Видео 1.

Framework vs. Library

Dependency Injection

Inversion of Control

Core container

* Beans
* Core
* Context
* Expression

AOP

* AOP
* Aspects

Instrumentation

* Instrument – Spring Boot??

Data access and integration

* JDBC
* JMS
* ORM
* Transaction

Web and Remoting – MVC тут

* Web
* Servlet
* Struts

Test

* Testing

Модули:

spring-context

spring-context-support

spring-tx - транзакции

spring-jdbc

Вариант приложения 1 – Client – имеет id и name, ConsoleEventLogger – имеет метод logEvent, который выводит сообщение на консоль, App – создает Client и ConsoleEventLogger внутри себя и пользуется их методами.

Проблемы:

1. Сложно редактировать – нужно перекомпилировать при изменении данных
2. Плохое масштабирование – только один логгер, который жестко зашит в приложение. Для изменения нужно перекомпилировать.
3. Сложно тестировать – при тестировании App.eventLogger будет тестироваться и ConsoleEventLogger.

Причина всех проблем – жесткое связывание (tight coupling).

Видео 2.

Решение проблем:

1. Всю статическую информацию лучше вынести во внешние файлы.
2. Сделать интерфейсную развязку - реализация должна зависить от интерфейсов, а не от классов. Но интерфейсы не нужны для классов, которые только хранят информацию (как Client).
3. Внедрение зависимостей – вместо создания экземпляров вручную, просим это сделать у Spring.

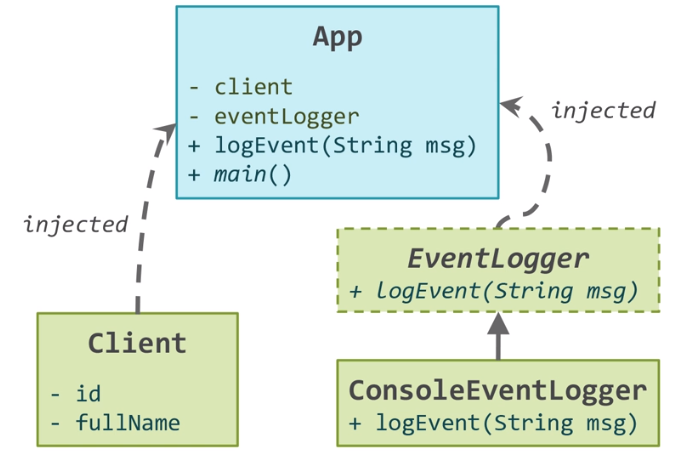
SVN???

Внедрение зависимостей решает проблемы:

1. ~~Плохое масштабирование~~ – можно подсунуть логгер с другой реализацией, рассчитанной на масштабирование.
2. ~~Сложно тестировать~~ – внедряем тестовую реализацию вместо ConsoleEventLogger и можно протестировать App.eventLogger в изоляции

Дорабока приложения

Вариант приложения 2 – ConsoleEventLogger реализует интерфейс EventLogger, App не создает Client и ConsoleEventLogger, а принимает их в качестве параметров конструктора.

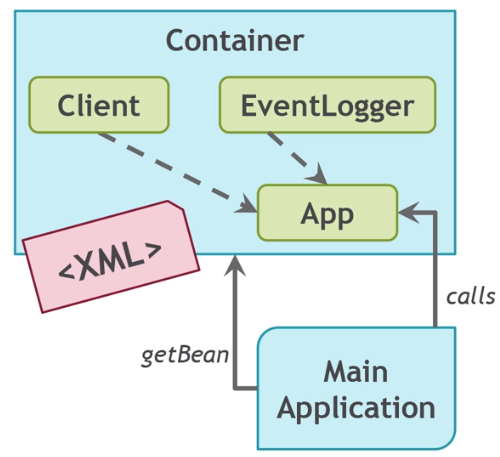


За создание классов и их связывание отвечает spring-контейнер. Мы обращаемся к нему для получения необходимой информации. Экземпляры классов – бины.

Если нужен бин, то возможны 2 варианта:

1. Обращение к контейнеру для получения
2. Навигация по графу объектов (получаем 1й бин, через него следующий и т.д.)

Как узнать какие бины создавать? XML-файл (или аннотации, или java, или groovy).



src/main/resources/Beans.xml

*<?***xml version="1.0" encoding="UTF-8"***?>*<**beans xmlns="http://www.springframework.org/schema/beans"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="  
 http://www.springframework.org/schema/beans  
 http://www.springframework.org/schema/beans/spring-beans-3.2.xsd"**>  
  
 <**bean id="client" class="com.epam.spring.core.beans.Client"**/>  
  
 <**bean id="eventLogger" class="com.epam.spring.core.loggers.ConsoleEventLogger"**/>  
  
 <**bean id="app" class="ua.epam.spring.core.com.epam.spring.core.App"**/>  
</**beans**>

Видео 3

2 способа именовать бины (можно вместе):

1. id – должен быть уникальным
2. name – можно указать несколько имен для одного бина. Полезно, когда бин реализует несколько интерфейсов и к нему удобно обращаться по разному.

Если указать несколько имен, то первый будет основным, а остальные будут алиасами. Можно указать алиас вручную:

<alias name=”aaa” alias=”bbb”>

Полезно, когда бины берутся из сторонних библиотек.

Имена нужны для инжектинга. Можно использовать и просто имена интерфейсов, но они часто реализуются несколькими классами, что приводит к ошибкам.

Внедрение зависимостей:

1. Через конструкторы
2. Через геттеры/сеттеры

Через конструкторы:

<**bean id="client" class="com.epam.spring.core.beans.Client"**>  
 <**constructor-arg value="1"**/>  
 <**constructor-arg value="John Smith"**/>  
</**bean**>

Аргументы одного типа инжектятся в том порядке, в котором перечислены.

Можно указать порядок вручную.

<**bean id="client" class="com.epam.spring.core.beans.Client"**>  
 <**constructor-arg index="0" value="1"**/>  
 <**constructor-arg index="1" value="John Smith"**/>  
</**bean**>

Если типы разные, то порядок не важен – спринг поймет.

Но, например тип 1 можно указать и для String и для Integer. Как Спрингу разобраться? Можно указать тип:

<**constructor-arg type="java.lang.Integer" value="1"**/>

<**constructor-arg type="java.lang.String" value="1"**/>

Можно указать имена переменных для параметров:

<**constructor-arg name="arg1" value="1"**/>

<**constructor-arg name="arg2" value="Dude"**/>

Но это хак, который работает через рефлекшн только когда включены дебаг-символы (???) и не используется в проде.

Инжектинг бинов:

<**bean id="app" class="com.epam.spring.core.App"**>  
 <**constructor-arg ref="client"**/>  
 <**constructor-arg ref="eventLogger"**/>  
</**bean**>

Если не указать аргументы, то спринг вызывает конструктор по умолчанию и он должен быть объявлен (если нужно).

Модфикатор доступа в конструкторе м.б. любым – спринг вызовет его даже если он private.

Описание контейнера Спринг – в xml (определены бины и зависимости).

1. Создать контейнер
2. Запустить приложение
3. Спринг создаст бины и проинжектит зависимости

В spring-framework 2 типа контейнеров:

1. BeanFactory – простейший контейнер.
   1. Поднимается
   2. Создает бины
   3. Инжектит их
2. ApplicationContext -
   1. Поднимается
   2. Создает бины
   3. Инжектит их
   4. Предоставляет сервисы для управления бинами

Создание контекста:

ApplicationContext ctx =

**new** ClassPathXmlApplicationContext(**"spring.xml"**);

Получение бина:

1. По имени – класс будет неизвестен, придется делать каст (почему? Все бины типа Object?)
2. По классу – не нужно кастить, используются дженерики
3. По классу и имени – как и у получения по классу
4. По аннотации, которая есть у класса

Можно указать несколько xml при создании контекста. Спринг объединит их.

???контекст == контейнер???

Дорабока приложения

Вариант приложения 3 – теперь в App создается контекст, а создание всех объектов указано в xml конфигурации.

Видео 4

Дорабока приложения

Вариант приложения 4 – появляется класс Event c 4 полями:

- id – генерируется через Random

- msg –через геттеры-сеттер

- date – через конструктор

- df – DateFormat – через конструктор, используется в toString().

EventLogger теперь принимает Event.

Event будет создаваться через Spring. Добавить в Event DateFormat, который принимается консту

Bean Scope / цикл жизни объекта в конейнере:

- scope по умолчанию – singleton

- scope задается в конфигурации, например так:

<**bean id="eventLogger" class="com.epam.spring.core.loggers.ConsoleEventLogger" scope="singleton"**/>

Если scope задан как синглтон, то при вызове getBean() будет возвращаться один и тот же бин.

- prototype – scope, при котором бин будет заново создаваться каждый раз при вызове getBean().

Другие scope:

* request – бин живет только на время запроса;
* session – бин живет пока есть сессия;
* global session – живет, пока есть глобальная http сессия.

Event требует объект типа Date как параметр конструктора. Можно создать его как простой bean и проинжектить. Но можно использовать внутренний бин:

<**bean id="event" class="com.epam.spring.core.events.Event" scope="prototype"**>  
 <**constructor-arg**>  
 <**bean class="java.util.Date"**/>  
 </**constructor-arg**>  
</**bean**>

Внутреннему бину можно назначить name и id, но он виден только там, где он опеределен и достучаться до него нельзя.

Все внутренние бины создаются в момент создания внешнего бина.

DateFormat – абстрактный класс для форматирования даты. У него нет конструктора, но есть статический метод, возвращающий объект этого класса. Для создания бина можно использовать factory-method:

<**bean class="java.text.DateFormat" factory-method="getDateTimeInstance"**/>

Если factory-method требует аргументы, то указываем их через constructor-args.

Видео 5

Дорабока приложения

Появляется новый логгер, который будет писать лог в файл. FileEventLogger (реализует интерфейс EventLogger). Имя файла в который будет записываться событие - полем, значение – через конструктор, создание экземляра – через Spring.

Если файл недоступен, то логгер вылетит с ошибкой при первом появлении эвента. Лучше найти такую ситуацию заранее.

Bean Initializers (правда так называется?) – работают после создания бина, но до его первого использования. Нужны для подготовки бина к работе. Для вызова такого метода нужно написать в xml:

<**bean id="eventLogger" class="com.epam.spring.core.loggers.FileEventLogger" init-method="init"**>  
 <**constructor-arg value="C:\Users\Sergei\_Morozov\Documents\Projects\spring-weaver\logfile.txt"**/>  
</**bean**>

Init-методы:

1. Не могут принимать аргументы
2. Могут иметь любой модификатор доступа (спрингу плевать)
3. Может возвращать что угодно (никто не прочитает) или кидать Exception

Bean Lifecycle (so far):

1. Создаются все зависимые бины (чтобы проинжектить)
2. Вызывается конструктор бина (с параметрами, если есть)
3. Устанавливаются properties (через сеттеры) (как именно??)
4. Вызывается init метод

Дорабока приложения

CacheFileLogger – записывает значение не сразу. Cache – лист eventов, cacheSize – размер листа (через конструктор).

Проблема – если по окончанию работы приложения в кеше остались события, то они не будут записаны. Нужен какой-то деструктор. Для этого есть destroy-method.

<**bean id="eventLogger" class="com.epam.spring.core.loggers.CacheFileEventLogger" init-method="init" destroy-method="destroy"**>  
 <**constructor-arg value="C:\Users\Sergei\_Morozov\Documents\Projects\spring-weaver\logfile.txt"**/>  
 <**constructor-arg value="2"**/>  
</**bean**>

Для destroy метода применяются те же ограничения, что и для init, но вызывется destroy метод только при правильном закрытии контекста.

ConfigurableApplicationContext *ctx* = **new** ClassPathXmlApplicationContext(**"spring.xml"**);

*ctx*.close();

У ApplicationContext нет метода close, он есть у ConfigurableApplicationContext

Ещё контексты???

ConfigurableApplicationContext так же содержит ShutdownHook – метод, для регистрации момента завершения приложения, который вызовется, если приложение завершится средствами java