CH01_Spring框架

三种配置元数据(描述数据的数据)的方式

1.xml方式

```
    <!-- 1.xml方式配置 -->
    <!-- name与Computer中的IntelCpu的属性名一致，ref指对应的id -->
    <bean id="IntelCpu" class="com.IntelCpu"></bean>
    <bean id="Computer" class="com.Computer">
        <property name="intelCpu" ref="IntelCpu"></property>
    </bean>
```

2.注解方式

```
<!-- 2.启用注解配置 -->
<!-- 扫描com包中的注解 -->
<context:annotation-config/>
<context:component-scan base-package="com"></context:component-scan>
```

```
package com;

import org.springframework.stereotype.Component;
//表示bean组件
@Component("IntelCpu")
public class IntelCpu {

    public void run() {
        System.out.println("intel cpu is running");
    }
}
```

```
package com;

import javax.annotation.Resource;

import org.springframework.stereotype.Component;

@Component("Computer")//容器中默认有一个小写的computer
public class Computer {
    @Resource
    private IntelCpu intelCpu;
```

```
    public Computer() {
        System.out.println("Computer无参构造");
    }

    public void setIntelCpu(IntelCpu intelCpu) {
        this.intelCpu = intelCpu;
    }

    public void play() {
        intelCpu.run();
        System.out.println("pc is running");
    }

}
```

3.Java代码方式

```
package com;

import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

/**
 * 3.测试java配置
 * @author 雨
 */
//将xml的两行代码，换成注解即可
@Configuration
@ComponentScan("com")
public class Run {

    public static void main(String[] args) {
        ApplicationContext ctx=new AnnotationConfigApplicationContext(Run.class);
        Computer pc=(Computer)ctx.getBean("Computer");
        pc.play();
    }

}
```

CH02_Spring容器

1.实例化bean的三种方式

```
<!--1.构造方法实例化-->
<bean id="IntelCpu" class="com.IntelCpu"></bean>

<!--2.实例工厂实例化 -->
<bean id="Desk" class="com.Desk" factory-method="createInstance" scope="prototype"></bean>

<!--3.通过实例工厂生产bean，告诉Chair的工厂在哪，工厂的哪个方法  -->
<bean id="ChairFactory" class="com.ChairFactory"></bean>
<bean id="Chair" factory-bean="ChairFactory" factory-method="create"></bean>
```

2.bean的构造器注入和setter方法注入

```
<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-4.0.xsd
        http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-4.0.xsd">

    <!-- 1.构造器注入
    优点：方便
    缺点：可读性差 -->
    <bean id="Teacher" class="spring3.Teacher">
        <constructor-arg index="1" value="zhangsan"></constructor-arg><!-- 自动进行类型转换 -->
        <constructor-arg index="0" value="1"></constructor-arg>
    </bean>

   <!--  2.setter方法注入
    -->
    <bean id="DiClass" class="spring3.DiClass">
        <property name="id" value="100"></property><!-- 基本数据类型 -->
        <property name="name" value="lisi"></property>
        <property name="teacher" ref="Teacher" ></property><!-- 引用数据类型注入 ， Teacher是id-->

        <property name="bookName">
            <list>
                <value>Java</value><!-- value基本数据类型，ref引用数据类型 -->
                <value>c#</value>
                <ref bean="Teacher"/>
            </list>
        </property>

        <property name="hobby">
            <set>
                <value>book</value>
                <value>football</value>
                <ref bean="Teacher"/>
                <value>book</value>
            </set>
        </property>

        <property name="map">
            <map>
                <entry key="a" value="100"></entry>
                <entry key-ref="Teacher" value="200"></entry>
                <entry key="b" value-ref="Teacher"></entry>
```

```
            </map>
        </property>

    </bean>

</beans>
```

CH03_数据验证和SpringEL

```
package com;

import java.util.List;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;

@Component
public class Computer {

    @Value("#{datas.stus.^[score >= 90]}")
    private List<Student> subList;

    @Value("#{datas.stus.?[score >= 90].![name]}")
    private List<String> nameList;

    @Value("#{datas.list1}")
    private List<String> list;

    @Value("#{datas.list1[1]}")
    private String listString;

    @Value("#{datas.map['c']}")
    private String mapString;

    @Value("#{5 > 4 && 3 > 6}")
    private boolean result1;

    @Value("#{3 * 6}")
    private int num1;
    @Value("#{computer.num1++}")//6.使用运算符
    private int num2;

    @Value("#{new int[]{1,3,4,5,6}}")//5.数组类型注入
    private int[] nums;

    @Value("#{mathUtil.area(4,5)}")//3.调用bean的自定义方法
    private int pcArea;

    @Value("#{'lenovl'.toUpperCase()}")//2.官方提供的字符串的方法
    private String brand;

    @Value("#{new com.Memory()}")//4.调用构造方法
    private Memory memory;

    //1.最简单的EL调用
```

```
    @Value("#{memory.memCount}")
    private int count;

    public String getBrand() {
        return brand;
    }

    public void setBrand(String brand) {
        this.brand = brand;
    }

    public Memory getMemory() {
        return memory;
    }

    public void setMemory(Memory memory) {
        this.memory = memory;
    }

    public int getCount() {
        return count;
    }

    public void setCount(int count) {
        this.count = count;
    }

    public int getNum1() {
        return num1;
    }

    public void setNum1(int num1) {
        this.num1 = num1;
    }

    public int getNum2() {
        return num2;
    }

    public void setNum2(int num2) {
        this.num2 = num2;
    }

    public int[] getNums() {
        return nums;
    }

    public void setNums(int[] nums) {
        this.nums = nums;
    }

    public int getPcArea() {
        return pcArea;
    }

    public void setPcArea(int pcArea) {
        this.pcArea = pcArea;
    }

    public void show() {
        System.out.println(brand);
```

```
        System.out.println(count);
        System.out.println(pcArea);
        System.out.println("======================");
        System.out.println(memory.getMemCount());
        System.out.println("num1:"+num1);
        System.out.println("num2:"+num2);
        System.out.println("======================");
        System.out.println(result1);

        System.out.println(listString);
        System.out.println(mapString);
        System.out.println("======================");
        for(Student stu : subList) {
            System.out.println(stu.getName());
        }
    }

}
```

CH04_Spirng AOP

AOP三种实现方式

1.实现SpringAPI的传统方式

1.写好通知

```
package adivce;

import java.lang.reflect.Method;

import org.springframework.aop.MethodBeforeAdvice;

import util.Md5Encode;
/**
 * 前置通知
 * @author 雨
 *
 */
public class Md5Advice implements MethodBeforeAdvice{

    @Override
    public void before(Method arg0, Object[] arg1, Object arg2) throws Throwable {
        System.out.println(arg0.getName());
        System.out.println(arg2.getClass().getName());

        arg1[1] = Md5Encode.getMD5(arg1[1].toString().getBytes());
        System.out.println("================================");
    }

}
```

2.在bean.xml进行配置

```xml
<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"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-4.3.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-4.3.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop-4.3.xsd
    ">
    <!-- 各种通知-->
    <bean id="Md5Advice" class="adivce.Md5Advice"></bean>
    <bean id="ScoreAdvice" class="adivce.ScoreAdvice"></bean>
    <bean id="TimeAdvice" class="adivce.TimeAdvice"></bean>
    <!-- 异常通知 -->
    <bean id="MyExAdvice" class="adivce.MyExAdvice"></bean>

    <bean id="UserServiceImpl" class="com.UserServiceImpl"></bean>

    <!-- 重点，代理的配置 -->
    <bean id="UserServiceProxy" class="org.springframework.aop.framework.ProxyFactoryBean">
       <property name="proxyInterfaces" value="com.UserService"></property>
       <property name="target" ref="UserServiceImpl"></property>
       <property name="interceptorNames">
           <list>
               <value>Md5Advice</value>
               <value>ScoreAdvice</value>
               <value>TimeAdvice</value>
               <value>MyExAdvice</value>
           </list>
       </property>
    </bean>
</beans>
```

2.纯POJO类（Advice不用实现任何接口）

写好通知

```java
package advice;

import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.Aspect;
import org.springframework.stereotype.Component;

import util.Md5Encode;

/**
 * 通知类，不需要实现Advice接口
 * 所有的通知的方法都可以写在这个类中
 * @author 雨
 *
 */

public class MyAdvice {
```

```
    public void beforeMethod(JoinPoint joinPoint) {
        System.out.println("前置通知执行");
        System.out.println(joinPoint.getSignature().getName());
        System.out.println(joinPoint.getArgs());//业务逻辑的参数
        System.out.println(joinPoint.getTarget());//被代理的对象
        System.out.println("==================================");
    }

    public void afterReturnningMethod(JoinPoint joinPoint, Object result) {
        System.out.println("后置通知执行，结果是："+result);
        System.out.println("==================================");
    }

    public Object aroundMethod(ProceedingJoinPoint joinPoint){
        System.out.println("环绕通知执行    开始");
        //得到第二个参数，加密
        Object[] param = joinPoint.getArgs();
        param[1] = Md5Encode.getMD5(param[1].toString().getBytes());
        //执行原有方法
        Object result = null;
        try {
            result = joinPoint.proceed(param);//记得传入参数
        } catch (Throwable e) {
            e.printStackTrace();
        }
        System.out.println("环绕通知执行    结束");
        return result;
    }

    public void thowExMethod(Exception e) {
        System.out.println("异常通知执行"+e.getMessage());
        System.out.println("==================================");
    }

    public void afterMethod(JoinPoint joinpoint) {
        System.out.println("最终通知执行");
    }

}
```

1.基于xml

```
<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"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-4.3.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-4.3.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop-4.3.xsd
    ">
    <!-- aop命名空间需要加上 -->
    <bean id="MyAdvice" class="advice.MyAdvice"></bean>
    <bean id="UserServiceImpl" class="com.UserServiceImpl"></bean>

    <!--
```

```
        ref：通知类
        before:前置通知，method，前置通知方法名，pointcut切点表达式，不管返回值，com包下的所有类的所有方法不管参数
        returning：后置通知方法参数中，返回值的参数名
     -->
    <aop:config>
        <aop:aspect id="MyAspect" ref="MyAdvice">
            <aop:before method="beforeMethod" pointcut="execution(* com.*.*(..))" />
            <aop:after-returning method="afterReturnningMethod" pointcut="execution(* com.*.*(..))" returning="result" />
            aop:around method="aroundMethod" pointcut="execution(* com.*.*(..))"/
            <aop:after-throwing method="thowExMethod" pointcut="execution(* com.*.*(..))" throwing="e"/>
            <aop:after method="afterMethod" pointcut="execution(* com.*.*(..))" />
        </aop:aspect>
    </aop:config>

</beans>
```

2.基于注解

```
    <context:annotation-config />
    <context:component-scan base-package="com" />
    <!-- 开启@AspectJ支持 -->
    <aop:aspectj-autoproxy />
```

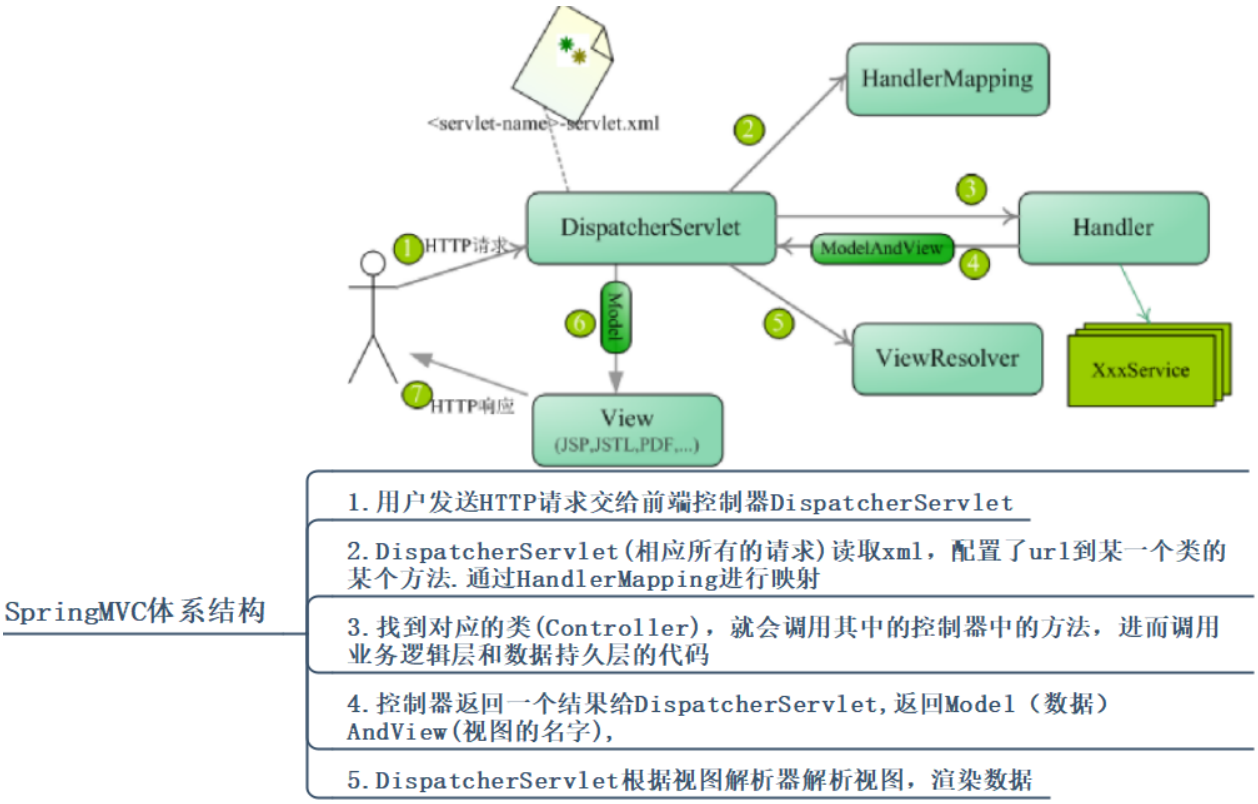
```
package com.advice;

import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.After;
import org.aspectj.lang.annotation.AfterReturning;
import org.aspectj.lang.annotation.AfterThrowing;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.springframework.stereotype.Component;

import util.Md5Encode;

@Component
@Aspect
public class MyAdvice {

    @Before("execution(* com.user.*.*(..))")
    public void beforeMethod(JoinPoint joinPoint) {
        System.out.println("前置通知执行");
        System.out.println(joinPoint.getSignature().getName());
        System.out.println(joinPoint.getArgs());
        System.out.println(joinPoint.getTarget());
        System.out.println("==================================");
    }

    @AfterReturning(pointcut="execution(* com.user.*.*(..))", returning="result")
    public void afterReturnningMethod(JoinPoint joinPoint, Object result) {
        System.out.println("后置通知执行，结果是："+result);
```

```
        System.out.println("==================================");
    }

    @Around("execution(* com.user.*.*(..))")
    public Object aroundMethod(ProceedingJoinPoint joinPoint){
        System.out.println("环绕通知执行    开始");
        Object[] param = joinPoint.getArgs();
        param[1] = Md5Encode.getMD5(param[1].toString().getBytes());
        Object result = null;
        try {
            result = joinPoint.proceed(param);
        } catch (Throwable e) {
            e.printStackTrace();
        }
        System.out.println("环绕通知执行    结束");
        return result;
    }
    @AfterThrowing(value="execution(* com.user.*.*(..))", throwing="e")
    public void thowExMethod(Exception e) {
        System.out.println("异常通知执行"+e.getMessage());
        System.out.println("==================================");
    }

    @After("execution(* com.user.*.*(..))")
    public void afterMethod(JoinPoint joinpoint) {
        System.out.println("最终通知执行");
    }

}
```

3.使用@AspectJ切面(如上注解方式)

CH05_Spring MVC(Controller的使用)

1.Controller与基本配置，注解



步骤:

1.创建web工程

2.先后编写spring-mvc.xml,applicationContext.xml,并将前两者配置到web.xml中

3.编写Controller

applicationContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- 这是人为分割的一个配置文件,
我们习惯把和web相关的东西配置到spring-mvc.xml,和web无关(有关数据库)的东西配置到这里
可用于和hibernate框架整合 -->
<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"
    xmlns:jee="http://www.springframework.org/schema/jee"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
        http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-4.0.xsd
        http://www.springframework.org/schema/jee
http://www.springframework.org/schema/jee/spring-jee-4.0.xsd">
    <description>Spring公共配置 </description>

    <!-- 配置Spring上下文的注解 -->
    <context:annotation-config />
    <!-- 使用annotation 自动注册bean, 并保证@Required、@Autowired的属性被注入 -->
    <context:component-scan base-package="com.abc.cakeonline">
        <context:exclude-filter type="annotation"
            expression="org.springframework.stereotype.Controller" />
    </context:component-scan>
```

```
</beans>
```

spring-mvc.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"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xsi:schemaLocation="http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd
        http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
        http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-4.0.xsd">
    <!-- 自动扫描且只扫描@Controller -->
    <context:component-scan base-package="com.abc.cakeonline">
        <context:include-filter type="annotation"
            expression="org.springframework.stereotype.Controller" />
    </context:component-scan>

    <!-- 如果@Mapping发生400错误，需要加这一行，调用注解方式的解析 -->
    <mvc:annotation-driven enable-matrix-variables="true" />

    <!-- 视图解析器，定义JSP文件的位置，前缀+返回值+后缀，就是要跳转的视图 -->
    <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/" />
        <property name="suffix" value=".jsp" />
    </bean>

</beans>
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">
  <display-name>springmvc1</display-name>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>

  <!-- 加载配置 -->
  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>
            classpath*:/applicationContext.xml
        </param-value>
  </context-param>

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

  <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/spring-mvc.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>springmvc</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>
</web-app>
```

HelloController.java

各种注解使用、返回值（根据视图解析器）

```
package com.abc.cakeonline.hello;

import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.CookieValue;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.Mapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestHeader;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller//这个类是一个控制器
@RequestMapping("/hi")//控制请求路径，类的级别和方法级别上的路径拼接
public class HelloController {

    @RequestMapping("/header")
    @ResponseBody
    public String testHeader(@RequestHeader("Accept") String accept) {
        System.out.println(accept);
        //跳转到/home.jsp
        return "home";
    }


    /**
     * 获取客户端的cookie
     * @param sid
     * @param request
     * @param response
     * @return
     */
    @RequestMapping("/cookie")
    public String testCookie(@CookieValue("JSESSIONID") String sid,
            HttpServletRequest request, HttpServletResponse response) {
//      //以前的实现，将cookie发送到客户端
```

```
//      Cookie cookie = new Cookie("name","zs");
//      response.addCookie(cookie);
//      //获取客户端穿过来的cookie
//      Cookie[] cs = request.getCookies();
//      for(Cookie c : cs) {
//          if(c.getName().equals("name")) {
//              String val = c.getValue();
//          }
//      }
        System.out.println(sid);
        return "";
    }


    @RequestMapping(value = "/hello", method=RequestMethod.GET) //url-pattern
//  @GetMapping("/hello")
    /**
     * required必须参数
     * defaltValue参数默认值
     * @param name
     * @param request
     * @return
     */
    public String helloSomebody(@RequestParam(value = "name", required = true, defaultValue =
"lisi") String name,
            HttpServletRequest request) {
//      String name = request.getParameter("name");

//      String p = request.getParameter("pageNum");
//      if(p==null)
//          p="1";
        System.out.println(name);
        request.setAttribute("n", name);
        return "home";
    }


    //hi/hello1/zhangsan/20 现在常用的url路径，就不会透露数据库表信息
    @RequestMapping("/hello1/{username}/{age}")
    public String hello1(@PathVariable("username") String un,
            @PathVariable("age") int age, HttpSession session) {
        System.out.println(un);
        System.out.println(age);
        session.setAttribute("n", un);
        return "home";
    }

}
```

2.拦截器与文件上传

spring-mvc.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"
    xmlns:context="http://www.springframework.org/schema/context"
```

```
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xsi:schemaLocation="http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd
        http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
        http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-4.0.xsd">
    <mvc:annotation-driven />

    <!-- 自动扫描且只扫描@Controller -->
    <context:component-scan base-package="com">
        <context:include-filter type="annotation"
            expression="org.springframework.stereotype.Controller" />
    </context:component-scan>

    <!-- Spring对Servlet3.0以上自带的文件上传支持 -->
    <bean id="multipartResolver"
class="org.springframework.web.multipart.support.StandardServletMultipartResolver"></bean>
    <!-- 文件上传，Spring针对Apache的一个文件上传解析器 -->
    <!-- bean id="multipartResolver"
class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
       <property name="defaultEncoding" value="UTF-8"></property>
       <property name="maxUploadSize" value="20000000"></property>
    </bean-->

    <!-- 拦截那些请求，不拦截什么，拦截器的类在哪儿(完整路径) 先注释，避免影响文件上传 -->
    <mvc:interceptors>
       <mvc:interceptor>
           <mvc:mapping path="/**"/>
           <mvc:exclude-mapping path="/login"/>
           <bean class="com.user.controller.RegistInterceptor" />
       </mvc:interceptor>
    </mvc:interceptors>

    <!-- 定义JSP文件的位置 -->
    <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/" />
        <property name="suffix" value=".jsp" />
    </bean>

</beans>
```

1.拦截器

```
package com.user.controller;

import java.util.Calendar;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.handler.HandlerInterceptorAdapter;
/**
 * 拦截器
 * @author 雨
```

```
 *
 */
public class RegistInterceptor extends HandlerInterceptorAdapter{

    //整个请求处理完了
    @Override
    public void afterCompletion(HttpServletRequest request, HttpServletResponse response, Object
handler, Exception ex)
            throws Exception {
        System.out.println("aftercompletion");
    }

    //控制器处理之后
    @Override
    public void postHandle(HttpServletRequest request, HttpServletResponse response, Object
handler,
            ModelAndView modelAndView) throws Exception {
        System.out.println("posthandle");
    }

    //控制器处理之前
    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object
handler)
            throws Exception {
        System.out.println("prehandle");
        Calendar c = Calendar.getInstance();
        int hour = c.get(Calendar.HOUR_OF_DAY);
        if(hour>= 11 && hour<=13) {
            return true;
        }else {
            response.sendRedirect("index.jsp");
            return false;
        }
    }

}
```

2.文件上传

```
package com.util;

import java.io.File;

import javax.servlet.http.HttpServletRequest;

import org.springframework.stereotype.Controller;
import org.springframework.util.FileCopyUtils;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.multipart.MultipartFile;

/**
 * 文件上传
 * 1.Spring框架下封装的Apache的单文件上传
 * 2.Apache多文件上传
 * 3.Spring对Servlet3.0以上自带的文件上传支持
```

```
 * @author 雨
 *
 */
@Controller
public class UploadController {

    /**
     *
     * @param title
     * @param file
     * @param request
     * @return
     */
    @RequestMapping("/upload1")
    public String upload1(@RequestParam("title") String title,
            @RequestParam("upFile") MultipartFile file,
            HttpServletRequest request) {
        //麻烦
//      InputStream is = file.getInputStream();
//      FileOutputStream fos = new FileOutputStream("d:/a.txt");
        System.out.println(title);

        //Spring框架简化
        String path = request.getServletContext().getRealPath("/");//项目根目录的物理路径
        try {
            //将file转化为字节数组、上传路径、上传名称  ;参数多是为了给我们更多改变空间，不要怕参数多
            FileCopyUtils.copy(file.getBytes(), new File(path+"/upload",
file.getOriginalFilename()));
        }catch(Exception e) {
            e.printStackTrace();
        }
        return "";

    }

    /**
     * 多文件上传
     * @param title
     * @param file
     * @param request
     * @return
     */
    @RequestMapping("/upload2")
    public String upload2(@RequestParam("title") String title,
            @RequestParam("upFile") MultipartFile[] file,//多文件上传数组
            HttpServletRequest request) {
        System.out.println(title);
        String path = request.getServletContext().getRealPath("/");
        try {
            //遍历上传文件数组
            for(MultipartFile temp : file) {
                FileCopyUtils.copy(temp.getBytes(), new File(path+"/upload",
temp.getOriginalFilename()));
            }
        }catch(Exception e) {
            e.printStackTrace();
        }
        return "";

    }
```

```
    @RequestMapping("/upload3")
    public String upload3(@RequestParam("title") String title,
            @RequestParam("upFile") MultipartFile file,
            HttpServletRequest request) {
        System.out.println(title);
        String path = request.getServletContext().getRealPath("/");
        try {
            FileCopyUtils.copy(file.getBytes(), new File(path+"/upload",
file.getOriginalFilename()));
        }catch(Exception e) {
            e.printStackTrace();
        }
        return "";

    }
}
```

CH06_SpringMVC2

视图解析

1.InternalResourceViewResolver(常用)

2.BeanNameViewResolver（Excel、Pdf文档视图）

```
<bean class="org.springframework.web.servlet.view.BeanNameViewResolver"></bean>
<bean id="userListExcel" class="com.otherview.ExcelView"></bean>
<bean id="userListPdf" class="com.otherview.PdfView"></bean>
```

ExcelController.java

```
package com.otherview;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class ExcelController {

    @RequestMapping("/excel")
    public String excel() {
        //跳转到userListExcel的Bean的id，理解到了BeanNameViewResolver
        return "userListExcel";
    }
```

```
    @RequestMapping("/pdf")
    public String pdf() {
        return "userListPdf";
    }
}
```

ExcelView.java

```
package com.otherview;

import java.util.Map;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.poi.hssf.usermodel.HSSFRow;
import org.apache.poi.hssf.usermodel.HSSFSheet;
import org.apache.poi.hssf.usermodel.HSSFWorkbook;
import org.springframework.web.servlet.view.document.AbstractExcelView;

public class ExcelView extends AbstractExcelView{

    @Override
    protected void buildExcelDocument(Map<String, Object> arg0, HSSFWorkbook workbook,
HttpServletRequest arg2,
            HttpServletResponse response) throws Exception {
        response.setHeader("Content-Disposition", "inline;filename="+new String("用户列
表.xls".getBytes(), "iso8859-1"));
        HSSFSheet sheet = workbook.createSheet("user");//sheet的名称
        HSSFRow row = sheet.createRow(0);
        row.createCell(0).setCellValue("用户编号");
        row.createCell(1).setCellValue("联系电话");
        for(int i=0;i<10;i++) {
            HSSFRow row1 = sheet.createRow(i+1);
            row1.createCell(0).setCellValue(i+1);
            row1.createCell(1).setCellValue("1551234123"+(i+1));
        }

    }

}
```

PdfView.java

```
package com.otherview;

import java.awt.Color;
import java.util.Map;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.springframework.web.servlet.view.document.AbstractPdfView;

import com.lowagie.text.Document;
```

```
import com.lowagie.text.Font;
import com.lowagie.text.Phrase;
import com.lowagie.text.Table;
import com.lowagie.text.pdf.BaseFont;
import com.lowagie.text.pdf.PdfWriter;

public class PdfView extends AbstractPdfView {

    @Override
    protected void buildPdfDocument(Map<String, Object> arg0, Document document, PdfWriter arg2,
HttpServletRequest arg3,
            HttpServletResponse response) throws Exception {
        //文件名、编码方式等
        response.setHeader("Content-Disposition", "inline;filename="+new String("用户列
表.pdf".getBytes(), "iso8859-1"));

        Table table = new Table(2);
        table.setWidth(100);
        table.setBorder(1);
        BaseFont baseFont=BaseFont.createFont("STSongStd-Light","UniGB-UCS2-H",false);
        Font cnFont = new Font(baseFont, 10, Font.NORMAL, Color.red);//字体、字号、是否斜体加粗
等、颜色
        table.addCell(new Phrase("学号", cnFont));
        table.addCell(new Phrase("联系方式", cnFont));

        for(int i=0;i<10;i++) {
            table.addCell(""+i);
            table.addCell("155123456"+i);
        }

        document.add(table);
    }

}
```

Spring Form标签(ResourceBundleViewResolver（返回properties中绑定的页面）)

双向绑定

UserForm.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
    v-----c------v双向绑定
```

```
<!--path写User类中的属性
    radiobuttons，从数据库检索专业集合绑定，使用items属性取出来
    commandName="user"指定formBackingObject 表单绑定对象，取出作用域的user对象，绑定到表单中
-->

    <form:form commandName="user" action="/springform/user/${action }" method="post" >
        姓名：<form:input path="name" /><br>
        密码：<form:password path="password"/><br>
        性别：<form:radiobutton path="gender" value="m"/>男
        <form:radiobutton path="gender" value="f"/>女<br>
        专业：<form:radiobuttons path="spe" items="${special }"/><br>
        学院：<form:radiobuttons path="collegeId" items="${colleges }" itemLabel="name"
itemValue="id" /><br>
        爱好：<form:checkboxes items="${hobbys }" path="hobby"/><br>
        城市：<form:select path="cityId">
            <form:option value="0">请选择</form:option>
            <form:options items="${citys }" itemLabel="name" itemValue="id"/>
        </form:select><br>
        <input type="submit" value="save" />
    </form:form>
</body>
</html>
```

UserController.java

```
package com;

import java.sql.Array;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;

import javax.servlet.http.HttpServletRequest;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

/**
 * 测试ResourceBundleViewResolver
 * @author 雨
 *
 */
@Controller
@RequestMapping("/user")
public class UserController {

    @RequestMapping("/login")
    public String login() {
        //Spring-mvc中的视图解析器、根据其中配置的资源文件，找到test.url=/test.jsp，从而跳到test.jsp
        return "test";
    }

    /**
     * 准备注册
     * @param request
```

```
     * @return
     */
    @RequestMapping(value="/regist", method=RequestMethod.GET)
    public String toRegist(HttpServletRequest request) {
        request.setAttribute("action", "regist");
        //双向绑定c------>v，把user对象绑定到页面的表单元素
        User user = new User();
        request.setAttribute("user", user);
        //从数据库中检索出专业的集合
        List<String> special = new ArrayList<>();
        special.add("计算机专业");
        special.add("美术专业");
        special.add("音乐专业");
        request.setAttribute("special", special);
        //从数据库中检索出所有的学院
        List<College> colleges = new ArrayList<>();
        College c1 = new College();
        c1.setId(1);
        c1.setName("计算机学院");
        College c2 = new College();
        c2.setId(2);
        c2.setName("美术学院");
        College c3 = new College();
        c3.setId(3);
        c3.setName("音乐学院");
        colleges.add(c1);
        colleges.add(c2);
        colleges.add(c3);
        request.setAttribute("colleges", colleges);

        List<String> hobbys = new ArrayList<>();
        hobbys.add("pc");
        hobbys.add("read");
        hobbys.add("football");
        request.setAttribute("hobbys", hobbys);

        List<City> citys = new ArrayList<>();
        City ci1 = new City();
        ci1.setId(1);
        ci1.setName("北京");
        City ci2 = new City();
        ci2.setId(2);
        ci2.setName("天津");
        City ci3 = new City();
        ci3.setId(3);
        ci3.setName("石家庄");
        citys.add(ci1);
        citys.add(ci2);
        citys.add(ci3);
        request.setAttribute("citys", citys);

        return "userForm";
    }

    /**
     * 注册
     * @param user
     * @return
     */
    @RequestMapping(value="/regist", method=RequestMethod.POST)
    //双向绑定v------>v,把页面的表单元素，绑定到控制器的user参数
```

```
public String regist(User user) {
    System.out.println(user.getName());
    System.out.println(user.getPassword());
    System.out.println(user.getGender());
    System.out.println(user.getSpe());
    System.out.println(user.getCollegeId());
    for(String h : user.getHobby()) {
        System.out.println(h);
    }
    System.out.println("======================");
    System.out.println(user.getCityId());
    return "";
}

/**
 * 准备编辑
 * @param uid
 * @param request
 * @return
 */
@RequestMapping(value = "/edit/{userId}", method=RequestMethod.GET)
public String toEdit(@PathVariable("userId") String uid,
        HttpServletRequest request) {
    request.setAttribute("action", "edit");
    //从数据库中检索出对应用户
    User user = new User();
    user.setName("张三");
    user.setPassword("123");
    user.setGender("f");
    user.setSpe("音乐专业");
    user.setCollegeId(2);
    user.setCityId(1);
    user.setHobby(new String[] {"pc","read"});
    request.setAttribute("user", user);

    //从数据库中检索出专业的集合
    List<String> special = new ArrayList<>();
    special.add("计算机专业");
    special.add("美术专业");
    special.add("音乐专业");
    request.setAttribute("special", special);

    //从数据库中检索出所有的学院
    List<College> colleges = new ArrayList<>();
    College c1 = new College();
    c1.setId(1);
    c1.setName("计算机学院");
    College c2 = new College();
    c2.setId(2);
    c2.setName("美术学院");
    College c3 = new College();
    c3.setId(3);
    c3.setName("音乐学院");
    colleges.add(c1);
    colleges.add(c2);
    colleges.add(c3);
    request.setAttribute("colleges", colleges);

    List<String> hobbys = new ArrayList<>();
    hobbys.add("pc");
    hobbys.add("read");
```

```
        hobbys.add("football");
        request.setAttribute("hobbys", hobbys);

        List<City> citys = new ArrayList<>();
        City ci1 = new City();
        ci1.setId(1);
        ci1.setName("北京");
        City ci2 = new City();
        ci2.setId(2);
        ci2.setName("天津");
        City ci3 = new City();
        ci3.setId(3);
        ci3.setName("石家庄");
        citys.add(ci1);
        citys.add(ci2);
        citys.add(ci3);
        request.setAttribute("citys", citys);
        return "userForm";
    }

    /**
     * 编辑
     * @param user
     * @return
     */
    @RequestMapping(value = "/edit", method=RequestMethod.POST)
    public String edit(User user) {
        System.out.println(user.getName());
        System.out.println(user.getPassword());
        System.out.println(user.getGender());
        System.out.println(user.getSpe());
        System.out.println(user.getCollegeId());
        for(String h : user.getHobby()) {
            System.out.println(h);
        }
        System.out.println("======================");
        System.out.println(user.getCityId());
        return "";
    }

}
```

views.properties

```
test.(class)=org.springframework.web.servlet.view.InternalResourceViewResolver
test.url=/test.jsp



test1.(class)=org.yyyyyyy
test1.url=exlceview



controller--test--resourcebundleviewresolver---views.properties----test.jsp
```

spring-mvc,启用视图解析器

```
<!-- views:资源文件的名字 -->
<bean class="org.springframework.web.servlet.view.ResourceBundleViewResolver">
       <property name="basename" value="views"></property>
</bean>
```

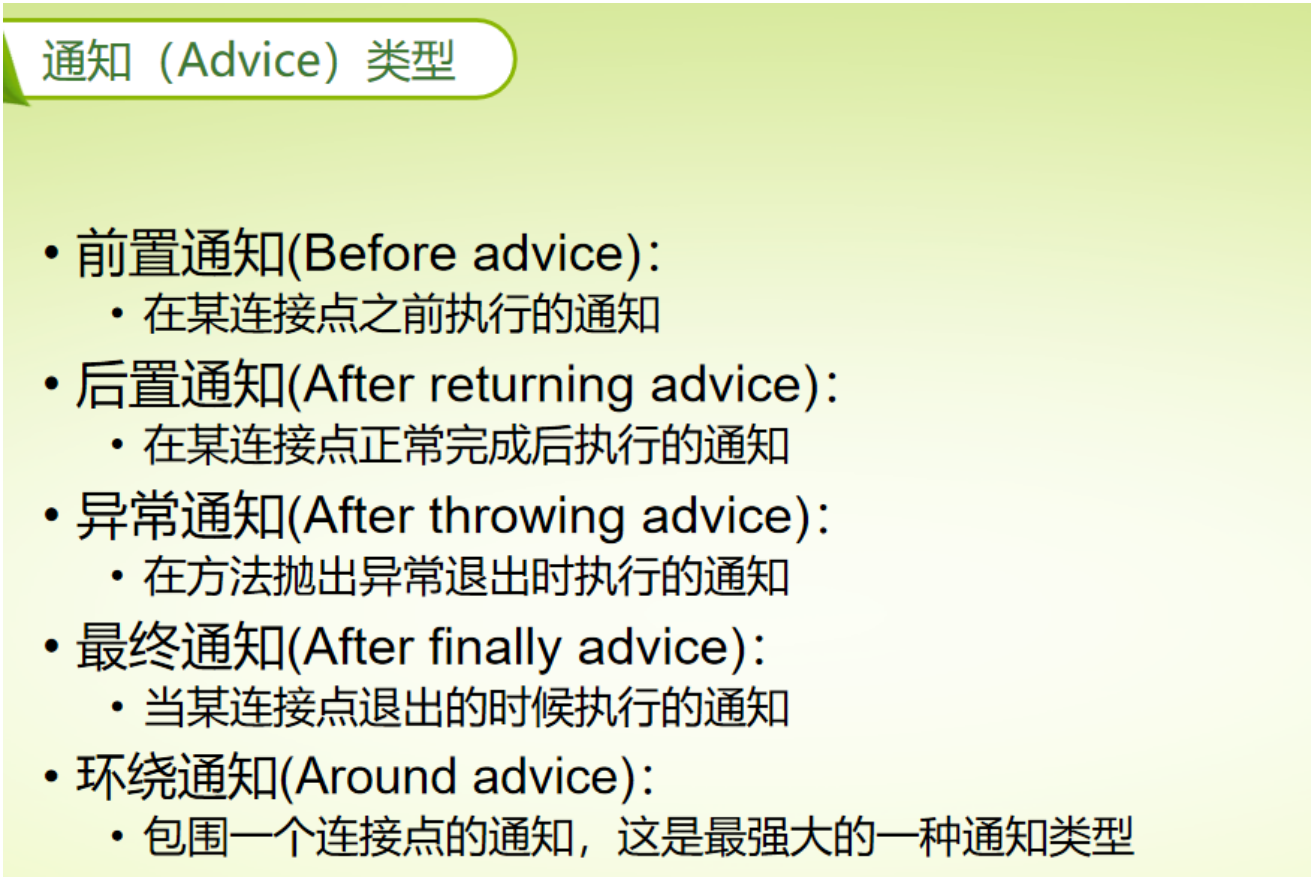
CH08_Spring对JDBC的支持

简答

IOC和DI的概念与关系

- IoC（Inversion of Control，控制反转）：
  - 设计原则，解耦组件之间的依赖关系
- DI（DI(Dependency Injection，依赖注入）：
  - 具体的设计模式，体现了IoC的设计原则
  - 因为DI是IoC最典型的实现，所以术语IoC与DI经常被混用

AOP编程：通知类型

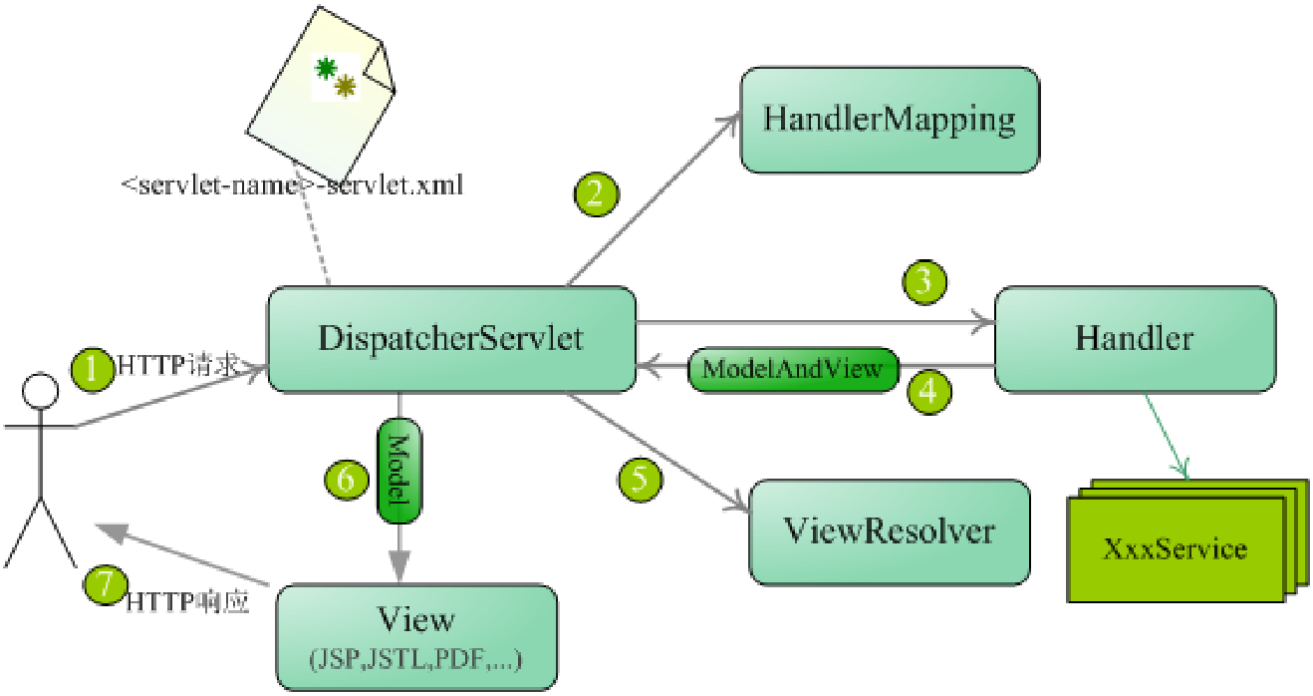


BeanFatory与ApplicationContext区别

BeanFactoty	**ApplicationContext**
功能：基本功能	增加企业特定功能
关系：接口	子接口，超集
Bean载入方式：延迟加载,getBean()	立即加载

Bean的生命周期

步 骤	**说 明**
1.实例化	Spring实例化Bean
2.设置属性	Spring注入Bean的属性
3.设置Bean名称，Bean工厂，应用上下文	如果Bean实现了XXXAware接口，执行对应方法
4.预处理(在初始化之前)	调用BeanPostProcessor对象的postProcessBeforeInitialization()方法
5.初始化Bean	实现InitializingBean接口的afterPropertiesSet()方法声明了初始化方法，将调用声明的
6.预处理(在初始化之后)	调用BeanPostProcessor对象的postProcessAfterInitialization()方法
7.Bean已经准备好	默认以单例的形式存在Spring容器中
8.销毁Bean	实现DisposableBean接口的destroy()方法声明了销毁方法，将调用声明的



SpringMVC体系结构

- 1、客户端发出请求，交给DispatcherServlet处理
- 2、DispatcherServlet根据请求信息及HandlerMapping的配置找到处理请求的处理器（Handler）
- 3、DispatcherServlet通过HandlerAdapter对Handler进行封装，再以统一的适配器接口调用Handler
- 4、处理器完成业务逻辑，返回一个ModelAndVIew给DispatcherServlet，ModelAndView包含视图逻辑名和模型数据信息
- 5、DispatcherServlet借由ViewResolver完成逻辑视图名到真实视图的解析工作
- 6、得到View真实视图后，DispatcherServlet就使用这个View对象对ModelAndView中的模型数据进行渲染
- 7、最终客户得到响应