<https://www.w3cschool.cn/wkspring/>

一、Spring 概述

（1）DI依赖注入

（2）IoC控制反转

（3）AOP面向切面编程，通过预编译方式和运行期动态代理实现程序功能的统一维护。允许定义方法拦截器和切入点对代码进行干净地解耦。

（4）POJO有一些private属性及其getter setter方法，没有业务逻辑的类

二、Spring 体系结构

（1）核心容器，由核心，Bean，上下文和表达式语言模块组成，细节如下：

核心模块提供了框架的基本组成，包括 IoC 和依赖注入。

Bean模块提供BeanFactory。

上下文模块建立在核心和 Bean 模块基础上，是访问定义和配置的任何对象的媒介。ApplicationContext 接口是上下文模块的重点。

表达式语言模块。

（2）数据访问/集成

（3）Web

三、Spring 环境配置

四、Spring Hello World 实例

五、Spring IoC 容器

Spring 提供两种容器，BeanFactory 容器和ApplicationContext 容器。

BeanFactory 容器是最简单的容器，给 DI 提供基本的支持。

ApplicationContext 容器添加更多企业特定的功能，且包括 BeanFactory 容器的所有功能

五（一）、Spring BeanFactory 容器

最常使用的实现了BeanFactory 接口的类是 XmlBeanFactory 类。

XmlBeanFactory factory = new XmlBeanFactory(new ClassPathResource("Beans.xml"));

五（二）、Spring ApplicationContext 容器

最常使用的实现了BeanFactory 接口的类是：

（1）FileSystemXmlApplicationContext：该类从XML文件中加载已定义的bean。需要提供给构造器 XML 文件的完整路径

（2）ClassPathXmlApplicationContext：该类从XML文件中加载已定义的bean。不需要提供给构造器 XML 文件的完整路径，需要配置CLASSPATH环境变量，该类会从CLASSPATH环境变量中搜索XML文件。

（3）WebXmlApplicationContext：该类在一个web应用程序的范围内从XML文件中加载已定义的bean。

五（三）、Spring Bean 定义

bean定义包含配置元数据，包括class，id，name，scope，constructor-arg，property，autowire，lazy-init，init-method，destroy-method

以下三个方法把配置元数据提供给 Spring 容器：

基于 XML 的配置文件

基于注解的配置

基于 Java 的配置

五（四）、Spring Bean 作用域

singleton单例并存储高速缓存中（默认）

prototype每次创建新的实例

request HTTP请求

session HTTP会话

global-session 全局HTTP会话

五（五）、Spring Bean 生命周期

（1）初始化回调

init-method（建议使用）

实现InitializingBean接口的afterPropertiesSet方法

（2）销毁回调

destroy-method（建议使用）

实现DisposableBean接口的destroy方法

AbstractApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");

context.registerShutdownHook();触发销毁回调

（3）在<beans>默认的初始化方法是default-init-method，默认的销毁方法是default-destroy-method

五（六）、Spring Bean 后置处理器

ApplicationContext会自动检测BeanPostProcessor接口的实现类

public class Test implements BeanPostProcessor {

public Object postProcessBeforeInitialization(Object bean, String beanName) throws BeansException {

return bean;

}

public Object postProcessAfterInitialization(Object bean, String beanName) throws BeansException {

return bean;

}

}

<bean class="Test" />

五（七）、Spring Bean 定义继承

<bean id="" class="" parent="父bean的id">

六、Spring依赖注入

包括基于构造函数的依赖注入和基于设值函数的依赖注入

六（一）、基于构造函数的依赖注入

（1）

public class TestA {

private TestB testB;

public TextEditor(TestB testB) {

this.testB = testB;

}

}

<bean id="testA" class="TestA">

<constructor-arg ref="testB"/>

</bean>

<bean id="testB" class="TestB">

</bean>

（2）

public class TestA {

public TestA(TestB testB, TestC testC) {

}

}

<bean id="testA" class="TestA">

<constructor-arg ref="testB"/>

<constructor-arg ref="testC"/>

</bean>

（3）

public class TestA {

public TestA(int testB, String testC) {

}

}

<bean id="testA" class="TestA">

<constructor-arg type="int" value="1" />

<constructor-arg type="java.lang.String" value="2" />

</bean>

（4）推荐

<bean id="testA" class="TestA">

<constructor-arg index="0" ref="testB"/>

<constructor-arg index="1" ref="testC"/>

<constructor-arg index="2" value="1" />

<constructor-arg index="3" value="2" />

</bean>

六（二）、基于设值函数的依赖注入

（1）

public class TestA {

private TestB testB;

private String testC;

public void setTestB(TestB testB) {

this.testB = testB;

}

public TestB getTestB() {

return testB;

}

public void setTestC(String testC) {

this.testC = testC;

}

public String getTestC() {

return testC;

}

}

<bean id="testA" class="TestA">

<property name="testB" ref="testB"/>

<property name="testC" value="c"/>

</bean>

（2）

<bean id="testA" class="TestA"

p:testB-ref="testB"

p:testC="c" />

</bean>

六（三）、Spring 注入内部 Beans

public class TestA {

private TestB testB;

public void setTestB(TestB testB) {

this.testB = testB;

}

public TestB getTestB() {

return testB;

}

}

<bean id="testA" class="TestA">

<property name="testB">

<bean id="testB" class="TestB"/>

</property>

</bean>

六（四）、Spring 注入集合

（1）四中集合List（允许重复）、Set（不能重复）、Map（键和值可以是任何类型）和 Properties（键和值都是字符串类型）

（2）

public class TestA {

List testB;

Set testC;

Map testD;

Properties testE;

public void setTestB(List testB) {

this.testB = testB;

}

public List getTestB() {

return testB;

}

public void setTestC(Set testC) {

this.testC = testC;

}

public Set getTestC() {

return testC;

}

public void setTestD(Map testD) {

this.testD = testD;

}

public Map getTestD() {

return testD;

}

public void setTestE(Properties testE) {

this.testE = testE;

}

public Properties getTestE() {

return testE;

}

}

<bean id="..." class="...">

<property name="testB">

<list>

<ref bean="beanA"/>

<value>stringA</value>

</list>

</property>

<property name="testC">

<set>

<ref bean="beanA"/>

<value>stringA</value>

</set>

</property>

<property name="testD">

<map>

<entry key ="beanA" value-ref="beanA"/>

<entry key="stringA" value="stringA"/>

</map>

</property>

</bean>

（3）注入空字符串和null的值

<property name="testA" value=""/>

<property name="testB"><null/></property>

七、Spring Beans 自动装配

<bean>元素的 autowire 属性值为：

no（默认），必须使用显式装配

byName，基于设值函数的依赖注入时，根据名称自动装配

byType，基于设值函数的依赖注入时，根据类型自动装配

constructor，基于构造函数的依赖注入时，根据类型自动装配

autodetect，先使用constructor，如果不执行，使用byType

八、Spring 基于注解的配置

（1）先执行注解配置，后进行XML配置

（2）注解配置默认不打开，使用<context:annotation-config/>打开

八（一）、Spring @Required 注释

@Required 注释应用于bean属性的setter方法，该bean属性必须存在于XML配置文件中

八（二）、Spring @Autowired 注释

<beans>里有：

<context:annotation-config/>

<bean id="textA" class="TextA">

</bean>

（1）@Autowired应用在Setter，除去<property>，执行byType自动装配

private TextB textB;

@Autowired

public void setTextB( TextB textB ){

this.textB = textB;

}

（2）@Autowired应用在属性，除去<property>和Setter

@Autowired

private TextB textB;

（3）@Autowired应用在构造函数，除去<property>

private TextB textB;

@Autowired

public TextA(TextB textB){

this.textB = textB;

}

（4）@Autowired注释默认依赖是必须的，@Autowired（required=false）关闭默认。

八（三）、Spring @Qualifier 注释

@Qualifier("textA")

private Text text;

<bean id="textA" class="Text">

<property name="name" value="a" />

</bean>

<bean id="textB" class="Text">

<property name="name" value="b" />

</bean>

八（四）、Spring JSR-250 注释

包括@PostConstruct，@PreDestroy和@Resource，用和不用没区别

八（五）、Spring 基于 Java 的配置

用代码替换xml进行配置

八（六）、Spring 中的事件处理

（1）ContextStartedEvent，使用ConfigurableApplicationContext接口中的start()方法启动ApplicationContext时，该事件被发布

（2）ContextStoppedEvent，使用ConfigurableApplicationContext接口中的stop()方法停止ApplicationContext时，发布这个事件。接受到这个事件后可以做清理工作。

（3）ContextClosedEvent，使用ConfigurableApplicationContext接口中的close()方法关闭 ApplicationContext 时，发布这个事件。一个已关闭的上下文到达生命周期末端，不能被刷新或重启。

（4）ContextRefreshedEvent，使用ConfigurableApplicationContext接口中的refresh()方法刷新ApplicationContext时，发布这个事件。ApplicationContext 被初始化时，发布这个事件。

（5）RequestHandledEvent，告诉所有bean，HTTP请求已被服务。

（6）Spring 的事件处理是单线程的。如果一个事件被发布，所有接收者得到该消息前，进程被阻塞。

（7）

public class Test implements ApplicationListener<ContextStartedEvent>{

public void onApplicationEvent(ContextStartedEvent event) {

}

}

<bean id="test" class="Test"/>

八（七）、Spring 中的自定义事件

public class TestEvent extends ApplicationEvent{

public TestEvent(Object source) {

super(source);

}

}

public class TestEventPublisher implements ApplicationEventPublisherAware {

private ApplicationEventPublisher applicationEventPublisher;

public void setApplicationEventPublisher(ApplicationEventPublisher applicationEventPublisher){

this.applicationEventPublisher = applicationEventPublisher;

}

public void publish() {

TestEvent testEvent = new TestEvent(this);

applicationEventPublisher.publishEvent(testEvent);

}

}

public class TestEventHandler implements ApplicationListener<TestEvent>{

public void onApplicationEvent(TestEvent event) {

}

}

public static void main(String[] args) {

ConfigurableApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");

TestEventPublisher testEventPublisher = (TestEventPublisher) context.getBean("TestEventPublisher");

testEventPublisher.publish();

}

<bean id="testEventPublisher" class="TestEventPublisher"/>

<bean id="testEventHandler" class="TestEventHandler"/>

九、Spring 框架的 AOP

五种通知

前置通知，在一个方法执行之前，执行通知。

后置通知，在一个方法执行之后，不考虑结果，执行通知。

返回后通知，在一个方法执行之后，只有在方法成功完成时，才能执行通知。

抛出异常后通知，在一个方法执行之后，只有在方法抛出异常时，才能执行通知。

环绕通知，在一个方法执行之前和执行之后，执行通知。

九（一）、Spring 中基于 AOP 的 XML架构

（1）在<beans>中导入spring-aop架构，在CLASSPATH中使用AspectJ库文件

（2）

public class Test {

public void testBefore(){

}

public void testAfter(){

}

public void testAfterReturning(Object retVal){

}

public void testAfterThrowing(IllegalArgumentException ex){

}

}

<aop:config>

<aop:aspect id="testAspect" ref="test">

<aop:pointcut id="testPointcut" expression="execution(\* 包名.\*.\*(..))"/> 或者 <aop:pointcut id="testPointcut" expression="execution(\* 包名.类名.方法名(..))"/>

<aop:before pointcut-ref="testPointcut" method="testBefore"/>

<aop:after pointcut-ref="testPointcut" method="testAfter"/>

<aop:after-returning pointcut-ref="testPointcut" returning="retVal" method="testAfterReturning"/>

<aop:after-throwing pointcut-ref="testPointcut" throwing="ex" method="testAfterThrowing"/>

</aop:aspect>

</aop:config>

<bean id="test" class="Test"/>

九（二）、Spring 中基于 AOP 的 @AspectJ

（1）在<beans>中导入spring-aop架构，在CLASSPATH中使用AspectJ库文件

（2）

@Aspect

public class Test {

@Pointcut("execution(\* 包名.\*.\*(..))") 或者 @Pointcut("execution(\* 包名.类名.方法名(..))")

private void testPointcut(){}

@Before("testPointcut()")

public void testBefore(){

}

@After("testPointcut()")

public void testAfter(){

}

@AfterReturning(pointcut = "testPointcut()", returning="retVal")

public void testAfterReturning(Object retVal){

}

@AfterThrowing(pointcut = "testPointcut()", throwing = "ex")

public void testAfterThrowing(IllegalArgumentException ex){

}

}

<aop:aspectj-autoproxy/>

<bean id="test" class="Test"/>

十、Spring JDBC 框架

十（一）、Spring JDBC 示例

public class TestJDBCTemplate implements TestDAO {

private DataSource dataSource;

private JdbcTemplate jdbcTemplate;

public void setDataSource(DataSource dataSource) {

this.dataSource = dataSource;

this.jdbcTemplate = new JdbcTemplate(dataSource);

//使用jdbcTemplate不同方法执行不同sql语句

}

}

<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">

<property name="driverClassName" value="com.mysql.jdbc.Driver"/>

<property name="url" value="jdbc:mysql://localhost:3306/数据库名"/>

<property name="username" value="username"/>

<property name="password" value="password"/>

</bean>

<bean id="testJDBCTemplate" class="TestJDBCTemplate">

<property name="dataSource" ref="dataSource" />

</bean>

十（二）、Spring 中 SQL 的存储过程

十一、Spring 事务管理

（1）局部事务如JDBC事务，全局事务如分布式系统事务

（2）事务管理包括声明式事务管理和Spring 编程式事务管理

十一（一）、Spring 声明式事务管理

public class TestJDBCTemplate implements TestDAO {

private DataSource dataSource;

private JdbcTemplate jdbcTemplate;

public void setDataSource(DataSource dataSource) {

this.dataSource = dataSource;

this.jdbcTemplate = new JdbcTemplate(dataSource);

//使用jdbcTemplate不同方法执行不同sql语句

}

public void test(){

try {

} catch (DataAccessException e) {

throw e;

}

return;

}

}

<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">

<property name="driverClassName" value="com.mysql.jdbc.Driver"/>

<property name="url" value="jdbc:mysql://localhost:3306/数据库名"/>

<property name="username" value="username"/>

<property name="password" value="password"/>

</bean>

<bean id="testJDBCTemplate" class="TestJDBCTemplate">

<property name="dataSource" ref="dataSource" />

</bean>

<bean id="dataSourceTransactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">

<property name="dataSource" ref="dataSource" />

</bean>

<aop:config>

<aop:pointcut id="testPointcut" expression="execution(\* 包名.TestJDBCTemplate.test(..))"/>

<aop:advisor advice-ref="testAdviceRef" pointcut-ref="testPointcut"/>

</aop:config>

<tx:advice id="testAdviceRef" transaction-manager="dataSourceTransactionManager">

<tx:attributes>

<tx:method name="test"/>

</tx:attributes>

</tx:advice>

十一（二）、Spring 编程式事务管理

public class TestJDBCTemplate implements TestDAO {

private DataSource dataSource;

private JdbcTemplate jdbcTemplate;

private PlatformTransactionManager platformTransactionManager;

public void setDataSource(DataSource dataSource) {

this.dataSource = dataSource;

this.jdbcTemplate = new JdbcTemplate(dataSource);

//使用jdbcTemplate不同方法执行不同sql语句

}

public void setPlatformTransactionManager(PlatformTransactionManager platformTransactionManager) {

this.platformTransactionManager = platformTransactionManager;

}

public void test(){

TransactionDefinition transactionDefinition = new DefaultTransactionDefinition();

TransactionStatus transactionStatus = platformTransactionManager.getTransaction(transactionDefinition);

try {

platformTransactionManager.commit(transactionStatus);

} catch (DataAccessException e) {

platformTransactionManager.rollback(transactionStatus);

throw e;

}

return;

}

}

<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">

<property name="driverClassName" value="com.mysql.jdbc.Driver"/>

<property name="url" value="jdbc:mysql://localhost:3306/数据库名"/>

<property name="username" value="username"/>

<property name="password" value="password"/>

</bean>

<bean id="testJDBCTemplate" class="TestJDBCTemplate">

<property name="dataSource" ref="dataSource" />

<property name="dataSourceTransactionManager" ref="dataSourceTransactionManager" />

</bean>

<bean id="dataSourceTransactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">

<property name="dataSource" ref="dataSource" />

</bean>

十二、Spring Web MVC 框架

http://www.kaixin001.com/124439585/photo/view\_124439585\_735014864\_32\_18\_50.html#735014864\_124439585\_0\_32

（1）WebApplicationContext是ApplicationContext 的扩展。HandlerMapping、Controller和ViewResolver是WebApplicationContext的一部分

（2）web.xml中有：

<servlet>

<servlet-name>Test</servlet-name>

<servlet-class>

org.springframework.web.servlet.DispatcherServlet

</servlet-class>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>Test</servlet-name>

<url-pattern>\*.jsp</url-pattern>

</servlet-mapping>

web.xml文件保留在应用程序的WebContent/WEB-INF目录下。

初始化<servlet-name> DispatcherServlet时，将加载位于该应用程序WebContent/WEB-INF目录下的 <servlet-name>-servlet.xml文件。

<url-pattern>是所有以.jsp结束的HTTP请求将由<servlet-name> DispatcherServlet处理。

（3）<servlet-name>-servlet.xml中有：

<context:component-scan base-package="包名" />

<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">

<property name="prefix" value="/WEB-INF/jsp/" />

<property name="suffix" value=".jsp" />

</bean>

<context:component-scan />标签用于激活Spring MVC注释扫描功能，该功能允许使用注释

InternalResourceViewResolver规定视图名称。若return "test" 则为/WEB-INF/jsp/test.jsp

（4）

@Controller

@RequestMapping("/test")

public class HelloController{

@RequestMapping(method = RequestMethod.GET)

public String test() {

return "test";

}

}