SpringBoot实战

> ### 本节目标
>
> 1.了解Spring Boot集成功能 2.了解Spring Boot应用的监控 3.掌握Spring Boot中使用数据库，REST，缓存 4.掌握Spring Boot应用的打包，部署，运维

1. 常用功能实践

1.1. SQL数据库

Spring Boot中使用数据库相关的功能引入`spring-boot-starter-jdbc`即可。这样Spring Boot在启动的时候会自动配置数据源，前提是需要在application.properties中配置数据库的地址，用户名，密码等信息。

示例：

- 在pom.xml中添加spring-boot-starter-jdbc依赖

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

- 使用MySQL数据库则需要额外添加MySQL的JDBC驱动

```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
</dependency>
```

- 在application.properties中添加数据库的配置

```
spring.datasource.data-username=root
spring.datasource.data-password=root
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/test
```

在我们未指定数据库连接池的时候，Spring Boot 2.x版本开始默认使用的是HikariCP (光，日本开发者贡献，号称世界上最快的数据库连接池)

- 启用了数据库的支持之后，Spring Boot完成自动配置，`JdbcTemplate` Bean就已经在IoC容器中存在了，接下来就可以直接通过@Autowired使用

1.2. NoSQL数据库

Spring框架关于数据处理方面主要集中在Spring Data这个项目。 Spring Data项目分为主要模块和社区模块。

主要模块有（主要有Spring IO 组织贡献）：

- Spring Data JDBC : 基于JDBC的数据库操作
- Spring Data Redis : 提供简洁的Redis配置和操作
- Spring Data MongoDB : 基于MongoDB进行文档对象操作
- 等等

社区模块（主要有开源社区开发贡献）：

- Spring Data Neo4j : 基于Neo4j的图数据库操作
- Spring Data Elasticsearch :基于Elasticsearch服务的搜索引擎操作
- 等等

Spring Data项目子模块很多，这里我们挑出Spring Data Redis子模块进行讲解。Spring Boot开启Redis操作，非常简单，基本三步就可以完成配置和操作。

> 备注：**准备Redis服务器并且安装**，Windows平台下的安装包window redis

示例：

- 在pom.xml中添加spring-boot-starter-data-redis依赖

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

- 在application.properties中添加redis的配置（可选，默认信息如下）

```
spring.redis.database=0
spring.redis.host=localhost
spring.redis.port=6379
```

当我们添加了spring-boot-starter-data-redis依赖后，如果不配置任何信息，这时候Spring Boot自动配置时就会去使用本地的redis（默认端口号，无认证要求，第0个数据库），如果本地没有安装redis则启动失败。

- Spring Boot正常启动之后就会在Spring IoC容器中创建一个`RedisTemplate` Bean RedisTemplate即就是我们操作Redis数据库的模版客户端。

示例：通过RedisTemplate访问Redis数据库

```
@RestController
@RequestMapping(value = "/redis")
public class RedisController {

    @Autowired
    private StringRedisTemplate redisTemplate;

    @RequestMapping(value = "/add")
    public Map<String, String> add(@RequestParam("key") String key,
@RequestParam("value") String value) {
        redisTemplate.opsForValue().set(key, value);
        Map<String, String> data = new HashMap<>();
        data.put(key, value);
```

```
        return data;
    }

    @RequestMapping(value = "/query")
    public Map<String, String> query(@RequestParam("key") String key) {
        Map<String, String> data = new HashMap<>();
        String value = redisTemplate.opsForValue().get(key);
        data.put(key, value);
        return data;
    }

    @RequestMapping(value = "/delete")
    public Map<String, String> delete(@RequestParam("key") String key) {
        Map<String, String> data = new HashMap<>();
        String value = redisTemplate.opsForValue().get(key);
        redisTemplate.delete(key);
        data.put(key, value);
        return data;
    }

    @RequestMapping(value = "/list")
    public Map<String, String> list(@RequestParam("key") String key) {
        Map<String, String> data = new HashMap<>();
        //要求是key的类型必须是string
        Set<String> keys = redisTemplate.keys(key);
        for (String k : keys) {
            if (redisTemplate.type(k) == DataType.STRING) {
                data.put(k, redisTemplate.opsForValue().get(k));
            }
        }
        return data;
    }
}
```

1.3. 邮件发送

Spring提供了一个简单的发送邮件的工具库，我们只需要依赖它，然后配置邮件服务器就可以进行发送邮件的操作。通过借用第三方邮箱（比如：网易邮箱，QQ邮箱）需要设置开启客户端发送邮件。

> 备注：**需要准备好测试邮箱和邮件服务**

示例：

- 在pom.xml中添加spring-boot-starter-mail依赖

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

- 在application.properties中添加发送邮件的服务器和帐号信息

```
spring.mail.protocol=smtp
spring.mail.host=smtp.qq.com
spring.mail.port=465
spring.mail.password=kkntpjueytjfbaad
spring.mail.username=1286072183
spring.mail.properties.mail.smtp.socketFactory.class=javax.net.ssl.SSLSocketFactory
```

- Spring Boot正常启动之后就会在Spring IoC容器中创建一个`JavaMailSender` Bean。之后我们就可以使用JavaMailSender发送邮件了。

1.3.1 发送简单邮件示例

下面代码示例展示了两种发送简单邮件的方式,第一种直接使用`Java Mail API`,第二种使用Spring提供的发送邮件的帮助类,第二种的API相对比较简洁,使用起来较为容易理解.

```
@Component
public class MailComponent {

    @Autowired
    private JavaMailSender mailSender;

    public void sendBasic1() throws MessagingException {
        MimeMessage mimeMessage = mailSender.createMimeMessage();
        mimeMessage.setRecipient(
                Message.RecipientType.TO,
                new InternetAddress("939694006@qq.com")
        );
        mimeMessage.setFrom(new InternetAddress("1286072183@qq.com"));
        mimeMessage.setSubject("SpringBoot send email by style1");
        mimeMessage.setText("Hello this is a simple mail.");
        mailSender.send(mimeMessage);
    }

    public void sendBasic2() throws MessagingException {
        MimeMessage mimeMessage = mailSender.createMimeMessage();
        MimeMessageHelper helper = new MimeMessageHelper(mimeMessage);
        helper.setFrom("1286072183@qq.com");
        helper.setTo("939694006@qq.com");
        helper.setSubject("SpringBoot send email by style2");
        helper.setText("Hello this is a simple mail.");
        mailSender.send(mimeMessage);
    }
}
```

1.3.2 发送复杂邮件示例

下面代码示例展示了发送带附件邮件和内联资源邮件.

```
//带附件
public void sendAttachments() throws MessagingException {
    MimeMessage message = mailSender.createMimeMessage();
    MimeMessageHelper helper = new MimeMessageHelper(message, true);
```

```
    helper.setFrom("1286072183@qq.com");
    helper.setTo("939694006@qq.com");
    helper.setSubject("sendAttachments");
    helper.setText("Check out this image!");
    ClassPathResource file = new ClassPathResource("secondriver.jpg");
    helper.addAttachment("CoolImage.jpg", file);
    mailSender.send(message);
}

//内联资源
public void sendInnerResources() throws MessagingException {
    MimeMessage message = mailSender.createMimeMessage();
    MimeMessageHelper helper = new MimeMessageHelper(message, true);
    helper.setFrom("1286072183@qq.com");
    helper.setTo("939694006@qq.com");
    helper.setSubject("sendInnerResources");
    helper.setText("Check out this image!");
    //资源引用处设置cid,资源标识
    helper.setText("<html><body><img src='cid:img123'></body></html>", true);
    ClassPathResource file = new ClassPathResource("secondriver.jpg");
    //添加资源时指定cid,资源标识符
    helper.addInline("img123", file);
    mailSender.send(message);
}
```

1.4. 测试

Spring Boot提供很多有用的测试应用的工具。spring-boot-starter-test的Starter提供Spring Test, JUnit, Hamcrest和Mockito的依赖。在spring-boot核心模块`org.springframework.boot.test`包下也有很多有用的测试工具。

如果使用`spring-boot-starter-test`的`Starter POM`（在test作用域内),将会提供下面库：

- Spring Test： 对Spring应用的集成测试支持
- JUnit： 用于Java应用的单元测试，事实上的标准。

如果它们不能满足你的要求，你可以随意添加其他的测试用的依赖库。

1.4.2 测试Spring Boot应用

一个Spring Boot应用只是一个Spring ApplicationContext，所以在测试它时除了正常情况下处理一个`vanilla Spring context`外不需要做其他特别事情。唯一需要注意的是，如果你使用SpringApplication创建上下文，外部配置，日志和Spring Boot的其他特性只会在默认的上下文中起作用。

Spring Boot提供一个@SpringBootTest注解用来替换标准的spring-test @ContextConfiguration注解。如果使用@SpringBootTest来设置你的测试中使用的ApplicationContext，它最终将通过SpringApplication创建，并且你将获取到Spring Boot的其他特性。

```
package com.bittech.boot;

import org.junit.Test;
```

```
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit4.SpringRunner;

import static org.assertj.core.api.Assertions.assertThat;

@RunWith(SpringRunner.class)
@SpringBootTest
public class TestSpringApplication {

    @Autowired
    private BookProperties bookProperties;

    @Test
    public void test() {
        assertThat("Hello Spring
Boot").isEqualToIgnoringCase(bookProperties.getName());
    }
}
```

1.4.3 测试运行的WEB服务

如果想让一个WEB应用启动，并且监听它的正常端口，可以使用HTTP来测试它（比如：使用TestRestTemplate），并且使用@SpringBootTest注解标识测试类。

```
package com.bittech.boot;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.boot.test.web.client.TestRestTemplate;
import org.springframework.test.context.junit4.SpringRunner;

import static org.assertj.core.api.Assertions.assertThat;

@RunWith(SpringRunner.class)
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
public class TestSpringBootWebApplication {

    @Autowired
    private TestRestTemplate testRestTemplate;

    @Test
    public void test() {
        String body = this.testRestTemplate.getForObject("/", String.class);
        assertThat(body).isEqualTo("Hello World");
    }
}
```

上面代码中我们使用了@SpringBootTest的webEnvironment属性，取值为RANDOM_PORT，其中webEnvironment取值有四种，分别代表了不同的测试环境。

- **MOCK**：加载WebApplicationContext和提供一个模拟的Servlet环境，嵌入式Servlet容器不启动；如果Servlet API不在classpath，这种模式会创建一个非WEB的ApplicationContext; 它可以 与@AutoConfigureMockMvc一起使用，用于对应用程序进行基于mockmvc的测试。（默认值）
- **RANDOM_PORT**: 加载ServletWebServerApplicationContext和提供真实的Servlet环境，嵌入式Servlet容器启动和监听随机端口
- **DEFINED_PORT**: 加载ServletWebServerApplicationContext和提供真实的Servlet环境，嵌入式Servlet容器启动和监听端口号来自application.properties定义或者使用默认端口8080
- **NONE**: 通过使用SpringApplication加载ApplicationContext，但是不提供任何的Servlet环境

> 注意：Spring测试框架在每次测试时会缓存应用上下文。因此，只要测试共享相同的配置，不管实际运行多少测试，开启和停止服务器只会发生一次。

2. 应用发布部署

SpringBoot应用的构建方式比较多,比如:基于Maven插件,基于Gradle插件,基于AntLib模块等等.由于Maven目前是Java项目构建工具中最为流行,也是Spring Boot项目本身主选的构建工具.所以我们主讲基于Maven插件的发布部署.

2.1 配置Maven插件

在pom.xml文件的中添加Maven Plugin依赖.

```
<build>
    <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <version>2.0.2.RELEASE</version>
                <executions>
                    <execution>
                        <goals>
                            <goal>repackage</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
    </plugins>
</build>
```

spring-boot-maven-plugin插件包含了如下几个goal(目标):

- spring-boot:help : 插件的帮助目标,可以查看插件的基本信息以及各个目标的描述信息
- spring-boot:build-info : 基于当前的Maven项目,生成build-info.properties信息
- spring-boot:repackage : 基于已经构建存在的jar重新打包成fatjar,如果有主类就可以执行
- spring-boot:run : 运行应用
- spring-boot:start : 启动应用,用于集成测试
- spring-boot:stop : 停止应用,用于集成测试

2.2 打包可执行fatjar运行

打包可执行jar的前提是我们需要将项目的打包格式设置为jar,如下设置:

```
<groupId>com.bittech</groupId>
<artifactId>spring-boot</artifactId>
<version>1.0.0</version>
<packaging>jar</packaging> <!-- 这里设置为jar -->
```

- 执行`mvn package`命令，构建可执行jar
- 切换目录到`target`下，执行`jar -jar spring-boot-1.0.0.jar`运行程序

2.3 打包war文件, 部署到容器

我们知道当引入spring-boot-starter-web这个starter的时候, web容器的依赖已经添加到项目中去了。打包成war文件(包)部署到容器中的时候，我们需要注意以下三点：

- war包不能包含WEB容器的jar包（因为一旦包含部署到容器中运行会出现jar冲突，比如：javax.servlet-api）
- 打包格式需要是war类型。
- 修改主类，继承SpringBootServletInitializer

下面我们对pom.xml文件做响应的调整，使得满足构建包是war包类型，并且可以部署到容器中正常运行。

- 修改打包类型

```
<groupId>com.bittech</groupId>
<artifactId>spring-boot</artifactId>
<version>1.0.0</version>
<packaging>war</packaging> <!-- 这里设置为war-->
```

- 使得web容器依赖的jar的scope范围为provided

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-tomcat</artifactId>
    <scope>provided</scope>
</dependency>
```

- 修改启动类

```
@SpringBootApplication
public class Application extends SpringBootServletInitializer {
    @Override
    protected SpringApplicationBuilder configure(SpringApplicationBuilder
application) {
        return application.sources(Application.class);
    }
    public static void main(String[] args) throws Exception {
        SpringApplication.run(Application.class, args);
    }
}
```

> 注意：如果WEB容器不是tomcat,那么需要修改的是对应的容器依赖starter(可能是：spring-boot-starter-jetty，spring-boot-starter-undertow)

修改完pom.xml文件和主类之后，我们就可以继续进行打包，部署

- 执行`mvn package`命令，构建可执行war
- 切换目录到`target`下，将`spring-boot-1.0.0.war`部署到Tomcat的安装目录下的webapp即可完成部署。

3. Proudction-ready特性

Spring Boot包含很多其他的特性，它们可以帮助我们监控和管理发布到生产环境的应用。我们可以选择使用HTTP端点，JMX或者远程Shell(SSH或者Telnet)来管理和监控应用程序。审计（Auditing）,健康（Health）和数据采集（Metrics Gathering）会自动应用到应用。

3.1 产品特性

`spring-boot-actuator`模块提供了Spring Boot所有的product-ready特性。启用该特性的最简单方式就是添加`spring-boot-actuator`的`Starter POM`的依赖。

Actuator（执行器）的定义：执行器是一个制造业术语，指的是用于移动或控制东西的一个机械装置。一个很小的改变就能让执行器产生大量的运动。

基于Maven的项目想要添加执行器只需要添加下面的`Starter`依赖：

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

3.2 端点

执行器端点允许你监控应用及与应用进行交互。Spring Boot包含很多内置的端点，你也可以添加自己的。例如，health端点提供了应用的基本健康信息。

端点暴露的方式取决于你采用的技术类型。大部分应用选择HTTP监控，端点的ID映射到一个URL。例如，默认情况下，health端点将被映射到/health。

spring-boot-actuator提供的如下可用端点：

ID	描述	敏感（Sensitive）
autoconfig	显示一个auto-configuration的报告，该报告展示所有auto-configuration候选者及它们被应用或未被应用的原因	true
beans	显示一个应用中所有Spring Beans的完整列表	true
configprops	显示一个所有@ConfigurationProperties的整理列表	true
dump	执行一个线程转储	true
env	暴露来自Spring ConfigurableEnvironment的属性	true
health	展示应用的健康信息（当使用一个未认证连接访问时显示一个简单的'status'，使用认证连接访问则显示全部信息详情）	false
info	显示任意的应用信息	false
metrics	展示当前应用的'指标'信息	true
mappings	显示一个所有@RequestMapping路径的整理列表	true
shutdown	允许应用以优雅的方式关闭（默认情况下不启用）	true
trace	显示trace信息（默认为最新的一些HTTP请求）	true

注：根据一个端点暴露的方式，sensitive参数可能会被用做一个安全提示。例如，在使用HTTP访问sensitive端点时需要提供用户名/密码（如果没有启用web安全，可能会简化为禁止访问该端点）。

启用Web端点：

```
#包含所有的web的端点
management.endpoints.web.exposure.include=*
#排除beans的端点
management.endpoints.web.exposure.exclude=beans
```

3.2.1 自定义端点

使用Spring属性可以自定义端点。你可以设置端点是否开启（enabled），是否敏感（sensitive），甚至它的id。例如，下面的application.properties改变了敏感性和beans端点的id，也启用了shutdown。

```
endpoints.beans.id=springbeans
endpoints.beans.sensitive=false
endpoints.shutdown.enabled=true
```

> 前缀`endpoints + . + name`被用来唯一的标识被配置的端点。

默认情况下，除了shutdown外的所有端点都是启用的。如果希望指定选择端点的启用，可以使用`management.endpoint.<id>.enabled`属性。

下面示例是启用shutdown的endpoint:

```
management.endpoint.shutdown.enabled=true
```

下面的配置禁用了除info外的所有端点：

```
management.endpoints.enabled-by-default=false
management.endpoint.info.enabled=true
```

3.2.2 健康信息

健康信息可以用来检查应用的运行状态。它经常被健康软件用来提醒人们生产系统是否停止。health端点暴露的默认信息取决于端点是如何被访问的。对于一个非安全，未认证的连接只返回一个简单的`status`信息。对于一个安全或者认证过的连接其他详情信息也会展示。

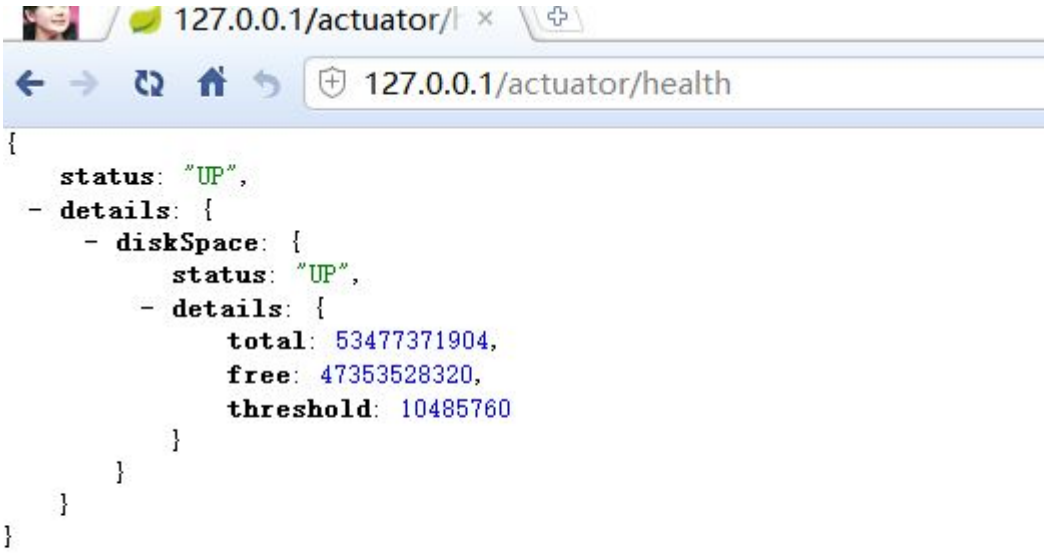
健康信息是从ApplicationContext中定义的所有HealthIndicator beans收集过来的。Spring Boot包含很多auto-configured的HealthIndicators，开发者也可以自定义HealthIndicator。

- 安全与HealthIndicators

HealthIndicators返回的信息通常性质上有点敏感。例如：你可能不想将数据库服务器的详情发布到外面。因此，在使用一个未认证的HTTP连接时，默认会暴露健康状态（Health Status）。

暴露健康状态：

```
//取值如下: when_authorized, never, always
//when_authorized:  认证用户可以访问
//never: 健康状态不可查看
//always: 健康状态总是可以查看
management.endpoint.health.show-details=always
```



为防止**拒绝服务（Denial of Service 简称：DoS）**攻击，Health响应会被缓存。你可以使用management.endpoint.health.cache.time-to-live属性改变默认的缓存时间（比如：1000ms）。

- 自动配置的HealthIndicators

Spring Boot会在合适的时候自动配置下面的HealthIndicators。

名称	描述
DiskSpaceHealthIndicator	低磁盘空间检测
DataSourceHealthIndicator	检查是否能从DataSource获取连接
MongoHealthIndicator	检查一个Mongo数据库是否可用（up）
RabbitHealthIndicator	检查一个Rabbit服务器是否可用（up）
RedisHealthIndicator	检查一个Redis服务器是否可用（up）
SolrHealthIndicator	检查一个Solr服务器是否可用（up）

- 自定义HealthIndicators

如果想为应用程序自定义健康信息，需要注册一个实现`org.springframework.boot.actuate.health.HealthIndicator`接口的Spring Beans。需要提供一个`health()`方法的实现，并返回一个Health响应。Health响应需要包含一个status和可选的用于展示的详情。

```
import org.springframework.boot.actuate.health.Health;
import org.springframework.boot.actuate.health.HealthIndicator;
import org.springframework.stereotype.Component;
@Component
public class MyHealthIndicator implements HealthIndicator {
    @Override
    public Health health() {
        int errorCode = check(); // perform some specific health check
        if (errorCode != 0) {
        return Health.down().withDetail("Error Code", errorCode).build();
        }
        return Health.up().build();
    }
```

```
}
```

除了Spring Boot预定义的Status类型，Health也可以返回一个代表新的系统状态的自定义Status。在这种情况下，需要提供一个HealthAggregator接口的自定义实现，或使用management.health.status.order属性配置默认的实现。

例如，假设一个新的，代码为FATAL的Status被用于你的一个HealthIndicator实现中。为了配置严重程度，你需要将下面的配置添加到application属性文件中：

```
management.health.status.order=FATAL, DOWN, OUT_OF_SERVICE, UNKNOWN, UP
```

总结

知识块	知识点	分类	掌握程度
SpringBoot实战	1. SQL数据库 2.Redis数据库 3.邮件发送 4.集成测试	实战型	掌握
事务管理	1.配置pom.xml 2.打包部署	实战型	掌握
产品特性	1. 端点以及配置 2.健康检查	实战型	掌握

课后作业

- 创建一个Spring Boot项目，练习发送邮件
- 创建一个Spring Boot Web项目，写几个简单的接口，打包成war包，部署到Tomcat，熟悉整个Spring Boot项目开发流程