Конфигурация Spring

1. Только XML

2. XML + код(аннотации)

3. Только код

Создание начального проекта Spring

1. Создаём проект Maven
2. Подгружаем Spring core, Spring bean, Spring context.
3. Собираем конфигурационный файл для Spring

Внедрение зависимостей (через конструктор)

<?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 https://www.springframework.org/schema/beans/spring-beans.xsd">  
  
 <bean id="musicBean"  
 class="ru.javas.ClassicMusic">  
 </bean>  
   
 <bean id="playerBean" class="ru.javas.PlayerMusic">  
 <constructor-arg ref="musicBean"/>  
 </bean>  
   
</beans>

(Тут мы внутри конструктора бина (playerBean) внедряем бин

(musicBean)), внизу java код

package ru.javas;  
  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
  
public class Start {  
  
 public static void main(String[] args) {  
  
 ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext(  
 "application.xml"  
 );  
  
 PlayerMusic player = context.getBean("playerBean", PlayerMusic.class);  
   
 System.*out*.println(player.music.getSongs());  
 System.*out*.println("end work");  
 }  
}

(Это внедрение зависимости через конструктор)

Внедрение зависимостей (через setter)

<?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 https://www.springframework.org/schema/beans/spring-beans.xsd">  
  
 <bean id="musicBean"  
 class="ru.javas.ClassicMusic">  
 </bean>  
  
 <bean id="playerBean" class="ru.javas.PlayerMusic">  
<!-- <constructor-arg ref="musicBean"/>-->  
 <property name="music" ref="musicBean" />  
 </bean>  
  
</beans>

(Внедрили ссылку на бин через setter)

package ru.javas;  
  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
  
public class Start {  
  
 public static void main(String[] args) {  
  
 ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext(  
 "application.xml"  
 );  
  
 PlayerMusic player = context.getBean("playerBean", PlayerMusic.class);  
  
 System.*out*.println(player.music.getSongs());  
 System.*out*.println("end work");  
 }  
}

Внедрение простых значений

<?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 https://www.springframework.org/schema/beans/spring-beans.xsd">  
  
 <bean id="musicBean"  
 class="ru.javas.ClassicMusic">  
 </bean>  
  
 <bean id="playerBean" class="ru.javas.PlayerMusic">  
<!-- <constructor-arg ref="musicBean"/>-->  
  
 <property name="music" ref="musicBean" />  
 <property name="name" value="Topa"/>  
 <property name="age" value="35"/>  
 </bean>  
  
</beans>

package ru.javas;  
  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
  
public class Start {  
  
 public static void main(String[] args) {  
  
 ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext(  
 "application.xml"  
 );  
  
 PlayerMusic player = context.getBean("playerBean", PlayerMusic.class);  
  
 System.*out*.println(player.music.getSongs());  
 System.*out*.println(player.getName());  
 System.*out*.println(player.getAge());  
  
 System.*out*.println("end work");  
 }  
}

Внедрение зависимостей через файл

1. Создаём в ресурсах app.properties
2. <context:property-placeholder location="classpath:playerMusic.properties"/>
3. <property name="name" value="${musicName}">  
     
   </property>  
     
   <property name="age" value="${musicAge}">  
     
   </property>

Передача нескольких аргументов в метод

<?xml version="1.0" encoding="UTF-8"?>  
<beans xmlns="http://www.springframework.org/schema/beans"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xmlns:context="http://www.springframework.org/schema/context"  
 xsi:schemaLocation="http://www.springframework.org/schema/beans  
 http://www.springframework.org/schema/beans/spring-beans.xsd  
 http://www.springframework.org/schema/context  
 http://www.springframework.org/schema/context/spring-context-3.0.xsd">  
  
 <context:property-placeholder location="classpath:playerMusic.properties"/>  
  
 <bean id="musicBean1"  
 class="ru.javas.ClassicMusic">  
 </bean>  
  
 <bean id="musicBean2"  
 class="ru.javas.RockMusic">  
 </bean>  
  
 <bean id="playerBean" class="ru.javas.PlayerMusic">  
  
 <property name="music">  
 <list>  
 <ref bean="musicBean1"/>  
 <ref bean="musicBean2"/>  
 <ref bean="musicBean2"/>  
 </list>  
 </property>  
   
 </bean>  
  
</beans>

1. Scope
   1. Singleton

(Это один из паттернов программирования, мы создаём один объект класса и всё время работаем с ним)

(Минус в том, что при изменении поля, значения изменятся во всех переменных, которые ссылаются на данный объект)

(Поэтому используется только для создания объектов без изменяемых полей)

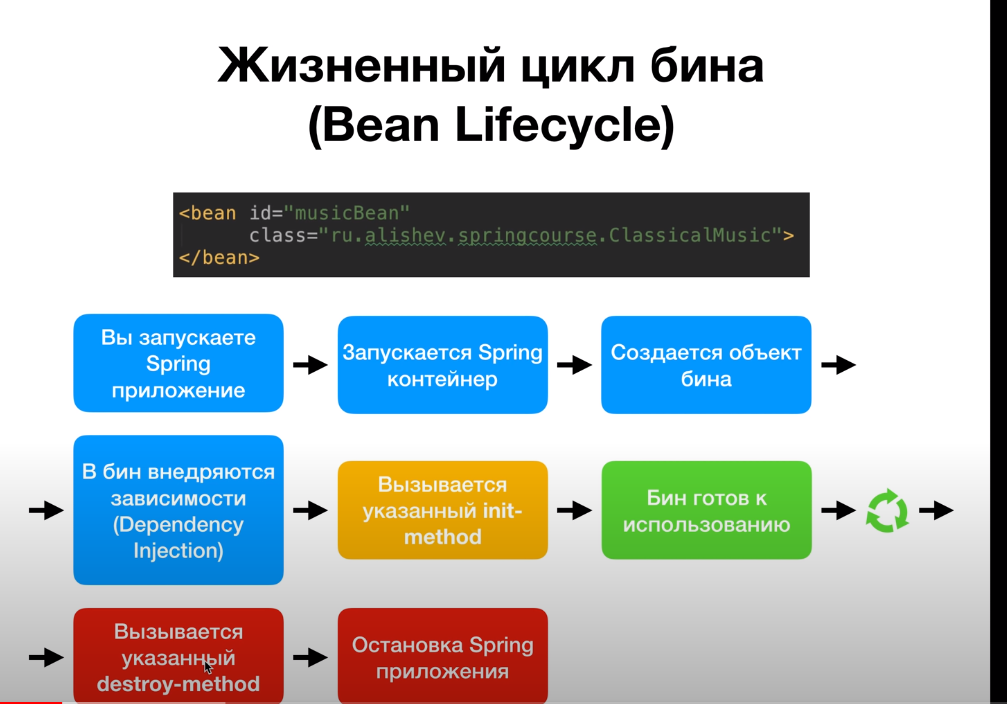
package ru.SpringSingleton;  
  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
import useClass.Music;  
import useClass.MusicPlayer;  
  
public class StartClass {  
 public static void main(String[] args) {  
  
 ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext("application.xml");  
  
 MusicPlayer player1 = context.getBean("musicPlayer", MusicPlayer.class);  
 MusicPlayer player2 = context.getBean("musicPlayer", MusicPlayer.class);  
 player2.getCompos();  
   
 player1.setCompos("Corsica");  
  
 player1.getCompos();  
 player2.getCompos();  
  
 }  
}

* 1. Prototype

<!-- Bean musicPlayer -->  
<bean id="musicPlayer"  
 class="useClass.MusicPlayer"  
 scope="prototype">  
 <constructor-arg ref="musicBean"/>  
 <property name="compos" value="${name}"/>  
</bean>

(По умолчанию стоит Singleton, но в бине с помощью метода scope можно изменить на prototype (для каждой переменной свой объект).

1. Жизненный цикл бина



(init-method вызывается, когда бин уже создан, но не передан пользователю)

(destroy-method вызывается перед уничтожением бина)



public class MusicPlayer {  
 private Music music = null;  
 private String compos = "Unknown";  
  
 MusicPlayer(Music music) {  
 this.music = music;  
 }  
  
 public void doInMyInit() {  
 System.*out*.println("Init method");  
 }  
  
 public void doInMyDestroy() {  
 System.*out*.println("Destroy method");  
 }

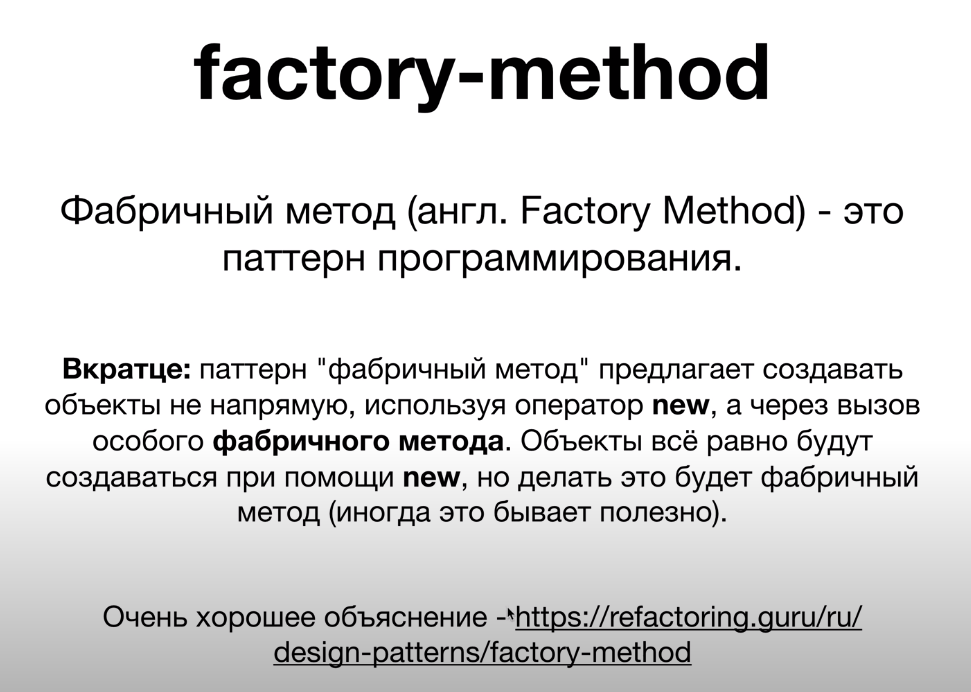
(Доопределяем методы в нашем классе бина)

<!-- Bean musicPlayer -->  
<bean id="musicPlayer"  
 class="useClass.MusicPlayer"  
 scope="prototype"  
 init-method="doInMyInit"  
 destroy-method="doInMyDestroy">  
  
 <constructor-arg ref="musicBean"/>  
 <property name="compos" value="${name}"/>  
</bean>

(Определяем методы в xml файле)

(При scope=”prototype” мы сами должны уничтожать наши объекты, метод destroy-method не работает)

Factory method (Паттерн программирования)



Java аннотации

1. Чтобы не конфигурировать классы в xml вручную, опишем их в коде, но для этого в xml, тоже укажем, где искать описанные классы.

<beans xmlns="http://www.springframework.org/schema/beans" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xmlns:aop="http://www.springframework.org/schema/aop"  
 xmlns:context="http://www.springframework.org/schema/context" xmlns:jee="http://www.springframework.org/schema/jee"  
 xsi:schemaLocation="  
 http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-3.0.xsd  
 http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.0.xsd  
 http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-3.0.xsd  
 http://www.springframework.org/schema/jee http://www.springframework.org/schema/jee/spring-jee-3.0.xsd">  
  
 <context:component-scan base-package="Start"/>  
</beans>

Код класса:

package Start;  
import org.springframework.stereotype.Component;  
  
  
@Component  
public class ClassicMusic implements Music {  
 @Override  
 public String getSongs() {  
 return "Bohemia Rhapsody";  
 }  
}

Аннотация @Autowired

1. Аннотация нужна для внедрения зависимостей, в бины.
2. С помощью рефлексии может придавать значения даже приватным полям.

package Start;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Component;  
  
@Component  
public class MusicPlayer {  
 private Music music = null;  
  
 @Autowired  
 public MusicPlayer(RockMusic music) {  
 this.music = music;  
 }  
  
 public Music getMusic() {  
 return music;  
 }  
  
 @Autowired  
 public void setMusic(ClassicMusic musics){  
 System.*out*.println(musics.getSongs());  
 }  
  
 public void playMusic(){  
 System.*out*.println(music.getSongs());  
 }  
   
}

(Использовали @Authowired)

package Start;  
  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
  
public class main {  
 public static void main(String[] args) {  
 ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext("application.xml");  
  
 MusicPlayer player = context.getBean("musicPlayer", MusicPlayer.class);  
 player.playMusic();  
 }  
}

(код класса)

@Autowired в методах

package Start;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Component;  
  
@Component("mus")  
public class MusicPlayer {  
 private Music music = null;  
  
 @Autowired  
 public MusicPlayer(ClassicMusic music) {  
 this.music = music;  
 System.*out*.println(this.music.getSongs());  
 }  
  
 public Music getMusic() {  
 return music;  
 }  
  
 @Autowired  
 public void setMusic(RockMusic musics) {  
 this.music = musics;  
 System.*out*.println(this.music.getSongs());  
 }  
  
 public void playMusic() {  
 System.*out*.println(music.getSongs());  
 }  
  
}

(При создании бина все методы, помеченные @Autowired будут выполнены, в порядке их появления в коде, но начиная с создания конструктора).

Несколько параметров в конструкторе

package Start;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Component;  
  
@Component("mus")  
public class MusicPlayer {  
 private RockMusic music;  
 private ClassicMusic music1;  
 private int id;  
  
 @Autowired  
 public MusicPlayer(RockMusic mus1, ClassicMusic mus2){  
 this.music = mus1;  
 this.music1 = mus2;  
   
 }  
  
 public void playMusic() {  
 System.*out*.println(music.getSongs());  
 System.*out*.println(music1.getSongs());  
 }  
  
}

Аннотация @Qualifier

1. Что бы компилятор понял, какой бин нужен нам в (конструкторе, методе, переменной) используем @Qualifier

@Component("mus")  
public class MusicPlayer {  
 @Autowired  
 @Qualifier("rockMusic")  
 private Music music;  
   
 public void playMusic() {  
 System.*out*.println(music.getSongs());  
 }  
}

Теперь рассмотрим пример для нескольких переменных

package Start;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.beans.factory.annotation.Qualifier;  
import org.springframework.stereotype.Component;  
  
@Component("mus")  
public class MusicPlayer {  
 private Music music1;  
 private Music music2;  
  
  
 public MusicPlayer(@Qualifier("rockMusic") Music music1, @Qualifier("classicMusic") Music music2) {  
 this.music1 = music1;  
 this.music2 = music2;  
 }  
  
 public void playMusic() {  
 System.*out*.println(music1.getSongs());  
 System.*out*.println(music2.getSongs());  
 }  
}