МИНОБРНАУКИ РОССИИ

ФГБОУ ВО «СГУ имени Н. Г. ЧЕРНЫШЕВСКОГО»

Факультет компьютерных наук и информационных технологий

УТВЕРЖДАЮ

Зав. кафедрой ИиП

Огнева М.В.

_____

__

(подпись, дата)

ОТЧЕТ О ПРАКТИКЕ

студента(ки) 4 курса 441 группы факультета КНиИТ

Вартаняна Готти Вартгесовича

вид практики: производственная

кафедра: информатики и программирования

курс: 3

семестр: 6

продолжительность: 2 нед. с 24.06.2019 г. по 07.07.2019 г.

Руководитель практики:

должность _____ ФИО

(подпись, дата)

Spring framework

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Spring Framework обеспечивает комплексную модель разработки и конфигурации для современных бизнес-приложений на Java - на любых платформах. Ключевой элемент Spring - поддержка инфраструктуры на уровне приложения: основное внимание уделяется "водопроводу" бизнес-приложений, поэтому разработчики могут сосредоточиться на бизнес-логике без лишних настроек в зависимости от среды исполнения.

Основная цель практики – создание Spring приложений, RESTful сервиса, внедрение (H2).

ОБЩИЕ СВЕДЕНИЯ О SPRING FRAMEWORK

Spring Framework (или коротко **Spring**) — универсальный фреймворк с открытым исходным кодом для Java-платформы.

Первая версия была написана Родом Джонсоном, который впервые опубликовал её вместе с изданием своей книги «Expert One-on-One Java EE Design and Development» (Wrox Press, октябрь 2002 года).

Фреймворк был впервые выпущен под лицензией Apache 2.0 license в июне 2003 года. Первая стабильная версия 1.0 была выпущена в марте 2004. Spring 2.0 был выпущен в октябре 2006, Spring 2.5 — в ноябре 2007, Spring 3.0 в декабре 2009, и Spring 3.1 в декабре 2011. Текущая версия — 5.1.2.

Несмотря на то, что Spring не обеспечивал какую-либо конкретную модель программирования, он стал широко распространённым в Java-сообществе главным образом как альтернатива и замена модели Enterprise JavaBeans. Spring предоставляет бóльшую свободу Java-разработчикам в проектировании; кроме того, он предоставляет хорошо документированные и лёгкие в использовании средства решения проблем, возникающих при создании приложений корпоративного масштаба.

Между тем, особенности ядра Spring применимы в любом Java-приложении, и существует множество расширений и усовершенствований для построения веб-приложений на Java Enterprise платформе. По этим причинам Spring приобрёл большую популярность и признаётся разработчиками как стратегически важный фреймворк.

БИНЫ

Начнем срывать покровы с самых базовых понятий Spring. Бин (bean) — это не что иное, как самый обычный объект. Разница лишь в том, что бинами принято называть те объекты, которые управляются Spring-ом. Бином является почти все в Spring — сервисы, контроллеры, репозитории, по сути все приложение состоит из набора бинов. Их можно регистрировать, получать в качестве зависимостей, проксировать, мокать и т.п.

Spring бины инициализируются при инициализации Spring контейнера и происходит внедрение всех зависимостей. Когда контейнер уничтожается, то уничтожается и всё содержимое. Если нам необходимо задать какое-либо действие при инициализации и уничтожении бина, то нужно воспользоваться методами init() и destroy().

МОДУЛИ

Spring Framework разработан с учетом модульной архитектуры, поэтому мы можем добавлять в свой проект ту функциональность, которая нам действительно необходима. Этим мы можем уменьшить итоговый размер дистрибутива программы и возможно, уменьшить время инициализации и старта приложения. Spring Framework насчитывает более 20 модулей.

Первое, что нам необходимо, это добавить модуль для работы с контекстом и Dependency Injection(DI). Делается это так же, как и для всех остальных зависимостей для Maven в наш pom.xml:

```
1. <dependencies>
2.     <dependency>
3.         <groupId>org.springframework</groupId>
4.         <artifactId>spring-context</artifactId>
5.         <version>4.1.5.RELEASE</version>
6.     </dependency>
7. </dependencies>
```

Аналогичным образом добавляются другие модули, только вместо spring-context подставляется название необходимого модуля.

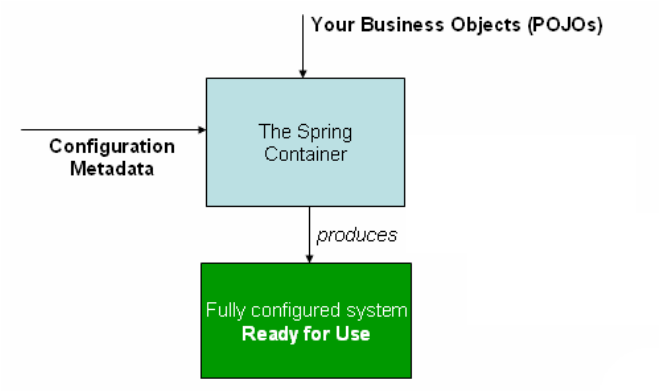
INVERSION OF CONTROL

Inversion of Control (IoC), также известное как Dependency Injection (DI), является процессом, согласно которому объекты определяют свои зависимости, т.е. объекты, с которыми они работают, через аргументы конструктора/фабричного метода или свойства, которые были установлены или возвращены фабричным методом. Затем контейнер inject(далее "внедряет") эти зависимости при создании бина. Этот процесс принципиально противоположен, поэтому и назван Inversion of Control, т.к. бин сам контролирует реализацию и расположение своих зависимостей, используя прямое создание классов или такой механизм, как шаблон Service Locator.

Основными пакетами Spring Framework IoC контейнера являются org.springframework.beans и org.springframework.context. Интерфейс BeanFactory предоставляет механизм конфигурации по управлению любым типом объектов. ApplicationContext - наследует интерфейс BeanFactory и добавляет более специфичную функциональность.

Описание работы IoC контейнера

Ниже представлена диаграмма, отражающая, как работает Spring. Ваши классы приложения совмещаются с метаданными конфигурации, в результате чего будет создан и инициализирован ApplicationContext, а на выходе вы получите полностью настроенное и готовое к выполнению приложение.


Your Business Objects (POJOs)
Configuration Metadata
The Spring Container
produces
Fully configured system
Ready for Use

ИНИЦИАЛИЗАЦИИ

В настоящий момент Spring framework поддерживает три разных способа конфигурирования контекста, каждый из которых заслуживает отдельного рассмотрения.

XML

Исторически конфигурирование контекста с использованием XML было первым методом конфигурирования, появившемся в Spring. Конфигурирование с помощью XML заключается в создании xml файла (традиционно носящего названия вида «context.xml», «applicationContext.xml» и т.д.), описывающего Spring beans, процесс их создания и взаимосвязи между ними.

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans xmlns="http://www.springframework.org/schema/beans"
3.        xmlns:context="http://www.springframework.org/schema/context"
4.        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5.        xsi:schemaLocation="http://www.springframework.org/schema/beans
   http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
6. http://www.springframework.org/schema/context
   http://www.springframework.org/schema/context/spring-context-3.0.xsd">
7.
8.     <bean id="coin" class="ru.java.spring.CoinImpl">
9.         <constructor-arg type="java.util.Random">
10.             <bean class="java.util.Random"/>
11.         </constructor-arg>
12.     </bean>
13.
14.     <bean id="target" class="ru.java.spring.GreeterTargetImpl">
15.         <constructor-arg index="0" ref="coin"/>
16.     </bean>
17.
18.     <bean id="greeter" class="ru.java.spring.Greeter">
19.         <constructor-arg name="newTarget" ref="target"/>
20.     </bean>
21.
22. </beans>
```

В приложении заменим создание контекста на ClassPathXmlApplicationContext("/applicationContext.xml") и можно проверять результат:

Использование XML конфигурации конечно неудобно тем, что приходится кроме написания непосредственно кода описывать ещё и его использование в Spring, при этом не имея возможности проверить ошибки в конфигурации до запуска тестов (или даже приложения). Кроме того, в декларативном xml файле достаточно сложно реализовать конфигурацию, требующую активных действий во время создания контекста.

С другой стороны, xml конфигурация представляет собой централизованное описание приложения, которое может хранится отдельно от кода, позволяя менять структуру приложения без пересборки. С помощью xml можно использовать в качестве Spring beans сторонний код.

Annotations

Конечно, использование аннотаций для определения бинов и их зависимостей, весьма удобно и упрощает разработку, но недостатков у этого подхода больше всего. Конфигурация контекста получается децентрализованной, так что неосторожное добавление нового бина может внезапно изменить работу всего приложения (и это, опять таки, не узнать до запуска тестов/приложения). Так же в код попадают Spring специфичные вещи, которые потом могут затруднить смену платформы. С аннотациями использование стороннего кода либо невозможно, либо требует определённых подпорок и костылей. Кроме того, единственной возможностью изменить поведение приложения будет его пересборка.

Java configuration

Третий подход — программное создание бинов, реализующий модный принцип convention over configuration, соглашения по конфигурации. Стоит отметить, что под программным созданием бинов я понимаю создание бинов на стадии формирования контекста, а не после того, как приложение уже запущено.

Все бины будут теперь определяться в ContextConfiguration:

public class ContextConfiguration {

```
1.      /**
2.       * "Random" service bean.
3.       * @return Java's built-in random generator.
4.       */
5.      @Bean
6.      public Random random() {
7.          return new Random();
8.      }
9.
10.       @Bean
11.       public Coin coin() {
12.           return new CoinImpl(random());
13.       }
14.
15.       @Bean
16.       public GreeterTarget greeterTarget() {
17.           return new GreeterTargetImpl(coin());
18.       }
19.
20.       @Bean
21.       public Greeter greeter() {
22.           return new Greeter(greeterTarget());
23.       }
24.  }
```

Spring вызывает в конфигурационных классах методы с аннотацией @Bean. Объекты, возвращённые этими методами, регистрируются как Spring бины. Названия бинов соответствуют названиям методов, которые их порождают.

Вызов методов, создающих бины, вручную, вполне безопасен, потому что Spring изменяет код создания бина, чтобы пытаться вернуть уже существующий подходящий бин и только если это невозможно, вызывать создающий код. Поэтому в методе coin() не создаётся второй экземпляр бина random, а используется ранее созданый бин. По этой причине методы, имеющие аннотацию @Bean, не должны объявляться final.

Важно и название класса конфигурации: несмотря на искушение назвать конфигурационный класс Context, делать этого не следует. Дело в том, что

Spring создаст бин и из этого класса, а в качестве имени использует имя класса. А имя «Context» уже занято самим Spring.

Java конфигурация выглядит наилучшим образом, если сравнивать её достоинства и недостатки. Это и централизованность как в xml; и безопасность типов; и простой рефакторинг; и отсутствия spring специфичных вещей в коде; и возможность выполнения каких-либо действий на этапе конфигурации. К недостаткам, пожалуй, относится необходимость ручного создания бинов и необходимость пересборки для переконфигурации приложения.

БАЗА ДАННЫХ H2

Spring – достаточно многофункциональный framework, и если мы надумали делать enterprise проект, то нам не обойтись без возможности работать с базами данных.

Для работы с базой данных в Spring используем возможности Spring Data. Нам необходимо:

- Добавить зависимости от соответствующей версии Spring.
- Добавить в зависимости библиотеку коннектора к h2 в pom.xml
- Добавить Hibernate к зависимостям проекта.
- Создать конфигурационный файл и прописать свойства БД H2

SPRING BOOT

Авторы Spring решили предоставить разработчикам некоторые утилиты, которые автоматизируют процедуру настройки и ускоряют процесс создания и развертывания Spring-приложений, под общим названием Spring Boot

Spring Boot — это полезный проект, целью которого является упрощение создания приложений на основе Spring. Он позволяет наиболее простым способом создать web-приложение, требуя от разработчиков минимум усилий по его настройке и написанию кода

Особенности Spring Boot

Spring Boot обладает большим функционалом, но его наиболее значимыми особенностями являются: управление зависимостями, автоматическая конфигурация и встроенные контейнеры сервлетов.

О REST И ЕГО ПОДДЕРЖКЕ

REST — это архитектурный стиль взаимодействия приложений в сети. Но в отличии от SOAP, у REST отсутствует какой-либо стандарт, а данные между клиентом и сервером могут предаваться в любом виде, будь то JSON, XML, YAML и т.д.

Spring Framework предоставляет богатый набор инструментов, упрощающий разработку REST API: инструменты для маршрутизации запросов, классы-кодеки для преобразования JSON/XML в объекты требуемых типов и т.д. Для маршрутизации запросов в Spring Framework используется аннотация @RequestMapping с указанием HTTP-метода при помощи свойства method или более простые аннотации вроде @GetMapping, @PostMapping, @DeleteMapping и т.д.