СБЕРБАНК ТЕХНОЛОГИИ

Spring framework

УЗНАЕМ

СБЕРБАНК ТЕХНОЛОГИИ

- Что такое Spring
- IoC контейнер Spring
- Spring Expression Language (SpEL)
- AOP в Spring

ЧТО ТАКОЕ SPRING

СБЕРБАНК ТЕХНОЛОГИИ

Spring Framework – облегчённая платформа для построения enterprise приложений на JAVA.

- Можно применять к любому java приложению, не привязана к WEB
- Облегчённая – не размер дистрибутива, а степень воздействия на код
- Модульная структура (IoC, WEB, Data Access, Messaging, …)

ЧТО ТАКОЕ SPRING

СБЕРБАНК ТЕХНОЛОГИИ

Позволяет создавать приложения из POJO объектов и инвазивно применять enterprise сервисы к нему.

Примеры:

- Метод работающий в DB транзакции без явного управления ими
- Метод работающий как RPC без явного воздействия через remote API
- Метод обрабатывающий сообщения без явного воздействия через JMS API

СБЕРБАНК ТЕХНОЛОГИИ

Ядро Spring Framework основано на принципе *инверсии управления* (Inversion of Control - IoC), когда создание и управление зависимостями между компонентами становятся внешними.

Мартин Фаулер назвал процесс внедрения зависимостей во время выполнения, приводящее к инверсии управления внедрением зависимостей (Dependency Injection - DI).

Реализация DI в Spring основана на двух концепциях:

- JavaBean
- Интерфейсы

В Spring любой управляемый ресурс – это bean.

С помощью интерфейсов можно получить максимальную отдачу от DI: бины могут использовать любую реализацию интерфейса для удовлетворения их зависимости.

Конфигурирование через XML или классы Java, или аннотациии в коде, или через Groovy.

СБЕРБАНК ТЕХНОЛОГИИ

Основные преимущества DI:

- Сокращение объема связующего кода
- Упрощенная конфигурация приложения
- Возможность управления общими зависимостями в единственном репозитории
- Улучшенная возможность тестирования

ЧТО ЕЩЁ?

СБЕРБАНК ТЕХНОЛОГИИ

Широчайший набор средств и инструментов:

- АОП (аспектно-ориентированное программирование)
- SpEL (Spring Expression Language)
- Валидация - Проверка достоверности
- Доступ к данным реляционным БД, NoSQL, графовые базы данных и документные базы данных
- Управление транзакциями
- MVC на веб-уровне
- Поддержка WebSocket

ЧТО ЕЩЁ?

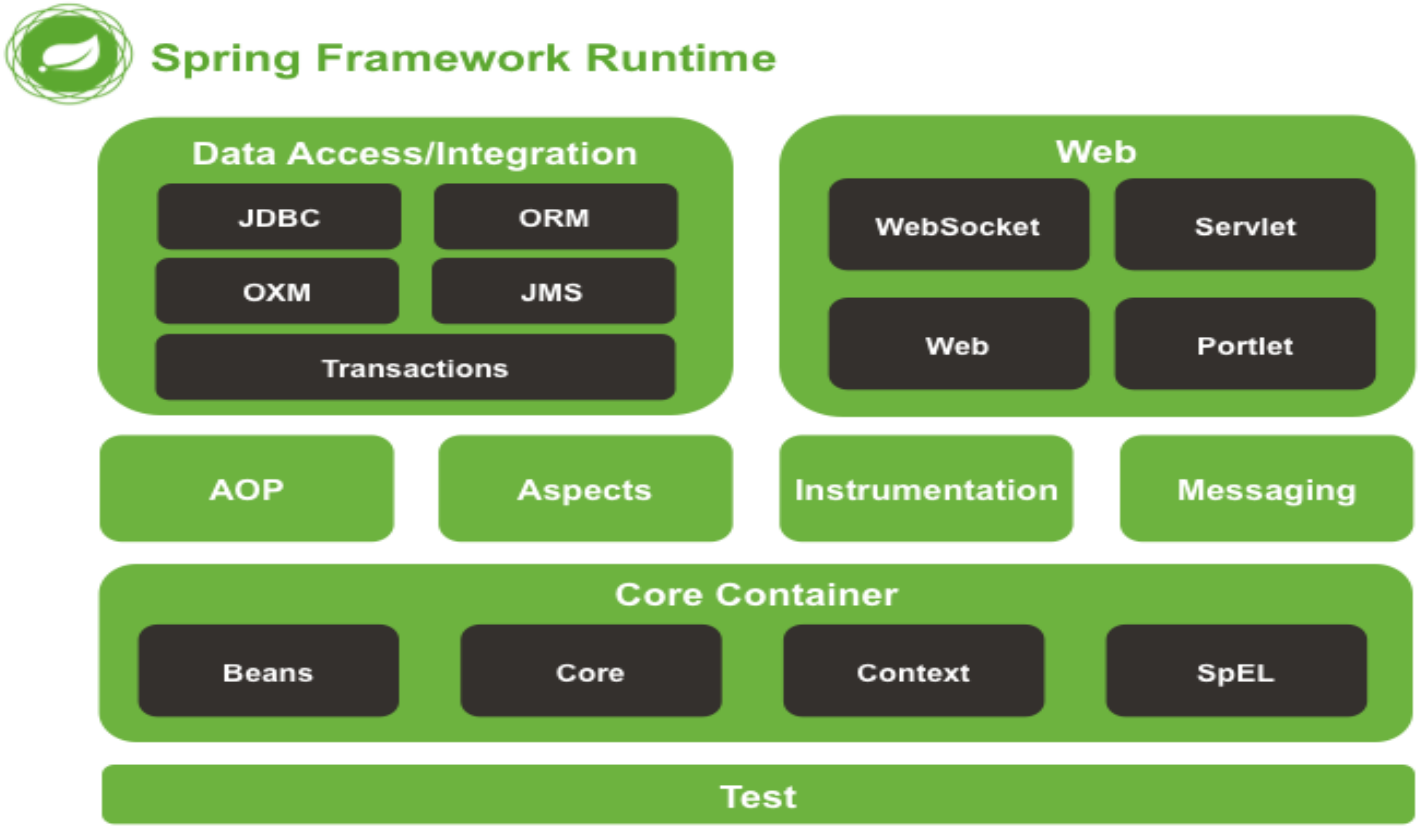
СБЕРБАНК ТЕХНОЛОГИИ

а так же:

- Поддержка удаленных технологий (RMI, JAX-WS, JMS, AMQP, REST, …)
- Поддержка электронной почты
- Поддержка планирования заданий
- Упрощенная обработка исключений

МОДУЛИ SPRING

СБЕРБАНК ТЕХНОЛОГИИ



ТИПЫ IOC

СБЕРБАНК ТЕХНОЛОГИИ

Инверсия управления делится на типа:

1. Dependency Lookup - компонент должен получить ссылку на зависимость
2. Dependency Injection - зависимости внедряются в компонент контейнером IoC

ТИПЫ DEPENDENCY LOOKUP

СБЕРБАНК ТЕХНОЛОГИИ

Разновидности Dependency Lookup:

- Dependency Pull (Извлечение зависимостей)
- Contextualized Dependency Lookup (Контекстуализированный поиск зависимостей)

DEPENDENCY PULL

СБЕРБАНК ТЕХНОЛОГИИ

Пример Dependency Pull в Spring:

```
ApplicationContext ctx = new ClassPathXmlApplicationContext
        ("META-INF/spring/app-context.xml");

MessageRenderer mr = ctx.getBean("renderer",
MessageRenderer.class);

mr.render();
```

CONTEXTUALIZED DEPENDENCY LOOKUP

СБЕРБАНК ТЕХНОЛОГИИ

Пример Contextualized Dependency Lookup в Spring:

```
public class MessageRenderer
                        implements ApplicationContextAware {
    private Dependency dependency;

    @Override
    public void setApplicationContext(ApplicationContext ctx)
            throws BeansException {
        this.dependency = ctx.getBean(Dependency.class);
    }
}
```

ТИПЫ DEPENDENCY INJECTION

СБЕРБАНК ТЕХНОЛОГИИ

Разновидности Dependency Injection:

- Constructor Dependency Injection
- Setter Dependency Injection

CONSTRUCTOR DEPENDENCY

СБЕРБАНК ТЕХНОЛОГИИ

Пример Constructor Dependency в Spring:

```
@Component
public class ConstructorInjection {
    private final Dependency dependency;

    @Autowired
    public ConstructorInjection(Dependency dependency) {
        this.dependency = dependency;
    }
}
```

SETTER DEPENDENCY

СБЕРБАНК ТЕХНОЛОГИИ

Пример Setter Dependency в Spring:

```
// SetterInjection.java
public class SetterInjection {
    private Dependency dependency;
    public void setDependency(Dependency dependency) {
        this.dependency = dependency;
    }
}
<!-- spring-context.xml-->
<bean id="dependency" class="ru.sbrf.Dependency" />

<bean id="setterInjection" class="ru.sbrf.SetterInjection">
    <property name="dependency" ref="dependency"/>
</bean>
```

ЧТО ИСПОЛЬЗОВАТЬ?

СБЕРБАНК ТЕХНОЛОГИИ

Почему Dependency Injection:

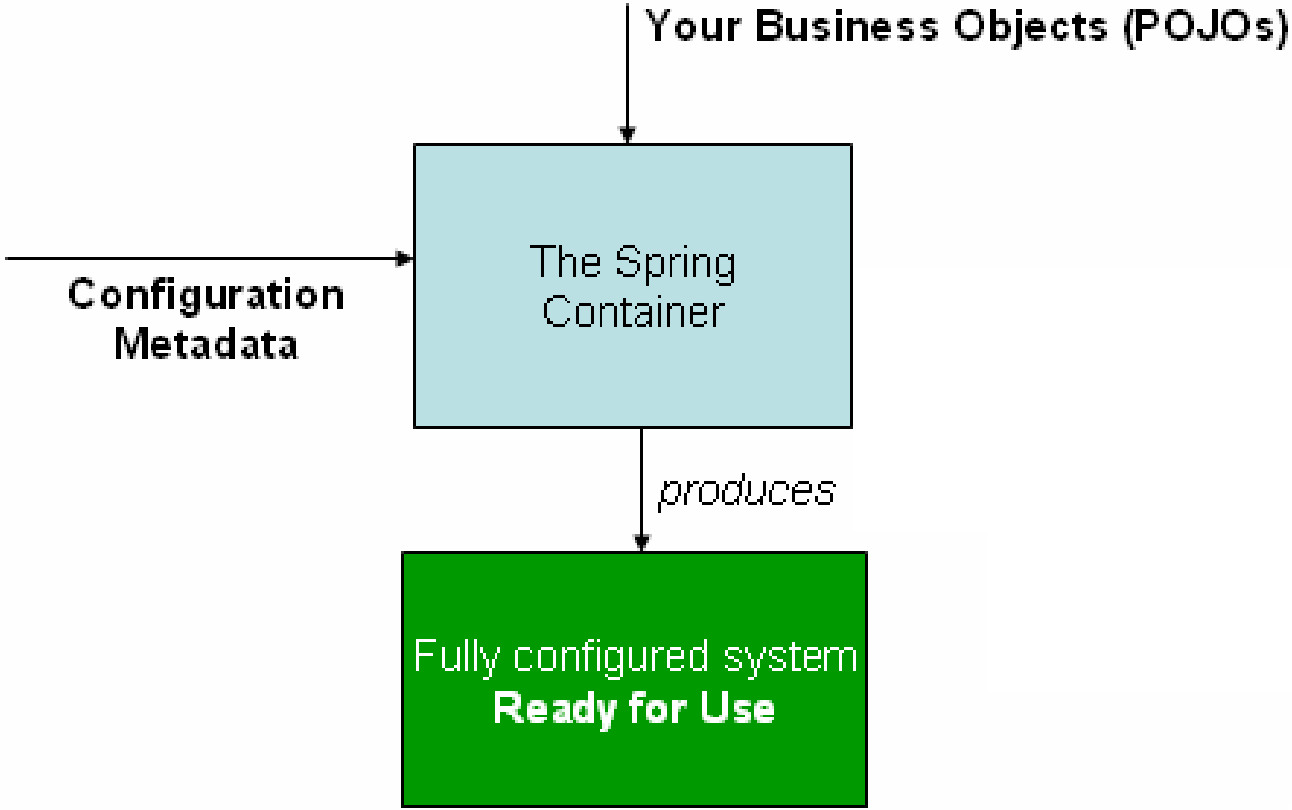
- Нулевое воздействие на код
- Лёгкость тестирования без привязки к контейнеру

Dependency Injection:

- Constructor Dependency Injection – гарантия предоставления всех зависимостей неизменяемому компоненту
- Setter Dependency Injection – ненавязчивое предоставление, у компонента есть стандартные настройки + возможность менять на лету

КАК РАБОТАЕТ В SPRING

СБЕРБАНК ТЕХНОЛОГИИ



SPRING IOC КОНТЕЙНЕР

СБЕРБАНК ТЕХНОЛОГИИ

Пакеты:

- org.springframework.beans
- org.springframework.context

Интерфейсы:

- BeanFactory - отвечает за управление компонентами, в том числе их зависимостями и жизненными циклами
- ApplicationContext – расширение beanfactory, дополнение к службам DI также предлагает другие службы, такие как служба транзакций и АОП, обработка событий приложения

Варианты конфигурирования:

- PropertiesBeanDefinitionReader – читает конфигурацию из файла свойств
- XmlBeanDefinitionReader – читает конфигурацию из XML

BEANFACTORY - ПРИМЕР

СБЕРБАНК ТЕХНОЛОГИИ

Вариант конфигурирования через XmlBeanDefinitionReader:

```
<!-- xml-bean-factory-config.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.xsd">

    <bean id="console" class="ru.sbrf.ConsoleMessageRenderer" />

</beans>
```

BEANFACTORY - ПРИМЕР

СБЕРБАНК ТЕХНОЛОГИИ

Вариант создание фабрики через XmlBeanDefinitionReader:

```
//Main.class
DefaultListableBeanFactory factory =
                                new DefaultListableBeanFactory();
XmlBeanDefinitionReader rdr = new XmlBeanDefinitionReader(factory);

rdr.loadBeanDefinitions(new
        ClassPathResource("xml-bean-factory-config.xml"));

MessageRenderer messageRender =
        factory.getBean("console", MessageRenderer.class);
```

Основные варианты конфигурирования:

- XML файл
- На базе аннотаций
- С помощью Java классов

КОНФИГУРАЦИЯ ЧЕРЕЗ XML ФАЙЛ

СБЕРБАНК ТЕХНОЛОГИИ

```
<!-- spring-configuration.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.xsd">
    <bean id="..." class="...">
        <!-- collaborators and configuration for this bean go here -->
    </bean>
    <!-- more bean definitions go here -->
</beans>
```

КОНФИГУРАЦИЯ ЧЕРЕЗ XML ФАЙЛ

СБЕРБАНК ТЕХНОЛОГИИ

```
//Main.class
ApplicationContext context =
        new ClassPathXmlApplicationContext("spring-configuration.xml");
// retrieve configured instance
MessageRenderer service =
                context.getBean("console", MessageRenderer.class);
```

СВОЙСТВА В <BEAN/>

СБЕРБАНК ТЕХНОЛОГИИ

Основные свойства BeanDefinition, которые можно задать:

- Имя класса имплементации
- Поведение бина (scope, lifecycle callbacks, and so forth)
- Ссылки на зависимые бины
- Другие специфичные свойства

ИМЕНОВАНИЯ БИНОВ

СБЕРБАНК ТЕХНОЛОГИИ

Приоритет разыменования бинов:

```
<bean id="someBean" ...
<bean name="someBean" ...
<bean class="examples.Command" ...
```

Создание псевдонима имени бина:

```
<alias name="fromName" alias="toName"/>
```

ВАРИАНТЫ СОЗДАНИЯ БИНОВ

СБЕРБАНК ТЕХНОЛОГИИ

```
<!-- Создание через конструктор-->
<bean id="exampleBean" class="examples.ExampleBean"/>

<!-- Создание через статический метод-->
<bean id="clientService"
      class="examples.ClientService"
      factory-method="createInstance"/>

<!-- Создание через метод другого бина -->
<bean id="clientService"
      factory-bean="serviceLocator"
      factory-method="createClientServiceInstance"/>
```

ВАРИАНТЫ ВНЕДРЕНИЯ ЗАВИСИМОСТЕЙ

СБЕРБАНК ТЕХНОЛОГИИ

```
<!-- Внедрение через конструктор-->
<bean id="dependsBean" class="examples.DependsBean"/>
<bean id="targetBean" class="examples.TargetBean">
    <constructor-arg ref="dependsBean"/>
    <constructor-arg type="int" value="1"/>
</bean>

<!-- Внедрение через метод -->
<bean id="targetBean" class="examples.TargetBean">
    <property name="beanOne" ref="dependsBean"/>
</bean>
```

ПАРАМЕТРЫ ВНЕДРЕНИЯ

СБЕРБАНК ТЕХНОЛОГИИ

```
<!-- Внедрение простых значений -->
<bean id="targetBean" class="examples.TargetBean"
      p:name="Andrey" p:age="31"/>

<!-- Внедрение значений через SpEL -->
<bean id="simpleCfg" class="examples.SimpleConfig"/>
<bean id="targetBean" class="examples.TargetBean"
      p:name="#{simpleCfg.name}" p:age="#{simpleCfg.age}"/>

<!-- Внедрение коллекций (map, list, set) -->
<bean id="targetBean" class="examples.TargetBean">
    <property name="map">
        <map>
            <entry key="key" value="value"/>
        </map>
    </property>
</bean>
```

СБЕРБАНК ТЕХНОЛОГИИ

Внедрение через метод (Lookup Method)

```
//CommandManager.java
public abstract class CommandManager {
    protected abstract Command createCommand();
    public void run() {
        while(...){
            Command cmd = createCommand();
            cmd.execute();
        }
    }
}
```

```
<!-- spring-configuration.xml -->
<bean id="command" class="examples.Command" scope="prototype"/>
<bean id="targetBean" class="examples.TargetBean">
    <lookup-method name="createCommand" bean="command"/>
</bean>
```

ДОПОЛНИТЕЛЬНОЕ УПРАВЛЕНИЕ ВНЕДРЕНИЕМ

СБЕРБАНК ТЕХНОЛОГИИ

Неявные зависимости можно задать:

```
<bean id="dependsBean" class="examples.DependsBean"/>
<bean id="exampleBean" class="examples.ExampleBean"
                                    depends-on="dependsBean"/>
```

СБЕРБАНК ТЕХНОЛОГИИ

Ленивая инициализация бинов:

```
<!-- В рамках одного бина -->
<bean id="lazy" class="com.foo.ExpensiveToCreateBean"
                                                lazy-init="true"/>

<!-- В рамках скопа бинов -->
<beans default-lazy-init="true">
    <!-- no beans will be pre-instantiated... -->
</beans>
```

АВТОСВЯЗЫВАНИЕ БИНОВ

СБЕРБАНК ТЕХНОЛОГИИ

- Spring может автоматически определить связи между бинами

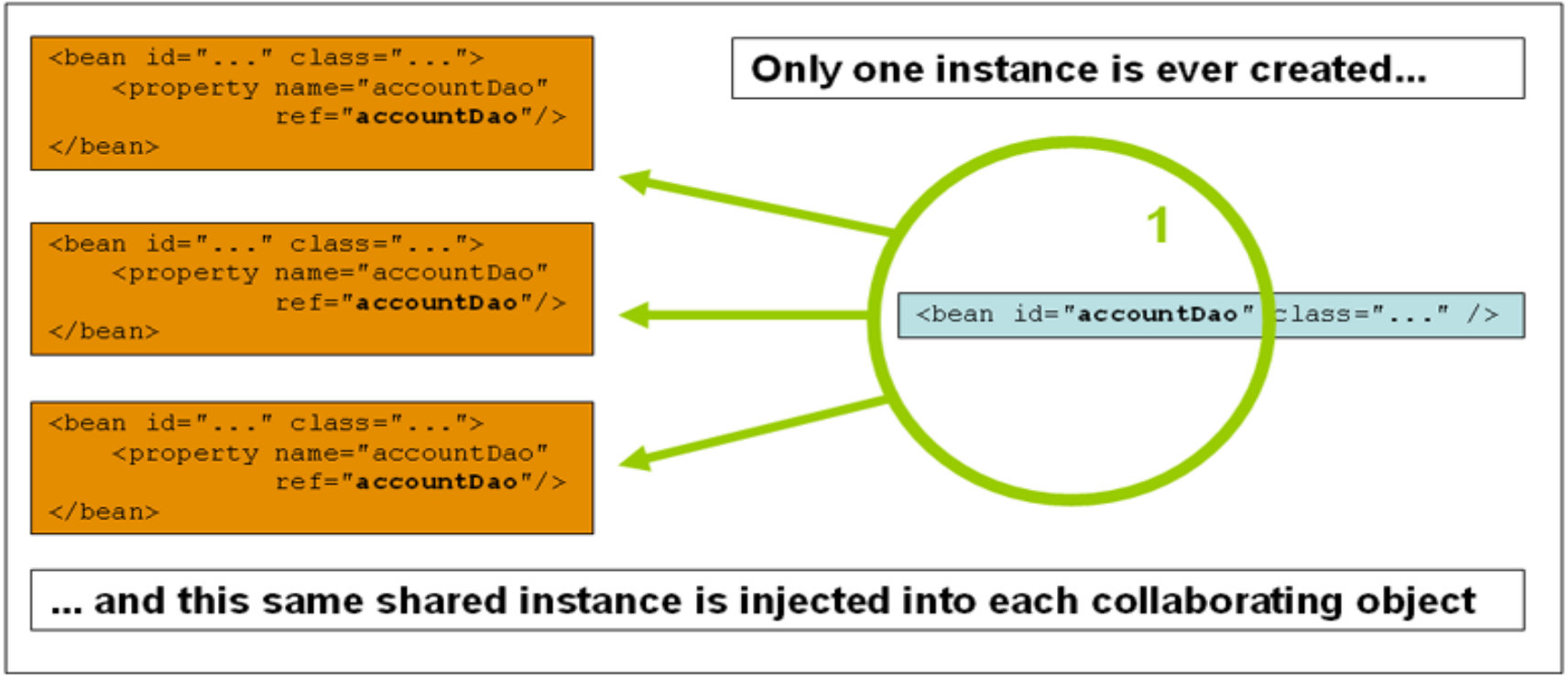
```
<bean id="exampleBean" class="examples.ExampleBean"
                                    autowire="constructor"/>
```

- Режимы автосвязывания:
  - No – отключено (по-умолчанию)
  - byName – свойство == имени бина
  - byType – тип свойства == типу бина в контексте
  - Constructor – аналог byType только для конструктора

РЕЖИМЫ СОЗДАНИЯ БИНОВ

СБЕРБАНК ТЕХНОЛОГИИ

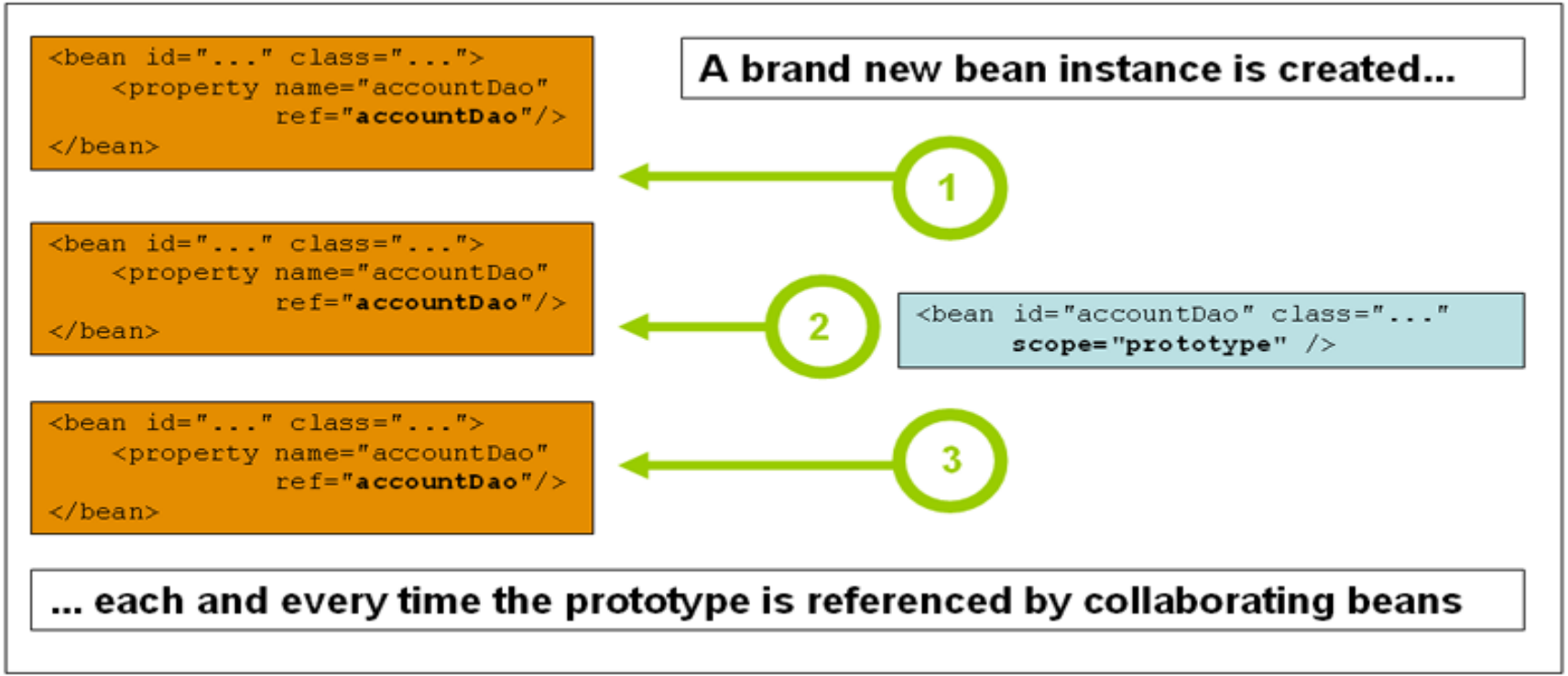
Singleton – один инстанс в рамках всего приложения



РЕЖИМЫ СОЗДАНИЯ БИНОВ

СБЕРБАНК ТЕХНОЛОГИИ

Prototype – новый объект при каждом его запросе (getBean())



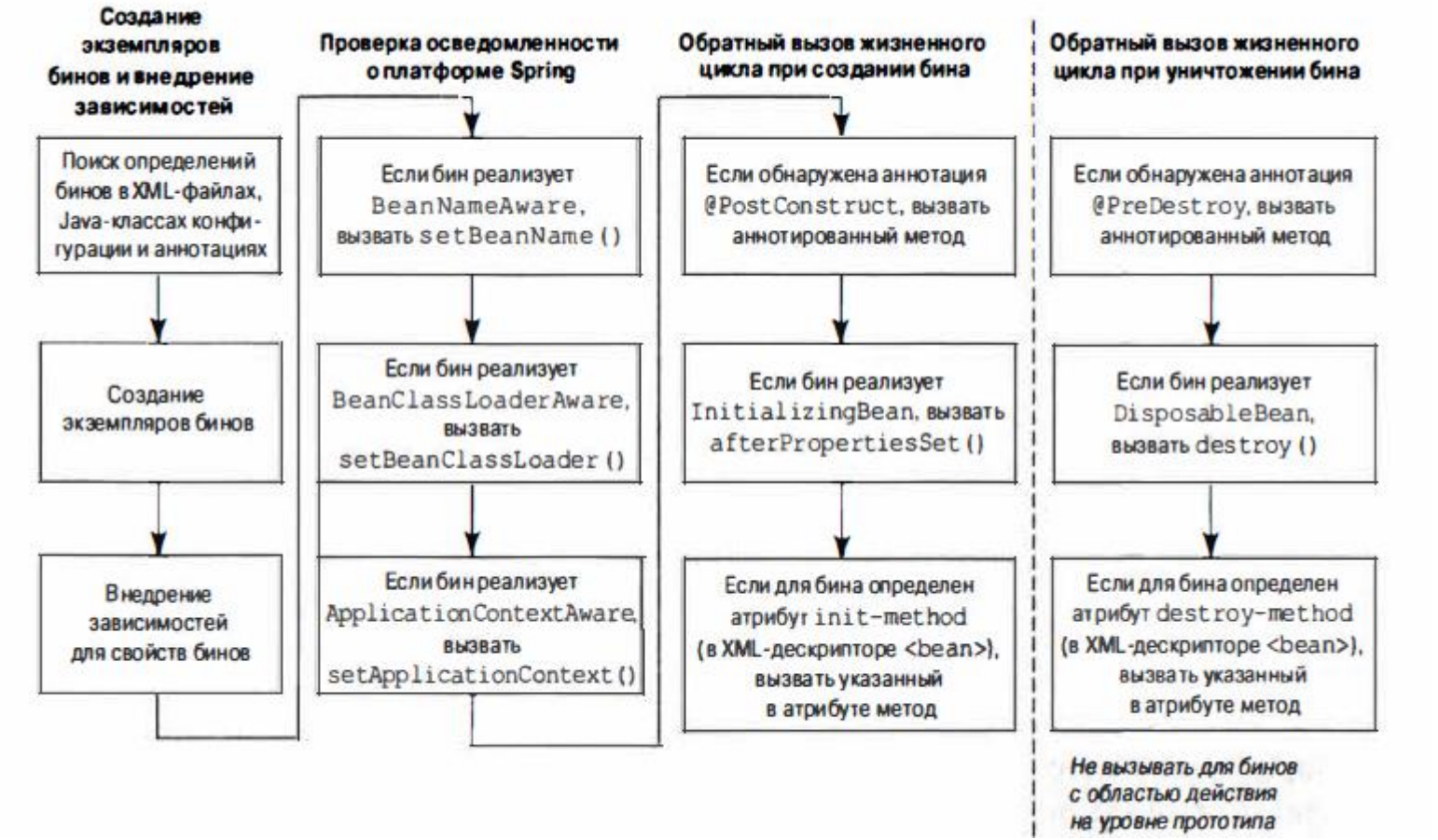
РЕЖИМЫ СОЗДАНИЯ БИНОВ - ИТОГИ

СБЕРБАНК ТЕХНОЛОГИИ

Область	Назначение
singleton	Будет создаваться только один объект на контейнер Spring IoC.
prototype	Платформа Spring будет создавать новый экземпляр, когда он запрашивается приложением
request	В Spring MVC – на время обработки HTTP запроса
session	В Spring MVC – на время обработки HTTP сеанса
globalSession	Для портлетных приложений – шарится между портлетами
websocket	На время жизни WebSocket
<собственная>	Реализация Scope интерфейса и его регистрация в ConfigurableBeanFactory

УПРАВЛЕНИЕ ЖИЗНЕННЫМ ЦИКЛОМ БИНОВ

СБЕРБАНК ТЕХНОЛОГИИ



CALLBACK НА ИНИЦИАЛИЗАЦИЮ БИНА

СБЕРБАНК ТЕХНОЛОГИИ

- Реализовать интерфейс InitializingBean:

```
void afterPropertiesSet() throws Exception;
```

- Указать метод инициализации:

```
<bean id="exampleBean" class="examples.ExampleBean"
                                    init-method="init"/>
```

- * Использование аннотации @PostConstruct на самом методе

- * Использование атрибута initMethod в аннотации @Bean при конфигурации через Java классы

CALLBACK НА УНИЧТОЖЕНИЕ БИНА

СБЕРБАНК ТЕХНОЛОГИИ

- Реализовать интерфейс DisposableBean:

```
void destroy() throws Exception;
```

- Указать метод инициализации:

```
<bean id="exampleBean" class="examples.ExampleBean"
                                destroy-method="destroy"/>
```

- * Использование аннотации @PreDestroy на самом методе

- * Использование атрибута destroyMethod в аннотации @Bean при конфигурации через Java классы

ДОПОЛНИТЕЛЬНЫЕ CALLBACK-И

СБЕРБАНК ТЕХНОЛОГИИ

Callback	Назначение
ApplicationContextAware	Получение в бине ссылку на текущий контекст
BeanNameAware	Получение ссылки собственного имени
BeanPostProcessor	Кастомизация логики создания бинов

КОНФИГУРИРОВАНИЕ НА БАЗЕ АННОТАЦИЙ

СБЕРБАНК ТЕХНОЛОГИИ

Определять бины можно так:

```
@Component
@Scope(BeanDefinition.SCOPE_PROTOTYPE)
public class AutoDetectedBean {

    private final Dependency dependency;

    @Autowired
    public AutoDetectedBean(Dependency dependency) {
        this.dependency = dependency;
    }
}
```

КОНФИГУРИРОВАНИЕ НА БАЗЕ АННОТАЦИЙ

СБЕРБАНК ТЕХНОЛОГИИ

Включить можно так:

```
<?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.xsd">

<context:annotation-config/>

</beans>
```

ОСНОВНЫЕ АННОТАЦИИ

СБЕРБАНК ТЕХНОЛОГИИ

Аннотация	Назначение
@Required	Внедряет обязательные зависимости по типу
@Autowired	Внедряет зависимости по типу
@Qualifier	Идентифицирует бины одинакового типа
@Resource	Внедряет зависимости по имени бина
@Component	Идентифицирует бин.
@Service	Идентифицирует бин на сервисном уровне
@Repository	Идентифицирует бин на DAO уровне
@Controller	Идентифицирует бин на уровне представления
@Scope	Определяет область действия бина
@Value	Внедряет дефолтные значения

КОНФИГУРИРОВАНИЕ С ПОМОЩЬЮ JAVA КЛАССОВ

СБЕРБАНК ТЕХНОЛОГИИ

Класс агрегирующий конфигурацию выглядит так:

```
@Configuration
public class ConfigExample {

    @Bean
    public Dependency dependency() {
        return new Dependency();
    }

    @Bean
    public MessageRenderer messageRenderer(Dependency dependency) {
        return new MessageRendererImpl(dependency);
    }
}
```

СОЗДАНИЕ КОНТЕКСТА

СБЕРБАНК ТЕХНОЛОГИИ

Инстанцировать контейнер на базе конфигурационного класса:

```
final ApplicationContext ctx =
        new AnnotationConfigApplicationContext(ConfigExample.class);

final MessageRenderer render = ctx.getBean(MessageRenderer.class);
render.execute();
```

ОСНОВНЫЕ АННОТАЦИИ

СБЕРБАНК ТЕХНОЛОГИИ

Аннотация	Назначение
@Configuration	Идентифицирует класс как фабрику бинов
@Bean	Идентифицирует метод создающий бин (аналог XML <bean/>)
@Scope	Определяет область действия бина
@PropertySource	Импортирования XML с конфигурацией
@ImportResource	Подключение файлов ресурсов
@Import	Импортирование других классов с конфигурацией
@ComponentScan	Аналог <context: component-scan>
@Profile	Подключает компонент только в заданном профиле

ENVIRONMENT И PROPERTYSOURCE

СБЕРБАНК ТЕХНОЛОГИИ

Environment – описывает окружение в котором работает приложение

Содержит следующую информацию:

- Текущие активные профили
- Текущие свойства полученные из разных источников (системные свойства, св-ва из файлов, JNDI, …)

СБЕРБАНК ТЕХНОЛОГИИ

Подключить собственный ресурс можно так:

```
@Configuration
@PropertySource("classpath:app.properties")
public class ConfigExample {
    @Autowired
    private Environment env;

    @Bean
    public TestBean testBean() {
        return new TestBean(env.getProperty(
                                    "ru.sbrf.testbean.timeout",
                                    Integer.class,
                                    10));
    }
```

SPRING EXPRESSION LANGUAGE (SPEL)

СБЕРБАНК ТЕХНОЛОГИИ

SpEL – унифицированный язык выражений, поддерживающий запросы и манипулирования графом объектов JAVA в runtime.

Что поддерживает:

- Переменные
- Различные операции
- Регулярные выражения
- Вызовы методов
- Присваивания
- Доступ к бинам контекста
- Массивы, списки, мапы

SPRING EXPRESSION LANGUAGE (SPEL)

СБЕРБАНК ТЕХНОЛОГИИ

Используется:

- В XML конфигурации
- В теле @Value
- На JSP страничках (Spring Web MVC)
- Для внесения изменяемой в рантайме логики в приложение

SPEL – ПРИМЕР

СБЕРБАНК ТЕХНОЛОГИИ

Пример использования напрямую через API:

```
ExpressionParser parser = new SpelExpressionParser();

Expression expr =
        parser.parseExpression("toUpperCase().substring(1, 5)");

EvaluationContext ctx =
        new StandardEvaluationContext(new String("Hello world"));

String result = expr.getValue(ctx, String.class);
```

SPEL – ПРИМЕР

СБЕРБАНК ТЕХНОЛОГИИ

Пример использования в XML конфигурации:

```
<bean id="exampleBean" class="ru.sbrf.ExampleBean">
    <property name="defLocale" value=
            "#{systemProperties['ru.sbrf.ExampleBean.defLocale']}"/>
</bean>
```

Пример использования в @Value:

```
@Value("#{ systemProperties['user.region'] }")
private String defaultLocale;
```

АОП – аспектно-ориентированное программирование. Инструмент внедрения сквозной функциональности.

СБЕРБАНК ТЕХНОЛОГИИ

Базовые понятия:

- Join point (точка соединения) – точка в runtime (в Spring АОП - метод)
- Advice (совет) – фрагмент кода выполняющийся в отдельной точке соединения
- Pointcut (срез) – условия выбора точек соединения (имя метода, …)
- Aspect (аспект) – комбинация совета и среза
- Target (объект цель) – объект к которому применяется аспект
- Weaving (связывание) – процесс регистрации аспектов с объектами целями
- AOP proxy (прокси объект) – объект с внедрёнными аспектами

Типы советов для точек соединения:

- Before advice – совет перед
- After returning advice – совет после нормального завершения
- After throwing advice – совет после выкидывания исключения
- After (finally) advice – совет после любого завершения
- Around advice – совет вместо

Типы связывания:

- Статическое - в compile time (напрмер - AspectJ)
- Динамическое – в run-time (Spring AOP)

Типы динамических прокси в Spring AOP:

- Стандартный JDK прокси (только для интерфейсов)
- CGLIB – для классов

СБЕРБАНК ТЕХНОЛОГИИ

Реализации АОП в Spring:

- Spring AOP API
- Поддержка @AspectJ

Spring AOP API – через реализацию интерфейсов, сильная связность с платформой, устарвшее API

@AspectJ – через набор аннотации библиотеки AspectJ

ПРИМЕР ЧЕРЕЗ SPRING AOP API

СБЕРБАНК ТЕХНОЛОГИИ

Определим совет around через MethodInterceptor:

```
public static class DisplayTimeIntercepter
                                    implements MethodInterceptor {
    @Override
    public Object invoke(MethodInvocation invocation)
                                                throws Throwable {
        try {
            System.out.println("Before: " +
                                    invocation.getMethod().getName());
            return invocation.proceed();
        }finally {
            System.out.println("After: " +
                                    invocation.getMethod().getName());
        }
    }
}
```

ПРИМЕР ЧЕРЕЗ SPRING AOP API

СБЕРБАНК ТЕХНОЛОГИИ

Использование:

```
final MessageRenderer target = new MessageRendererImpl();
final ProxyFactory pf = new ProxyFactory();
pf.setTarget(target);
pf.addAdvice(new DisplayTimeIntercepter());

final MessageRenderer proxy = (MessageRenderer)pf.getProxy();
proxy.execute();
```

СОВЕТЫ SPRING AOP API

СБЕРБАНК ТЕХНОЛОГИИ

Совет	Интерфейс
Before	MethodBeforeAdvice
After returning	AfterReturningAdvice
Around	MethodInterceptor
Throws	ThrowsAdvice
Introduction	IntroductionInterceptor

СРЕЗЫ SPRING AOP API

СБЕРБАНК ТЕХНОЛОГИИ

Основные интерфейсы для описания срезов:

```
public interface Pointcut {
    ClassFilter getClassFilter();
    MethodMather getMethodMatcher();
}

public interface ClassFilter {
    boolean matches(Class clazz);
}

public interface MethodMather {
    boolean matches(Method m, Class targetClass);
    boolean matches(Method m, Class targetClass, Object[] args);
    boolean isRuntime();
}
```

СРЕЗЫ ИЗ КОРОБКИ ДЛЯ SPRING AOP API

СБЕРБАНК ТЕХНОЛОГИИ

Класс, реализующий Pointcut	Описание
AspectJExpressionPointcut	Поддержка языка AspectJ
StaticMethodMatcherPointcut	Определяет статические точки
DynamicMethodMatcherPointcut	Определяет динамические точки (зависимость от аргументов)
ComposablePointcut	Объединение точек
JdkRegexpMethodPointcut	Поддержка регулярных выражений
AnnotationMatchingPointcut	Поддержка аннотаций Spring

ПРИМЕР ЧЕРЕЗ @ASPECTJ

СБЕРБАНК ТЕХНОЛОГИИ

Определим бин с аспектами:

```
@Component
@Aspect
public class AnnotatedLogInterceptor {

    @Around("execution(* longRunningMethod(..)")
    public Object invoke(MethodInvocation invocation)
                                                    throws Throwable {
        try {
            System.out.println("Before");
            return invocation.proceed();
        }finally {
            System.out.println("After");
        }
    }
}
```

ПРИМЕР ЧЕРЕЗ @ASPECTJ

СБЕРБАНК ТЕХНОЛОГИИ

Подключение аспекта в контекст:

```
@Configuration
@EnableAspectJAutoProxy
public class ConfigExample {
    //Other beans
}
```

Подключение зависимостей:

- org.springframework => spring-aop
- org.aspectj => aspectjrt
- org.aspectj => aspectjweaver

ПОДДЕРЖИВАЕМЫЕ ВЫРАЖЕНИЯ @ASPECTJ

СБЕРБАНК ТЕХНОЛОГИИ

Выражение @AspectJ	Описание
execution	На основе имени метода
within	Задаёт тип объекта
this	Интерфейс для прокси объекта
target	Интерфейс для целевого объекта
args	Задаёт типы аргументов метода
bean	Join point для бинов спринга
@annotation	Методы отмеченные аннотацией

ПОДДЕРЖИВАЕМЫЕ ВЫРАЖЕНИЯ @ASPECTJ

СБЕРБАНК ТЕХНОЛОГИИ

Советы	Выражение @AspectJ
Before	@Before
After returning	@AfterReturning
Throws	@AfterThrowing
After	@After
Around	@Around
Introduction	@DeclareParents

ЛИТЕРАТУРА

СБЕРБАНК ТЕХНОЛОГИИ

http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/