IOC 控制反转=DI 依赖注入

* 反转获取对象的方式，从SpringIOC容器中获取，而不是自己创建new，然后setId赋值

一个bean就是一个对象

在applicationContext.xml中, <bean>创建对象，<property>给属性赋值

<bean id="ssss" class="org.linran.Student"> 包名.类名

<property name="id" value="12"></property>

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

</bean>

获取对象

ApplicationContext context=new ClassPathXmlApplicationContext("applicationContext.xml");

Student s1=(Student)context.getBean("ssss");

给属性赋值方法1：setID方法

如果ssss的teacher属性是一个对象，则property要用ref而不是value

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

<bean id="teacher" class="org.linran.Teacher">

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

</bean>

这种给属性赋值的方式，就是调用了setId, setTeacher方法。这方法名错一个字母都不行

Spring通过反射实现。通过类找方法

给属性赋值方法2：构造器

Student类中

public Student() {

super();

}

//这个没有参数的构造器得写，否则方法1中的就不能用了

public Student(int id, String name, Teacher teacher) {

super();

this.id = id;

this.name = name;

this.teacher = teacher;

}

applicationContext.xml中

<bean id="s2" class="org.linran.Student">

<constructor-arg value="11"></constructor-arg>

<constructor-arg value="寇鹏飞"></constructor-arg>

<constructor-arg ref="teacher"></constructor-arg>

</bean>

如果赋值和构造器参数顺序不一致，则

* 1.可以用index指定顺序

<constructor-arg ref="teacher" index="2"></constructor-arg>

<constructor-arg value="11" index="0"></constructor-arg>

<constructor-arg value="寇鹏飞" index="1"></constructor-arg>

* 2.可以用name指定参数名

<constructor-arg value="11" name="id"></constructor-arg>

<constructor-arg ref="teacher" name=""teacher"></constructor-arg>

<constructor-arg value="寇鹏飞" name="name"></constructor-arg>

* 3.用type指定参数

<constructor-arg value="11" type="int"></constructor-arg>

<constructor-arg ref="teacher" type="Teacher"></constructor-arg>

<constructor-arg value="寇鹏飞" type="String"></constructor-arg>

* 给属性赋值方法3：p命名空间
* Namespaces里勾选p – <http://www.springframework.org/schema/p>
* 创建对象时直接在标签里写p:id=, 如果是引用类型要加-ref

<bean id="s3" class="org.linran.Student" p:id="1" p:name="林然" p:teacher-ref="teacher">

</bean>

给list array set map注入，分别有各自的标签<list><set><map>

<property name="listProperty">

<list>

<value>string here</value>

</list>

</property>

<map>

<entry>

<key><value></value></key>

<value></value>

</entry>

</map>

<property>

<props>

<prop key="foot4">足球4</prop>

<props>

<property>

AOP

# 搭建环境

1.下载jar包

* 去官网下spring-framework-5.1.4.RELEASE

libs jar包 教程上是dics，应该是4以前的版本，现在变了

docs 文档

* aop.jar
* beans.jar
* context.jar
* core.jar
* expression.jar spring表达式
* 还有一个commons-logging.jar，需要另外下载，日志jar

新建java project——6个jar包复制到src文件夹——然后这6个右键，build path——add to build path——然后这6个jar包就变到了referenced libraries里边

新建dynamic java web project——这几个包复制到/WebContent/WEB-INF/lib目录里边，然后就惊奇的发现，这几个包自动添加到了Java Resources/Libraries必须类库里边了

* 编写配置文件

安装插件spring tool suite

springsource-tool-suite-3.9.7.RELEASE-e4.9.0-updatesite.zip

失败了，直接下载sts，相当于一个整合了spring的eclipse

新建java project——build path上一步——新建Spring bean configuration file, 命名为applicationContext

<?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">

</beans>

# 配置监听器

* SpringIOC容器初始化：

1.将容器中所有bean实例化为对象

2.将各个bean依赖的属性值注入

* Web项目初始化SpringIOC容器：

当服务Tomcat启动时，通过监听器将SpringIOC容器初始化一次

spring-web.jar提供

1.新建Dynamic Web Project（如果不勾选Web.xml，会在Web-Content/WEB-INF位置下少一个Web.xml文件，内容为下）

<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://xmlns.jcp.org/xml/ns/javaee" xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app\_4\_0.xsd" id="WebApp\_ID" version="4.0">

<display-name>lll</display-name>

<welcome-file-list>

<welcome-file>index.html</welcome-file>

<welcome-file>index.htm</welcome-file>

<welcome-file>index.jsp</welcome-file>

<welcome-file>default.html</welcome-file>

<welcome-file>default.htm</welcome-file>

<welcome-file>default.jsp</welcome-file>

</welcome-file-list>

</web-app>

2.把7个（6+1个spring-web.jar）jar包放进Web-Content/WEB-INF/lib目录下，add to build path

3.在web.xml里添加

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener.class</listener-class>

</listener>

配置spring-web.jar提供的监听器，此监听器可以在服务器启动时初始化SpringIOC容器

4告诉IOC容器的位置

在ContextLoaderListener.class的父类ContextLoader.class的源码中有一行

**public** **static** **final** String ***CONFIG\_LOCATION\_PARAM*** = "contextConfigLocation";

CONFIG\_LOCATION\_PARAM就是一个属性，属性值为contextConfigLocation，保存着容器配置文件applicationContext.xml的位置

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:applicationContext.xml</param-value>

</context-param>

不指定contextConfigLocation也行，放到默认位置WebContent/WEB-INF下就行。文件名也不能改，只能是applicationContext.xml

5.在Java Resources/src位置下new一个Spring Bean Configuration File，就是IOC容器配置文件。src就是classpath类路径

拆分Spring配置文件

<param-value>

classpath: applicationContext.xml,

classpath: applicationContext-DAO.xml,

classpath: applicationContext-Service.xml

</param-value>

或者

classpath: applicationContext.xml,

classpath: applicationContext-\*.xml（推荐）

或者在applicationContext.xml中

<import resource="applicationContext-\*.xml" />

Model JavaBean

View JSP

Controller Servlet

# DispatcherServlet

web.xml中配置**DispatcherServlet**，充当controller。servlet把拦截下来的请求，依据相应的规则分发到目标Controller来处理

* 加载这个DispatcherServlet，会需要一个Spring MVC配置文件。文件名是[servlet-name]-servlet.xml
* 默认情况下去WEB-INF文件夹下（与web.xml同级）查找springmvc-servlet.xml

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>/WEB-INF/springmvc-config.xml</param-value>

</init-param>

也可以这样改变spring mvc配置文件的位置和名称

解析该文件内容，创建一个WebApplicationContext，继承自ApplicationContext

<!-- 配置DispatcherServlet -->

<servlet>

<servlet-name>springmvc</servlet-name>

<servlet-class>

org.springframework.web.servlet.DispatcherServlet

</servlet-class>

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>/WEB-INF/springmvc-config.xml</param-value>

</init-param>

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

</servlet>

<!-- 让springmvc的前端控制器拦截所有请求 -->

<servlet-mapping>

<servlet-name>springmvc</servlet-name>

<url-pattern>/</url-pattern>

</servlet-mapping>

springmvc-config.xml

<bean name="/hello" class="org.fkit.controller.HelloController">

声明HelloController业务控制类，将其映射到/hello请求

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:p="http://www.springframework.org/schema/p"

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">

xmlns:namespace-prefix="namespaceURI"

其中namespace-prefix为自定义前缀，只要在这个XML文档中保证前缀不重复即可；namespaceURI是这个前缀对应的XML Namespace的定义。

org.fkit.controller.HelloController

public class HelloController implements Controller {

@Override

public ModelAndView handleRequest(HttpServletRequest request, HttpServletResponse response) throws Exception{

System.out.println("你好世界");

}

}

HelloController实现Controller接口，是一个控制器

基于注解的控制器

springmvc-config.xml

<context:component-scan base-package="org.fkit.controller" />

org.fkit.controller.HelloController.java

**@Controller**

public class HelloController{

**@RequestMapping(value="/hello")**

public ModelAndView hello() {

System.out.println("hello方法被调用");

ModelAndView mv=new ModelAndView();

mv.addObject("message", "fffffffff");

mv.setViewName("WEB-INF/content/welcome.jsp");

return mv;

}

}

Spring扫描一遍base-package包里所有的类，带有@Controller注解的类成为Spring的bean

springmvc-config.xml中，还可以写

<bean class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping" />

<bean class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter" />

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

配置一堆处理器映射器、处理器适配器、视图解析器…………如果没写spring就用默认的

视图放在/WebContent/WEB-INF/content/welcome.jsp这个位置

xmlns:mvc=<http://www.springframework.org/schema/mvc>

xsi:schemaLocation="http://www.springframework.org/schema/mvc

http://www.springframework.org/schema/mvc/spring-mvc.xsd"

加上这几行，spring-mvc.config.xml里头就可以写

<mvc:annotation-driven></mvc:annotation-driven>

<mvc:default-servlet-handler></mvc:default-servlet-handler>

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

p:prefix="/WEB-INF/content/" p:suffix=".jsp"></bean>

* <mvc:annotation-driven>自动注册RequestMappingHandlerMapping和RequestMappingHandlerAdapter两个bean，@Controller必须

（处理器映射器和处理器适配器）

* <mvc:default-servlet-handler> spring mvc的静态资源处理器，定义一个org.springframework.web.servlet.resource.DefaultServletHttpRequestHandler，对进入DispatcherServlet的url检查，如果是静态资源的请求，转由Web应用服务器默认的servlet处理，如果不是，由dispatcherservlet处理？？？？？？？
* InternalResourceViewResolver解析视图，prefix表示视图前缀，suffix表示后缀。因为返回的字符串是welcome，解析之后显示/WEB-INF/content/welcome.jsp

@Controller

public class HelloController{

@RequestMapping(value="/hello")

public String hello(Model model) {

model.addAttribute("message", "HHHHHHHHHFFFFFFF");

return "welcome";

}

}

## RequestMapping

@RequestMapping(value="hello")

@RequestMapping("/hello")

效果相同

@RequestMapping(value="/hello", method="RequestMethod.POST")

指定http请求方式，有POST，GET……

效果等同于使用@GetMapping @PostMapping

@RequestMapping(value="/hello", method="RequestMethod.POST", consumes="application/json")

指定请求的content-type为特定类型，比如application/json

@RequestMapping(value="/hello", method="RequestMethod.POST", consumes="application/json") produces="application/json"

指定返回的content-type

还有params属性，headers属性等等

public String hello(Model model){

return "h1";

}

处理方法的参数可以有很多种类型，比如javax.servlet.ServletRequest, javax.servlet.HttpServletRequest, org.springframework.ui.Model。写哪个调用的时候就创建哪个，然后传递给方法。

org.springframework.web.context.request.WebRequest比较常用

## Spring配置可选方案

* 在XML中显式配置
* 在Java中显式配置
* 隐式bean发现机制和自动装配

推荐使用自动装配，然后JavaConfig，最后xml

package soundSystem;

@Configuration

@ComponentScan

public class CDPlayerConfig {

}

@ComponentScan默认扫描与配置类相同的包。比如此例中会扫描soundSystem包以及子包。查找带有@Component注解的类，然后自动创建一个bean。

@Configuration表示这是一个配置类？

@Component

public class SgtPeppers implements CompactDisc{

public void play() {

System.out.println("1233211233321");

}

}

测试类CDPlayerTest

@RunWith(SpringJUnit4ClassRunner.class)

@ContextConfiguration(classes=CDPlayerConfig.class)

public class CDPlayerTest {

@Autowired

private CompactDisc cd;//没有任何一句给cd赋值的语句，但是cd就是创建出来了还能用

@Test

public void cdShouldNotBeNull() {

assertNotNull(cd);

}

}

* SpringJUnit4ClassRunner.class表示测试开始时自动创建spring的应用上下文
* @ContextConfiguration表示在哪个类中加载配置
* @Autowired暂时不知道什么意思
* @RunWith(SpringJUnit4ClassRunner.class)让测试运行于Spring测试环境。看不懂

这个例子过程：创建上下文，扫描，创建bean，然后赋给@autowired？

## @Component

Spring应用上下文中每个bean都有一个ID。

上例没有明确指出SgtPeppers这个bean的id，默认就是sgtPeppers

@Component("hello")

这样设置ID

## @ComponentScan

如果要把配置类放在单独的包中，那么就得在ComponentScan添加参数了

ComponentScan("soundSystem")

@ComponentScan(basePackages="soundSystem")

@ComponentScan(basePackages={"soundSystem", "video"))

@ComponentScan(basePackageClasses={CDPlayer.class, DVDPlayer.class})

最后一种，指定类或接口，而不是包了。

## 自动装配

让Spring自动满足bean依赖，比如一个bean的参数是另一个bean

@Component

public class CDPlayer{

private CompactDisc cd;

**@Autowired**

public CDPlayer(CompactDisc cd){

this.cd=cd;

}

}

@Autowired注解用在A的构造器上或者其他方法上，参数是B，spring创建A的bean的时候就会传入一个B的bean。应该是创建完A, B这两个bean之后再尝试满足依赖

如果有多个bean满足依赖关系，Spring抛出异常，表明没有明确指明使用哪一个bean装配

@AutoWired

private MediaPlayer player

变量声明的时候也能用，这个MediaPlayer是个接口，但也没有关系，把实现了MediaPlayer的CDPlayer的bean注入到player了

## 通过Java代码显示装配bean

@Configuration

public class CDPlayerConfig{

**@Bean**

public CompactDisc sgtPeppers(){

return new SgtPeppers();

}

@Bean

public CDPlayer cdPlayer(){

return new CDPlayer(sgtPeppers());

}

}

* @Bean注解告诉spring这个方法将返回一个对象，该对象要注册为Spring应用上下文中的bean
* 默认这个bean的id和方法名相同，即sgtPeppers。如果想改名字@Bean(name="newName")
* 假如有依赖，则return new CDPlayer(sgtPeppers());传统写法认为新建了一个SgtPeppers对象给传到构造器里，其实不是。带有@Bean注解的方法，这样调用Spring会拦截，返回该方法创建的bean

@Bean

public CDPlayer cdPlayer(CompactDisc cd){

return new CDPlayer();

}

或者这样写，有一丝自动装配的味道在里边。

# Spring Boot

@RestController

public class UserController {

@GetMapping("/hello")

public String getUserByGet(String userName){

return "Hello nihaoshijie";

}

}

GetMapping处理Get请求，路径/hello