

Модуль 08 - Бассейн Java Spring

Резюме: сегодня вы узнаете о разработке Java на уровне предприятия и основах среды Spring.

Содержание

I Предисловие

II Инструкции

III Упражнение 00: Spring контекст

IV Упражнение 01: JdbcTemplate

V Упражнение 02: AnnotationConfig

Глава I

Предисловие

Spring Framework является неотъемлемой частью большинства корпоративных систем на основе Java. Эта структура значительно упрощает настройку приложений и связывание компонентов друг с другом. Благодаря этому разработчик может полностью сосредоточиться на реализации бизнес-логики.

Принцип работы Spring полностью основан на шаблонах DI/loC, с которыми вам следует ознакомиться перед использованием этой технологии.

Центральным понятием в среде Spring является корзина (компонент), представляющая объект внутри контейнера ApplicationContext. Контейнер также создает связи между бинами.

Существует несколько способов настройки бинов:


1. Использование xml.file.
2. Использование конфигурации Java (настройка с аннотациями).
3. Комбинированная конфигурация.

Конфигурация XML позволяет изменить поведение приложения без пересборки. В свою очередь, конфигурация Java делает код более удобным для разработчиков.



Глава III

Упражнение 00: Spring контекст

	Exercise 00
	Spring Context
	Turn-in directory : ex00/
	Files to turn in : Spring-folder
	Allowed functions : All

Давайте реализуем слабосвязанную систему, состоящую из набора компонентов (бинов) и соответствующую принципам IoC/DI.

Предположим, что есть интерфейс принтера, предназначенный для отображения определенного сообщения.

Этот класс имеет две реализации: `PrinterWithDateTimeImpl` и `PrinterWithPrefixImpl`.

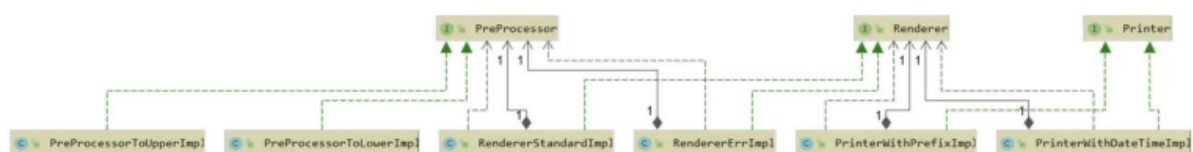
Первый класс выводит сообщения, указывая дату и время вывода с помощью `LocalDateTime`, а второй класс можно использовать для установки текстового префикса для сообщения.

В свою очередь, обе реализации `Printer` зависят от интерфейса `Renderer`, который отправляет сообщения на консоль. `Renderer` также имеет две реализации: `RendererStandardImpl` (выводит сообщение через стандартный `System.out`) и `RendererErrImpl` (выводит сообщение через `System.err`).

`Renderer` также зависит от интерфейса `PreProcessor`, который выполняет предварительную обработку сообщений.

Реализация `PreProcessorToUpperImpl` переводит все буквы в верхний регистр, а реализация `PreProcessorToLower` переводит все буквы в нижний регистр.

UML-диаграмма классов показана ниже:



Пример кода, использующего эти классы стандартным образом:

```
public class Main {
    public static void main(String[] args) {
        PreProcessor preProcessor = new PreProcessorToUpperImpl();
        Renderer renderer = new RendererErrImpl(preProcessor);
        PrinterWithPrefixImpl printer = new PrinterWithPrefixImpl(renderer);
        printer.setPrefix("Prefix ");
        printer.print("Hello!");
    }
}
```

Запуск этого кода даст следующий результат:

PREFIX HELLO


- Вам необходимо описать файл context.xml для Spring, где будут указаны все настройки для каждого компонента и связи между ними.

Использование этих компонентов с Spring выглядит следующим образом:

```
public class Main {  
    public static void main(String[] args) {  
        ApplicationContext context = new ClassPathXmlApplicationContext("context.xml");  
        Printer printer = context.getBean("printerWithPrefix", Printer.class);  
        printer.print("Hello!");  
    }  
}
```

Глава IV

Упражнение 01: JdbcTemplate

	Exercise 01
	JdbcTemplate
	Turn-in directory : <i>ex01/</i>
	Files to turn in : Service-folder
	Allowed functions : All

JdbcTemplate и его расширение NamedParameterJdbcTemplate — удобные механизмы для работы с базами данных. Эти классы позволяют исключить написание шаблонного кода для выполнения и обработки запросов, а также необходимость перехватывать проверяемые исключения.

Кроме того, они предоставляют удобную концепцию RowMapper для обработки ResultSet и преобразования результирующих таблиц в объекты.

Теперь вам нужно реализовать модель пользователя со следующими полями:

- Identifier
- Email

Вам также необходимо реализовать интерфейс CrudRepository<T> со следующими методами:

- T findById(Long id)
- List findAll()
- void save(T entity)
- void update(T entity)
- void delete(Long id)

Интерфейс UsersRepository, объявленный как UsersRepository extends CrudRepository<User>, должен содержать следующий метод:

- Optional findByEmail(String email)

Кроме того, требуются две реализации UsersRepository: UsersRepositoryJdbcImpl (использует стандартные механизмы операторов) и UsersRepositoryJdbcTemplateImpl (основан на JdbcTemplate/NamedParameterJdbcTemplate).

Оба класса должны принимать объект DataSource в качестве аргумента конструктора.

В файле context.xml должны быть объявлены бины для обоих типов репозиториев с разными идентификаторами, а также два бина типа DataSource: DriverManagerDataSource и HikariDataSource.

Кроме того, данные для подключения к БД должны быть указаны в файле db.properties и включены в context.xml с использованием заполнителей \${db.url}.

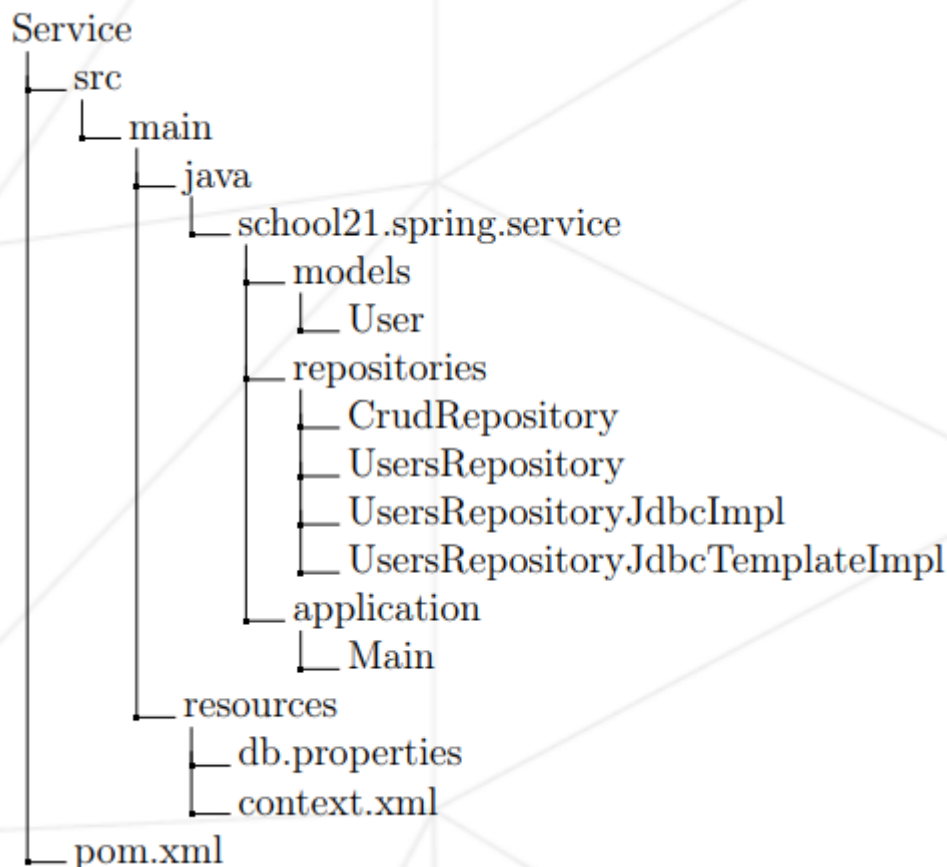
Пример db.properties:

```
db.url=jdbc:postgresql://localhost:5432/database
db.user=postgres
db.password=qwerty007
db.driver.name=org.postgresql.Driver
```

В классе Main работа метода findAll должна быть продемонстрирована с использованием обоих репозиториев:


```
ApplicationContext context = new ClassPathXmlApplicationContext("context.xml");
UsersRepository usersRepository = context.getBean("usersRepositoryJdbc", UsersRepository.class);
System.out.println(usersRepository.findAll());
usersRepository = context.getBean("usersRepositoryJdbcTemplate", UsersRepository.class);
System.out.println(usersRepository.findAll());
```

Структура проекта:



Глава V

Упражнение 02: AnnotationConfig

	Exercise 02
AnnotationConfig	
Turn-in directory : ex02/	
Files to turn in : Service-folder	
Allowed functions : All	

Теперь вам нужно настроить механизмы конфигурации Spring-приложения с помощью аннотаций.

Для этого используйте класс конфигурации, помеченный как `@Configuration`. Внутри этого класса нужно описать бины для подключения к БД `DataSource` с помощью аннотации `@Bean`. Как и в предыдущей задаче, данные о подключении должны находиться внутри файла `db.properties`. Вам также необходимо убедиться, что `context.xml` отсутствует.

Также реализуйте пару интерфейс/класс `UserService/UsersServiceImpl` с объявленной в ней зависимостью от `UsersRepository`. Вставка правильного бина репозитория должна быть реализована с помощью аннотации `@Autowired` (аналогично вам нужно привязать `DataSource` внутри репозитория).

Коллизии при автоматической привязке разрешаются с помощью аннотации `@Qualifier`.

Корзины для `UserService` и `UsersRepository` должны быть определены с использованием аннотации `@Component`.

В `UsersServiceImpl` реализуйте метод `String signUp (String email)`, который регистрирует нового пользователя и сохраняет его данные в БД. Этот метод возвращает временный пароль, присвоенный пользователю системой (эта информация также должна быть сохранена в базе данных).

Чтобы проверить, правильно ли работает ваш сервис, выполните интеграционный тест для `UsersServiceImpl`, используя базу данных в памяти (H2 или HSQLDB). Конфигурация контекста для тестовой среды (`DataSource` для базы данных в памяти) должна быть описана в отдельном классе `TestApplicationConfig`. Этот тест должен проверить, был ли возвращен временный пароль в методе регистрации.

Структура проекта:

Service
└─ src

