1. 技术选型分析

1.1 Mock框架

做测试，本质上就是在一个可控的环境下对被测系统 / 组件进行各种试探。

怎么把不可控变成可控?

第一步是隔离，

第二步用一个可控的组件代替不可控的组件。换言之，用一个假的组件代替真的组件。

典型实例

二方服务；业务配置；MySQL。

1.2 Mock工具选型「Mockito&PowerMock&TestableMock」

选型结果：对Mockito、PowerMock、TestableMock对比分析，选择Mockito + PowerMock。

用途：基于实践，Mockito用于替换Spring Bean；PowerMock用于替换静态方法。

选型分析

工具	原理	最小Mock单元	对被Mock方法的限制	上手难度	IDE支持
Mockito	动态代理	类	不能Mock私有/静态和构造方法	**较容易**	**很好**
PowerMock	自定义类加载器	类	**任何方法皆可**	较复杂	**较好**
TestableMock	运行时字节码修改	方法	**任何方法皆可**	**很容易**	一般

Mockito

优势：spring-boot-starter-test默认集成Mockito，与Spring Boot有更好的兼容性。

劣势：基于动态代理/运行时替换对象等，不能Mock私有/静态和构造方法。

PowerMock

优势：基于Mockito扩展，完美兼容Mockito。功能强大。

劣势：

- 使用了自定义类加载器，会导致Jacoco在默认的`on-the-fly`模式下覆盖率跌零。
- 使用了自定义类加载器，由于JDK一些类严格限定了所用的类加载器，需要通过配置精确排除一些类，避免应用启动报错。
- 使用了自定义类加载器，部分二方/三方组件未考虑到应用使用自定义类加载器的场景，会出现无法预料的异常「类加载器不同导致一些对象无法存入不同"同类型"的容器类。」

TestableMock

优势：`TestabledMock`的功能与`PowerMock`基本平齐，且极易上手。

劣势：

- 官方推荐使用`Maven`/`Gradle`方式`run/debug`应用。使用Maven方式运行调试，与IDEA适配性不好，原因见附录。
- 如果不使用`Maven`/`Gradle`方式`run/debug`应用，则需要变更java启动参数「增加 javaagent参数」，稍有复杂性。
- 编写Mock方法时IDE无法即时提示方法参数是否正确匹配，必须在运行时通过日志和排查文档辅助，入门门槛高/成本高。这个功能理论上能够通过扩展主流IDE插件来补充，但目前暂无相关开发计划。

原理：基于 javaagent 技术，在 main 方法启动前拦截类的加载并对类进行重写，在单元测试运行时替换对被Mock方法的调用。

- javaagent使用指南
- TestableMock的设计和原理

网站：https://alibaba.github.io/testable-mock

附录

TestableMock使用`Maven`方式`run/debug`应用

1. Debug模式功能不丰富，没有IDEA原生功能丰富。
2. 运行测试用例时，只有命令行输出，页面可视化标识均没有，IDEA原生的标识均作废了。

配置实例如下。

1. 依赖maven插件：maven-surefire-plugin

```
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-surefire-plugin</artifactId>
            <configuration>
                <argLine>-
javaagent:${settings.localRepository}/com/alibaba/testable/testable-
agent/${testable.version}/testable-agent-${testable.version}.jar</argLine>
            </configuration>
        </plugin>
    </plugins>
</build>
```

2. 使用Maven运行测试用例

如果您只想运行单个测试而不是项目中声明的所有测试，请使用 Maven`-Dtest=TestName test`命令为单个测试创建 Maven 运行配置。运行配置将保存在“**运行配置**”节点下。

1. 在**Maven**工具窗口中，在**Lifecycle**节点下，右键单击**test**目标。
2. 从上下文菜单中，选择**Create 'name of the module/project and name of a goal'**。
3. 在打开的对话框中，指定包含要运行的测试的工作目录，并在**命令行**字段中指定阶段（自动指定）和`-Dtest=TestName test`命令。

3. **debug一个Maven的测试用例**

参考maven-surefire-plugin插件的Debug章节，使用分离进程「Forked Tests」方式运行，执行命令：

```
mvn -Dmaven.surefire.debug test
```

IDEA会自动暂时attach命令，点击后即可进入debug模式。

1.2 内存数据库选型「MariaDB4J&H2&Testcontainers」

典型实践

- 使用嵌入式内存数据库：可断网执行，无需事务回滚，但可能遇到SQL不兼容的风险。
- 使用事务回滚的机制：我们的代码面对的是同样的数据库引擎，不必担心 SQL 不兼容的问题。

选型结果及说明

1. 核心诉求是断网执行，可尽力规避SQL不兼容的风险，所以选择「嵌入式内存数据库」
2. 重点是给测试提供不同的配置「JDBC URL」，保证代码不变。

选型结果

对H2、Wix Embedded MySql、MariaDB4j、Testcontainers对比分析，选择MariaDB4j；备选H2。

用途：替换MySQL数据库

选型分析

H2

优势：

- 开源的嵌入式数据库引擎，java语言编写
- 提供兼容模式，可以兼容一些主流的数据库

劣势：与MySQL语法有兼容性问题，很多SQL「如建表SQL」均需要重新适配。

资料：H2数据库攻略

Wix Embedded MySql

优势：MySQL的内存&嵌入式数据库版本，可以完美替换MySQL数据库。

劣势：项目已废弃，官方推荐了 Testcontainers，官方关联了MariaDB4j

MariaDB4j

优势：

- MariaDB的内存&嵌入式数据库版本。
- MariaDB是MySQL的衍生项目，兼容MySQL语法，可以替换MySQL数据库

劣势：

- 由于模拟了MariaDB的使用方式，需要绑定一个本地端口，驱动的连接字符串为：jdbc:mariadb://localhost:3306/lendengine
- 客户端驱动需要替换为：org.mariadb.jdbc.Driver

- 与MySQL语法是否完全兼容，仍然存疑。

资料：

- https://juejin.cn/post/6868239324932997133

Testcontainers

优势：

- 支持 JUnit 测试的 Java 库，提供轻量级、一次性的通用数据库实例
- Spring Boot 官方推荐的集成测试工具
- 方便控制容器的生命周期。
- 可以做任何事情，理论上可以模拟一切外部组件。

劣势：

- 完全基于Docker，Mac使用前需要先安装Docker；Windows使用前需要安装 Docker + WSL2，初次使用成本高。
- 公司的Jenkins CI「持续集成」方式未知，需要研究是否可以兼容。不过主流的CI方式都是兼容的。

三、集成测试启动全流程

基于演示项目 bitkylin-spring-boot ，以及应用 lendengine 进行演示。

1. 基于Spring SPI机制，加载 `SysConfigListener`

2. 基于Spring SPI机制，加载 SimbusinessStarterAutoConfiguration -> SimbusinessPostProcessor

调用方法，AbstractConfigEntrypointProcessor#postProcessBeanFactory，将配置从远端拉到本地。

四、公司配置中心的Mock

1. 配置中心原理分析

1. 加载本地配置文件

application-local.properties「SpringBoot：ConfigFileApplicationListener」「此环节没有实际作用，因为结果被后面的覆盖了」

2. 从configservice中读取配置「系统配置『proplus』」

SysConfigListener

req: pod基本信息「应用名、集群、主机名、子环境」

http://configservice.apps01.ali-bj-sit03.shuheo.net/configservice/sysconf/pull?r_e=lendengine

作用：

1. 远端加载系统配置
2. 存入ConfigService，处理系统配置热更新相关事宜「系统配置热更新本集成测试不涉及」

3. 远端系统配置和本地配置文件合并「优先级：`application-local.properties` > 远端系统配置 > `application.properties`」
4. 合并后的 `PropertySource` 作为 Spring Boot 优先级最高的配置源

3. SimBusiness

基于Spring SPI机制，加载 SimbusinessStarterAutoConfiguration -> SimbusinessPostProcessor

调用方法，AbstractConfigEntrypointProcessor#postProcessBeanFactory，将配置从远端拉到本地。

作用：

1. 全量拉取远端业务配置
2. 设定定时任务，周期性更新本地业务配置。

五、@MockBean原理

org.springframework.boot.test.mock.mockito.MockitoPostProcessor

1. 上述 bean 初始化时即获知所有加注了 @MockBean 的 bean
2. 在执行 #postProcessBeanFactory 时，创建并注册了需 mock 的 bean 的 RootBeanDefinition，以及 Bean 实例

#postProcessBeanFactory -> #registerMock

六、jacoco多module集成研究

使用项目 `bitkylin-integration-test` 进行演示。

1. parent的pom.xml中添加：

```
    <build>
        <plugins>
            <plugin>
                <groupId>org.jacoco</groupId>
                <artifactId>jacoco-maven-plugin</artifactId>
                <version>0.8.7</version>
                <executions>
                    <execution>
                        <id>prepare-agent</id>
                        <goals>
                            <goal>prepare-agent</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
```

2. 业务module正常使用，不需要体现jacoco的任何信息
3. 新增 `report` module，pom.xml中写入如下信息：

```
    <dependencies>
```

```
        <dependency>
            <artifactId>subModule1</artifactId>
            <groupId>cc.bitky.testky</groupId>
            <version>0.0.1-SNAPSHOT</version>
        </dependency>
        <dependency>
            <artifactId>subModule2</artifactId>
            <groupId>cc.bitky.testky</groupId>
            <version>0.0.1-SNAPSHOT</version>
        </dependency>
        <dependency>
            <groupId>org.jacoco</groupId>
            <artifactId>jacoco-maven-plugin</artifactId>
            <version>0.8.7</version>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.jacoco</groupId>
                <artifactId>jacoco-maven-plugin</artifactId>
                <version>0.8.7</version>
                <executions>
                    <execution>
                        <id>report-aggregate</id>
                        <phase>verify</phase>
                        <goals>
                            <goal>report-aggregate</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
```

关键点:

1. 依赖欲收集覆盖率信息的业务module
2. 添加jacoco插件的execution `report-aggregate`