**SPRING SOME MATTERS**

**На заметку!** Аннотация @Autowired может быть применена только к одному из методов

конструкторов. В случае ее применения к нескольким методам конструкторов

платформа Spring выдаст соответствующее сообщение во время начальной загрузки

ApplicationContext.

@Service("constructorConfusion")

puЫic class ConstructorConfusion {

private String someValue;

puЫic ConstructorConfusion(String someValue) {

System.out.println("ConstructorConfusion(String) called");

this.someValue = someValue;

**@Autowired**

puЫic ConstructorConfusion (@Value (" 90") int someValue) {

**Внедрение значений с использованием SpEL (стиль XML)**

<bean id="injectSimpleConfig"

class="com.apress.prospring4.ch3.xml.InjectSimpleConfig"/>

<bean id="injectSimpleSpel"

class="com.apress.prospring4.ch3.xml.InjectSimpleSpel"

p:name="#{injectSimpleConfig.name}"

p:age="#{injectSimpleConfig.age}"

p:height="#{injectSimpleConfig.height}"

p:programmer="#{injectSimpleConfig.programmer}"

p:ageinSeconds="#{injectSimpleConfig.ageinSeconds}"/>

puЫic class **InjectSimpleConfig** {

private String name **=**

11 Chris Schaefer11;

private int age **=** 32;

private float height = 1.778f;

private boolean programmer = true;

private Long ageinSeconds **=** 1009843200L;

puЫic String getName ()

return name;

puЫic void setName (String name) {

this.name = name;

puЫic int getAge ()

return age;

puЫic void setAge ( int age)

this.age = age;

puЬlic float getHeight()

return height;

puЫic **class InjectSimpleSpel**

private String name;

private int age;

private float height;

private boolean programmer;

private Long ageinSeconds;

puЬlic String getName()

return this.name;

puЫic void setName(String name)

this.name **=** name;

puЫic int getAge ()

return this.age;

puЫic void setAge (int age)

this.age =

**Внедрение значений с использованием SpEL (стиль аннотаций)**

@Service("injectSimpleSpel")

puЫic class InjectSimpleSpel {

@Value("#{injectSimpleConfig.name}")

private String name;

@Value("#{injectSimpleConfig.age + 1}")

private int age;

@Value("#{injectSimpleConfig.height}")

private float height;

@Value("#{injectSimpleConfig.programmer}")

private boolean programmer;

@Value("#{injectSimpleConfig.ageinSeconds}")

private Long ageinSeconds;

**атрибуга local and bean and parent дескриптора <ref>.**

Как будет

показано позже в разделе "Именование бинов", бину можно назначать более одного

имени и ссылаться на него с помощью множества псевдонимов. Применение

атрибуга local означает, что дескриптор <ref> всегда просматривает только идентификатор

бина и никогда не принимает во внимание его псевдонимы. Более того,

определение бина должно существовать в том же самом ХМL-файле конфигурации.

Чтобы внедрить бин по любому имени или импортировать бин из другого ХМLфайла

конфигурации, вместо атрибута local в дескрипторе <ref> необходимо

применять атрибуг bean.

<bean id="oracle"

name="wiseworm" class="com.apress.prospring4.ch3.Bookworm0racle"/>

<bean id="injectRef" class="com.apress.prospring4.ch3.xml.InjectRef">

<property name="oracle">

**<ref local**="oracle"/>

</property>

</bean>

<bean id="oracle" name="wiseworm" class="com.apress.prospring4.ch3.

BookwormOracle"/>

<bean id="injectRef" class="com.apress.prospring4.ch3.xml.InjectRef">

<property name="oracle">

<**ref bean**="wiseworm" />

</property>

</bean>

<bean id="target2" class="com.apress.prospring4.ch3.SimpleTarget"

p:val-ref="injectBean"/>

<bean id="targetЗ" class="com.apress.prospring4.ch3.SimpleTarget">

<property name="val ">

<**ref parent="**injectBean"/>

</property>

</bean>

<bean id="injectBean" class="java.lang.String" с: O="Child In Bean"/>

***Применение аннотации @Resource при внедрении коллекций***

<context:annotation-config/>

<context:component-scan

base-package="com.apress.prospring4.ch3.annotation"/>

<util:map id="map" map-class="java.util.HashMap">

<entry key="someValue">

<value>Hello World!</value>

</entry>

<entry key="someBean">

<ref bean="oracle"/>

</entry>

</util:map>

<util:properties id="props">

<prop key="firstName">Chris</prop>

<prop key="secondName">Schaefer</prop>

</util:properties>

<util: set id="set">

<value>Hello World!</value>

<ref bean="oracle"/>

</util: set>

<util:list id="list">

<value>Hello World!</value>

<ref bean="oracle"/>

</util:list>

мы внедряем ранее показанные бины и используем аннотации

**@Resource стандарта JSR-250** с указанием в них имен

puЫic class Collectioninjection {

@Resource(name="map")

private Map<String, Object> map;

@Resource(name="props")

private Properties props;

@Resource(name="set")

private Set set;

@Resource(name="list")

private List list;

**На заметку!** Вас может удивить, почему вместо @Au tow i red применяется аннотация

@Resource. Причина в том, что аннотация @Autowired семантически определена так, что

она всегда трактует массивы, коллекции и карты как наборы соответствующих бинов, с целевым

типом бина, производным от объявленного типа значений в коллекции. Таким образом,

например, если класс имеет атрибут типа List<Oracle> и определена аннотация

@Autowired, то платформа Spring попытается внедрить все бины типа Oracle из текущего

контекста ApplicationContext в этот атрибут (вместо <util: list>, объявленного

в конфигурационном файле), что приведет либо к внедрению неожиданных зависимостей,

либо к генерации платформой Spring исключения при отсутствии бинов типа Oracle.

В результате для внедрения типа коллекции понадобится явно сообщить Spring о необходимости

выполнить внедрение, указав имя бина, что и поддерживает аннотация @Resource.

***внедрением через метод (Method lnjection)***

***Lookup Method Injection (Внедрение через метод поиска) и Method Replacement (Замена метода).***

Форма **Lookup Method lnjection** поддерживает еще один механизм, с помощью которого бин может получить одну из своих зависимостей.

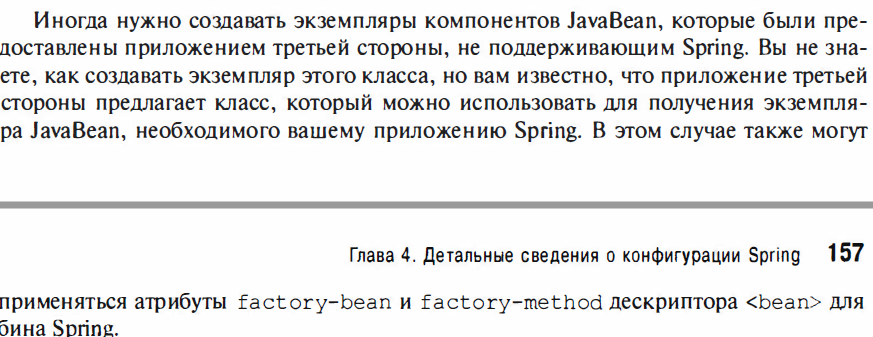
Форма **Method Replacement** позволяет заменять реализацию любого метода

бина произвольным образом, без необходимости в изменении первоначального

исходного кода. Для обеспечения этих двух средств Spring применяет возможности

динамического расширения байт-кода, обеспечиваемые библиотекой **CGLIB**.

***Yow to use factory-bean and factory-method attributes in Spring?***

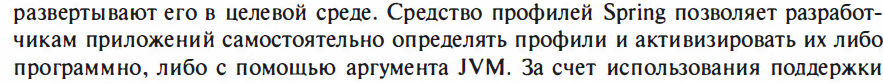


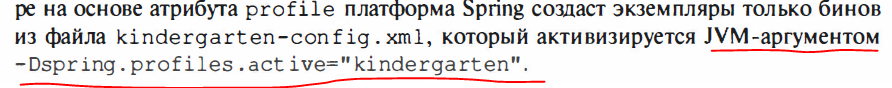


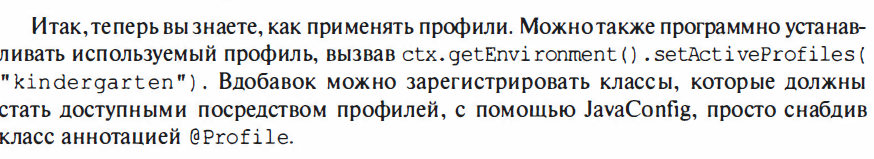
***How to use profile conception in Spring?***



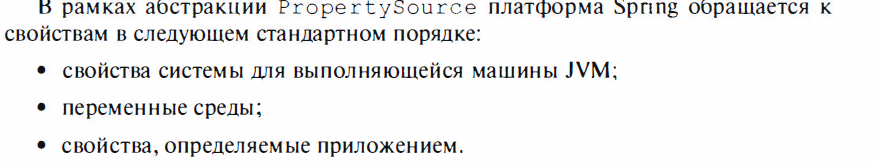
***How to set profile in Spring?***



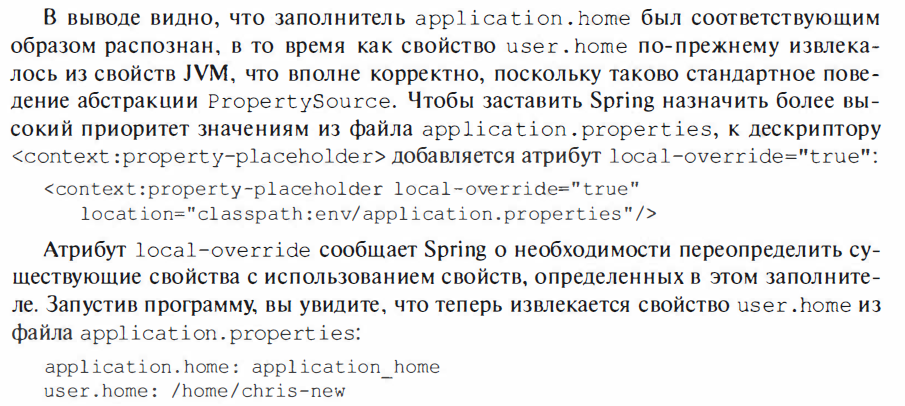


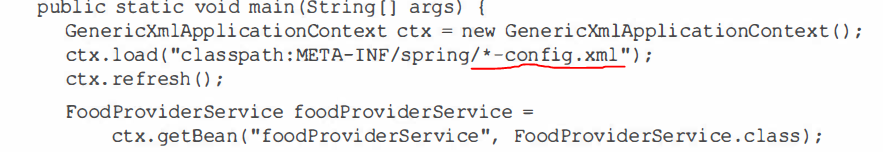


***How does spring read properties, I mean, in what order spring reads properties?***



***How to read data from property file and override jvm properties with local ones from the property file?***





**Управление транзакциями**

***Локальные транзакции***специфичны для отдельного транзакционноrо

ресурса (к примеру, подключения JDBC), тогда как ***глобальные транзакции***

управляются контейнером и могут охватывать несколько транзакционных

ресурсов.

-транзакции, управляемые контейнером (**container-managed transaction - СМТ**), для управления транзакциями декларативным путем.

-иметь больший контроль над транзакцией и работают с транзакциями, управляемыми бином (**bean managed transaction - ВМТ**), чтобы управлять ими программным путем.

АРl-интерфейса транзакций Java (**Java Transaction APl - JTA**).

управление транзакциями является сквозной функциональностью и не должно кодироваться внутри бизнеслоrики. Самый подходящий способ реализации управления транзакциями – дать возможность разработчикам определить требования к транзакциям декларативным путем и позволить такой инфраструктуре, как Spring, JEE или АОП, самостоятельно связать логику управления транзакциями.

***Декларативные и программные транзакции в Spring***

В Spring есть три варианта управления транзакциями. Два из них предназначены

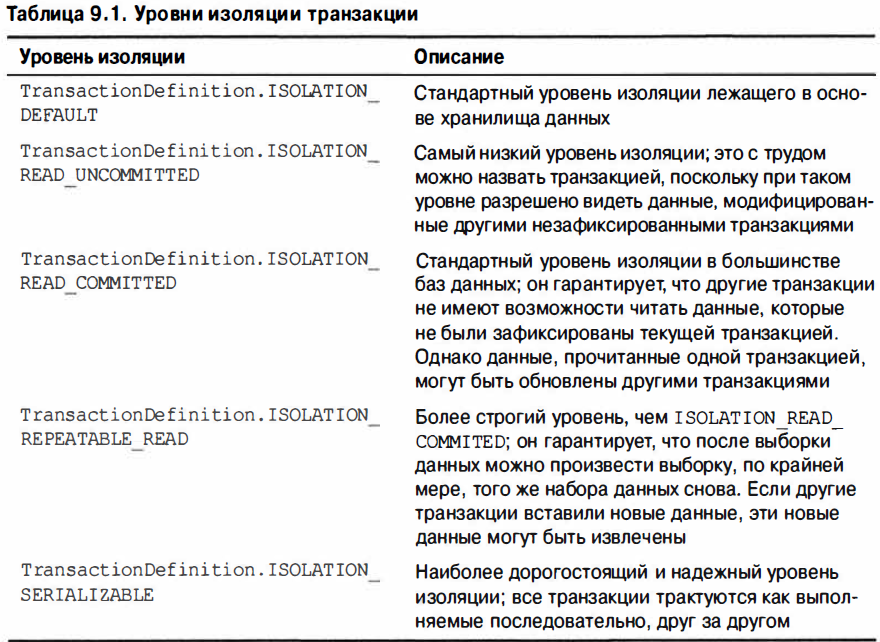
для декларативного управления транзакциями, причем в одном варианте используются

Jаvа-аннотации, а в другом - ХМL-конфигурация. Третий вариант предусматривает

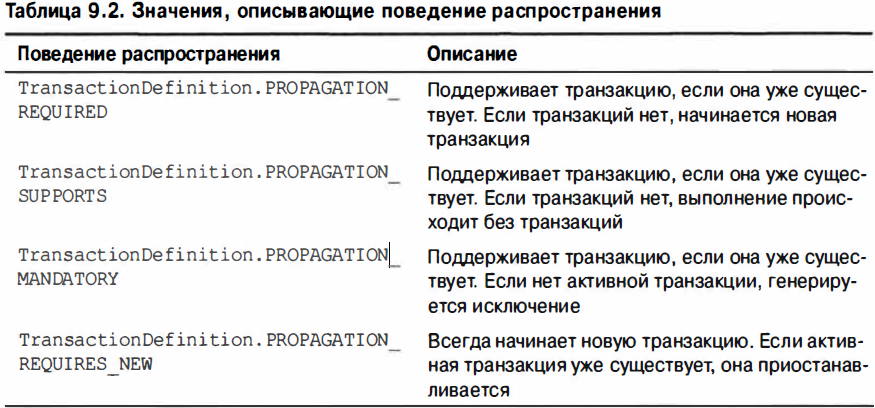
программное управление транзакциями. Все три варианта рассматриваются в

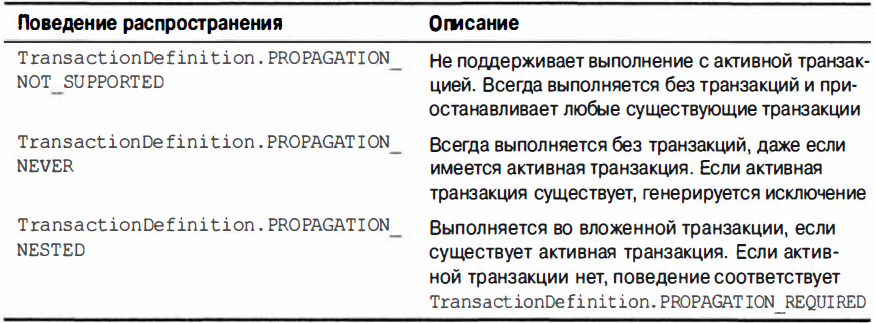
последующих разделах.

***Уровни изоляции транзакции***



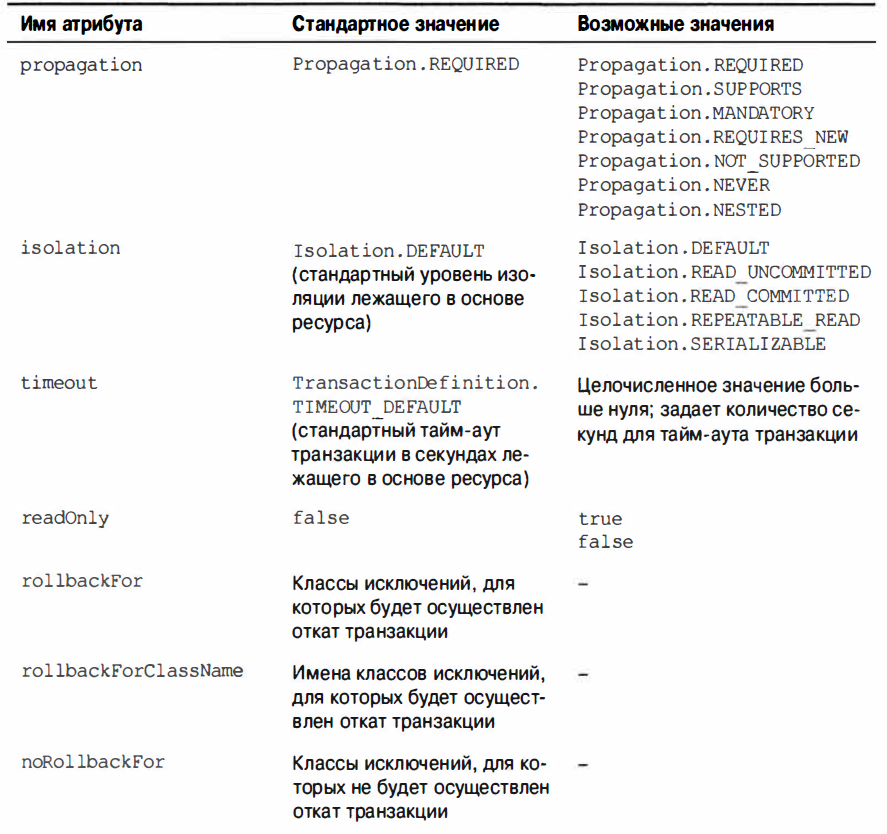
***Propagation***

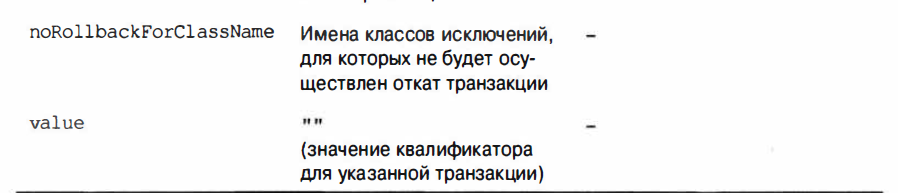




***Атрибуты для аннотации @Transactional***

В специальных случаях может понадобиться определять тайм-аут, уровень изоляции , выполнять ли откат (или нет) для специфических исключений, а также другие параметры .





**Но мы не хотим, чтобы в транзакции участвовала функция** countAll ().

**Причина в том, *что* ее результат не должен управляться диспетчером сущностей JPA.**

**Вместо этого нужно просто получить счетчик и забыть о нем. В таком случае можно**

**переопределить поведение распространения транзакции на** Propagation. NEVER.

