ＳＴＥＰ０９-１

Ｓｐｒｉｎｇフレームワーク入門

コードで理解するＳｐｒｉｎｇの世界

jUnit/Spring/Mockitoを使用したテスト

STEP09-1各クラスを個別にテストする方法を試してみる（単体テスト的）

目次

[1 単体テストの準備 1](#_Toc467598838)

[1.1 Spring のTestで必要なライブラリ登録 （Maven） 1](#_Toc467598840)

[1.2 テスト用にAOPを使用しないSpring MVC 設定ファイルを用意 2](#_Toc467598841)

[2 各クラスを個別にテストする 3](#_Toc467598842)

[2.1 書籍一覧Ｄａｏクラスのテストケース（正常系）新規作成 3](#_Toc467598844)

[2.2 書籍一覧Ｄａｏクラスのテストケース（正常系）実行 4](#_Toc467598845)

[2.3 書籍一覧Ｄａｏクラスのテストケース（異常系）追加 5](#_Toc467598846)

[2.4 書籍一覧Ｄａｏクラスのテストケース（正常系＋異常系）実行 5](#_Toc467598847)

[2.5 書籍登録Ｄａｏクラスのテストケース（正常系＋異常系）を新規作成 6](#_Toc467598848)

[2.6 書籍登録Ｄａｏクラスのテストケース（正常系＋異常系）実行 7](#_Toc467598849)

[2.7 書籍登録Ｄａｏクラスのテストケース（正常系別パターン）を追加 7](#_Toc467598850)

[2.8 書籍登録Ｄａｏクラスのテストケース（正常系＋異常系）実行 8](#_Toc467598851)

[2.9 プロジェクト全体のテストケースを一気に実行 8](#_Toc467598852)

# 単体テストの準備



## Spring のTestで必要なライブラリ登録 （Maven）

１）Maven設定ファイルに必要なライブラリを追記

今回のテストでは、Spring 用のTestライブラリと、Mockitoというモックライブラリ等が必要。

|  |
| --- |
| /pom.xml |
| ● 108行目付近 |
| <!-- Test -->  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <scope>test</scope>  <version>4.11</version>  </dependency>  115  <!-- Spring + junit -->  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-test</artifactId>  <version>${org.springframework-version}</version>  <scope>test</scope>  </dependency>  <!-- Mockito -->  <dependency>  <groupId>org.mockito</groupId>  <artifactId>mockito-core</artifactId>  <version>1.10.19</version>  <scope>test</scope>  </dependency>    </dependencies> |

## Maven設定ファイルを更新するとプロジェクトがリビルドする。

## ビルドの進捗は右下に地味に表示されているだけなので見落とさないこと

## ビルドに数秒～数分がかかる場合があるが、終わるまで待っているほうが吉。

## テスト用にAOPを使用しないSpring MVC 設定ファイルを用意

１）Spring MVC 設定ファイル（application-context-biz.xml）をテスト用にコピー

●コピー元

|  |
| --- |
| /src/main/webapp/WEB-INF/spring/application-context-biz.xml |

●コピー先 ＜ファイルコピー＋ペースト後、リファクタリングでリネームします＞

|  |
| --- |
| /src/test/resources/test-context-biz.xml |

２）Spring MVC 設定ファイル（application-context-web.xml）をテスト用にコピー

●コピー元

|  |
| --- |
| /src/main/webapp/WEB-INF/spring/application-context-web.xml |

●コピー先 ＜ファイルコピー＋ペースト後、リファクタリングでリネームします＞

|  |
| --- |
| /src/test/resources/test-context-web.xml |

３）テスト用Spring設定ファイル（test-context-biz.xml）を編集してAOPを無効にする。

|  |
| --- |
| /src/test/resources/test-context-biz.xml |
| ● 17行目付近 |
| <!-- Enables the Spring AOP -->  <!-- AOPを無効化するためコメントアウト  <aop:aspectj-autoproxy />  --> |

# 各クラスを個別にテストする

**STEP09-1では、作成した各クラスを個別にテストする方法を学習します。**

例としてＤＡＯクラスのテストケースを作成してみる。



## 書籍一覧Ｄａｏクラスのテストケース（正常系）新規作成

１）書籍一覧Ｄａｏクラスのテストクラス（ListBookDaoSpringJdbcTest.java）新規作成

|  |
| --- |
| /src/**test**/java で右クリック → New → Class（もしくは jUnitTestCase）    ● Java Class 画面  ---------------  Package: 「jp.sample.bookmgr.biz.dao」  Name: 「ListBookDaoSpringJdbcTest」 と入力して Finish  ---------------  クラスのひな形が自動作成される |

２）書籍一覧Ｄａｏクラスのテストクラス（ListBookDaoSpringJdbcTest.java）コーディング

|  |
| --- |
| /src/test/java/jp.sample.bookmgr.biz.dao.ListBookDaoSpringJdbcTest.java （新規作成） |
| package jp.sample.bookmgr.biz.dao;  import static org.junit.Assert.\*;  import static org.mockito.Matchers.\*;  import static org.mockito.Mockito.\*;  import java.util.ArrayList;  import java.util.List;  import org.hamcrest.CoreMatchers;  import org.junit.Before;  import org.junit.Test;  import org.junit.runner.RunWith;  import org.mockito.InjectMocks;  import org.mockito.Mock;  import org.mockito.MockitoAnnotations;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.jdbc.core.JdbcTemplate;  import org.springframework.jdbc.core.RowMapper;  import org.springframework.test.context.ContextConfiguration;  import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;  import jp.sample.bookmgr.biz.domain.Book;  /\*\*  \* 書籍一覧データアクセス実装クラステストケース  \*  \* @author 長住@NTT-AT  \* @version 1.0  \*/  // SpringによるJunitテストランナー  @RunWith(SpringJUnit4ClassRunner.class)  // Spring定義ファイルの取り込み  @ContextConfiguration("classpath:test-context-biz.xml")  public class ListBookDaoSpringJdbcTest {  /\*\*  \* JdbcTemplate オブジェクトのモック  \*/  @Mock(name="template")  private JdbcTemplate mockJdbc;    /\*\*  \* テスト対象となる ListBookDao クラス  \*/  @InjectMocks  @Autowired  private ListBookDao listBookDao;  /\*\*  \* モック化のおまじない  \*/  @Before  public void setUp() {  MockitoAnnotations.initMocks(this);  }  /\*\*  \* ListBookDao.getBookList() のテストケース 正常系その１  \* 取得データが正しく返却されているかを検証  \*/  @SuppressWarnings("unchecked")  @Test  public void testGetBookList\_normal\_1() {  List<Book> expected = new ArrayList<Book>();  Book book1 = new Book();  book1.setId(1);  book1.setIsbn("978-4-7741-5380-3");  book1.setName("Spring3入門");  book1.setPrice(3900);  expected.add(book1);  String sql = "SELECT id, isbn, name, price FROM book ORDER BY id";  // JdbcTemplate#queryの戻り値をMock動作に差替え  when(mockJdbc.query(eq(sql), (RowMapper<Book>)anyObject())).thenReturn(expected);    List<Book> actual = listBookDao.getBookList();  assertThat(actual, CoreMatchers.is(expected));  }  } |

## 書籍一覧Ｄａｏクラスのテストケース（正常系）実行

|  |
| --- |
| ① クラス単位でテストケース実行する方法。  テストケースのクラスを選択して右クリック → Run as → JUnit Test  ② テストケース実行結果確認  EclipseのJUnitビューが表示され、テストを実行。  テスト実行が完了すると、その結果が表示される。  テストコードにエラーがある場合や、テスト結果にバグがある場合は、アイコンで識別できる。  また、エラーをダブルクリックすると、該当テストケースのコードが表示される。 |

テストケースを実行し、結果を確認する。

## 書籍一覧Ｄａｏクラスのテストケース（異常系）追加

１）書籍一覧Ｄａｏクラスのテストケース（異常系の例）コード追加

|  |
| --- |
| /src/test/java/jp.sample.bookmgr.biz.dao.ListBookDaoSpringJdbcTest.java |
| ● 83行目付近 |
| 83  /\*\*  \* ListBookDao.getBookList() のテストケース 異常系その１  \* 例外発生時の挙動確認  \*/  @SuppressWarnings({ "unchecked", "serial" })  @Test(expected=DataAccessException.class)  public void testGetBookList\_abnormal\_1() {  // ListBookDao#getList  when(mockJdbc.query(anyString(), any(RowMapper.class))).thenThrow(new DataAccessException(""){});  listBookDao.getBookList();  } |

## 書籍一覧Ｄａｏクラスのテストケース（正常系＋異常系）実行

テストケース（ListBookDaoSpringJdbcTest.java）をjUnitで実行し、結果を確認する。

## 書籍登録Ｄａｏクラスのテストケース（正常系＋異常系）を新規作成

１）書籍登録Ｄａｏクラスのテストケース作成＋コーディング

|  |
| --- |
| /src/test/java/jp.sample.bookmgr.biz.dao.AddBookDaoSpringJdbcTest.java （新規作成） |
| package jp.sample.bookmgr.biz.dao;  import static org.junit.Assert.\*;  import static org.mockito.Matchers.\*;  import static org.mockito.Mockito.\*;  import org.hamcrest.CoreMatchers;  import org.junit.Before;  import org.junit.Test;  import org.junit.runner.RunWith;  import org.mockito.InjectMocks;  import org.mockito.Mock;  import org.mockito.MockitoAnnotations;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.dao.DataAccessException;  import org.springframework.jdbc.core.namedparam.MapSqlParameterSource;  import org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate;  import org.springframework.test.context.ContextConfiguration;  import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;  import jp.sample.bookmgr.biz.domain.Book;  /\*\*  \* 書籍登録データアクセス実装クラステストケース  \*  \* @author 長住@NTT-AT  \* @version 1.0  \*/  // SpringによるJunitテストランナー  @RunWith(SpringJUnit4ClassRunner.class)  //Spring定義ファイルの取り込み  @ContextConfiguration("classpath:test-context-biz.xml")  public class AddBookDaoSpringJdbcTest {  /\*\*  \* NamedParameterJdbcTemplate オブジェクトのモック  \*/  @Mock(name="template")  private NamedParameterJdbcTemplate mockJdbc;    /\*\*  \* テスト対象となる AddBookDao クラス  \*/  @InjectMocks  @Autowired  private AddBookDao addBookDao;    /\*\*  \* モック化のおまじない  \*/  @Before  public void setUp() {  MockitoAnnotations.initMocks(this);  }    /\*\*  \* addBookDao.addBook() のテストケース 正常系その１  \* NamedParameterJdbcTemplate の update メソッドリターン値による挙動確認  \*/  @Test  public void testAddBook\_normal\_1() {  Book book1 = new Book();  book1.setIsbn("978-4-7741-5380-3");  book1.setName("Spring3入門");  book1.setPrice(3900);  MapSqlParameterSource pmap = new MapSqlParameterSource();  pmap.addValue("ISBN", book1.getIsbn());  pmap.addValue("NAME", book1.getName());  pmap.addValue("PRICE", book1.getPrice());    String sql = "INSERT INTO book (isbn, name, price) VALUES (:ISBN, :NAME, :PRICE)";  when(mockJdbc.update(eq(sql), refEq(pmap))).thenReturn(1);  assertThat(addBookDao.addBook(book1), CoreMatchers.is(1));  }  /\*\*  \* addBookDao.addBook() のテストケース 異常系その１  \* 例外発生時の挙動確認  \*/  @SuppressWarnings("serial")  @Test(expected=DataAccessException.class)  public void testAddBook\_abnormal\_1() {  Book book1 = new Book();  book1.setId(1);  book1.setIsbn("978-4-7741-5380-3");  book1.setName("Spring3入門");  book1.setPrice(3900);  // AddBookDao#addBook  when(mockJdbc.update(anyString(), (MapSqlParameterSource)anyObject())).thenThrow(new DataAccessException(""){});  addBookDao.addBook(book1);  }  } |

## 書籍登録Ｄａｏクラスのテストケース（正常系＋異常系）実行

テストケース（AddBookDaoSpringJdbcTest.java）をjUnitで実行し、結果を確認する。

## 書籍登録Ｄａｏクラスのテストケース（正常系別パターン）を追加

１）書籍登録Ｄａｏクラスのテストケース追加

|  |
| --- |
| /src/test/java/jp.sample.bookmgr.biz.dao.AddBookDaoSpringJdbcTest.java |
| ● 80行目付近 （testAddBook\_normal\_1() と testAddBook\_abnormal\_1() の間に挿入） |
| 80  /\*\*  \* addBookDao.addBook() のテストケース 正常系その２  \* パラメータが正しく NamedParameterJdbcTemplate の update メソッドにわたっているか？  \*/  @Test  public void testAddBook\_normal\_2() {  Book book1 = new Book();  book1.setId(1);  book1.setIsbn("978-4-7741-5380-3");  book1.setName("Spring3入門");  book1.setPrice(3900);  MapSqlParameterSource pmap = new MapSqlParameterSource();  pmap.addValue("ISBN", book1.getIsbn());  pmap.addValue("NAME", book1.getName());  pmap.addValue("PRICE", book1.getPrice());  String sql = "INSERT INTO book (isbn, name, price) VALUES (:ISBN, :NAME, :PRICE)";  addBookDao.addBook(book1);  verify(mockJdbc, times(1)).update(eq(sql), refEq(pmap));  }    /\*\*  \* addBookDao.addBook() のテストケース 正常系その３  \* パラメータが正しく NamedParameterJdbcTemplate の update メソッドにわたっているか？  \*/  @Test  public void testAddBook\_normal\_3() {  Book book1 = new Book();  book1.setIsbn("978-4-7741-5380-3");  book1.setName("Spring3入門");  book1.setPrice(3900);  MapSqlParameterSource pmap = new MapSqlParameterSource();  pmap.addValue("ISBN", "978-4-7741-5380-4");  pmap.addValue("NAME", "Spring4入門");  pmap.addValue("PRICE", 4900);    String sql = "INSERT INTO book (isbn, name, price) VALUES (:ISBN, :NAME, :PRICE)";  addBookDao.addBook(book1);  verify(mockJdbc, never()).update(eq(sql), refEq(pmap));  } |

## 書籍登録Ｄａｏクラスのテストケース（正常系＋異常系）実行

テストケース（AddBookDaoSpringJdbcTest.java）をjUnitで実行し、結果を確認する。

## プロジェクト全体のテストケースを一気に実行

|  |
| --- |
| ① プロジェクト全体のテストケースを一気に実行する。  Package Explorerでプロジェクト「bookMgr」を選択して右クリック → Run as → JUnit Test  ② テストケース実行結果確認  EclipseのJUnitビューが表示され、テストを実行。  テスト実行が完了すると、その結果が表示される。  テストコードにエラーがある場合や、テスト結果にバグがある場合は、アイコンで識別できる。  また、エラーをダブルクリックすると、該当テストケースのコードが表示される。 |