单元测试：

1 、可测试性代码

2、dao层的单元测试；jpa不自动配置数据源的设置

mockito Mock MockBean 区别

**Plain Mockito library**

import org.mockito.Mock;

...

@Mock

MyService myservice;

and

import org.mockito.Mockito;

...

MyService myservice = Mockito.mock(MyService.class);

come from the Mockito library and are functionally equivalent.

They allow to mock a class or an interface and to record and verify behaviors on it.

The way using annotation is shorter, so preferable and often preferred.

Note that to enable Mockito annotations during test executions, the MockitoAnnotations.initMocks(this) static method has to be called.

To avoid side effect between tests, it is advised to do it before each test execution :

@Before

public void initMocks() {

MockitoAnnotations.initMocks(this);

}

Another way to enable Mockito annotations is annotating the test class with @RunWith by specifying the MockitoJUnitRunner that does this task and also other useful things :

@RunWith(org.mockito.runners.MockitoJUnitRunner.class)

public MyClassTest{...}

**Spring Boot library wrapping Mockito library**

This is indeed a [Spring Boot class](https://docs.spring.io/spring-boot/docs/current/api/org/springframework/boot/test/mock/mockito/MockBean.html):

import org.springframework.boot.test.mock.mockito.MockBean;

...

@MockBean

MyService myservice;

The class is included in the spring-boot-test library.

It allows to add Mockito mocks in a Spring ApplicationContext.

If a bean, compatible with the declared class exists in the context, it **replaces** it by the mock.

If it is not the case, it **adds** the mock in the context as a bean.

Javadoc reference :

Annotation that can be used to add mocks to a Spring ApplicationContext.

...

If any existing single bean of the same type defined in the context will be replaced by the mock, if no existing bean is defined a new one will be added.

**When use classic/plain Mockito and when use @MockBean from Spring Boot ?**

Unit tests are designed to test a component in isolation from other components and unit tests have also a requirement : being as fast as possible in terms of execution time as these tests may be executed each day dozen times on the developer machines.

Consequently, here is a simple guideline :

As you write a test that doesn't need any dependencies from the Spring Boot container, the classic/plain Mockito is the way to follow : it is fast and favors the isolation of the tested component.

If your test needs to rely on the Spring Boot container **and** you want also to add or mock one of the container beans : @MockBean from Spring Boot is the way.