



下载APP



## 14 | 在 Spring 项目如何进行集成测试？

2021-09-03 郑晔

《程序员的测试课》

课程介绍 >



讲述：郑晔

时长 11:42 大小 10.72M



你好，我是郑晔！

上一讲我们讲了 Spring 对轻量级开发的支持。不同于传统的开发方式，Spring 希望可以做到开发不依赖于应用服务器。为了达成这个目标，Spring 提供了各种支持，能够让你在部署到容器之前完成所有代码的基础验证工作。在核心业务部分，只要我们能够不过分依赖于 Spring 的种种特性，测试就和普通的单元测试差别不大。

不过在真实世界的软件开发中，我们总要与其它的外部组件集成。一旦牵扯到集成，测试的难度就上来了。不过正如前面所说，Spring 要尽可能让你在不依赖于容器的情况下测试。Spring 的做法就是提供一套自己的方案，替代掉对于容器的依赖。

这一讲，我们就来看看采用 Spring 的项目如何做集成测试。

## 数据库的测试


今天数据库几乎成了所有商业项目的标配，所以，Spring 也提供了对于数据库测试很好的支持。我们之前说过，一个好的测试要有可重复性，这句话放到数据库上就是要保证测试之前的数据库和测试之后的数据库是一样的。怎么做到这一点呢？

### 测试配置

通常有两种做法，一种是采用嵌入式内存数据库，也就是在测试执行之后，内存中的数据一次丢掉。另一种做法就是采用真实的数据库，为了保证测试前后数据库是一致的，我们会采用事务回滚的方式，并不把数据真正地提交进数据库里。

我们做测试的一个关键点就是不能随意修改代码，切记，**不能为了测试的需要而修改代码**。如果真的要修改，也许应该修改的是设计，而不仅仅是代码。

虽然不能修改代码，但我们可以提供不同的配置。只要我们给应用提供不同的数据库连接信息，它就会连到不同的数据库上。Spring 就给了我们一个提供不同配置的机会，只要我们在测试中声明一个不同的属性配置即可，下面就是一个例子。

 复制代码


```
1 @ExtendWith(SpringExtension.class)
2 @DataJpaTest
3 @AutoConfigureTestDatabase(replace = AutoConfigureTestDatabase.Replace.NONE)
4 @TestPropertySource("classpath:test.properties")
5 public class TodoItemRepositoryTest {
6     ...
7 }
```

在这段代码里，我们提供了一个测试用的配置，也就是 `@TestPropertySource` 给出的一个配置。这是在用 `classpath` 上的 `test.properties` 这个文件中的配置，去替换掉我们缺省的配置（也就是我们真实的数据库）。

### 嵌入式内存数据库


正如我们前面所说，我们要保证数据库的可重复性有两种做法：嵌入式内存数据库和事务回滚。要想使用嵌入式内存数据库，我们需要提供一个嵌入式内存数据库的配置。在 Java

世界中，常见的嵌入式内存数据库有 H2、HSQLDB、Apache 的 Derby 等。我们配置一个测试的依赖就好，以 H2 为例，像下面这样。

 复制代码

```
1 testImplementation "com.h2database:h2:$h2Version"
```

然后，再提供一个相应的配置，像下面这样。

 复制代码

```
1 jdbc.driverClassName=org.h2.Driver
2 jdbc.url=jdbc:h2:mem:todo;DB_CLOSE_DELAY=-1
3 hibernate.dialect=org.hibernate.dialect.H2Dialect
4 hibernate.hbm2ddl.auto=create
```

如果运气好的话，你的测试就可以顺利地运行了。是的，运气好的话。

之所以把软件开发这么严肃认真的事归结到运气，这就不得不说说使用嵌入式内存数据库的问题了。


严格地说，这不是嵌入式内存数据库的问题，这其实是只要运行在不同的数据库上都会有问题，也就是 SQL 的不一致。虽然我们知道 SQL 有一个统一的标准，然而，几乎每个数据库引擎为了某些特点都有一些特殊的处理。造成的结果就是，虽然理论上说 SQL 可以运行在所有的数据库引擎上，然而真实情况却是总有一部分 SQL 只能运行在特定的引擎上。

如果你用的是 JPA 这种技术，因为 JPA 会根据数据库引擎替我们生成真实的 SQL，这个问题体现得还不是特别明显。但如果你用的 MyBatis 或者是其它需要手写 SQL 的技术，一旦发现了不能运行的 SQL，你就不得不再此权衡一下，如何去面对两个有差异的数据库。

所以，嵌入式内存数据库这种技术看上去很美，但我在实际的项目中用得并不多，我更多会采用事务回滚的方式。


## 事务回滚

在事务回滚的方式中，我们的配置几乎与标准的应用配置是一样的，下面是我们在实战中所采用的配置。

 复制代码

```
1 spring.datasource.url=jdbc:mysql://localhost:3306/todo_test?useUnicode=true&ch
2 spring.datasource.username=todo
3 spring.datasource.password=geektime
4 spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
```

通常来说，为了不让测试过程和开发过程造成数据冲突，我们会创建两个不同的数据库，在 MySQL 中，这就是两条 SQL 语句。

 复制代码


```
1 create database todo_dev;
2 create database todo_test;
```

这样，一个用来做手工测试用，另外一个交由自动化测试使用，你从数据库后缀名上就可以看出二者的差异。顺便说一下，这种做法在业界的普遍流行是源自 Ruby on Rails（一个 Ruby 的 Web 开发框架），当年它在软件开发实践上给整个行业带来了极大的颠覆。

采用这种做法，我们的代码面对的是同样的数据库引擎，也就不必担心 SQL 不兼容的问题了。

我们所说的事务回滚体现在 @DataJpaTest 上，它把数据库回滚做成缺省的配置，所以我们什么都不用做，就可以获得这样的能力。

与大多数测试一样，测试与数据库的集成时，我们也要做一些准备。需要准备的往往是一些数据，提前插入到数据库里。我们可以使用 Spring 给我们准备的基础设施（TestEntityManager）向数据库中完成这个工作，下面是一个例子。

 复制代码

```
1 @ExtendWith(SpringExtension.class)
2 @DataJpaTest
3 public class ExampleRepositoryTests {
4     @Autowired
5     private TestEntityManager entityManager;
6 }
```



```
7    @Test
8    public void should_work() throws Exception {
9        this.entityManager.persist(new User("sboot", "1234"));
10       ...
11    }
12 }
```

如果你用的不是 JPA 而是其它的数据访问方式，Spring 也给我们提供了 `@JdbcTest`，这相当于是一个更基础的配置，因为只要有 `DataSource`，它就可以很好地工作起来，这适用于绝大多数的测试情况。相应地，数据工作也更加地直接，采用 SQL 就可以，下面是一个例子。

[复制代码](#)

```
1  @JdbcTest
2  @Sql({"test-data.sql"})
3  class EmployeeDAOIntegrationTest {
4      @Autowired
5      private DataSource dataSource;
6
7      ...
8  }
```

## Web 接口测试

除了数据库，另外一个几乎成了今天标配的就是 Web。Spring 对于 Web 测试也提供了非常好的支持。

如果按照我在实战中的方式工作，你会发现到了编写 Web 接口这步，我们基本上完成了几乎所有的工作，只差给外界一个接口让它和我们的系统连接起来。在前面的实战中，我们采用整体集成的方式对系统进行测试，这里的关键点就是 `@SpringBootTest`，它把所有的组件都连接了起来。

[复制代码](#)

```
1  @SpringBootTest
2  @AutoConfigureMockMvc
3  @Transactional
4  public class TodoItemResourceTest {
5      ...
6  }
```


在讲集成测试的时候我曾经说过，集成测试分为两种，一种把所有代码都集成起来的测试，另外一种是针对外部组件的集成。从代码上来看，后一种测试只是针对一个单元在测试，所以它兼具单元测试和集成测试的特点。其实，测试 Web 接口也有一种类似于单元测试的集成方式，它采用的 `@WebMvcTest`。

 复制代码

```
1 @WebMvcTest(TodoItemResource.class)
2 public class TodoItemResourceTest {
3     ...
4 }
```

正如你在这段代码中看见的那样，这里我们指定了要测试的组件 `TodoItemResource`。在这个测试里，它不会集成所有的组件，只会集成与 `TodoItemResource` 相关的部分，但整个 Web 处理过程是完整的。

如果把它视为单元测试，服务层后面的代码都是外部的，我们可以采用模拟对象把它控制在可控范围内，这个时候，上一讲遗漏的 `MockBean` 就开始发挥作用了。

 复制代码

```
1 @WebMvcTest(TodoItemResource.class)
2 public class TodoItemResourceTest {
3     @MockBean
4     private TodoItemService service;
5
6     @Test
7     public void should_add_item() throws Exception {
8         when(service.addTodoItem(TodoParameter.of("foo"))).thenReturn(new TodoItem
9         ...
10    }
11 }
```

在这里，`@MockBean` 标记的 `TodoItemService` 模拟对象会参与到组件组装的过程中，成为 `TodoItemResource` 的组成部分，我们就可以设置它的行为。如果 Web 接口同服务层有比较复杂的交互，那这种做法就能够很好的处理。当然，正如我们一直在说的，我不建议这里做得过于复杂。

`@WebMvcTest` 这种偏向于单元测试的做法，执行速度相对于 `@SpringBootTest` 这种集成了所有组件的做法而言要快一些。所以如果测试的量大的话，采用 `@WebMvcTest` 会

有一定的优势。

理解 Web 接口测试还有一个关键点。正如我在之前内容中说过，当年 Spring 摆脱了大部分对于应用服务器的依赖，但是 Web 却是它一直没有摆脱的。所以，怎么更好地不依赖于 Web 服务器进行测试，就是摆在 Spring 面前的问题。答案是 **Spring 提供了模拟的 Web 环境**。

具体到我们的测试上，它就是 MockMvc 对象发挥的作用。我们用下面的代码回顾一下它的用法。

 复制代码

```
1 @SpringBootTest
2 @AutoConfigureMockMvc
3 @Transactional
4 public class TodoItemResourceTest {
5     @Autowired
6     private MockMvc mockMvc;
7     ...
8
9     @Test
10    public void should_add_item() throws Exception {
11        String todoItem = "{ " +
12            "\"content\": \"foo\"" +
13            "}";
14        mockMvc.perform(MockMvcRequestBuilders.post("/todo-items")
15            .contentType(MediaType.APPLICATION_JSON)
16            .content(todoItem))
17            .andExpect(status().isCreated());
18        assertThat(repository.findAll()).anyMatch(item -> item.getContent().eq
19    }
20 }
```

这里的关键是 `@AutoConfigureMockMvc`，它为我们配置好了 MockMvc，剩下的就是我们使用这个配置好的环境进行访问。

从程序库的角度看，MockMvc 可以理解成客户端的 Moco，同样是设置请求和应答。和 Moco 不同的点在于，它的请求是设置好的，而应答要匹配。

从实现的角度理解，它就是那个模拟的 Web 环境。所谓模拟的环境，是因为它根本没有启动真正的 Web 服务器，而是直接去调用了我们的代码，省略了请求在网络上走一遭的过

程。但请求进到服务器之后的主要处理都在，所以相应的处理都在（无论是各种 Filter 的处理，还是从请求体到请求对象的转换）。现在你应该明白了，MockMvc 是 Spring 轻量级开发的一个重要的组成部分。

到这里，我给你介绍了 Spring 集成测试中最常用到的两种：数据库测试和 Web 接口测试。这里介绍的也是推荐你去使用的做法。还有一些细节的做法我在这里没有提到，比如可以取消数据的回滚，再比如使用真实的 Web 环境（走网络的那种），不提是因为它们并不是值得推荐的做法。

正如我在最近两讲一直说的那样，Spring 在支持轻量级开发上做了很大的努力，所以，在把整个系统集成起来之前，绝大部分内容我们都已经验证过了。我在这里介绍的只是其中最为典型的用法，Spring 的测试绝对是一个值得挖掘的宝藏，你可以阅读它的文档去发掘更多有趣的用法。

现在我们对怎样在真实项目中做好单元测试和集成测试已经有了一个基本的理解，但在实际的项目中，不同类型的测试该怎么配比呢？这就是我们下一讲要讨论的内容。

## 总结时刻

今天我们讨论了在 Spring 项目中怎么进行集成测试，主要讲解了如何做数据库和 Web 接口的集成测试。

做数据库测试，难点在于如何在测试之后恢复环境。有两种典型的做法：使用嵌入式内存数据库或是使用事务回滚的机制。无论是哪种做法，重点是给测试提供不同的配置，保证代码不变。

因为不同数据库引擎对 SQL 兼容程度不同，我更建议你使用事务回滚的做法。

Web 接口测试通常是最外层的测试，可以做整体的集成测试（@SpringBootTest），或对一个单元进行测试的集成测试（@WebMvcTest）。

在 Web 接口测试中，一个关键点是采用模拟 Web 环境，这样可以在不启动 Web 服务器的前提下进行测试。这种做法不依赖于部署过程，测试速度可以大幅度提升。



如果今天的内容你只能记住一件事，那请记住：**采用轻量级的测试手段，保证代码的正确性。**

## 思考题

今天我们讲了 Spring 对于集成测试的支持，希望你可以通过阅读文档，了解它的更多特性。如果你在阅读文档的过程中发现了哪些有趣的特性，欢迎在留言区分享你的所得。

分享给需要的人，Ta订阅后你可得 **20** 元现金奖励

👍 赞 0    💡 提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇   13 | 在 Spring 项目中如何进行单元测试？

下一篇   15 | 测试应该怎么配比？

## 精选留言 (3)

💬 写留言



邓志国

2021-09-04

引入cucumber后，实际上是启动了web服务，通过http来测试，这样更加真实。不知道mockmvc有没有什么坑，毕竟它稍微假了一点

展开 ▾



webmin

2021-09-03

H2本身支持设定为模拟oracle,mysql等数据库(url中加上MODE=Oracle)，可以支持特定DB常用的函数和机制，到是准备测试用的初始化数据是比较考验耐心和毅力，后期如果调整schema，需要对测试用到的SQL逐一调整也一项体力活。

展开 ▾

作者回复: 多谢补充



**grandgraph**

2021-09-03

老师, 对于Go语言来说是否也有类似的测试工具推荐使用呢？

