**摘要**

asdasdasdasd

**ABSTRACT**

asdasdasdasdasdasdasdasd

目录

[第一章 绪论 1](#_Toc396528637)

[1.1选题的目的与意义 1](#_Toc815045418)

[1.2国内外研究现状 1](#_Toc1817639760)

[1.3本报告主要工作 1](#_Toc1116567745)

[第二章 相关技术研究 1](#_Toc1441982729)

[2.1相关技术介绍 1](#_Toc1050769908)

[第三章 系统需求分析 1](#_Toc1531814475)

[3.1功能需求 1](#_Toc1171921089)

[3.1.1功能概述 1](#_Toc1905216186)

[3.1.2功能需求 1](#_Toc1987261332)

[3.2可行性研究 1](#_Toc88045133)

[第四章 系统概要设计 2](#_Toc158317548)

[3.1系统总体架构设计 2](#_Toc110790603)

[3.2系统功能模块设计 2](#_Toc189342672)

[第五章 系统详细设计 2](#_Toc1859007097)

[第六章 系统功能实现 2](#_Toc592699076)

[4.1系统初始化实现 2](#_Toc1464215546)

[1. 为何需要初始化？ 2](#_Toc1055570649)

[2.初始化详细流程 2](#_Toc613489876)

[4.2处理器模块实现 2](#_Toc855356685)

[4.3请求参数自动注入模块实现 2](#_Toc724271777)

[4.4处理器映射模块实现 2](#_Toc898444843)

[4.5处理器适配器模块实现 2](#_Toc1204954244)

[4.6视图处理器模块实现 2](#_Toc895187698)

[第七章 系统测试 2](#_Toc149209404)

[第八章 结束语 3](#_Toc1649036979)

[7.1全文总结 3](#_Toc2088041518)

[7.2不足与下一步工作 3](#_Toc1683517399)

[致谢 3](#_Toc1779875768)

**第一章 绪论**

**1.1选题的目的与意义**

MVC框架是一种在应用程序开发中使用的一种框架，它将业务逻辑，

数据和视图分离的方式来组织代码。互联网发达的今天，

MVC框架广泛应用于各种程序中。MVC框架具有极高的可复用性，

因此，对MVC框架的研究与实现具有十分重要的意义。

**1.2国内外研究现状**

**1.3本报告主要工作**

1.实现IoC容器, 并且可以通过注解配置bean以及配置bean是原型还是单例

2.实现DispatcherServler，DispatcherServlet做到良好控制MVC框架的运行流程。

3.实现HandlerMapping，HandlerMapping做到能够根据请求找到特定的Controller。

4.实现ModelAndView，ModelAndView能够封装数据和视图信息。

5.实现ViewResolver，ViewResolver能够根据ModelAndView生成页面。

**第二章 相关技术研究**

**2.1相关技术介绍**

1. Java

Java是一种语言

**第三章 系统需求分析**

**3.1功能需求**

**3.1.1功能概述**

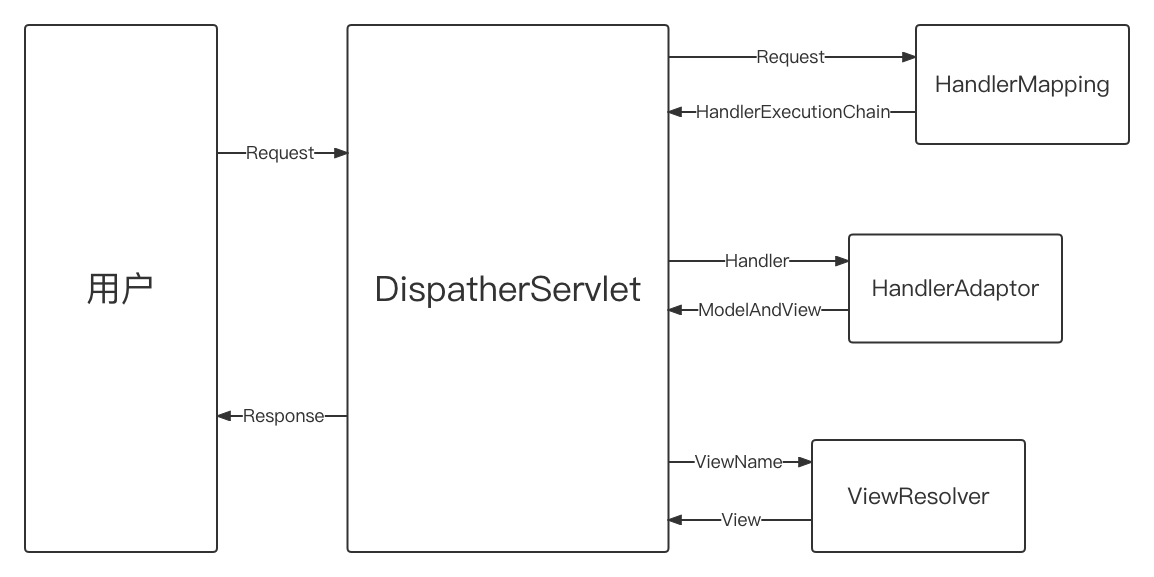
**3.1.2功能需求**

**3.2可行性研究**

**第四章 系统概要设计**

**3.1系统总体架构设计**

系统总体架构图如下

每次 HTTP请求执行流程：

1. 用户的HTTP请求由Servlet容器处理并分配线程传入DispatcherServlet。

2. DispatcherServlet接受请求后，向HandlerMapping传入Request对象，HandlerMapping模块根据Request对象的URI和HTTP请求方法，将处理器和所有符合条件的拦截器封装为HandlerExecutionChain对象返回给DispatcherServlet。

3. DispathcerServlet拿到处理器执行链后，调用所有拦截器的前置处理。

4. DispatcherServlet使用处理器适配器调用处理器，返回ModelAndView对象，其中包含要响应的数据和视图名。

5. DispatcherServlet向视图处理器发送视图名接受视图对象，

再由视图对象渲染视图将数据渲染到视图中。

6. 最后Servlet容器返回响应对象。

该系统属于框架项目，用于帮助开发人员提高开发效率，

该系统基于MVC模式的实现，架构图如下：

**3.2** **系统模块架构设计**

DispatcherServlet是整个MVC框架最顶层的类，也是整个框架最核心的控制者，它作为管理者具有如下功能：

1. 初始化HandlerMapping模块，HandlerAdapter模块，ViewResolver模块，确保这些模块能够正常运行。

2. 实现完整的框架流程控制，作为整个框架饿管理者，此类必须实现良好的流程控制。

3. 此类是依托与ServletAPI实现的类，所以它必须实现Servlet的接口，并提供必要的功能。

**第五章 系统详细设计**

**5.1 处理器映射器模块详细设计**

**5.1.2 处理器映射器功能详细设计**

1. 处理器映射器的定位为管理所有的处理器，并为DispatcherServlet提供处理器映射，所以再初始化阶段，处理器映射器必须完成所有处理器创建与管理。

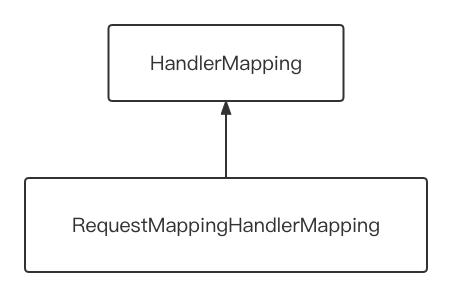
2. 处理器映射器因为要向DispatcherServlet提供获取处理器的服务，所以理所当然应当具有获取处理器执行链的功能。

3. 处理器映射器不仅仅提供最基本的功能，还应当实现同时实现拦截的功能，故处理器映射器应当具有注册拦截器已经在DispatcherServlet请求处理器执行链时，将处理器以及其对应的所有拦截器封装为一个对象返回给DispatcherServlet。

**5.1.2 处理器映射器代码架构详细设计**

处理器映射器代码继承架构图：

代码架构说明：

1. HandlerMapping 是一个接口，定义与DispatcherServlet交互的接口规范。

2. RequestMappingHandlerMapping 是一个继承HandlerMapping接口的具体实现类，它是针对使用@RequestMApping注解定义的处理器的处理器映射器。

**5.2 处理器适配器模块详细设计**

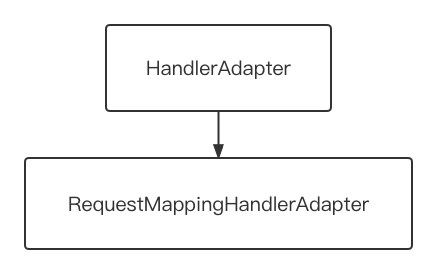
**5.2.1 处理器适配器功能详细设计**

1. 处理器适配器存在的目的是为了针对不同的处理器进行适配，故处理器适配器需要具有适配并执行处理器的功能。

2. 同时，由于在处理器适配器执行处理器的过程中，处理器需要各种各样的参数，所以处理器适配具有参数解决器的子模块，用于根据请求生成参数以便调用处理器。

**5.2.2 处理器适配器代码架构详细设计**

处理器适配器代码架构详细设计：

代码架构说明：

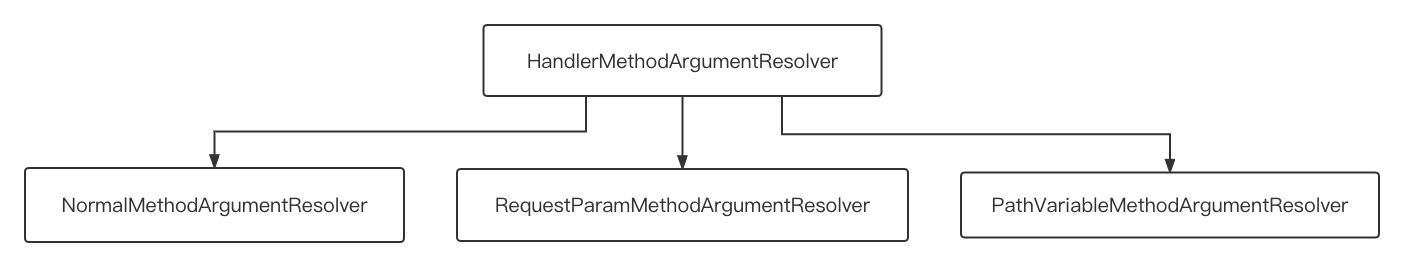
1. HandlerAdapter是一个接口，定义了处理器适配器与DispacherServlet交互的规范。

2. RequestMappingHandlerAdapter是一个基于HandlerAdapter接口的实现类，是针对HandlerMethod初期实现的处理器适配器，但处理器是HandlerMethod类型时可以使用此适配器调用处理器。

**5.2.3 参数解决器子模块详细设计**

参数解决器是作为处理器适配器的子模块存在的，它存在的目的是为了解决在处理器中需要手动获取参数的问题，手动获取参数是每一个请求所必须经历的过程，所以参数解决器封装了这样一个过程，提高了代码的复用性，加快了开发的效率。

在此框架中参数解决器总共有三种类型，它们的代码架构图如下：

1. HandlerMethodArgumentResolver是一个接口，定义了参数解决器所必须的功能，以及其与处理器适配器交互的接口。

2. NormalMethodArgumentResolver是一个基于HandlerMethodArgumentResolver的实现类，此实现类是为了自动注入一些常用的与业务无关的参数，比如Request和Response，此实现类自动生效不需要使用注解定义。

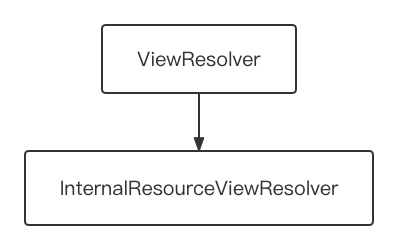
3. RequestParamMethodArgumentResolver是一个基于HandlerMethodArgumetnResolver的实现类，此实现类是为了将request中蕴含的参数自动注入给处理器，使用此功能需要使用@RequestParam注解绑定到处理器的参数上，处理器适配器会根据这个注解自动使用此功能。

4. PathVariableMethodArgumentResolver是一个基于HandlerMethodArgumentResolver的实现类，此实现类是为了将URI路径中蕴含的参数自动注入给处理器使用，使用此功能要在处理器的参数前面使用@PathVariable注解，使用此注解后，在处理器适配器调用此处理器时，会将路径中的参数自动注入处理器的参数中。

**5.3 视图解决器模块的详细设计**

在处理器适配调用完成处理器，处理器适配器会返回ModelAndView对象，但是在这个对象中，只蕴含了Model和视图名，所以根据MVC模式的定义，还需要通过视图名创建一个视图对象，所以视图解决器模块被设计了出来，他的主要功能是通过视图名创建一个视图对象。

其代码架构图如下：

1. ViewResolver是一个接口，它定义了视图解决器的规范，及其与DispatcherServlet的交互的接口。

2. InternalResourceViewResolver是一个基于ViewResolver的实现类，它存在的目的是为了在MVC模式中的视图模块引入JSP模块，使用此实现类即可将JSP的视图模块整合进入框架。

**第六章 系统功能实现**

**6.1系统初始化实现**

为何需要初始化？

系统本质为MVC框架，会通过注解识别程序员编写的代码简化程序员需要编写的代码，所以在程序开始运行时会进行许多的初始化工作，其中包括创建以及初始化IOC容器，初始化处理器映射器，初始化拦截器等一些列初始化工作。

2.初始化详细流程

**6.2处理器映射器模块实现**

1. 初始化关键代码

2. 流程功能关键代码

protected boolean addHandler(HandlerExecutionChain chain, HttpServletRequest request) {

String URI = request.getRequestURI();

String requestMethod = request.getMethod();

HandlerMethod handlerMethod = this.handlerMethodMap.getOrDefault(URI, null);

if (handlerMethod != null) {

for (RequestMethod method : handlerMethod.getRequestMethods()) {

if ( method.toString().equals(requestMethod) ) {

chain.setHandler(handlerMethod);

return true;

}

}

return false;

}

for (Map.Entry<String,HandlerMethod> entry : this.handlerMethodMap.entrySet()) {

boolean isMatching = URI.matches( entry.getKey() );

if (isMatching) {

for ( RequestMethod method : entry.getValue().getRequestMethods() ) {

if ( method.toString().equals(requestMethod) ) {

chain.setHandler( entry.getValue() );

return true;

}

}

return false;

}

}

return false;

}

**6.3请求参数自动注入模块实现**

关键代码

**6.5处理器适配器模块实现**

关键代码

Method method = handlerMethod.getMethod();

Parameter[] parameters = handlerMethod.getParameters();

Object[] arguments = new Object[parameters.length];

for (int i = 0; i < parameters.length; i++) {

MethodParameter methodParameter = new MethodParameter(method, i);

for (HandlerMethodArgumentResolver argumentResolver : this.argumentResolverList) {

if ( argumentResolver.supportsParameter(methodParameter) ) {

arguments[i] = argumentResolver.resolveArgument(request, response, handlerMethod, methodParameter);

break;

}

}

}

Object returnObject = null;

Object controller = handlerMethod.getController();

try {

returnObject = method.invoke(controller, arguments);

}

catch (IllegalAccessException | InvocationTargetException e) {

e.printStackTrace();

}

if (returnObject == null) {

return null;

}

else if (returnObject.getClass() != ModelAndView.class && returnObject.getClass() != String.class) {

return null;

}

else {

return returnObject;

}

**6.6视图处理器模块实现**

关键代码

for (Map.Entry<String,Object> entry : model.entrySet()) {

request.setAttribute(entry.getKey(), entry.getValue());

}

try {

request.getRequestDispatcher(this.URL).forward(request, response);

}

catch (ServletException | IOException e) {

e.printStackTrace();

}

**第七章 系统测试**

**7.1 测试方法**

**7.1.2 黑盒测试**

黑盒测试

**7.1.3 白盒测试**

白盒测试

**7.2 功能测试**

**7.2.1 初始化功能测试**

测试在框架初始化的过程中会不会出现任何的问题，需要保证系统能够正常的初始化，初始化功能测试如表7.1所示：

表7.1初始化功能测试

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 测试目的 | 测试在系统其中后，初始化工程能否正确执行 | | | | | | |
| 编写人 | 张炼 | 时间 | 2018-5-16 | | 状态 | 检测结果为合格 | |
| 前置条件 | 系统能够正常启动 | | | | | | |
| 序号 | 测试步骤 | | | 期待结果 | | | 是否通过 |
| 1 | 直接启动系统，不配置初始化配置类 | | | 启动初始化中断，并其实需要配置初始化配置类 | | | 通过，和预期结果一致 |
| 2 | 配置初始化配置类，然后启动系统 | | | 系统能够正常启动 | | | 通过，和预期结果一致 |
| 3 | 不配置Controller，启动系统 | | | 启动成功，尝试化正常 | | | 通过，和预期结果一致 |
| 4 | 配置Controller，启动系统 | | | 启动成功，初始化正常 | | | 通过，和预期结果一致 |
| 4 | 配置Controller，不配置URI，启动系统 | | | 启动成功，初始化正常 | | | 通过，和预期结果一致 |
| 5 | 配置Controller，配置URI，启动系统 | | | 启动成功，初始化正常 | | | 通过，和预期结果一致 |
| 6 | 配置Controller，配置URI，不配置请求方法 | | | 启动成功，初始化正常 | | | 通过，和预期结果一致 |
| 7 | 配置Controller，配置URI，配置请求方法 | | | 启动成功，初始化正常 | | | 通过，和预期结果一致 |
| 8 | 正确配置Controller，不配置拦截器，启动系统 | | | 启动成功，初始化正常 | | | 通过，和预期结果一致 |
| 9 | 正确配置Controller，配置拦截器 | | | 启动成功，初始化正常 | | | 通过，和预期结果一致 |

**7.2.2处理器映射模块器模块测试**

测试在系统启动后，处理器映射模块能否正常运行，处理器映射器模块测试如表7.2所示：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 测试目的 | 测试处理器映射器能否正常配置使用 | | | | | | |
| 编写人 | 张炼 | 时间 | 2018-5-16 | | 状态 | 检测结果为合格 | |
| 前置条件 | 系统能够正常启动 | | | | | | |
| 序号 | 测试步骤 | | | 期待结果 | | | 是否通过 |
| 1 | 不进行任何配置，在代码中输出已经管理的处理器URI | | | 不输出任何处理器URI | | | 通过，和预期结果一致 |
| 2 | 配置一个处理器，在代码中输出已经管理的URI | | | 输出一个对应的处理器的URI | | | 通过，和预期结果一致 |
| 3 | 配置多个处理器，在代码中输出已经管理的URI | | | 输出多个对应的处理器的URI | | | 通过，和预期结果一致 |
| 4 | 在处理中使用路径参数，在代码中输出URI | | | 输出对应的URI | | | 通过，和预期结果一致 |
| 4 | 配置一个处理器，使用多个请求方法，在代码中输出所以请求方法 | | | 正确输出处理器的所有请求方法 | | | 通过，和预期结果一致 |
| 5 | 配置一个处理器，使用一个请求方法，在代码中输出所以请求方法 | | | 正确输出一个处理器的请求方法 | | | 通过，和预期结果一致 |
| 6 | 配置拦截器，在代码中输出拦截器的类名 | | | 正确输出拦截器的类名 | | | 通过，和预期结果一致 |
| 7 | 配置多个拦截器，在代码中输出拦截器的类名 | | | 正确输出多个拦截器的雷名 | | | 通过，和预期结果一致 |
| 8 | 配置处理器和拦截器，在Servlet中输出处理器执行链 | | | 正确输出处理器执行链 | | | 通过，和预期结果一致 |

**7.2.3 处理器适配器模块功能测试**

处理器适配器模块功能测试如表7.3所示：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 测试目的 | 测试处理器映射器能否正常配置使用 | | | | | | |
| 编写人 | 张炼 | 时间 | 2018-5-16 | | 状态 | 检测结果为合格 | |
| 前置条件 | 系统能够正常启动 | | | | | | |
| 序号 | 测试步骤 | | | 期待结果 | | | 是否通过 |
| 1 | 不进行任何配置，在代码中输出已经管理的处理器URI | | | 不输出任何处理器URI | | | 通过，和预期结果一致 |
| 2 | 配置一个处理器，在代码中输出已经管理的URI | | | 输出一个对应的处理器的URI | | | 通过，和预期结果一致 |
| 3 | 配置多个处理器，在代码中输出已经管理的URI | | | 输出多个对应的处理器的URI | | | 通过，和预期结果一致 |
| 4 | 在处理中使用路径参数，在代码中输出URI | | | 输出对应的URI | | | 通过，和预期结果一致 |
| 4 | 配置一个处理器，使用多个请求方法，在代码中输出所以请求方法 | | | 正确输出处理器的所有请求方法 | | | 通过，和预期结果一致 |
| 5 | 配置一个处理器，使用一个请求方法，在代码中输出所以请求方法 | | | 正确输出一个处理器的请求方法 | | | 通过，和预期结果一致 |
| 6 | 配置拦截器，在代码中输出拦截器的类名 | | | 正确输出拦截器的类名 | | | 通过，和预期结果一致 |
| 7 | 配置多个拦截器，在代码中输出拦截器的类名 | | | 正确输出多个拦截器的雷名 | | | 通过，和预期结果一致 |
| 8 | 配置处理器和拦截器，在Servlet中输出处理器执行链 | | | 正确输出处理器执行链 | | | 通过，和预期结果一致 |

**第八章 结束语**

**7.1全文总结**

本文从相关技术研究，系统需求分析，系统概要设计，系统功能实现，系统测试5个方面阐述了MVC框架的信息。

1.首先对相关技术进行调研，了解了目前相关的主要MVC框架的设计，以及实现技术

2.根据调研获取的信息分析实现的框架的主要需要实现的功能.

3.根据需求的分析和低耦合，高内聚的设计思想设计除了整体的架构以及实现功能的各个模块。

4.整个框架使用了许多代码设计模式，以便于提高代码的扩展性和可维护性。

**7.2不足与下一步工作**

本系统

# 致谢

论文是在唐开山老师的指导下完成的，十分感谢唐开山老师，在编写论文的过程中提供了很多的帮助。