

Курс: Spring Framework

Сборник упражнений
(версия 2.2, March, 2012)

Учебный курс: Spring Framework

Авторы курса:	Евгений Кривошеев (EKrivosheev@luxoft.com) Дарья Линник (DLinnik@luxoft.com) Максим Потанин (MPotantin@luxoft.com) Вячеслав Яковенко (VYakovenko@luxoft.com)
---------------	--

Версия продукта:	Spring 3.x
---------------------	------------

Спецификация упражнений

Свойства виртуальной машины

<i>IMAGE_SOURCE_LOCATION</i>	spring31.ova
<i>VM_NAME</i>	jva-spring
<i>GUEST_OS_TYPE</i>	Linux/Ubuntu
<i>GUEST_OS_RAM</i>	>1Gb
<i>GUEST_LOGIN</i>	user
<i>GUEST_PASS</i>	user

Свойства упражнений в рамках виртуальной машины

<i>ECLIPSE_HOME</i>	~/bin/eclipse
<i>LABS_WORKSPACE</i>	~/Documents/labs
<i>SOLUTIONS_WORKSPACE</i>	~/Documents/solutions
<i>TOMCAT_HOME</i>	~/bin/tomcat
<i>SPRING_HOME</i>	~/bin/spring-framework-2.5.5
<i>SPRING3_HOME</i>	~/bin/spring-framework-3.1.1.RELEASE
<i>SPRING3_HOME_SLINK</i>	~/bin/spring
<i>HSQL_HOME</i>	~/bin/hsqldb/lib/hsqldb.jar

Упражнение 1.

Установка и настройка рабочей среды.

Длительность

0,5 часа

Цели упражнения.

Установить и настроить рабочую среду.

Описание.

Рабочая среда реализована с использованием концепции *виртуальной машины*. Виртуальная машина – это эмуляция полноценного ПК (машины) в рамках уже установленной ОС. Наиболее популярные программные продукты, реализующие описанные виртуальные машины – это Microsoft VirtualPC, VMWare (Server и Player) и Oracle Virtual Box. В данной рабочей среде используется Oracle Virtual Box.

На виртуальной машине, как и на обычной реальной, должна быть установлена ОС. ОС из-под которой запускается виртуальная машина, называется *хостовой*, а ОС, установленная на виртуальной машине, называется *гостевой*. В рамках хостовой системы можно запускать множество гостевых систем, количество одновременно запущенных гостевых ОС ограничивается только ресурсами хостовой системы.

Виртуальная машина, как и обычная, использует жесткий диск (HDD) для хранения данных и установленного ПО. Но HDD виртуальной машины организован как *образ*, т.е. для хостовой системы это один файл. В случае Oracle VirtualBox это файл с расширением ova (vdi).

Этот образ можно настроить – установить ОС, установить нужное ПО, скопировать необходимые данные. После этого образ HDD можно будет просто скопировать на необходимые хостовые системы и настроить там виртуальные машины на использование этого образа.

В данном курсе рабочая среда реализована, как образ HDD для продукта Oracle VirtualBox. Для установки рабочей среды следует запустить Oracle VirtualBox и создать новую виртуальную машину. Эта виртуальная машина – набор настроек, и машин может быть определено множество. В эти настройки входит имя машины, размер выделяемой ей памяти и месторасположение образа HDD.

После настройки новой виртуальной машины её можно запустить, при этом её окно будет обычным окном хостовой системы. Для переключения в полноэкранный режим и обратно следует нажать комбинацию клавиш RightCtrl+F.

Если просто свернуть окно виртуальной машины, то она все равно продолжит функционировать. Для того, чтобы она не занимала процессорное время, когда она не нужна, следует нажать комбинацию клавиш RightCtrl+P.

Для остановки виртуальной машины можно воспользоваться следующими вариантами:

- Завершить работу гостевой ОС (машина корректно остановится);
- Жесткий останов (эмуляция отключения питания);

- Останов с сохранением состояния (при следующем запуске состояние восстановится).

Задачи упражнения.

1. Скопировать образ виртуального диска на локальную машину, если необходимо*.
2. Распаковать его, если он в архиве*.
3. Настроить новую виртуальную машину с использованием данного виртуального диска.
4. Запустить виртуальную машину.
5. Заморозить её.

Подробное руководство.

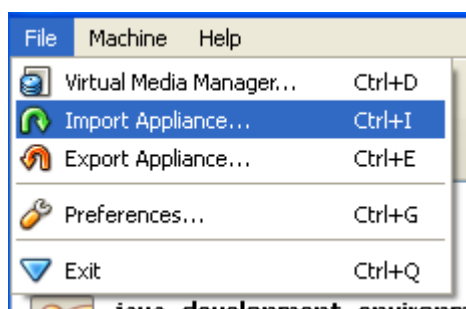
Часть 1. Импорт виртуальной машины.

Необходимое ПО:

Oracle VirtualBox 4.1.* и выше (www.virtualbox.org)

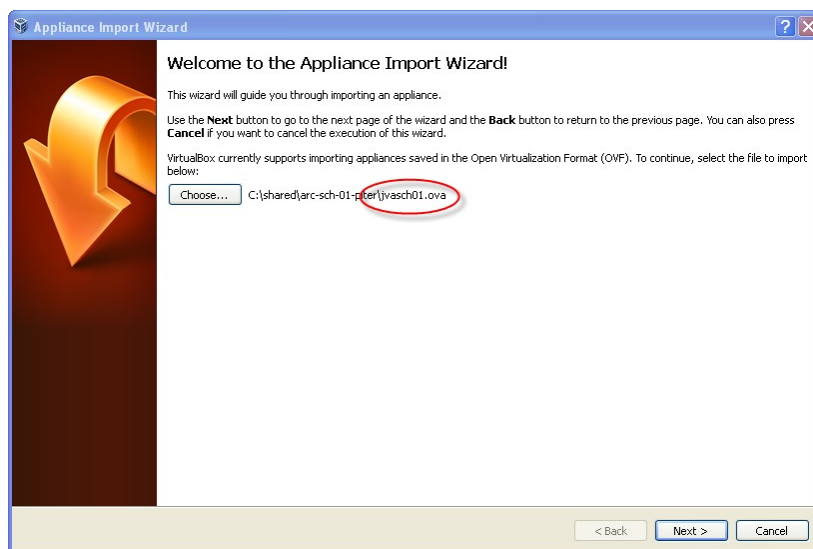
Установка:

1. Запустить VirtualBox
2. Из меню {File} , выбрать пункт {Import Appliance...};



* Необязательная часть

3. Кликнув на кнопку [Choose...], выбрать соответствующий образ и кликнуть на кнопку Next>];

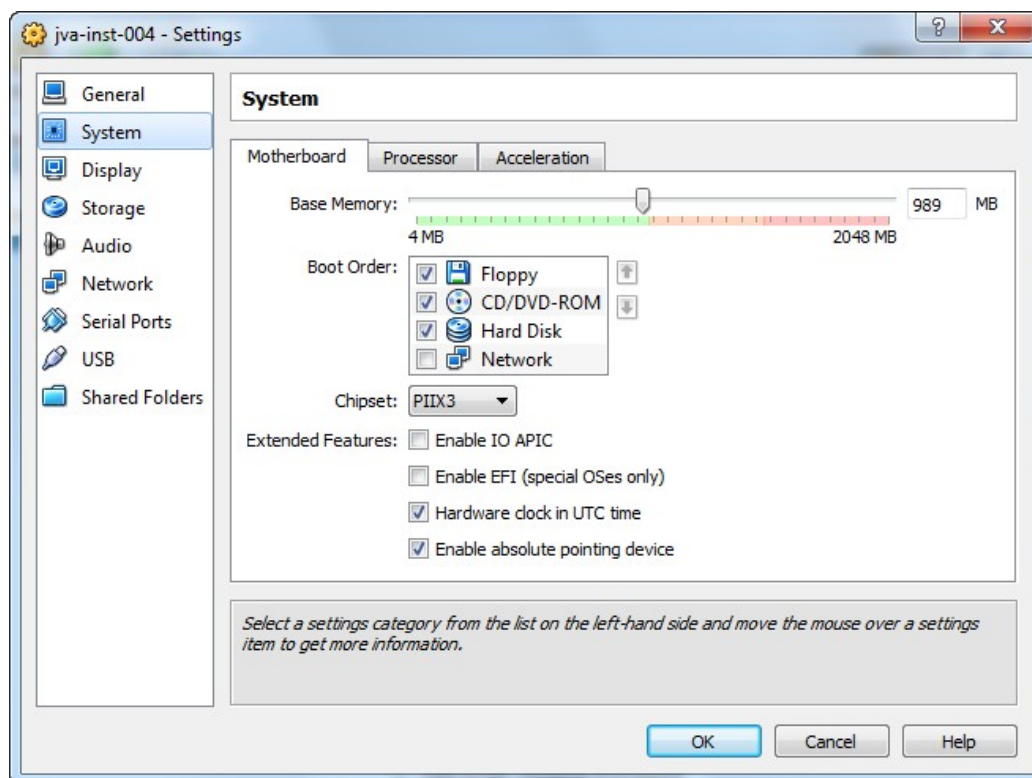


4. Указать название виртуальной машины, которое будет использоваться менеджером VirtualBox (по умолчанию, “java-spring”) и кликнуть на кнопке [Import];

Запустить виртуальную машину и убедиться в ее работоспособности.

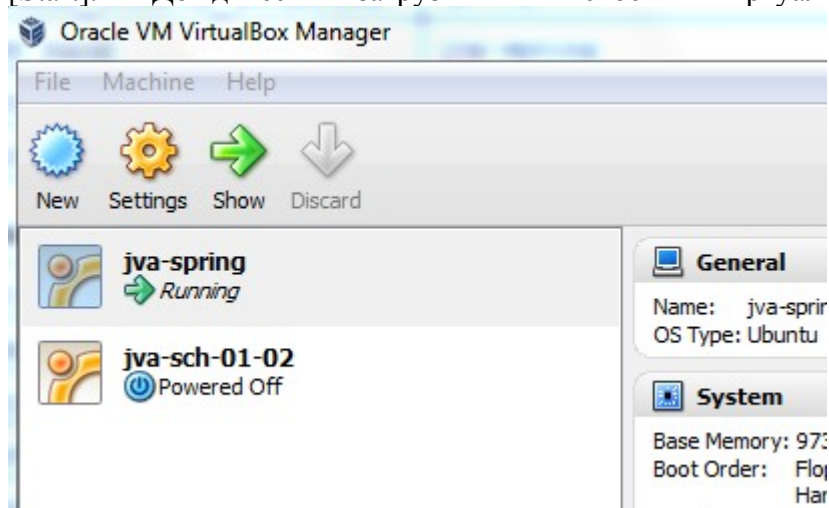
Часть 2. Настройка параметров виртуальной машины

Зайдите в раздел System (не запуская виртуальную машину) и убедитесь в том, что ползунок «Base Memory» находится в максимальном положении «зеленой» зоны. Желательно чтобы вы могли выделить под гостевую OS не менее 1Гб оперативной памяти. Кликните на кнопку [Ок].



Часть 3. Запуск виртуальной машины

1. Запустите Oracle VM VirtualBox Manager. В левой части окна выберите необходимую вам виртуальную машину и кликните мышкой на кнопку [Start]. Дождитесь загрузки своей виртуальной машины.



Упражнение 2.

Настройка вспомогательных сервисов.

Длительность

20 мин

Цели упражнения.

Ознакомиться со вспомогательными средствами разработки.

Описание.

В качестве среды разработки используется Eclipse. Практические задания находятся в окружении LABS_WORKSPACE. Решения находятся в окружении SOLUTIONS_WORKSPACE.

В качестве СУБД используется HSQL. Для демонстрации работы с данными через Spring необходима уже созданная база данных. Все действия будут производиться в ней.

В качестве контейнера используется Apache Tomcat.

Задачи упражнения.

1. Ознакомиться со средой разработки Eclipse.
2. Ознакомиться с СУБД HSQL.

Подробное руководство.

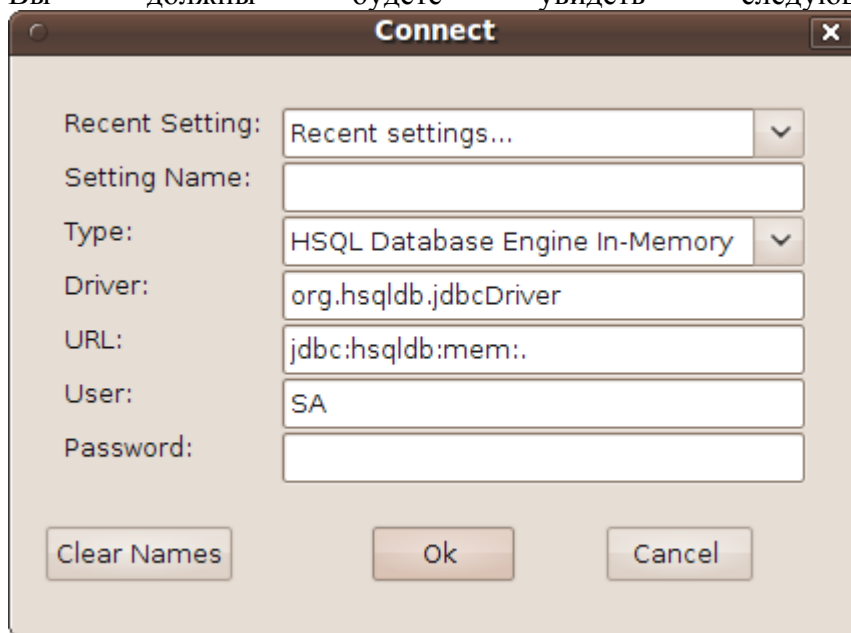
Часть 1. Среда разработки eclipse.

1. Запустите файл ECLIPSE_HOME/eclipse
2. Ознакомьтесь со структурой заготовочных проектов и решений, переключаясь между окружениями (File -> Switch Workspace).
3. Проверьте переключение между Java EE и Java перспективами (Window -> Open Perspective).

Часть 2. СУБД HSQL.

4. Запустите текстовый клиент `java -cp ~/bin/hsqldb/lib/hsqldb.jar org.hsqldb.util.DatabaseManagerSwing`

5. Вы должны будете увидеть следующее окно:



запомните

пожалуйста эти параметры - они понадобятся нам в дальнейшем для конфигурирования DataSource. Так же обратите внимание на [url:jdbc:hsqldb:mem:.](#) Этот url указывает на то, что временная [...] база данных будет создана в оперативной памяти, которая будет уничтожена после завершения работы, что очень удобно использовать в целях тестирования.

Упражнение 3. Приложение HelloWorld.

Проект

lab-3-core

Длительность

Самостоятельная работа: 20..25 мин

Обсуждение: 5 мин

Цели упражнения.

Ознакомиться с использованием Spring IoC на примере простейшего приложения. Понять простейшие приемы внедрения зависимостей и создания контекста приложения.

Ознакомиться с приложением, основными сущностями и связями между ними.

Описание.

Spring предоставляет возможность декларативного внедрения зависимостей между классами и создания общего контекста приложения. Spring позволяет объявлять в качестве бинов обычные java-классы (POJO).

Задачи упражнения.

1. Разобраться с основными принципами описания контекста приложения в xml-файле.
2. Разобраться с внедрением зависимостей и установкой значений полей бина.
3. Проанализировать, каким образом создается контекст приложения из xml-файла.
4. Проанализировать уже созданные сущности и связи между ними.

Подробное руководство.

Часть 1. Основные принципы описания контекста в xml-файле.

1. Откройте проект lab-3-core;
2. Откройте файл {lab-3-core}/resources/application-context.xml.
3. Проанализируйте структуру xml-файла.

Часть 2. Внедрение зависимостей. Установка значений полей.

4. Обратите внимание на описание бинов person и country.
5. *Каким образом задаются поля бина person? Сравните описание бина и класс Person.*
6. *Как связаны бины person и country?*

7. Откройте тест `HelloWorldTest`. Сравните получение эталонного объекта (метод `getExpectedPerson()`) и его описание в `xml`-файле.

Часть 3. Получение контекста из `xml`-файла.

8. Откройте класс `HelloWorldTest`, метод `setUp()`.
9. *Каким образом получается контекст из `xml`-файла? Где должен находиться этот файл?*
10. Откройте тест `HelloWorldTest` и запустите его, с помощью фреймворка тестирования `jUnit 4.*`.

Часть 4. Анализ сущностей.

11. Откройте `package model`. *Какие классы и интерфейсы там описаны? Какие между ними связи?*
12. Изучите другие `package`' и приложения. *Какие сущности для чего могут использоваться?*

Часть 5. Анализ библиотек подключенных к проекту.

13. Зайдите в директорию `lib`. *Какие библиотеки там находятся?*
14. *К каким продуктам они относятся?*
15. *Почему не все библиотеки из `SPRING3_HOME/dist` попали в этот проект?*

Упражнение 4.

Разработка простейшего приложения.

Проект

lab-4-core

Длительность

30 мин

Цели упражнения.

- Научиться создавать простейшее приложение с использованием Spring.
- Научиться использовать на практике основные приемы внедрения зависимостей и связывания бинов при помощи Spring IoC контейнера.
- Научиться описывать и создавать контекст приложения.
- Научиться использовать автоматическое связывание.

Описание.

Spring предоставляет возможность декларативного внедрения зависимостей между классами и создания общего контекста приложения. Spring позволяет объявлять в качестве бинов обычные java-классы (POJO). Существует возможность автоматического связывания бинов, например, по типу или по имени.

Задачи упражнения.

1. Описать и создать контекст приложения для существующих модельных классов UsualPerson, Country и Contact.
2. Задать все значения полей всех классов так, чтобы тест SimpleAppTest успешно выполнялся.
3. Установить все необходимые зависимости любым подходящим способом, также в соответствии в тестом SimpleAppTest.
4. Создать контекст приложения.
5. Реализовать ситуацию: Человек, который проживает в конкретной стране и у него есть несколько контактов. Задать все начальные значения для всех параметров и вывести всю информацию об этом человеке. Для проверки использовать тест SimpleAppTest.

Подробное руководство.

Часть 1. Создание контекста приложения.

1. Откройте проект lab-4-core;
2. Откройте файл {lab-4-core}/resources/application-context.xml. В нем уже есть начальное описание бинов country и person.

3. Откройте тест SimpleAppTest, метод getExpectedPerson(). *Каких описаний не хватает в существующем xml-файле для соответствия эталонному классу?*

Часть 2. Задание значений полей.

4. Задайте поля height и isProgrammer для класса UsualPerson.
5. Это делается в теге <property name=... value=... > внутри тега <bean>. Простые property (String, int, boolean итп) задаются значениями (value=), их имена задаются атрибутом name=.

```
<property name="height" value="1.78"/>
<property name="isProgrammer" value="true"/>
```

6. Задайте список контактов Contacts. Для этого в теге property с name=contacts создается тег list, а внутри него теги value, в каждом из которых задается один контакт.

```
<property name="contacts">
    <list>
        <value>asd@asd.ru</value>
        <value>+7-234-456-67-89</value>
    </list>
</property>
```

Часть 3. Внедрение зависимостей.

7. Кроме простых property необходимо также задать ссылки на другие классы. *Какие есть способы это сделать?*
8. Один из способов связать бины между собой – указать явно ссылку на нужный бин <property name=... ref=...>. *Каковы достоинства и недостатки этого подхода?*
9. Другой способ связывания – автоматически (autowire, не забудьте добавить <context:annotation-config/>) для бина, например, по типу или по имени. *Каковы достоинства и недостатки этого подхода?*

Часть 4. Создание контекста приложения.

10. Откройте метод @Before setUp().
11. В нем есть переменная типа AbstractApplicationContext. Ей присваивается значение ClassPathXmlApplicationContext, параметром которого является путь на диске к файлу с описанием контекста.
12. Чем эта декларация отличается от декларации в предыдущем примере?

Часть 5. Реализация тестовой ситуации.

13. Откройте тест SimpleAppTest, метод testInitPerson().

14. Получить бин, отвечающий за класс UsualPerson из контекста. `context.getBean(имя бина)`. Обратите внимание, что этот метод вернет объект типа `Object`. Его необходимо привести к типу `UsualPerson`.
15. Обратите внимание на альтернативные возможности получения бинов из контекста. Для этого зайдите в документацию к классу `AbstractApplicationContext`: `{SPRING3-HOME}/docs/javadoc-api/index.html` и изучите его API.
16. Вызовите метод `toString()` у объекта у только что полученного бина.
17. Выполните тест. Если все сделано правильно, то тест выполнится без ошибок, а в консоль выведется вся информация о человеке.

Часть 6. Использование Spring TestContext Framework

18. Откройте класс `SpringTCFAppTest`;
19. Сравните его с предыдущим тестом (`SimpleAppTest`). В чем принципиальное отличие этих классов?
20. К какой библиотеке принадлежит аннотация `@RunWith`?
21. Изучите документацию к аннотации `@ContextConfiguration` (см. 10.3.5.2 Context management, Spring Reference v.3.1.1). Какие еще варианты, указания контекста приложения, она поддерживает?

Упражнение 5.

Использование Spring AOP. @AspectJ style.

Проект

lab-5-aop

Длительность

40 мин

Цели упражнения.

Научиться использовать Spring AOP с применением подхода @AspectJ style на примере Before, After и Around advice'ов.

Описание.

Существует два основных подхода использования AOP в Spring: @AspectJ style и Schema-based. Подход @AspectJ может быть использован начиная с JDK 5. Также при использовании этого подхода все описания делаются аннотациями, т.е. находятся внутри кода. В этом случае, нет возможности использовать в качестве advice'ов или pointcut'ов классы из сторонних библиотек.

Задачи упражнения.

1. На примере Before advice и After returning advice (класс Politeness) создайте остальные типы After advice'ов для метода sellSquishee(...) из класса ApeBar с использованием подхода @AspectJ. Проверьте правильность выполнения при помощи теста AopAspectJTest.
2. Создать Around advice для метода sellSquishee(...) с использованием подхода @AspectJ. Проверьте правильность выполнения при помощи теста AopAspectJTest.

Подробное руководство.

Часть 1. Создание after advice'ов.

1. Откройте проект lab-5-aop;
2. Откройте файл с конфигурацией application-context.xml. Добавить строчку <aop:aspectj-autoproxy/>. Это указание на то, что все описания аспектов будут находиться в декларациях.
3. Откройте класс Politeness. *Что значат параметры в аннотации методов sayHello и askOpinion? Как в методе askOpinion получено название проданного напитка?*
4. Создайте advice на базе метода sayGoodBye, который выполнится, после попытки продать напиток в любом случае. (After). Для этого необходимо добавить строчку
`@After("execution(* sellSquishee(..))")`
перед объявлением метода.

5. Запустите тест `AopAspectJTest` для разных конфигураций `Customer` (есть у него деньги или нет – параметр `broke`). В каком случае какой из тестов `testAfterReturningAdvice` выполнится правильно?

Часть 2. Создание `around advice`'а.

6. Создайте `Around advice` для метода `sellSquishee` таким образом, чтоб в нем был вызов `target`-метода. За основу возьмите метод `sayPoliteWordsAndSell`. Для этого необходимо добавить строчку `@Around("execution(* sellSquishee(..))")` перед объявлением метода.
7. Для проверки запустите тест `testAroundAdvice`.
8. Можно ли в `around advice`'е не вызывать `target`-метод?

Часть 3. Создание `after throwing advice`'ов.

9. Создайте `advice` на базе метода `sayYouAreNotAllowed`, который выполнится, если у клиента нет денег, т.е в случае, когда выбросится исключение `CustomerBrokenException` (After throwing). Для этого необходимо добавить строчку `@AfterThrowing("execution(* sellSquishee(..))")` перед объявлением метода;
10. Откройте `application-context.xml`, модифицируйте его так, чтобы у Клиента “не было денег”;
11. Откройте класс `AopAspectJExceptionTest` и выполните юнит-тест;

Упражнение 6

Использование JDBC в Spring при работе с данными.

Проект

lab-6-jdbc

Длительность

1 час

Цели упражнения.

Научиться использовать JDBC совместно со Spring. Научиться выполнять основные операции (SELECT, INSERT, UPDATE) JDBC через Spring.

Описание.

Spring предоставляет вспомогательное API для работы с JDBC.

Для работы с JDBC через Spring необходимо настроить DataSource. Это делается в application-context.xml.

В качестве базы данных используется HSQLDB.

Задачи упражнения.

1. Получить и напечатать полный список всех стран, используя CountryDao. *Все ли поля класса Country корректно заполняются?* Для проверки использовать тест JdbcTest, метод testCountryList.
2. Исправить CountryRowMapper так, чтобы заполнялись все поля Country.
3. Получить и напечатать полный список стран, названия которых начинаются с буквы А. Для проверки использовать тест JdbcTest, метод testCountryListStartsWithA.
4. Изменить название какой-либо страны в базе. Для проверки использовать тест JdbcTest, метод testCountryChange (необходимо изменить эталонную страну в этом тесте).

Подробное руководство.

Часть 1. Получение полного списка стран.

1. Откройте проект lab-6-jdbc;
2. Откройте директорию {lab-6-jdbc}/lib. *Какие библиотеки добавились в этом проекте? Почему?*
3. Откройте файл с конфигурацией application-countext.xml. Проанализируйте конфигурации всех бинов.
4. Откройте класс CountryDao. Реализуйте в нем метод getCountryList(), так чтобы он возвращал полный список всех стран. В качестве примера используйте уже существующие методы.

5. Для этого необходимо получить `JdbcTemplate` и у него вызвать метод `query(String s, RowMapper rowMapper)`
6. В качестве `sql`-запроса можно использовать `GET_ALL_COUNTRIES_SQL`.
7. В качестве `RowMapper`'а – `COUNTRY_ROW_MAPPER`.
8. Откройте тест `JdbcTest`, запустите на выполнение метод `testCountryList`.
9. *Правильно ли выполнен тест? Почему?*

Часть2. Исправление RowMapper'а

10. Откройте класс `CountryRowMapper`. *Что необходимо сделать, чтобы заполнялись все поля Country?*
11. В метод `mapRow` необходимо задать `codeName` для `Country`, по аналогии с `id` и `name`.
12. Выполните тест `JdbcTest`, метод `testCountryList` еще раз.

Часть 3 Получение списка стран, начинающихся с буквы А.

13. Откройте класс `CountryDao`, метод `getCountryListStartWith()`. *Что такое NamedParameterJdbcTemplate? Чем он отличается от JdbcTemplate?*
14. Откройте тест `JdbcTest`, метод `testCountryListStartsWithA`. *Правильно ли выполняется этот тест?*
15. Попробуйте теперь закомментировать аннотацию `@DirtiesContext` к некоторым тестам (в разных комбинациях). *Как теперь выполняются тесты? Почему?*

Часть 4 Изменение названия страны.

16. Откройте класс `CountryDao`.
17. Реализуйте метод для изменения названия страны по известному `codeName` – `updateCountryName`. В качестве `sql`-запроса можно использовать `UPDATE_COUNTRY_NAME_SQL_1`, `UPDATE_COUNTRY_NAME_SQL_2`.

Должно получиться:

```
getJdbcTemplate().execute(  
UPDATE_COUNTRY_NAME_SQL_1 + newCountryName + " " +  
UPDATE_COUNTRY_NAME_SQL_2 + codeName + " ");
```

18. Для проверки в тесте `JdbcTest`, метод `testCountryChange` создайте страну с новым названием и известным `codeName`.
19. Как можно изменить `application-context.xml` и `JdbcTest` так, чтобы не было необходимости вызывать `countryDao.loadCountries()`; в `@Before` методе? В чем преимущество этого подхода? (Дополнительная информация для ответа на вопрос: п.13.8 Embedded database support, Spring Reference);

Упражнение 7

Использование ORM в Spring при работе с данными.

Проект

lab-7-orm

Длительность

45 мин

Цели упражнения.

Научиться использовать ORM, на примере JPA (Hibernate v.4) при помощи Spring.

Описание.

Spring предоставляет поддержку различных ORM с применением одного и того же подхода. Использование ORM через Spring упрощает тестирование, обработку исключений, управление ресурсами. Spring поддерживает следующие ORM: JPA, Hibernate, JDO, iBATIS SQL Maps, и др...

В данном примере используется описание Entity-классов через аннотации, с использованием javax.persistence (JPA v.2.0 : JSR-317; JPA v.2.1 : JSR-338).

Задачи упражнения.

1. Создать `LocalContainerEntityManagerFactoryBean` и сконфигурировать на основе существующего `DataSource`. В качестве `@Entity` задать `lab.model.Country`.
2. Создать mapping для класса `Country`.
3. В классе `CountryJpaDaoImpl` реализовать метод сохранения страны, `save(Country country)`. Сохранить страну в базе. Проверить при помощи теста `CountryDaoImplTest`, `testSaveCountry()`.
4. В классе `CountryJpaDaoImpl` реализовать метод получения списка всех стран, метод `getAllCountries()`. Получить список всех стран. Проверить при помощи теста `CountryDaoImplTest`, `testGetAllCountries()`.
5. В классе `CountryJpaDaoImpl` реализовать метод получения страны по названию. Получить страну по названию. Проверить при помощи теста `CountryDaoImplTest`, `testGetCountryByName (String name)`.

Подробное руководство.

Часть 1. Создание и конфигурация `LocalContainerEntityManagerFactoryBean`.

1. Откройте файл с конфигурацией `application-context.xml`.

2. Создайте бин `lcmef`, в качестве класса укажите для него `LocalContainerEntityManagerFactoryBean`.
3. Задайте ему `dataSource`, `persistenceUnitName`, `persistenceProviderClass`.

У вас должна получиться следующая конфигурация:

```
<bean id="lcmef" class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
    <property name="loadTimeWeaver">
        <bean class="org.springframework.instrument.classloading.InstrumentationLoadTimeWeaver" />
    </property>
    <property name="dataSource" ref="dataSource"></property>
    <property name="persistenceUnitName" value="springframework.lab.orm.jpa" />
    <property name="persistenceProviderClass" value="org.hibernate.ejb.HibernatePersistence"/>
</bean>
```

Укажите в каких пакетах Spring Framework должен вести поиск компонент (`@Entity`, `@Repository`, etc.). Для этого добавьте в `application-context.xml` следующую строку:

```
<context:component-scan base-package="lab.model, lab.dao" />
```

Часть 2. Создание `mapping`'а класса `Country`.

4. Откройте класс `Country`. Добавьте аннотации `@Entity` и `@Table(name = "COUNTRY")` перед объявлением класса.
5. Добавьте аннотации `@Id` `@Column(nullable = false)` перед заданием метода `getId()`.
6. Добавьте аннотацию `@Column` перед заданием метода `getName()`.
7. Добавьте аннотацию `@Column(name="code_name")` перед описанием метода `getCodeName()`.
8. В итоге должно получиться код, содержащий следующие строки:

```
@Entity
@Table(name = "COUNTRY")
public class Country implements Serializable {
    private int id;
    private String name;
    private String codeName;

    public Country() { }

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    public int getId() {
        return id;
    }
}
```

```
    }  
    ...  
    public void setId(int id) {this.id = id;}  
  
    @Column(name = "NAME")  
    public String getName() {  
        return name;  
    }  
    ...  
    @Column(name = "CODE_NAME")  
    public String getCodeName() {  
        return codeName;  
    }  
    ...
```

Часть 3. Сохранение страны.

9. Откройте класс `CountryJpaDaoImpl`. Зачем этот класс наследуется от `AbstractJpaDao` и имплементирует интерфейс `CountryDao`?
10. Откуда и каким образом получается `EntityManagerFactory` (переменная `emf`)?
11. В методе `save` получите `EntityManager`, метод `emf.createEntityManager()`.
12. Вызовите у `entityManager` метод `persist()` в рамках программной транзакции, передав ему сохраняемый объект в качестве параметра.
13. Проверьте правильность реализации при помощи теста `CountryDaoImplTest, testSaveCountry()`.

Часть 4. Получение списка всех стран.

14. Откройте метод `getAllCountries()` в классе `CountryJpaDaoImpl`.
15. Получите `EntityManager`, вызовите у него метод `createQuery`, передав ему в качестве параметра строку запроса и `Country.class`.
16. Проверьте правильность реализации при помощи теста `CountryDaoImplTest, testGetAllCountries()`.

17. У вас должен получиться код, близкий к этому:

```
List<Country> countryList = null;
EntityManager em = emf.createEntityManager();
if (em != null) {
    countryList =
        em.createQuery("from Country", Country.class)
            .getResultList();
    em.close();
}
return countryList;
```

Часть 5. Получение страны по названию.

18. Откройте метод `getCountryByName(String name)` в классе `CountryJpaDaoImpl`.

19. Создайте запрос для поиска и передайте его в полученный `EntityManager` следующим образом:

```
Country country = em
    .createQuery("SELECT c FROM Country c WHERE c.name LIKE :name",
        Country.class).setParameter("name", name)
    .getSingleResult();
```

20. Проверьте правильность реализации при помощи теста `CountryDaoImpTest, testGetCountryByName()`.

21. В итоге, у вас должен получиться код, близкий к следующему:

```
EntityManager em = emf.createEntityManager();
Country country = null;
if (em != null) {
    country = em
        .createQuery("SELECT c FROM Country c WHERE c.name LIKE :name",
            Country.class).setParameter("name", name)
        .getSingleResult();
    em.close();
}
return country;
```

Дополнительные вопросы:

22. Какие еще стратегии получения первичного ключа (PK) возможны для сущностей? (`@GeneratedValue(strategy = GenerationType.AUTO)`)
23. Переведите корневой логгер в режим DEBUG (для этого отредактируйте файл `log4j.properties`)
24. Поэкспериментируйте с различными стратегиями получения PK. Какая между ними разница?

Упражнение 8

Управление транзакциями в Spring.

Проект

lab-8-tx

Длительность

30 мин

Цели упражнения.

Научиться использовать декларативное и программное управление транзакциями при помощи Spring.

Описание.

Spring предоставляет возможности для декларативного и программного управления транзакциями. Также в Spring есть возможность интеграции с абстракциями для доступа к данным через ORM или JDBC. В данном примере рассматривается управление транзакциями на примере JDBC.

Задачи упражнения.

1. Создать `TransactionManager`, основанный на `DataSource`. Объявить декларативную поддержку транзакций при помощи аннотаций.
2. Описать бин `countryService`. Сделать все методы класса `CountryServiceImpl` транзакционными. Задать разные значения `propagation` для методов, в соответствии с названиями методов (например, для метода `getAllCountriesRequired()` задать значение `Propagation.REQUIRED`).
3. Протестировать разные варианты `propagation` для методов `CountryService` при помощи теста `DeclarativeTransactionTest`. Сравнить поведение методов, вызванных внутри и вне транзакции для разных значений `propagation`.

Подробное руководство.

Часть 1. Создание `TransactionManager`'а.

1. Откройте файл с конфигурацией `application-context.xml`.
2. Добавьте в него следующую конфигурацию:

```
<context:annotation-config/>
<context:component-scan base-package="lab.service" />
```

Какую роль выполняет каждая из этих строк?

3. Создайте бин `transactionManager` для класса `org.springframework.jdbc.datasource.DataSourceTransactionManager`. Задайте ему ссылку на `DataSource`.

4. Добавьте строчку `<tx:annotation-driven />`, которая включает декларативную поддержку транзакций при помощи аннотаций.

Часть 2. Конфигурация `CountryServiceImpl`.

5. Создайте бин `countryService` для класса `CountryServiceImpl`. Задайте ему ссылку на `dao`.
6. Откройте класс `CountryServiceImpl`. Добавьте строчку `@Transactional` перед описанием класса. Это делает все методы этого класса транзакционными с настройками по умолчанию.
7. Чтобы изменить настройки для конкретных методов необходимо добавить описание

```
@Transactional(readOnly = false, propagation = Propagation.REQUIRED)
```

перед объявлением метода. Задайте соответствующие описания всем методам вида `getAllCountriesRequired()`, изменяя только значение `propagation` в соответствии с названием метода.

Часть 3. Тестирование декларативного описания транзакций.

8. *Каково должно быть поведение метода с `propagation REQUIRED`, вызванного внутри или вне транзакции? С `propagation MANDATORY`?*
9. Проверьте правильность при помощи теста `DeclarativeTransactionTest`.

Упражнение 9

Разработка простейшего приложения с использованием Spring 3 MVC .

Проект

lab-9-mvc

Длительность

1 час

Кодовая база

lab-9-mvc

Цели упражнения.

Создать минимальное работоспособное приложение с использованием каркаса Spring 3 MVC.

Описание.

Любое приложение, работающее на Spring MVC, должно иметь несколько обязательных элементов. В дескрипторе приложения необходимо зарегистрировать хотя бы один `DispatcherServlet` и определить обрабатываемые им запросы. Также нужно зарегистрировать «слушатель» типа `ContextLoaderListener` для инициализации контекста Spring.

Чтобы увидеть работу приложения, нужно создать контроллеры и представления. Для каждого контроллера нужно определить обрабатываемые им запросы. Для представлений нужно задать стратегию сопоставления строковых имён самим представлениям.

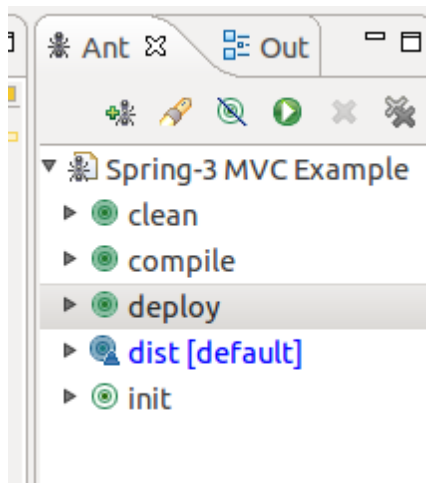
Задачи упражнения.

1. Изучить имеющееся приложение, выделить все имеющиеся в нем компоненты и изучить их.
2. Доработать имеющееся приложение таким образом, чтобы у приложения появилась возможность выполнить аналогичные операции для сущности Country (см. предыдущие упражнения).

Подробное руководство.

Часть 1

1. Убедиться в том, что системная переменная `$CATALINA_HOME` задана и указывает на директорию в которую предустановлен сервелет-контейнер TomCat. Для этого в командной строке ввести следующую команду: `echo $CATALINA_HOME`, ее вывод должен указывать на `~/bin/tomcat`
2. В Eclipse откройте окно управления сборкой приложения Window → Show view -> Ant



3. Выберите таржет «deploy» и запустите его на выполнение;
4. После успешной сборки приложения запустите сервлет-контейнер TomCat;
5. Откройте браузер и укажите в адресной строке следующий url:
<http://localhost:8080/lab-9-mvc/adduser.form>
6. Если все предыдущие шаги были выполнены вами правильно вы должны будете увидеть следующую форму:

Add User Form:

First Name:

Last Name:

7. Заполните поля формы и кликните на кнопку [Save Changes]
8. Если все было сделано правильно, вы должны будете увидеть следующий экран:

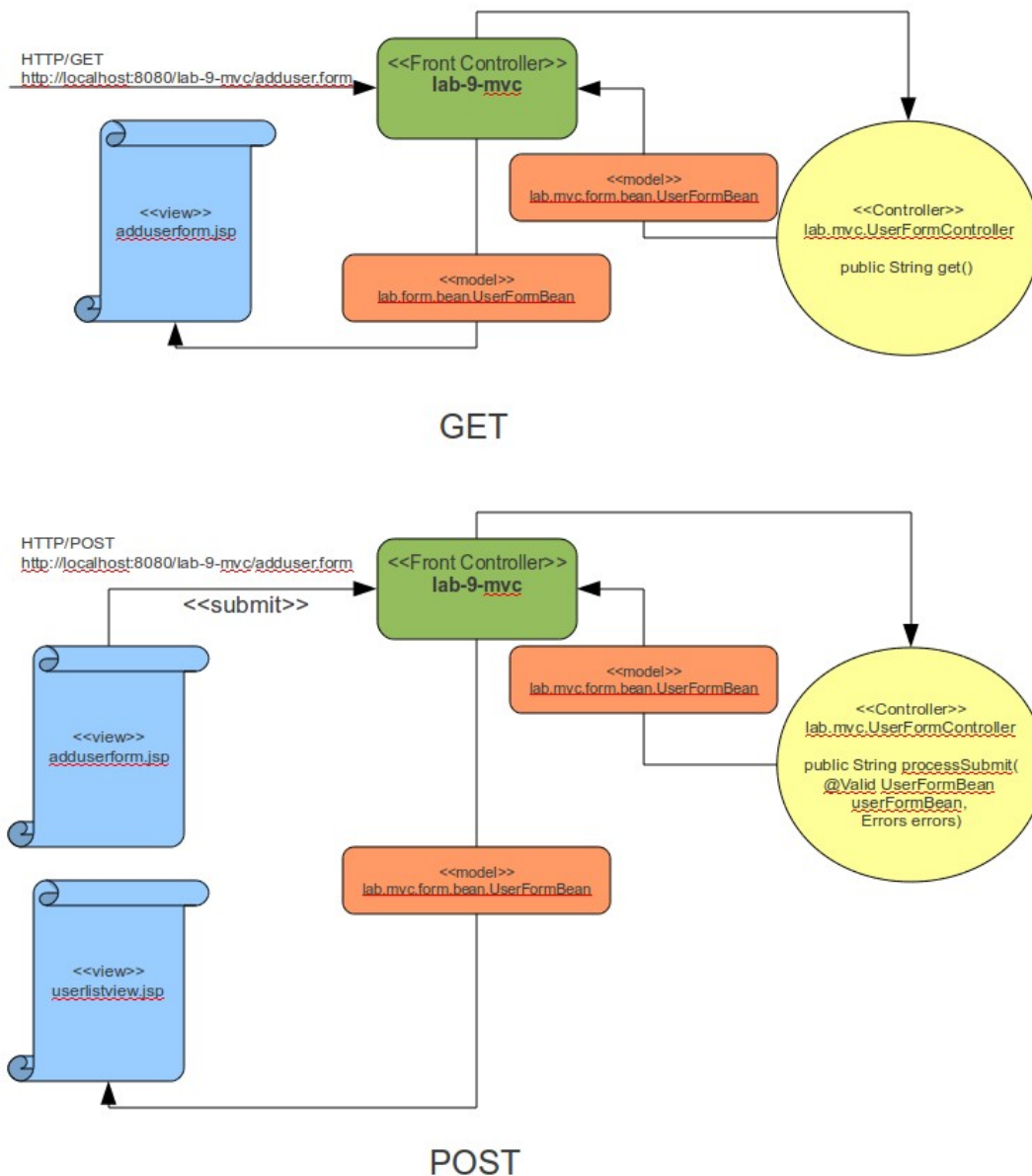
Users List:

Id	First Name	Last Name
0	Rod	Johnson

[Add User Form](#)

9. Повторить п.6,7 указав в качестве данных пустые поля и поля длиной менее 2-х символов. Какой результат вы получили?

10. Изучите внимательно все артефакты проекта и диаграмму зависимостей mvс-компонент:



Ответить на следующие вопросы:

1. В каком артефакте описаны все бины данного приложения?
2. На какой контроллер передается управление при обработке адреса <http://localhost:8080/lab-9-mvc/adduser.form> и почему?
3. Как в данный момент обрабатывается адрес <http://localhost:8080/lab-9-mvc/> и почему? Как можно изменить это поведение?
4. Почему если запрос приходит из браузера с использованием метода GET выполнение передается на `adduserform.jsp`, а при

использовании метода POST управление может вернуться как на `adduserform.jsp`, так и на `userlistview.jsp`?

5. Каким образом происходит проверка правильности заполнения полей формы?

Часть 2

Разработать следующие артефакты:

1. `lab.dao.CountryDao`
2. `lab.dao.HsqlCountryDao`
3. `lab.domain.Country`
4. `lab.mvc.form.bean.CountryFormBean`
5. `lab.mvc.CountryFormController`
6. `web/WEB-INF/views/addcountryform.jsp`
7. `web/WEB-INF/views/countrylistview.jsp`

Подробное руководство

1. Воспользовавшись кодовой базой предыдущей части упражнения и разработав вышеуказанные артефакты, выполнить сборку приложения.
2. Запустить сервлет-контейнер TomCat, указать в адресной строке браузера: <http://localhost:8080/lab-9-mvc/addcountry.form>
3. Убедиться в том, что приложение успешно сохраняет добавленную информацию о стране в базе данных.

Упражнение 10.

Использование планирования задач.

Проект

lab-10-quartz

Длительность

30 мин

Цели упражнения.

Научиться запускать задачи по таймеру с использованием Quartz scheduler при помощи Spring.

Описание.

Spring предоставляет вспомогательное API для работы с таймерами, позволяя настраивать их через конфигурационные файлы.

Quartz scheduler – сторонний продукт (<http://www.quartz-scheduler.org/>). Также может быть запущен через определенные промежутки времени, но кроме этого предоставляет возможность задать конкретный момент времени выполнения задачи (например, конкретная дата или день недели.)

Spring позволяет настраивать эти планировщики через конфигурационные файлы, а также избавляет от необходимости стартовать и останавливать таймер.

Задачи упражнения.

1. Измените конфигурацию бина schedulerFactoryBean из application-context.xml и добавьте необходимые параметры для reportTrigger так, чтобы задача PrintingJob выполнялась каждую секунду, начиная с пятой секунды от момента запуска. Для проверки используйте тест QuartzJobTest, метод testRepeatableJob.
2. Измените конфигурацию бина schedulerFactoryBean из application-context.xml и добавьте необходимые параметры для reportTrigger так, чтобы задача PrintingJob выполнялась раз в три секунды только во время текущего часа, в текущий день недели, в текущем месяце. Для проверки используйте тест QuartzCronJobTest, метод testCronJob.

Подробное руководство.

Часть 1. Новые возможности в Spring 3.1, облегчающие тестирование.

1. Откройте файл с конфигурацией application-context.xml .
2. Обратите внимание на то, что в этом упражнении в файле application-context.xml появились нововведения, некоторые бины объявлены в бло-

ках: `<beans profile="interval">` , а классы юнит-тестов дополнительно проаннотированы `@ActiveProfiles("interval")`. Таким образом, при выполнении конкретных юнит-тестов будут подгружаться только бины, объявленные в рамках определенного профиля.

3. Проанализируйте контекст! *Как вы думаете, почему именно в этом упражнении мы обратились к этой возможности?*
4. Внимательно изучите комментарии к декларациям бинов – Spring 3.1 поддерживает Quartz 2.*, в API которого произошли существенные изменения.

Часть 2. Конфигурация PrintingJob при помощи SimpleTriggerFactoryBean.

5. Откройте класс `PrintingJob`. Проанализируйте его. *От какого класса он унаследован? Какие методы переопределены?*
6. Найдите блок объявления бинов для профиля «interval»
7. Добавьте следующие свойства для бина `schedulerFactoryBean`

```
<property name="triggers">
  <list>
    <ref bean="reportTrigger" />
  </list>
</property>
```

8. Задайте конфигурацию для бина `reportTrigger`, добавив необходимые свойства:

```
<property name="jobDetail" ref="reportJob" />
<property name="repeatInterval" value="1000" />
<property name="startDelay" value="5000" />
```

9. *Что означает каждое из этих свойств?*
10. Выполните тест `QuartzJobTest`, метод `testRepeatableJob`.

Часть 3. Конфигурация PrintingJob при помощи CronReportTrigger из Quartz Scheduler.

11. Найдите блок объявления бинов для профиля «cron»

Задайте конфигурацию для бина `reportTrigger`, добавив необходимые свойства:

```
<property name="jobDetail" ref="reportJob" />
<property name="cronExpression" value="0/3 * * ? m dw" />
```

12. вместо `m` подставьте номер текущего месяца, вместо `dw` номер текущего дня недели (где 1- воскресенье, 2 – понедельник и т.д.).
13. *Что означает каждое из этих свойств?*
14. *Что означает каждый из параметров `cronExpression`?*
15. Выполните тест `QuartzCronJobTest`.

Часть 4. Использование аннотаций для планирования задач.

1. Откройте `application-context-ad.xml` – изучите конфигурацию и новые возможности, которые предоставляет Spring для планирования задач с помощью аннотаций;

2. Откройте класс `ScheduledTask`, изучите код. *Каким образом Spring определяет, что метод `doSomething()` должен выполняться планировщиком?*
3. Выполните тест `ScheduledTest`, метод `testRepeatableJob()`.