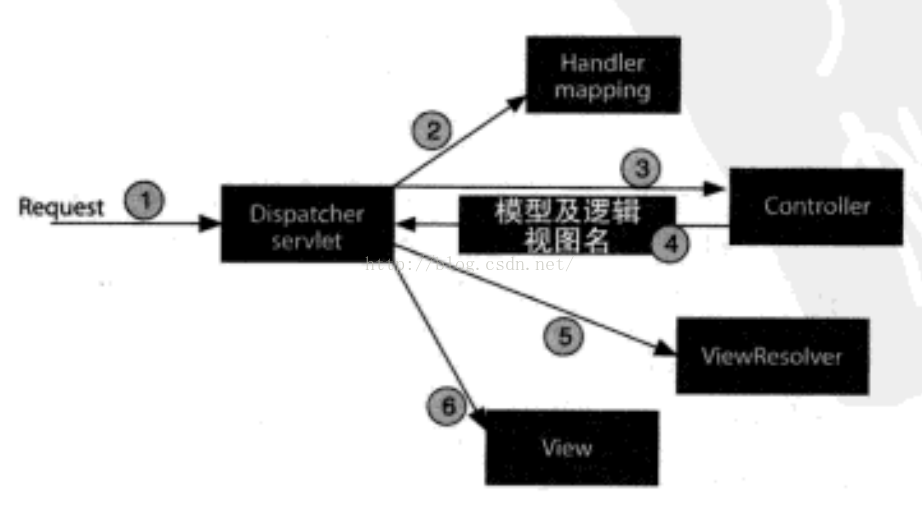
<http://www.cnblogs.com/wewill/p/5672232.html>

springMVC是一个MVC框架，他控制着请求相应的整个流程，从请求一进入到应用服务器到相应离开，都离不开mvc框架

# 请求在应用服务器中

先说说请求相应在应用服务器的整个过程



* DisptacherServlet接收到请求，并读取出请求中的数据
* DisptacherServlet根据请求中的信息(url)，去Handlermapping中查找url对应的资源（如果没有则报404）
* 如果资源指向一个Controller,则将请求和请求数据发送给Controller
* Controller处理请求（一般Controller将处理动作交给业务层处理），将处理好的数据和视图名返回给DisptacherServlet
* DisptacherServlet根据视图名去查找ViewResolver视图解析器，将数据交给对应的视图处理
* 视图(jsp,html等等)拿到数据后，渲染

清楚了请求在服务器中的流程后，知道了使用一个mvc框架我们要进行几步配置

* 初始化DisptacherServlet（配置Handlermapping和Controller的位置）
* 配置Handlermapping(ServletMapping)
* 标识Controler，配置Controler要处理的url以及返回的视图名
* 配置视图解析器

第二步和第三步是在一起的，因为Controler就是Servlet**Handlermapping和Controller的配置在web.xml中就相当于<servlet>和<servlet-mapping>**DisptacherServlet需要从web.xml中拿关于springmvc的配置(servlel,servletmapping)，才能调度整个流程

## 关于DisptacherServlet

**（可以先跳过这部分，往后面看）**DisptacherServlet是SpringMVC的调度中心，在这里请求会第一次进入到SpringMVCDisptacherServlet配置着SpringMVC运作的信息在以前，我们会在web.xml中配置SpringMVC,然后SpringMVC会根据web.xml中的配置初始化DisptacherServlet在servlet3.0之后，出现了ServletContainerInitializer，可以在项目启动最开始时，会自动扫描实现了ServletContainerInitializer接口的类，对Servlet、Listener、Filter进行动态注册，利用此技术，可以通过实现ServletContainerInitializer来配置DisptacherServlet（注册controller,ViewResolve），达到SpringMVC零配置的效果

## 关于ServletContainerInitializer

**（可以先跳过这部分，往后面看）ServletContainerInitializer的作用和web.xml差不多，都是部署servlet,listener,filter,只是一个是用java代码配置，一个用xml配置**在实现了ServletContainerInitializer接口的类上要注解上@HandlesTypes(//要动态注册的class，一定要是Servlet或者Listener或者Filter)ServletContainerInitializer接口中只有一个方法onStart(Set<Class>,ServletContext)当应用启动的时候，会调用onStart方法，参数Set<Class>就是@HandlesTypes中的类，意味着这些类可以被动态注册实现动态注册的核心方法：

* addServlet(String name,Servlet servlet)
* addFilter(String name,Filter filter)
* addListener(Listener listener)

这些都是注册Servlet、listener、filter的方法显然还是有点麻烦，所以随着spring的发展，有更多简单的方法去配置现在最简单就是继承AbstractAnnotationConfigDispatcherServletInitializer，重写getServletConfigClasses、getServletMappings、getRootConfigClasses来配置DisptacherServlet

# 配置DisptacherServlet

**这一步，跟配置web.xml作用差不多，就是要部署注册servlet**在这里我们采用继承重写AbstractAnnotationConfigDispatcherServletInitializer的方法，来配置DisptacherServlet

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | public class WebInitializer extends AbstractAnnotationConfigDispatcherServletInitializer{     //得到中间层(service、dao、aop、po等)的配置   @Override   protected Class<?>[] getRootConfigClasses() {     return new Class[]{DAOConf.class,ServiceConf.class,AOPConf.class};   }     //得到controler和ViewResolver的配置   @Override   protected Class<?>[] getServletConfigClasses() {     return new Class[]{SpringMvcConf.class};   }     //标识哪些url要经过这个DisptacherServlet处理   @Override   protected String[] getServletMappings() {     return new String[]{"/"}；//所有url都被DisptacherServlet处理   }    } |

# SpringMVC配置

这一步是完成SpringMvcConf.class，这个类配置了一些controller和ViewResolver

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | @Configuration  @EnableWebMvc  //扫描控制器  @ComponentScan(includeFilters=@ComponentScan.Filter(type=FilterType.ANNOTATION,value=Controller.class))  public class SpringMvcConf extends WebMvcConfigurerAdapter {     //配置视图解析器   //html解析   @Bean   public ViewResolver htmlResolver(){     InternalResourceViewResolver viewResolver=new InternalResourceViewResolver();     viewResolver.setPrefix("/WEB-INF/view/");     viewResolver.setSuffix(".html");     return viewResolver;   }   //静态资源处理   //当DisptacherServlet接收到了他匹配的请求，但是找不到相应的Controller，就会把这个请求返回给默认的处理（比如交给tomcat处理）   @Override   public void configureDefaultServletHandling(DefaultServletHandlerConfigurer configurer) {     configurer.enable();   }  } |

这里关键有几步：

* @EnableWebMvc，开启springmvc
* @ComponentScan开启自动扫包，扫描所有带有@Controller的Bean
* 添加ViewResolver，这里的配置是使，视图路径为"/WEB-INF/view/"+Controller返回的视图名+".html"
* 因为之前配置了所有路径都被DisptacherServlet接收，这会导致一些静态资源找不到controller去处理，结果出现404，configurer.enable()开启了之后，当springmvc不能处理的时候会交回给默认的处理

# 配置Controller

## 最简单配置

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | @Controller  public class IndexServlet {   @RequestMapping(value="/",method=RequestMethod.GET)//GET方法访问"/"的时候，调用此方法   public String get(Model model){    model.addAttribute("title","index");    return "index";   }  } |

这是一个最简单的控制器，几个要点

* 当类被@Controller注释时，该类就是一个控制器
* 控制器中的方法被@RequestMapping标识，配置了这个方法访问的url和method
* get方法有一个参数Model，这跟Request中的Attribute差不多，model.addAttribute就跟request.addAttribute一样

## 接收参数

接收参数形式有3种：？号传值，路径变量，提交表单

### ?号传值

访问形式：路径?参数名=值&参数名=值

|  |  |
| --- | --- |
| 1  2  3  4 | @RequestMapping(value="/login",method=RequestMethod.GET)  public String login(@RequestParam("username")String username,@RequestParam("password")String password){        return "login";  } |

这样子就能拿到参数了

## 路径变量

这种也是用的比较多的，通过占位符来实现访问形式：路径/参数值

|  |  |
| --- | --- |
| 1  2  3  4 | @RequestMapping(value="/logout/{id}",method=RequestMethod.GET)  public String logout(@PathVariable("id")int id){    return "logout";  } |

比如访问"/logout/123"，123就是id参数

## 表单变量

当表单要上传一个对象的时候，这种方式就很好用了，springmvc会自动将读取表单数据然后实例化相应bean

|  |  |
| --- | --- |
| 1  2  3  4 | @RequestMapping(method=RequestMethod.POST,path="/user")  public String Post(User user){  return "post";  } |

查看原文：<http://zswlib.com/2016/07/15/springmvc%e5%9f%ba%e6%9c%ac%e4%bd%bf%e7%94%a8/>