar hello.jar
```

就能不部署 J2EE 容器的情况下，启动一个 Web 应用。

这种把 Web 应用打包为单一可运行 jar 包同时内嵌 J2EE 容器的方式，主要目的是简化应用的部署、配置工作。

在实际运维中这种打包方式可以很好地配合 Docker 之类的虚拟化平台，达到简单、快速部署的目标。

3 Spring Boot 的主要功能特性

作为一个简化 Spring 开发、调试、部署的框架，Spring Boot 提供了许多好用的特性，这里仅介绍开发中主要特性。其他的诸多实用特性，比如：日志集成、安全技术、NoSQL 技术、消息（JMS）、邮件发送、JTA 分布式事务集成、JMX 的监控和管理技术集成、测试功能集成、使用条件的自动配置技术、WebSockets 开发等不在本文中介绍，读者在本文的基础上，可以自行参考第六章中给出的资源进行扩展学习。

Spring Boot 的提供微服务框架开发功能，除了将 Web 程序打包为 war 文件的部署方式。还可以将 Web 程序编译为可独立运行的 jar 包，同时还可以内嵌 J2EE 容器，使得 Web 程序也可以独立于容器外运行。

3.1 SpringApplication

SpringApplication 类提供了一种从 main()方法启动 Spring 应用的便捷方式。如下面的 SpringApplication 类代码所示，Hello 类在的 main()方法中，通过调用 SpringgApplication.run 这个静态方法启动应用本身。

SpringApplication 类代码：

```
package hello;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure. SpringBootApplication;
import org.springframework.web.bind.annotation. RequestMapping;
import org.springframework.web.bind.annotation. ResponseBody;
@SpringBootApplication
public class Hello {
    @RequestMapping("/test/v1")
    @ResponseBody
    String home() {
        return "Hello World";
}
public static void main(String[] args) throws Exception {
        SpringApplication.run(Hello.class, args);
  }
}
```

上述代码说明了，SpringApplication 可以简单地、很好地启动一个 Spring 应用。对于 Web 应用，Spring Boot 可以通过打包工具将 Tomcat、Jetty 等容器嵌入到可执行的 jar 包内，使 jar 包成为独立运行的 Web 应用。

3.2 外化配置

Spring 开发框架中允许将程序的运行参数放置到 properties 文件内，在 Spring Boot 中，这些可配置的参数的来源扩大了，不在局限于 properties 文件，可以来自其他外部环境（比如启动命令行、系统环境变量），这个功能称为外化配置（externalize）。

除了使用 properties 配置文件，还可以使用 YAML 配置文件，配置文件中参数可以系统环境变量、命令行参数等外部环境。

Spring Boot 的配置参数可以来源以下环境：

(1) 命令行参数；
(2) 来自于 java:comp/env 的 JNDI 属性；
(3) Java 系统属性(System.getProperties())；
(4) 操作系统环境变量；
(5) 只有在random.*里包含的属性会产生一个 RandomValuePropertySource；
(6) 在打包的 jar 外的应用程序配置文件(application.properties,包含 YAML 和 profile 变量)；
(7) 在打包的 jar 内的应用程序配置文件(application.properties,包含 YAML 和 profile 变量)；
(8) 在@Configuration 类上的@PropertySource 注解；
(9) 默认属性(使用 SpringApplication.setDefaultProperties 指定)。

下面是一个具体的示例,从外部环境中获取 db.port 的属性(数据库端口)：

代码示例 获取外部资源参数

```
public class Application {
    @Value("${dp.port}")
    private String extResource;
    public static void main(String[] args) throws Exception {
        SpringApplication.run(Application.class, args);
    }
}
```

Spring Boot 的配置文件有 2 种格式,分别是 properties 和 YAML。

3.2.1 Application 属性文件

SpringApplication 会从以下位置加载 application.properties 文件,并把它们添加到 Spring Environment 中：

(1) 当前目录下的一个/config 子目录；
(2) 当前目录；
(3) 一个 classpath 下的/config 包；
(4) classpath 根路径(root)。

这个列表是按优先级排序的 (列表中位置高的将覆盖位置低的)。

注：你可以使用 YAML('.yml')文件替代'.properties'。

如果不喜欢将 application.properties 作为配置文件名，你可以通过指定 spring.config.name 环境属性来切换其他的名称。你也可以使用 spring.config.location 环境属性来引用一个明确的路径(目录位置或文件路径列表以逗号分割)。

3.2.2 YAML

YAML 是 JSON 的一个超集,也是一种方便的定义层次配置数据的格式。无论你何时将 SnakeYAML 库放到 classpath 下,SpringApplication 类都会自动支持 YAML 作为 properties 的替换。

从下面的样例代码可知 YAML 易于人工书写、观察,缺点是 YAML 文件不能通过@PropertySource 注解加载。

YAML 样例(代码来自 http://docs.spring.io/)：

```
nvironments:
    dev:
      url: http://dev.bar.com
      name: Developer Setup
    prod:
      url: http://foo.bar.com
      name: My Cool App
```

对应的 Properties(代码来自 http://docs.spring.io/)

```
environments.dev.url=http://dev.bar.com
environments.dev.name=Developer Setup
environments.prod.url=http://foo.bar.com
environments.prod.name=My Cool App
```

3.2.3 Profiles

在 Spring Boot 可以通过@Profiles 来限制某些配置类的加载。如图 10 所示,代码中的配置,仅在环境的 Profile 为 production 才被加载,如果 Profile 为其他值,此配置类则不会被加载。

通过环境变量控制@Profiles，从而可以灵活控制程序的加载。

Profiles 示例：

```
@Configuration
@Profile("production")
public class ProductionConfiguration {
    // ...
}
```

如下面的代码,在 Application.properties 中的激活 Profile 为：dev, hsqldb。

```
spring.profiles.active=dev,hsqldb
```

3.2.4 开发 Web 应用

在传统的 spring MVC、RESTful 开发，需要手工处理 XML,进行许多配置,在 Spring Boot，仅需在 maven 配置添加一项：spring-boot-starter-web,如图 1 所示。

```
<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/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>test</groupId>
    <artifactId>appTest</artifactId>
    <version>0.0.1</version>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>1.3.5.RELEASE</version>
    </parent>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
    </dependencies>
```

图 1 Spring Boot Web 程序的 Maven 配置文件

3.2.5 使用 JPA 操作数据库

在 Spring Boot 使用 JSP 操作数据库,操作十分简单,仅需要在 Maven 配置加入 pom 项目：spring-boot-starter-data-jpa,如图 2 所示。

然后定义需要的实体类：Customer,代码如下：

```
@Entity
public class Customer implements Serializable {
@Id
  @GeneratedValue
  private Long id;
  @Column(nullable = false)
  private String name;
  @Column(nullable = false)
  private Integer age;
 // ... additional members, often include @OneToMany mappings
  protected Customer() {
    // no-args constructor required by JPA spec
```

```
        // this one is protected since it shouldn't be used directly
    }
    public Customer(String name, int age) {
            this.name = name;
            this.age = age;
    }
    public String getName() {
          return this.name;
      }
      // ... etc
    }
```

根据业务需要的创建数据接口：CustomerRepository，**代码如下：**

代码示例，CustomerRepository：

```
public interface  CustomerRepository extends Repository<
Customer, Long> {
Page<Customer> findAll(Pageable pageable);
Customer findByNameAndCountryAllIgnoringCase(String
name, String country);
}
```

如图 3 **所示，在配置类** Application **中对数据接口** CustomerRepository **使用**@Bean **注解配置后，**Spring Boot **会自动创建接口** CustomerRepository **的实现，在零编码的情况下，就能使用** CustomerRepository **接口。**

```
<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/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>test</groupId>
    <artifactId>appTest</artifactId>

    <version>0.0.1</version>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>1.3.5.RELEASE</version>
    </parent>

    <dependencies>

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

    </dependencies>
```

图 2 **增加** pom **项：**spring-boot-starter-data-jpa

```
@SpringBootApplication
public class Application {

        private static final Logger log = LoggerFactory.getLogger(Application.class

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

        @Bean
        public CommandLineRunner demo(CustomerRepository repository) {
                return (args) -> {
                        // save a couple of customers
                        repository.save(new Customer("Jack", "Bauer"));
                        repository.save(new Customer("Chloe", "O'Brian"));
                        repository.save(new Customer("Kim", "Bauer"));
                        repository.save(new Customer("David", "Palmer"));
                        repository.save(new Customer("Michelle", "Dessler"));

                        // fetch all customers
                        log.info("Customers found with findAll():");
                        log.info("-------------------------------");
                        for (Customer customer : repository.findAll()) {
                                log.info(customer.toString());
                        }
                log.info("");

                        // fetch an individual customer by ID
                        Customer customer = repository.findOne(1L);
                        log.info("Customer found with findOne(1L):");
                        log.info("--------------------------------");
                        log.info(customer.toString());
                log.info("");

                        // fetch customers by last name
                        log.info("Customer found with findByLastName('Bauer'):");
                        log.info("--------------------------------------------");
                        for (Customer bauer : repository.findByLastName("Bauer")) {
                                log.info(bauer.toString());
                        }
                log.info("");
```

图 3 **图** 3 **实际配置类中零编码的情况下即可使用接口** CustomerRepository

（图 3 **来自** https://spring.io/guides/gs/accessing-data-jpa/**）**

3.3 整合传统的基于 XML 配置的 Spring 旧代码

如果要把早期的基于 XML **配置的旧代码整合到** Spring Boot **新代码中，在条件许可的情况下，尽量对旧代码进行改造，这样可以避免在实际运行中出现问题。**

如果旧代码改造成本较高，也可以考虑使用@ImportResource **注解引入传统的** XML **配置，**XML **配置中的** bean **也能正常使用。**

在@ImportResource **注解接入传统的** Spring XML **配置文件，可以解决大部分** bean **的的创建文件，实际使用时仍然存在少量的代码会出现问题，这时需要进行大量、细致的测试，才能找出有问题的代码并修正问题，这样也能解决问题。**

4 Spring Boot 的那些坑

4.1 JSP 调试问题

使用内嵌容器时，仅在使用 Tomcat **可以支持** JSP **（需要增加配置），其他容器不支持** JSP**。**

打包为 war **文件部署到标准容器时，不存在调试问题，所以仅会影响** JSP **调试。**

目前使用 JSP **开发的的情况越来越少，所以这个问题对开发的影响较小。**

4.2 注解自动扫描问题

Spring Boot **仅仅扫描本身工程的源代码内的** spring **配置类，不会扫描引用的** jar **包内的进行** spring **配置类对象。对于引用的** jar **包的** spring **配置对象需要用**@import **注解进行。**

4.3 整合传统 Spring 的 XML 配置冲突问题

@ImportResource **注解导入** XML **配置，如果在** XML **使用了**<context:component-scan>**的自动扫描功能，会引发** Spring **启动冲突。**

解决方法：

在<context: component-scan>**加入**<exclude-filter>**字段排除冲突的配置类（**@Configuration**），如：**

<context: exclude-filter type="assignable" expression="**类的全路径**"/>

参考文献：

[1] Pivotal **团队**, Spring Boot Reference[ol].2016.

[2] Pivotal **团队及其他开源贡献者**. Spring Boot **源码、示例** [ol]. 2016.

[3] **作者：**Pivotal **团队，译者：**qibaoguang@gmail.com. Spring Boot **参考指南（翻译）**[ol]. 2016.

[4] **作者：**Dan Woods **，译者：张卫滨. 深入学习微框架：**Spring Boot[ol]. 2014.

[5] **成富. 使用** Spring Boot **快速构建** Spring **框架应用**[ol]. 2014.