Введение в Spring. Beans

№ урока: 9 **Курс:** Основы Java EE

Средства обучения: InteliJ Idea

Обзор, цель и назначение урока

Знакомство с Spring. Создание первого приложения на Spring. Понятия Beans, IoC, AOP. Шаблоны Proxy, Singleton, Prototype.

Изучив материал данного занятия, учащийся сможет:

- Создать приложение в виде ООП используя Spring.
- Разобраться, как внедряются объекты через Dependency Injection в Spring.

Содержание урока

1. Знакомство с Spring.
2. Понятия IoC.
3. Понятия AOP.
4. Понятия Dependency Injection и Service Locator.
5. Область видимости бинов: шаблоны Singleton и Prototype.
6. Наследование бинов.
7. Шаблон Proxy и библиотека CGLIB.
8. Типы внедрений объектов в бины.
9. Использование коллекций в бинах.

Резюме

- **Spring Framework** (или коротко **Spring**) — универсальный фреймворк с открытым исходным кодом для Java-платформы. Также существует форк для платформы .NET Framework, названный Spring.NET.
  Первая версия была написана Родом Джонсоном, который впервые опубликовал её вместе с изданием своей книги «Expert One-on-One Java EE Design and Development»[3] (Wrox Press, октябрь 2002 года).
  Фреймворк был впервые выпущен под лицензией Apache 2.0 license в июне 2003 года. Первый стабильный релиз 1.0 был выпущен 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 — 5.0.1.
  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.
- **Spring** может быть рассмотрен как коллекция меньших фреймворков или фреймворков во фреймворке. Большинство этих фреймворков может работать независимо друг от друга, однако они обеспечивают большую функциональность при совместном их использовании. Эти фреймворки делятся на структурные элементы типовых комплексных приложений:

CyberBionic systematics

CyberBionic Systematics ® 2018
19 Eugene Sverstyuk Str.,5 floor
Kyiv, Ukraine

Tel. 0 800 750 312
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

Inversion of Control-контейнер: конфигурирование компонентов приложений и управление жизненным циклом Java-объектов.
Фреймворк аспектно-ориентированного программирования: работает с функциональностью, которая не может быть реализована возможностями объектно-ориентированного программирования на Java без потерь.
Фреймворк доступа к данным: работает с системами управления реляционными базами данных на Java-платформе, используя JDBC- и ORM-средства и обеспечивая решения задач, которые повторяются в большом числе Java-based environments.
Фреймворк управления транзакциями: координация различных API управления транзакциями и инструментарий настраиваемого управления транзакциями для объектов Java.
Фреймворк MVC: каркас, основанный на HTTP и сервлетах, предоставляющий множество возможностей для расширения и настройки (customization).
Фреймворк удалённого доступа: конфигурируемая передача Java-объектов через сеть в стиле RPC, поддерживающая RMI, CORBA, HTTP-based протоколы, включая web-сервисы (SOAP).
Фреймворк аутентификации и авторизации: конфигурируемый инструментарий процессов аутентификации и авторизации, поддерживающий много популярных и ставших индустриальными стандартами протоколов, инструментов, практик через дочерний проект Spring Security (ранее известный как Acegi).
Фреймворк удалённого управления: конфигурируемое представление и управление Java-объектами для локальной или удалённой конфигурации с помощью JMX.
Фреймворк работы с сообщениями: конфигурируемая регистрация объектов-слушателей сообщений для прозрачной обработки сообщений из очереди сообщений с помощью JMS, улучшенная отправка сообщений по стандарту JMS API.
Тестирование: каркас, поддерживающий классы для написания модульных и интеграционных тестов.

- Центральной частью Spring является контейнер **Inversion of Control**, который предоставляет средства конфигурирования и управления объектами Java с помощью рефлексии. Контейнер отвечает за управление жизненным циклом объекта: создание объектов, вызов методов инициализации и конфигурирование объектов путём связывания их между собой.
  Объекты, создаваемые контейнером, также называются управляемыми объектами (beans). Обычно конфигурирование контейнера осуществляется путём загрузки XML-файлов, содержащих определение bean'ов и предоставляющих информацию, необходимую для создания bean'ов.
  Объекты могут быть получены одним из двух способов:
  Поиск зависимости — шаблон проектирования, в котором вызывающий объект запрашивает у объекта-контейнера экземпляр объекта с определённым именем или определённого типа.
  Внедрение зависимости — шаблон проектирования, в котором контейнер передает экземпляры объектов по их имени другим объектам с помощью конструктора, свойства или фабричного метода.
- **Внедрение зависимости** (англ. Dependency injection, DI) — процесс предоставления внешней зависимости программному компоненту. Является специфичной формой «инверсии управления» (англ. Inversion of control, IoC), когда она применяется к управлению зависимостями. В полном соответствии с принципом единственной ответственности объект отдаёт заботу о построении требуемых ему зависимостей внешнему, специально предназначенному для этого общему механизму.
  Работа фреймворка, обеспечивающая внедрение зависимости, описывается следующим образом. Приложение, независимо от оформления, исполняется внутри контейнера IoC, предоставляемого фреймворком. Часть объектов в программе по-прежнему создается обычным способом языка программирования, часть создается контейнером на основе предоставленной ему конфигурации.
  Условно, если объекту нужно получить доступ к определенному сервису, объект берет на себя ответственность за доступ к этому сервису: он или получает прямую ссылку на местонахождение сервиса, или обращается к известному «сервис-локатору» и запрашивает ссылку на реализацию определенного типа сервиса. Используя же внедрение зависимости, объект просто предоставляет свойство, которое в состоянии хранить ссылку на нужный тип сервиса; и когда объект создается, ссылка на реализацию нужного типа сервиса автоматически вставляется в это свойство (поле), используя средства среды.

CyberBionic systematics
CyberBionic Systematics ® 2018
19 Eugene Sverstyuk Str.,5 floor
Kyiv, Ukraine
Tel. 0 800 750 312
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

Внедрение зависимости более гибко, потому что становится легче создавать альтернативные реализации данного типа сервиса, а потом указывать, какая именно реализация должна быть использована в, например, конфигурационном файле, без изменений в объектах, которые этот сервис используют. Это особенно полезно в юнит-тестировании, потому что вставить реализацию «заглушки» сервиса в тестируемый объект очень просто.

С другой стороны, излишнее использование внедрения зависимостей может сделать приложения более сложными и трудными в сопровождении: так как для понимания поведения программы программисту необходимо смотреть не только в исходный код, а еще и в конфигурацию, а конфигурация, как правило, невидима для IDE, которые поддерживают анализ ссылок и рефакторинг, если явно не указана поддержка фреймворков с внедрениями зависимостей.

- **Локатор служб** (англ. service locator) — это шаблон проектирования, используемый в разработке программного обеспечения для инкапсуляции процессов, связанных с получением какого-либо сервиса с сильным уровнем абстракции. Этот шаблон использует центральный реестр, известный как «локатор сервисов», который по запросу возвращает информацию (как правило это объекты), необходимую для выполнения определенной задачи. Обратите внимание, что в некотором случае, локатор служб фактически является анти-шаблоном.

  Преимущества:

  - «Локатор служб» может действовать как простой компоновщик времени выполнения. Это позволяет управлять кодом программы во время выполнения без повторной компиляции приложения, а в некоторых случаях без необходимости его перезапуска.
  - Приложения могут оптимизировать себя во время выполнения путем выборочного добавления и удаления элементов из локатора служб. Например, приложение может обнаружить, что у него есть лучшая библиотека для чтения доступных изображений JPG, чем стандартная, и соответствующим образом изменить реестр.
  - Компоненты приложения или библиотеки, используемые в приложении, могут быть полностью разделены. Единственная связь между ними записывается в реестр.

  Недостатки:

  - Отношения между компонентами приложения, помещенные в реестр, скрывают зависимости частей программы (делают их неочевидными) и увеличивают сложность системы. Это затрудняет поиск ошибок в приложении и может сделать систему в целом менее надежной.
  - Реестр должен быть уникальным, что может стать узким местом для одновременного запуска несколько копий приложения.
  - Реестр может быть серьезной уязвимостью безопасности, поскольку он позволяет посторонним (в том числе злоумышленникам) вводить код в приложение.
  - Реестр скрывает зависимости класса, вызывая ошибки во время выполнения (англ.)русск., а не ошибки времени компиляции, когда при отсутствии необходимых зависимостей компилятор информирует об ошибке.
  - Реестр делает код более сложным для поддержания (против использования инъекции зависимостей), потому что становится неясным, когда вы вносите ошибочную запись в реестр или пропускаете необходимую запись.
  - Реестр делает код более сложным для тестирования, поскольку все тесты должны взаимодействовать с одним и тем же глобальным классом локатора служб для установки поддельных зависимостей тестируемого класса. Однако это легко преодолеть, введя классы приложений с помощью одного интерфейса локатора служб

- **Заместитель** (англ. Proxy) — структурный шаблон проектирования, предоставляющий объект, который контролирует доступ к другому объекту, перехватывая все вызовы (выполняет функцию контейнера).

  Проблема

  Необходимо контролировать доступ к объекту, не изменяя при этом поведение клиента. Необходимо иметь доступ к объекту так, чтобы не создавать реальные объекты непосредственно, а через другой объект, который может иметь дополнительную функциональность.

CyberBionic systematics

CyberBionic Systematics ® 2018
19 Eugene Sverstyuk Str.,5 floor
Kyiv, Ukraine

Tel. 0 800 750 312
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

Решение

Создать суррогат реального объекта. «Заместитель» хранит ссылку, которая позволяет заместителю обратиться к реальному субъекту (объект класса «Заместитель» может обращаться к объекту класса «Субъект», если интерфейсы «Реального Субъекта» и «Субъекта» одинаковы). Поскольку интерфейс «Реального Субъекта» идентичен интерфейсу «Субъекта», так, что «Заместителя» можно подставить вместо «Реального Субъекта», контролирует доступ к «Реальному Субъекту», может отвечать за создание или удаление «Реального Субъекта». «Субъект» определяет общий для «Реального Субъекта» и «Заместителя» интерфейс так, что «Заместитель» может быть использован везде, где ожидается «Реальный Субъект». При необходимости запросы могут быть переадресованы «Заместителем» «Реальному Субъекту».

Виды:

- **Протоколирующий прокси**: сохраняет в лог все вызовы «Субъекта» с их параметрами.
- **Удалённый заместитель** (англ. remote proxies): обеспечивает связь с «Субъектом», который находится в другом адресном пространстве или на удалённой машине. Также может отвечать за кодирование запроса и его аргументов и отправку закодированного запроса реальному «Субъекту»,
- **Виртуальный заместитель** (англ. virtual proxies): обеспечивает создание реального «Субъекта» только тогда, когда он действительно понадобится. Также может кэшировать часть информации о реальном «Субъекте», чтобы отложить его создание,
- **Копировать-при-записи**: обеспечивает копирование «субъекта» при выполнении клиентом определённых действий (частный случай «виртуального прокси»).
- **Защищающий заместитель** (англ. protection proxies): может проверять, имеет ли вызывающий объект необходимые для выполнения запроса права.
- **Кэширующий прокси**: обеспечивает временное хранение результатов расчёта до отдачи их множественным клиентам, которые могут разделить эти результаты.
- **Экранирующий прокси**: защищает «Субъект» от опасных клиентов (или наоборот).
- **Синхронизирующий прокси**: производит синхронизированный контроль доступа к «Субъекту» в асинхронной многопоточной среде.
- **«Умная» ссылка** (англ. smart reference proxy): производит дополнительные действия, когда на «Субъект» создается ссылка, например, рассчитывает количество активных ссылок на «Субъект»

Преимущества**:**

- удалённый заместитель;
- виртуальный заместитель может выполнять оптимизацию;
- защищающий заместитель;
- «умная» ссылка(указатель);

Недостатки

- резкое увеличение времени отклика.

Шаблон Proxy может применяться в случаях работы с сетевым соединением, с огромным объектом в памяти (или на диске) или с любым другим ресурсом, который сложно или тяжело копировать. Хорошо известный пример применения — объект, подсчитывающий число ссылок.

Закрепление материала

- Что такое Spring?
- Что такое IoC?
- Что такое AOP?
- Что такое Dependency Injection?
- В чем разница между Singleton и Prototype?
- Зачем нужен шаблон проектирования Proxy?
- Что такое bean?
- В чем разница между наследованием бинов и наследованием классов?
- Какие есть типы внедрения объектов в бины?

Дополнительное задание

CyberBionic systematics

CyberBionic Systematics ® 2018
19 Eugene Sverstyuk Str.,5 floor
Kyiv, Ukraine

Tel. 0 800 750 312
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

Задание
Создайте интерфейс транспортного средства и 3 реализации. Также транспортные средства должны иметь колеса и аудиосистему. Реализуйте бины к транспортным средствам, колесам и аудиосистеме.

Самостоятельная деятельность учащегося

Задание 1
Выучите основные понятия, рассмотренные на уроке.
Прочитать документацию pdf про IoC контейнеры, главы 3.1, 3.2.
Фундаментальную статью Мартина Флауера http://martinfower.com/articles/injection.html

Задание 2
Прочитать разделы 3.3, 3.4
Прочитать про c: namespace в документации, изменить конфиг с помощью c:namespace. Добавить bean для реализации 3-го конструктора.

Задание 3
Прочитать 3.5 Bean Scopes, 3.8 Container Extension Points. 3.7 Bean Definition Inheritance. Переопределить вызов методов init и destroy через интерфейс BeanPostProcessor

Задание 4
Изучите внедрение метода способом <replaced-method>
Прочитать http://phantompainray.blogspot.com/2013/11/spring_28.html
Замените реализацию метода action у объекта T1000 с помощью <replaced-method>

Задание 5
Прочитать главу Collections (стр. 48).
Реализовать пример с помощью коллекции Map у роботов. Ключ – поле year. Передать коллекцию через конструктор.

Рекомендуемые ресурсы

Spring docs pdf
https://docs.spring.io/spring/docs/3.0.x/spring-framework-reference/pdf/spring-framework-reference.pdf

IoC контейнеры
http://java-source.net/open-source/containers

Сравнение
http://www.silkdi.com/help/comparison.html

BeanFactory
http://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/beans/factory/BeanFactory.html

ApplicationContext
http://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/beans/context/ApplicationContext.html

Proxy
http://tutorials.jenkov.com/java-reflection/dynamic-proxies.html

CyberBionic systematics
CyberBionic Systematics ® 2018
19 Eugene Sverstyuk Str.,5 floor
Kyiv, Ukraine
Tel. 0 800 750 312
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

http://java.dzone.com/articles/power-proxies-java
https://www.javacodegeeks.com/2014/01/implementing-dynamic-proxies-a-comparison.html
https://walivi.wordpress.com/2012/12/20/jdk-dynamic-proxy/
http://tutorials.jenkov.com/java-reflection/index.html

ByteCode Libriaries
http://java-source.net/open-source/bytecode-libraries

CGLIB Maven
http://mvnrepository.com/artifact/cglib/cglib

CyberBionic Systematics ® 2018
19 Eugene Sverstyuk Str.,5 floor
Kyiv, Ukraine

Tel. 0 800 750 312
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com