Spring 核心总结

一、 IoC(Inversion of control): 控制反转

1、IoC：

概念：控制权由对象本身转向容器；由容器根据配置文件去创建实例并创建各个实例之间的依赖关系

核心：bean 工厂；在 Spring 中，bean 工厂创建的各个实例称作 bean

2、bean 工厂创建 bean 的三种方式：

- 通过构造方法直接创建：

  <bean id="" class="bean class name">

- 通过静态工厂方法创建：

  <bean id="" class="factory class name" factory-method="">

- 通过非静态工厂方法创建：

  <bean id="factory" class="factory class name">

  <bean id="" factory-bean=" factory" factory-method="">

3、Spring 中实现 IoC 的方式：依赖注入（Dependency Injection）

Spring 中依赖注入的两种方式：

- 通过 setter 方法注入：

  <property name=""></property>

  其中，name 属性的取值依 setter 方法名而定

- 通过构造方法注入：

  <constructor-arg index=""></ constructor-arg>

  其中，index 表示构造方法中的参数索引(第一个参数索引为 0)

4、设置属性时可选的标签：

- value：基本类型（封装类型）或 String 类型
- ref：引用工厂中其它的 bean
- list：List 或数组类型
- set：Set 类型
- map：Map 类型
- props：Properties 类型

5、自动装配：自动将某个 bean 注入到另一个 bean 的属性当中

（bean 标签的 autowire 属性）

6、依赖检查：检查 bean 的属性是否完成依赖关系的注入

（bean 标签的 dependency-check 属性）

7、生命周期方法：

init-method（也可实现接口 org.springframework.beans.factory.InitializingBean）

destroy-method(也可实现接口 DisposableBean)

8、单例 bean：默认情况下，从 bean 工厂所取得的实例为 Singleton

（bean 的 singleton 属性）

9、Aware 相关接口：可以通过实现 Aware 接口来获得 Spring 所提供的资源

- org.springframework.beans.factory.BeanNameAware
- org.springframework.beans.factory.BeanFactoryAware
- org.springframework.context.ApplicationContextAware

10、ApplicationContext 的功能扩展：

1）BeanPostProcessor：若想在 Spring 对 bean 完成依赖注入之后，再补充一些后续操作，则可以定义一个 bean 类来实现接口：org.springframework.beans.factory.config.BeanPostProcessor，然后在配置文件中定义该 bean 类；从而 Spring 容器会在每一个 bean 被初始化之前、之后分别执行实现了该接口的 bean 类的 postProcessBeforeInitialization（）方法和 postProcessAfterInitialization（）方法；

2）BeanFactoryPostProcessor：

- CustomEditorConfigurer：可借助实现了接口 java.beans.PropertyEditor 的类，并依据当中的实现，将字符串类型转换为指定类型的对象；
- PropertyPlaceholderConfigurer：可将配置文件中属性值用“${key}”形式表示，则会将其值替换成指定的属性文件中key所对应的value；
- PropertyOverrideConfigurer：可借助该类在指定的属性文件中设定一些优先的属性（将忽略配置文件中对应属性的设定值）；

（注意：属性文件中设定属性值的格式为：beanName.propertyName=value）

3) 国际化消息的支持：可通过 ApplicationContext 的 getMessage（）方法获得指定资源文件中的消息；配置如下：

```
<bean
  id="messageSource"
  class="org.springframework.context.support.ResourceBundleMessageSource">
   <property name="basename">
     <value>first/msg</value>
   </property >
</bean>
```

4) 事件的支持：

可发布的事件：ApplicationEvent

发布事件的方法：publishEvent（ApplicationEvent）

事件监听接口：ApplicationListener

11、ApplicationContext 管理 bean 的执行阶段：

- 创建 bean；
- 属性注入（依赖关系的注入）；
- BeanNameAware 接口实现类的 setBeanName（）方法；
- BeanFactoryAware 接口实现类的 setBeanFactory（）方法；
- ApplicationContextAware 接口实现类的 setApplicationContext（）方法；
- BeanPostProcessor 接口实现类中的 postProcessBeforeInitialization（）方法；
- InitializingBean 接口实现类的 afterPropertiesSet（）方法；
- Bean 定义文件中 init-method 属性所指定的方法；
- BeanPostProcessor 接口实现类中的 postProcessAfterInitialization（）方法；

12、FactoryBean：用来创建 bean

```
<bean id="myObject"
      class="org.springframework.beans.factory.config.
       MethodInvokingFactoryBean">
  <property name="staticMethod">
    <value>com.whatever.MyClassFactory.getInstance</value>
  </property>
</bean>
```

（注意：通过 bean 工厂的 getBean(“myObject”)得到的是 FactoryBean 所生产的产品；getBean(“&myObject”)得到的是 FactoryBean 实例本身）

二、AOP(Aspect-Oriented Programming)：面向方面编程

1、代理的两种方式：

静态代理：

- 针对每个具体类分别编写代理类；
- 针对一个接口编写一个代理类；

动态代理：

针对一个方面编写一个 InvocationHandler，然后借用 JDK 反射包中的 Proxy 类为各种接口动态生成相应的代理类

2、AOP 的主要原理：动态代理

3、AOP 中的术语：

- Aspect：方面，层面
- Joinpoint：结合点、联结点；加入业务流程的点或时机
- Pointcut：切入点、作用点；指定某个方面在哪些联结点织入到应用程序中
- Advice：通知、行为；某个方面的具体实现
- Advisor：由 Pointcut 和 Advice 共同构成

4、Pointcut：Spring 根据类名称及方法名称定义 Pointcut，表示 Advice 织入至应用程序的时机；

org.springframework.aop.Pointcut

5、ProxyFactoryBean：用来创建代理对象

```
<bean id="proxy"
    class="org.springframework.aop.framework.ProxyFactoryBean">
    <property name="target">
        <ref bean="target bean"/>
    </property>
    <property name="proxyInterfaces">
        <list>
            <value>interfaces of target bean implemented</value>
        </list>
```

```
        </property>
        <property name="interceptorNames">
            <list>
                <value>Advice/Advisor bean</value>
            </list>
        </property>
    </bean>
```

创建代理对象需指定的三要素：

- target：设定目标对象（只能是一个）；
- proxyInterfaces：设定代理接口（目标对象所实现的接口）；
- interceptorNames：设定拦截器的名字（各个 advice 或 advisor bean 的列表）

6、Advice：五种（根据织入的时机不同而划分）

- Before Advice：在目标对象的方法执行之前加入行为；

  要实现的接口：org.springframework.aop.MethodBeforeAdvice
- After Advice：在目标对象的方法执行之后加入行为；

  要实现的接口：org.springframework.aop.AfterReturningAdvice
- Throw Advice：在目标对象的方法发生异常时加入行为

  要实现的接口：org.springframework.aop.ThrowsAdvice
- Around Advice：在目标对象的方法执行前后加入行为

  要实现的接口：org.aopalliance.intercept.MethodInterceptor
- Introduction Advice：引入的行为（增加了可操作的方法）

  1. 声明要添加的功能接口；
  2. 创建引入 Advice；

  (继承 org.springframework.aop.support.DelegatingIntroductionInterceptor，并实现添加的功能接口）

  3. 在配置文件中定义一个 DefaultIntroductionAdvisor 的 bean

  （需要两个构造方法的参数：Advice 和添加的功能接口）

7、PointcutAdvisor：Pointcut 定义了 Advice 的应用时机；在 Spring 中，使用 PointcutAdvisor 将 Pointcut 和 Advice 结合为一个对象

- NameMatchMethodPointcutAdvisor：可用来指定 Advice 所要应用的目标对象上的方法名称

```
<bean id="nameAdvisor"
     class="org.springframework.aop.support.NameMatchMethodPointcutAdvisor">
    <property name="mappedName">
      <value>do*</value>
    </property>
    <property name="advice">
      <ref local="AdviceBeanName"/>
    </property>
</bean>
```

- RegexpMethodPointcutAdvisor：可使用 Regular expression 定义 Pointcut，在符合 Regular expression 的情况下应用 Advice，其 pattern 属性用来指定所要符合的完整类名（包括 package 名称）及方法名；定义该属性时可使用的符号包括：

符号	描述
.	符合任意一个字符
+	符合前一个字符一次或多次
*	符合前一个字符零次或多次
\	转义字符，用来转义正则表达式中使用到的字符

```
<bean id="regexpAdvisor"
class="org.springframework.aop.support.RegexpMethodPointcutAdvisor">
    <property name="pattern">
        <value>.*do.*</value>
    </property>
    <property name="advice">
        <ref local="AdviceBeanName"/>
    </property>
</bean>
```

8、AutoProxy：自动代理

- BeanNameAutoProxyCreator：根据 bean 的名字为其自动创建代理对象，bean 工厂自动返回其代理对象：

```
<bean
    class="org.springframework.aop.framework.autoproxy.BeanNameAutoP
    roxyCreator">
    <property name="beanNames">
        <list>
            <value>beanName</value>
        </list>
    </property>
    <property name="interceptorNames">
        <list>
            <value>Advice/Advisor Bean Name</value>
        </list>
    </property>
</bean>
```

- DefaultAdvisorAutoProxyCreator：自动将 Advisor 应用到符合 Pointcut 的目标对象上

```
<bean id="regexpAdvisor"
class="org.springframework.aop.support.RegexpMethodPointcutAdvisor">
    <property name="pattern">
        <value>.*do.*</value>
    </property>
    <property name="advice"><ref local="AdviceName"/></property>
</bean>
<bean
class="org.springframework.aop.framework.autoproxy.DefaultAdvisorAutoP
roxyCreator"/>
```

三、Spring 的 Web MVC：

1、DispatcherServlet：作为前端控制器，负责分发客户的请求到 Controller；

在 web.xml 中的配置如下：

```
<servlet>
  <servlet-name>mvc</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
  <init-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/mvc-servlet.xml</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
  <servlet-name>mvc</servlet-name>
  <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

2、Controller：负责处理客户请求，并返回 ModelAndView 实例；

Controller 必须实现接口 org.springframework.web.servlet.mvc.Controller，实现该接口中的方法 handleRequest（），在该方法中处理请求，并返回 ModelAndView 实例

3、HandlerMapping：DispatcherServlet 根据它来决定请求由哪一个 Controller 处理；

- 默认情况下，DispatcherServlet 将使用 org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping，即使用和客户请求的 URL 名称一致的 Controller 的 bean 实例来处理请求；
- 另外一种常用的 HandlerMapping 为 org.springframework.web.servlet.handler.SimpleUrlHandlerMapping，配置如下：

```
<bean
class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
    <property name="mappings">
      <props>
        <prop key="/add.do">add</prop>
```

```
            </props>
        </property>
    </bean>
```

在以上“mappings”属性设置中，“key”为请求的 URL，而“value”为处理请求的 Controller 的 bean 名称

4、ModelAndView：用来封装 View 与呈现在 View 中的 Model 对象；

DispatcherServlet 根据它来解析 View 名称，并处理呈现在 View 中的 Model 对象

5、ViewResolver：DispatcherServlet 委托 ViewResolver 来解析 View 名称；

常用的 ViewResolver 实例配置如下：

```
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix">
        <value>/</value>
    </property>
    <property name="suffix">
        <value>.jsp</value>
    </property>
</bean>
```

四、Spring Data Access

1、Template

2、Dao Support

3、Transaction Manager

4、Transaction Proxy