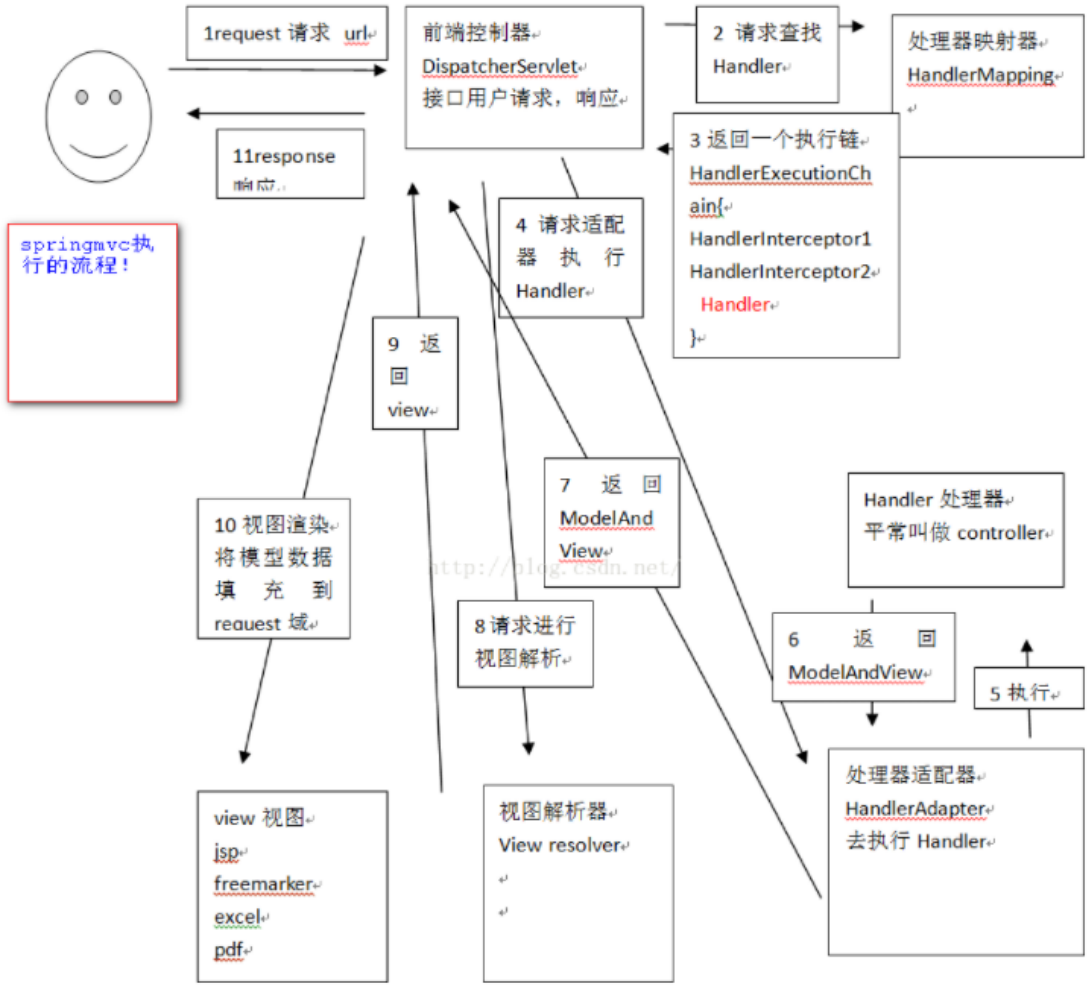
1.什么是Spring MVC？简单介绍下你对springMVC的理解?

Spring MVC是一个基于Java的实现了MVC设计模式的请求驱动类型的轻量级Web框架，通过把Model，View，Controller分离，将web层进行职责解耦，把复杂的web应用分成逻辑清晰的几部分，简化开发，减少出错，方便组内开发人员之间的配合。

2.SpringMVC的执行流程



第一步：用户发送请求被前端控制器(DispatcherServlet)接收

第二步：DispatcherServlet 接收到请求后, 将请求信息交给处理器映射器 （HandlerMapping），

可以根据xml配置、注解进行查找

第三步：HandlerMapping 根据用户的url请求 查找匹配该url的 Handler，并返回一个执行链**

第四步、第五步、第六部：DispatcherServlet 再请求 处理器适配器(HandlerAdapter) 调用相应的 Handler 进行处理并返回 ModelAndView 给 HandlerAdapter

第七步：HandlerAdapter向DispatcherServlet返回ModelAndView

ModelAndView是springmvc框架的一个底层对象，包括Model和view

第八步：前端控制器请求视图解析器去进行视图解析

根据逻辑视图名解析成真正的视图(jsp)

第九步：视图解析器向前端控制器返回View

第十步：前端控制器进行视图渲染

第十一步：前端控制器向用户响应结果

3. Springmvc的优点

1. 可以支持各种视图技术，而不仅仅局限于JSP；
2. 与Spring框架集成（如IoC容器、AOP等）；
3. 清晰的角色分配：前端控制器(dispatcherServlet)，请求到处理器映射（handlerMapping)，处理器适配器（HandlerAdapter)，视图解析器（ViewResolver）。
4. 支持各种请求资源的映射策略。

4. SpringMVC怎么样设定重定向和转发的

转发：在返回值前面加"forward:"，譬如"forward:user.do?name=method4"

重定向：在返回值前面加"redirect:"，譬如"redirect:http://www.baidu.com"

5. SpringMVC常用的注解有哪些

@RequestMapping：用于处理请求 url 映射的注解，可用于类或方法上。用于类上，则表示类中的所有响应请求的方法都是以该地址作为父路径。

@RequestBody：注解实现接收http请求的json数据，将json转换为java对象。

@ResponseBody：注解实现将conreoller方法返回对象转化为json对象响应给客户。

6. SpingMvc中的控制器的注解一般用哪个？有没有别的注解可以替代

一般用@Controller注解，也可以使用@RestController，@RestController注解相当于@ResponseBody + @Controller，表示是表现层，除此之外，一般不用别的注解代替

7. springMVC和struts2的区别有哪些

1. springmvc的入口是一个servlet即前端控制器（DispatchServlet），而struts2入口是一个filter过虑器（StrutsPrepareAndExecuteFilter）。
2. springmvc是基于方法开发(一个url对应一个方法)，请求参数传递到方法的形参，可以设计为单例或多例(建议单例)，struts2是基于类开发，传递参数是通过类的属性，只能设计为多例。
3. Struts采用值栈存储请求和响应的数据，通过OGNL存取数据，springmvc通过参数解析器是将request请求内容解析，并给方法形参赋值，将数据和视图封装成ModelAndView对象，最后又将ModelAndView中的模型数据通过reques域传输到页面。Jsp视图解析器默认使用jstl。

8. 如何解决POST请求中文乱码问题，GET的又如何处理呢

1. 解决post请求乱码问题：在web.xml中配置一个CharacterEncodingFilter过滤器，设置成utf-8

```
<filter>
    <filter-name>CharacterEncodingFilter</filter-name>
    <filter-
class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
    <init-param>
        <param-name>encoding</param-name>
        <param-value>utf-8</param-value>
    </init-param>
</filter>

<filter-mapping>
    <filter-name>CharacterEncodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

2. get请求中文参数出现乱码解决方法有两个:
    - 修改tomcat配置文件添加编码与工程编码一致，如下:

      ```
      <ConnectorURIEncoding="utf-8" connectionTimeout="20000" port="8080"
      protocol="HTTP/1.1" redirectPort="8443"/>
      ```

    - 另外一种方法对参数进行重新编码:

      ```
      String userName = new
      String(request.getParamter("userName").getBytes("ISO8859-1"),"utf-8")
      ```

      > ISO8859-1是tomcat默认编码，需要将tomcat编码后的内容按utf-8编码。

9. 拦截器

拦截器可以用于权限验证、解决乱码、操作日志记录、性能监控、异常处理等

自定义拦截器：可以通过继承Spring框架中的HandlerInterceptorAdapter类，然后重写preHandle、postHandle、afterCompletion三个方法，在三个方法中写我们自己要想实现的业务逻辑代码。

```
/**
 * 自定义拦截器
 * @author admol
 */
public class MyInterceptor extends HandlerInterceptorAdapter {
    /**
     * 在请求之前执行
     * @param request
     * @param response
     * @param handler 表示被拦截的请求目标
     * @return false:拦截请求,终止请求 true:继续执行请求
     */
    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse
response,
                             Object handler) throws Exception {
        //业务逻辑代码编写...(如:解决乱码,权限验证)
        request.setCharacterEncoding("utf-8");
```

```
        return true;
    }

    /**
     * @param modelAndView 可以对视图进行操作
     */
    @Override
    public void postHandle(HttpServletRequest request, HttpServletResponse
response,
                           Object handler, ModelAndView modelAndView) throws
Exception {
        //业务逻辑代码编写...(如:操作日志记录,更改视图信息)
    }

    /**
     * @param ex 异常
     */
    @Override
    public void afterCompletion(HttpServletRequest request, HttpServletResponse
response,
                                Object handler, Exception ex) throws Exception {
        //业务逻辑代码编写...(如:,资源销毁,异常处理)
    }
}
```

接着需要注册拦截器才可以使用：配置xml 或者**编写拦截器配置文件类并继承 WebMvcConfigurer类**

```
<!-- 配置拦截器 -->
<mvc:interceptors>
    <!-- 多个拦截器，按顺序执行 -->
    <mvc:interceptor>
        <mvc:mapping path="/**"/> <!-- 表示拦截所有的url包括子url路径 -->
        <bean id="myInterceptor" class="com.admol.web.MyInterceptor" />
    </mvc:interceptor>
    <mvc:interceptor>
        <mvc:mapping path="/**"/>
        <bean class="ssm.interceptor.HandlerInterceptor2"/>
    </mvc:interceptor>
    <mvc:interceptor>
        <mvc:mapping path="/**"/>
        <bean class="ssm.interceptor.HandlerInterceptor3"/>
    </mvc:interceptor>
</mvc:interceptors>
```

1.编写拦截器实现类，实现接口 HandlerInterceptor，

重写里面需要的三个比较常用的方法，实现自己的业务逻辑代码

(就是自己拦截器拦截时做什么处理)

```
import java.io.IOException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.servlet.HandlerInterceptor;
import org.springframework.web.servlet.ModelAndView;
import com.*.*.*.User;
public class AdminInterceptor implements  HandlerInterceptor {
```

```
    /**
     * 在请求处理之前进行调用（Controller方法调用之前）
     */
    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse
response, Object handler) {
//        System.out.println("执行了TestInterceptor的preHandle方法");
        try {
            //统一拦截（查询当前session是否存在user）(这里user会在每次登陆成功后，写入
session)
            User user=(User)request.getSession().getAttribute("USER");
            if(user!=null){
                return true;
            }
            response.sendRedirect(request.getContextPath()+"你的登陆页地址");
        } catch (IOException e) {
            e.printStackTrace();
        }
        return false;//如果设置为false时，被请求时，拦截器执行到此处将不会继续操作
                     //如果设置为true时，请求将会继续执行后面的操作
    }

    /**
     * 请求处理之后进行调用，但是在视图被渲染之前（Controller方法调用之后）
     */
    @Override
    public void postHandle(HttpServletRequest request, HttpServletResponse
response, Object handler, ModelAndView modelAndView) {
//         System.out.println("执行了TestInterceptor的postHandle方法");
    }

    /**
     * 在整个请求结束之后被调用，也就是在DispatcherServlet 渲染了对应的视图之后执行（主要是
用于进行资源清理工作）
     */
    @Override
    public void afterCompletion(HttpServletRequest request, HttpServletResponse
response, Object handler, Exception ex) {
//        System.out.println("执行了TestInterceptor的afterCompletion方法");
    }

}
```

2.编写拦截器配置文件类并继承 WebMvcConfigurer类，并重写其中的方法 addInterceptors并且在主类上加上注解 @Configuration

```
import org.springframework.context.annotation.Configuration;
import
org.springframework.web.servlet.config.annotation.InterceptorRegistration;
import org.springframework.web.servlet.config.annotation.InterceptorRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
import com.*.*.interceptor.AdminInterceptor;

@Configuration
public class LoginConfig implements WebMvcConfigurer {
```

```
    @Override
    public void addInterceptors(InterceptorRegistry registry) {
        //注册TestInterceptor拦截器
        InterceptorRegistration registration = registry.addInterceptor(new
AdminInterceptor());
        registration.addPathPatterns("/**");                    //所有路径都被拦
截
        registration.excludePathPatterns(                       //添加不拦截路径
                                        "你的登陆路径",          //登录
                                        "/**/*.html",           //html静态资源
                                        "/**/*.js",             //js静态资源
                                        "/**/*.css",            //css静态资源
                                        "/**/*.woff",
                                        "/**/*.ttf"
                                        );
    }
}
```

其他实现自定义拦截器的方式：

1. 实现接口 `implements HandlerInterceptor`
2. 实现接口 `implements WebRequestInterceptor`
   注册拦截器方法不变

拦截器与过滤器的区别：

1.过滤器是依赖于Servlet容器，基于回调函数；Intercepto依赖与框架，基于反射机制。
2.过滤器的过滤范围更大，还可以过滤一些静态资源，拦截器只拦截请求。

执行顺序：

1. 当两个拦截器都实现放行操作时，顺序为preHanle1、preHandle2、postHandle2、postHandle1、afterCompletion2、afterCompletion1;
2. 当第一个拦截器preHandle返回false,也就是对其进行拦截时，第二个拦截器是完全不执行的，第一个拦截器只执行preHandle部分。
3. 当第一个拦截器preHandle返回True，第二个拦截器preHandle返回false，顺序为preHandle1、preHandle2、afterCompletion1

10. SpringMVC怎么和AJAX相互调用的

通过Jackson框架就可以把Java里面的对象直接转化成Js可以识别的Json对象。具体步骤如下：

1. 加入Jackson.jar
2. 在配置文件中配置json的映射
3. 在接受Ajax方法里面可以直接返回Object、List等，但方法前面要加上@ResponseBody注解。

11. Spring MVC的异常处理

可以将异常抛给Spring框架，由Spring框架来处理；我们只需要配置简单的异常处理器，在异常处理器中添视图页面即可。

12. SpringMvc的控制器是不是单例模式？如果是，有什么问题？怎么解决

是单例模式，在多线程访问的时候有线程安全问题，解决方案是在控制器里面不能写可变状态量，如果需要使用这些可变状态，可以使用ThreadLocal机制解决，为每个线程单独生成一份变量副本，独立操作，互不影响。

13. 如果在拦截请求中，我想拦截get方式提交的方法，怎么配置

可以在@RequestMapping注解里面加上method=RequestMethod.GET。

14. 怎样在方法里面得到Request，或者Session

直接在方法的形参中声明request，SpringMvc就自动把request对象传入。

15. 如果想在拦截的方法里面得到从前台传入的参数，怎么得到

直接在形参里面声明这个参数就可以，但必须名字和传过来的参数一样。

16. 如果前台有很多个参数传入，并且这些参数都是一个对象的，那么怎么样快速得到这个对象

直接在方法中声明这个对象，SpringMvc就自动会把属性赋值到这个对象里面。

17. SpringMvc中函数的返回值是什么

返回值可以有很多类型，有String，ModelAndView。ModelAndView类把视图和数据都合并的一起的，但一般用String比较好。

18. SpringMvc用什么对象从后台向前台传递数据的

通过ModelMap对象，可以在这个对象里面调用put方法，把对象加到里面，前端就可以通过el表达式拿到。

19. 怎么样把ModelMap里面的数据放入Session里面

可以在类上面加上@SessionAttributes注解，里面包含的字符串就是要放入session里面的key

20. 说一下struts2和springMVC有什么不同?

\1. 核 心控制器（前端控制器、预处理控制器）：spring mvc核心控制器是Servlet，而Struts2是Filter。

2.控制器实例：Spring Mvc会比Struts快一些（理论上）。Spring Mvc是基于方法设计，而Sturts是基于对象，每次发一次请求都会实例一个action，每个action都会被注入 属性，而Spring更像Servlet一样，只有一个实例，每次请求执行对应的方法即可(注意：由于是单例实例，所以应当避免全局变量的修改，这样会产生线程安全问题)。

\3. 管理方式：大部分的公司的核心架构中，就会使用到spring,而spring mvc又是spring中的一个模块，所以spring对于spring mvc的控制器管理更加简单方便，而且提供了全 注解方式进行管理，各种功能的注解都比较全面，使用简单，而struts2需要采用XML很多的配置参数来管理（虽然也可以采用注解，但是几乎没有公司那 样使用）。

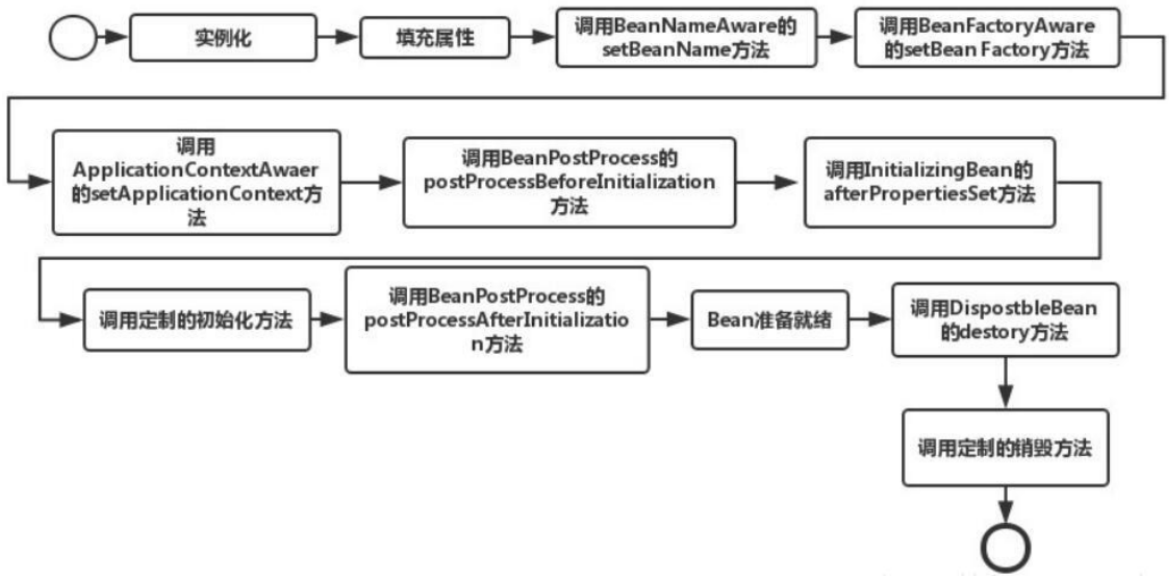
4.参数传递：Struts2中自身提供多种参数接受，其实都是通过（ValueStack）进行传递和赋值，而SpringMvc是通过方法的参数进行接收。

5.学习难度：Struts更加很多新的技术点，比如拦截器、值栈及OGNL表达式，学习成本较高，springmvc 比较简单，很较少的时间都能上手。

6.intercepter 的实现机制：struts有以自己的interceptor机制，spring mvc用的是独立的AOP方式。

7.spring mvc处理ajax请求,直接通过返回数据，方法中使用注解@ResponseBody，spring mvc自动帮我们对象转换为JSON数据。而struts2是通过插件的方式进行处理

SpringBean 的生命周期



（1）实例化Bean：

对于BeanFactory容器，当客户向容器请求一个尚未初始化的bean时，或初始化bean的时候需要注入另一个尚未初始化的依赖时，容器就会调用createBean进行实例化。对于ApplicationContext容器，当容器启动结束后，通过获取BeanDefinition对象中的信息，实例化所有的bean。

（2）设置对象属性（依赖注入）：

实例化后的对象被封装在BeanWrapper对象中，紧接着，Spring根据BeanDefinition中的信息 以及 通过BeanWrapper提供的设置属性的接口完成依赖注入。

（3）处理Aware接口：

接着，Spring会检测该对象是否实现了xxxAware接口，并将相关的xxxAware实例注入给Bean：

①如果这个Bean已经实现了BeanNameAware接口，会调用它实现的setBeanName(String beanId)方法，此处传递的就是Spring配置文件中Bean的id值；

②如果这个Bean已经实现了BeanFactoryAware接口，会调用它实现的setBeanFactory()方法，传递的是Spring工厂自身。

③如果这个Bean已经实现了ApplicationContextAware接口，会调用setApplicationContext(ApplicationContext)方法，传入Spring上下文；

（4）BeanPostProcessor：

如果想对Bean进行一些自定义的处理，那么可以让Bean实现了BeanPostProcessor接口，那将会调用postProcessBeforeInitialization(Object obj, String s)方法。

（5）InitializingBean 与 init-method：

如果Bean在Spring配置文件中配置了 init-method 属性，则会自动调用其配置的初始化方法。

（6）如果这个Bean实现了BeanPostProcessor接口，将会调用postProcessAfterInitialization(Object obj, String s)方法；由于这个方法是在Bean初始化结束时调用的，所以可以被应用于内存或缓存技术；

以上几个步骤完成后，Bean就已经被正确创建了，之后就可以使用这个Bean了。

（7）DisposableBean：

当Bean不再需要时，会经过清理阶段，如果Bean实现了DisposableBean这个接口，会调用其实现的destroy()方法；

（8）destroy-method：

最后，如果这个Bean的Spring配置中配置了destroy-method属性，会自动调用其配置的销毁方法。