未命名

**2021届本科毕业设计**

**题 目 基于Netty的WEB框架设计与实现**

**学 院 信息与电子工程学院**

**专 业 计算机科学与技术**

**班 级 计算机171**

**学 号 1170199037**

**学生姓名 程月甲**

**指导教师 金国英**

**教师职称 副教授**

**完成日期 2021年5月25日**

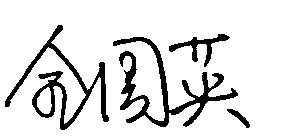
浙江科技学院毕业设计（论文）、学位论文

版权使用授权书

本人 程月甲 学号 1170199037 声明所呈交的毕业设计（论文）、学位论文《 基于Netty的WEB框架设计与实现 》，是在导师指导下进行的研究工作及取得的研究成果。除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，与我一同工作的人员对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

本毕业设计（论文）、学位论文作者愿意遵守**浙江科技学院** 关于保留、使用学位论文的管理办法及规定，允许毕业设计（论文）、学位论文被查阅。本人授权 **浙江科技学院** 可以将毕业设计（论文）、学位论文的全部或部分内容编入有关数据库在校园网内传播，可以采用影印、缩印或扫描等复制手段保存、汇编毕业设计（论文）、学位论文。

（保密的学位论文在解密后适用本授权书）



论文作者签名： 导师签名：

签字日期： 2021年5月25日 签字日期：2021年 5月 25日

# 摘要

当下，移动互联网方兴未艾，以区块链为代表的“互联网3.0”，势如破竹。这一切无不表明互联网代表的虚拟世界正在越来越大，这就需要更多人参与到互联网的建设中。而目前，市场上大多数中小型系统所使用的MVC WEB框架相对而言，比较复杂。并且在网络通讯模式上还是基于BIO，这就限制了服务的高并发。

并且，在前后端分离的大环境下，为了使数据展现过程更加直接，并且提供更好的用户体验，业务服务被拆分成了前端服务和后端服务。其具体流程为：首先从浏览器发送AJAX请求，然后服务端接受该请求并返回JSON数据返回给浏览器，最后在浏览器中进行界面渲染。

本文将在改进版MVC模式的基础上，构建一个轻量高效的WEB框架。

Netty 作为高性能的异步通信框架，常常被用来作为系统服务之间的通信组件，以此来提高系统的通信性能[1]。为了提高整体的高并发，本文将使用NIO杰出代表者Netty作为网络通讯框架。

除网络通讯功能外，具体功能由以下功能模块完成。其中，容器管理模块通过可插拔性IOC容器，进行bean对象管理。请求路由模块，通过基于注解的方式，实现网络的路由请求和参数传递；通过使用fastjson实现json响应；通过责任链模式，实现拦截器。Sql模块，通过依赖JDBC完成数据库连接以及操作；通过依赖sqlbuilder第三方开源库，实现自动生成sql。

最后，将基于框架，构建一个简单的用户管理系统，用来测试框架功能。

**关键词：** Netty，MVC，WEB框架，NIO，轻量级

# ABSTRACT

At present, mobile Internet is in the ascendant, and "Internet 3.0", represented by blockchain, is in full swing. Small and medium-sized systems are important. In the history of Intenet, the continuous development of small and medium-sized systems has promoted the market promotion and technology maturity. At present, the MVC web framework used by most small and medium-sized systems in the market is relatively complex and is based on bio, which limits the high concurrency of services.

This paper will build a lightweight and efficient web framework based on MVC mode.

In order to improve the overall high concurrency, this paper will use netty as the network communication framework. Netty has the characteristics of high concurrency, fast transmission and good encapsulation.

In addition, the specific functions are completed by the following modules. The container management module manages bean objects through pluggable IOC container. The request routing module realizes the routing request and parameter transfer of the network through annotation based method; The response of JSON is realized by using fastjson; Through the chain of responsibility mode, the interceptor is realized. SQL module, through relying on JDBC to complete the database connection and operation; By relying on the third-party open source library of sqlbuilder, the automatic generation of SQL is realized.

Finally, we will build a simple user management system based on the framework to test the framework function.

**Keywords:** Netty，MVC，WEB framework，NIO，lighting

目录

[摘要 I](#_Toc72843204)

[ABSTRACT II](#_Toc72843205)

[第1章 绪论 1](#_Toc72843206)

[1.1 研究背景及意义 1](#_Toc72843207)

[1.2 主要研究内容 2](#_Toc72843208)

[第2章 相关技术介绍 3](#_Toc72843209)

[2.1 Netty 3](#_Toc72843210)

[2.2 Fastjson 6](#_Toc72843211)

[2.3 SqlBuilder 7](#_Toc72843212)

[2.4 设计模式 7](#_Toc72843213)

[2.4.1 SPI 7](#_Toc72843214)

[2.4.2 Ioc 7](#_Toc72843215)

[2.4.3 责任链模式 8](#_Toc72843216)

[第3章 需求分析与设计 9](#_Toc72843217)

[3.1 业务需求 9](#_Toc72843218)

[3.2 功能需求 9](#_Toc72843219)

[3.2.1 容器管理功能 10](#_Toc72843220)

[3.2.2 请求响应功能 10](#_Toc72843221)

[3.2.3 sql功能 10](#_Toc72843222)

[3.3 架构设计 11](#_Toc72843223)

[3.4 功能模块设计 11](#_Toc72843224)

[3.4.1 容器管理模块 12](#_Toc72843225)

[3.4.2 请求响应模块 12](#_Toc72843226)

[3.4.3 sql模块 13](#_Toc72843227)

[第4章 系统实现 15](#_Toc72843228)

[4.1 框架开发环境 15](#_Toc72843229)

[4.2 容器模块实现 15](#_Toc72843230)

[4.2.1 容器管理 15](#_Toc72843231)

[4.2.2 Bean管理 16](#_Toc72843232)

[4.3 请求响应模块实现 17](#_Toc72843233)

[4.3.1 路由请求 17](#_Toc72843234)

[4.3.2 Json响应 19](#_Toc72843235)

[4.3.3 传递参数 20](#_Toc72843236)

[4.3.4 拦截器 21](#_Toc72843237)

[4.4 Sql模块实现 23](#_Toc72843238)

[4.4.1 数据库连接 23](#_Toc72843239)

[4.4.2 Sql语句生成 24](#_Toc72843240)

[第5章 框架功能测试 28](#_Toc72843241)

[5.1 测试案例构建 28](#_Toc72843242)

[5.1.1 数据库设计 28](#_Toc72843243)

[5.1.2 接口设计 29](#_Toc72843244)

[5.2 功能测试 29](#_Toc72843245)

[5.2.1 创建用户 29](#_Toc72843246)

[5.2.2 更新用户 30](#_Toc72843247)

[5.2.3 查找用户 30](#_Toc72843248)

[第6章 总结与展望 32](#_Toc72843249)

[6.1 总结 32](#_Toc72843250)

[6.2 展望 32](#_Toc72843251)

[第7章 结束语（致谢） 33](#_Toc72843252)

[第8章 参考文献 34](#_Toc72843253)

# 绪论

## 研究背景及意义

当下，移动互联网方兴未艾，以区块链为代表的“互联网3.0”，势如破竹。这一切无不表明互联网代表的虚拟世界正在越来越大，这就需要更多人参与到互联网的建设中。虽然在互联网发展的整个进程中，大型系统是其中的中流砥柱，但中小型系统也是不应该不可以被忽视淹没的。无论是互联网1.0出现伊始，还是互联网2.0爆发前夕，中小型系统的前仆后继才推动了市场的推广和技术的成熟。

历史有其发展规律，在软件开发的过程中，也是有迹可循。在低成本的原则下，发展出了低耦合，高内聚的开发原则。在此前提下，各类开发模式涌现，其中MVC(Model View Controller)模式率先问鼎中原。

MVC模式，也称为模型-视图-控制器模式。用一种业务逻辑、数据、界面显示分离的方法组织代码，将业务逻辑聚集到一个部件里面，在改进和个性化定制界面及用户交互的同时，不需要重新编写业务逻辑[2]。MVC被独特的发展起来用于映射传统的输入、处理和输出功能在一个逻辑的图形化用户界面的结构中。它把软件系统分为三个基本部分：模型（Model）：负责存储系统的中心数据。视图（View）：将信息显示给用户（可以定义多个视图）。控制器（Controller）：处理用户输入的信息。负责从视图读取数据，控制用户输入，并向模型发送数据，是应用程序中处理用户交互的部分。负责管理与用户交互交互控制。视图和控制器共同构成了用户接口。

MVC模式，不仅降低了开发成本，缩短了开发周期，增加了代码复用性；同时，也降低了开发人员的学习成本。

基于MVC模式开发的框架并非少数。从时间维度上来说，包括首先利用MVC完成职责划分的Struts框架，后来又有J2EE横扫千军般的攻占企业级开发，再到当下一统江湖的Spring框架。

Struts，Apache的Struts是一个用于开发现代Java Web应用程序的开源免费MVC框架。它包含一组类和接口，共同协作开发Web应用程序[3]。它适用于许多Java应用程序的网络框架，并且受到了很多社区的支持。为了进一步扩展Struts Java框架，Struts 2 作为后继者有了很大的改变。这个框架最适合于开发当代 Java EE web 应用程序的工程师，Apache 软件基金会借助 Struts 2 为开发人员提供了一个用于开发面向网络的，企业级应用程序，并优化整个开发流程的工具。优点：没有线程安全问题，依赖注入使得测试更简单，它可以使用“对象图导航语言”（OGNL），Struts 2框架的Servlet上下文以简单的地图形式表示 ，它支持通过验证方法进行手动验证，并且还包含子框架验证。使用MVC解决了职责划分问题，使Web应用迈向了新的台阶，开始飞速发展。

J2EE ，由Sun和Java 社区推出的面向企业级开发的新框架。除了基于Servlet/Jsp的MVC框架，还提供了满足了企业的数据库， 邮件， 消息，事务处理等需求的一些规范和标准，包括EJB, JMS , JDBC 等。Java EE这一系列富有野心的成果，因为它的便利性和封装性，发展的非常迅猛，J2EE迅速统治了Web应用开发市场。

Spring，当下赤手可热的企业级web开发框架。其通过IOC模式，极其巧妙的改变了java bean笨重的体系，将所有bean对象都灵活巧妙的进行了管理。并且使用AOP模式，可以将核心业务功能和非业务性功能巧妙的进行解耦。通过以上两个新模式的应用，使得java web开发一改往日臃肿局面。开创了轻量级开发的MVC框架。

## 主要研究内容

本文旨在介绍一款基于Netty的MVC WEB框架的设计与实现。其与目前主流生产环境中常见的 MVC框架不同的是，本文将设计并实现一种轻量级，简单易用的MVC 框架。本框架按功能可分为容器管理模块、请求响应模块、sql模块，根据模块的不同，研究的内容主要如下：

1）设计并实现系统的容器管理。本系统通过工厂模式实现可插拔式容器管理。系统引入IOC模式，并且默认使用IOC作为容器管理。

2）设计并实现系统的响应请求功能。该模块将实现路由控制，json响应，拦截器功能。

3）设计并实现数据库操作。该模块通过sqlBuilder实现自动生成sql语句，并且实现数据库的增删改查等基本操作。

# 相关技术介绍

为了实现高性能的MVC框架，在设计过程中，积极使用领域优秀的第三方开源框架，并积极采用领域流行的设计模式。在容器模块中，使用了SPI模式进行灵活选择容器管理模式，作为容器管理的主要模式的IOC模式，广泛采用了反射。在MVC模块中，主要应用了行业顶尖的网络I/O框架Netty进行网络通讯，使用了alibaba的fastjson实现restful格式的相应风格。在数据库模块中，则使用了jdbc的作为连接器，并使用sqlBuilder加反射的模式实现sql语句的动态生成。

## Netty

Netty 即是基于 Reactor 模型开发的应用框架，由 JBOSS 提供。Netty 提供异步的、事件驱动的网络应用程序框架和工具，用以快速开发高性能、高可靠性的网络服务器和客户端程序[4].

Netty是一个利用 Java NIO的能力，隐藏其背后的复杂性而提供一个易于使用的 API 的客户端/服务器框架。一经推出，斩获众多公司的青睐，迅速成为一个新星。Netty大放异彩无非缘由自身的三大特性：并发高，传输快，封装好。

**1）并发高**

首先介绍一下BIO与NIO的差异。在网络通讯中，在建立连接的后，会经过两个步骤，第一步是接收完客户端发过来的全部数据，第二步是服务端处理完请求业务之后返回response给客户端。BIO和NIO的区别在于处理接收完客户端发过来的全部数据时有不同的操作。

其中BIO等待客户端发数据这个过程是阻塞的，这就造成了一个线程只能处理一个请求的情况，而机器能支持的最大线程数是有限的，所以BIO不能支持高并发。如下图。

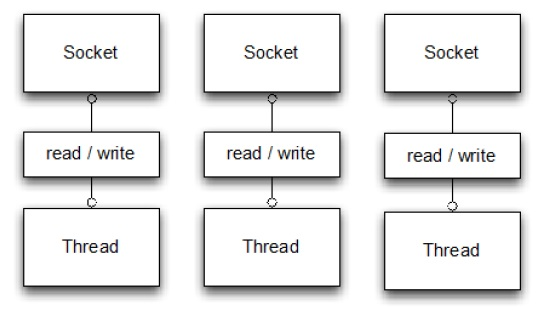


图2.1 BIO模型

而在NIO中，当一个Socket建立好之后，Thread并不会阻塞去接受这个Socket，而是将这个请求交给Selector，Selector会不断的去遍历所有的Socket，一旦有一个Socket建立完成，他会通知Thread，然后Thread处理完数据再返回给客户端。这个过程是不阻塞的，这样就能让一个Thread处理更多的请求。

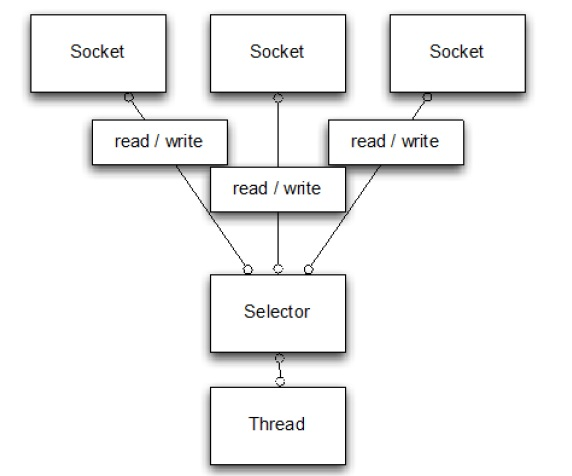
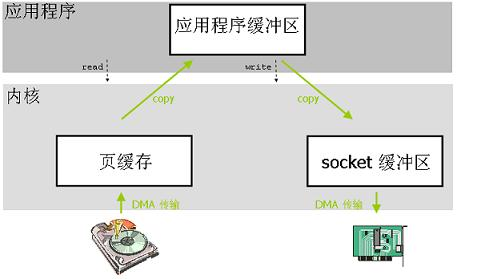


图2.2 BIO模型

Netty是一款基于NIO（Nonblocking I/O，非阻塞IO）开发的网络通信框架，对比于BIO（Blocking I/O，阻塞IO），它的并发性能得到了很大提高。

**2）传输快**

Netty的传输快主要依赖于NIO的一个特性——零拷贝。众所周知，JVM的内存有堆内存、栈内存和字符串常量池等。其中，作为Java对象存放的地方，堆内存是占用内存空间最大的一块。传统拷贝中，数据如果需要从IO读取到堆内存，中间需要经过Socket缓冲区，也就是说一个数据会被拷贝两次才能到达他的的终点，如果数据量大，就会造成不必要的资源浪费。

图2.3传统拷贝

针对这种情况，Netty使用了NIO中的另一大特性—零拷贝，当JVM需要接收数据的时候，虚拟机会在堆内存之外开辟一块内存，内存跟硬盘相映射，届时数据就直接从IO读到了堆外内存中去[5]。Netty可以通过ByteBuf直接对这些数据进行直接操作，从而加快了传输速度。

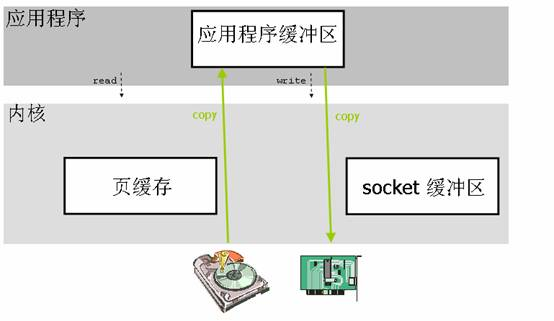


图2.4零拷贝

**3）封装好**

Netty以优美的方式对JavaNIO进行了一系列的封装，并重新抽象出了新的模型。这一切都使得代码的愈发简单明了，降低了开发人员的入手程度。此外，通过对Java NIO的封装，Netty还修复了Java NIO中的空轮询的问题。这收到了更多人的追捧。

## Fastjson

Fastjson是阿里巴巴开发的一款专门用于Java库，可以将 Java 对象转换为 JSON 格式，当然它也可以将 JSON 字符串转换为 Java 对象。Fastjson 可以操作任何 Java 对象，即使是一些预先存在的没有源码的对象[6]。

Fastjson有比较显著的优点。首先其提供服务器端、安卓客户端两种解析工具，性能表现较好。并且提供了 toJSONString() 和 parseObject() 方法来将 Java 对象与 JSON 相互转换。调用toJSONString方 法即可将对象转换成 JSON 字符串，parseObject 方法则反过来将 JSON 字符串转换成对象。而且Java泛型的广泛支持，允许对象的自定义表示、允许自定义序列化类。允许转换预先存在的无法修改的对象（只有class、无源代码）。此外，支持任意复杂对象（具有深厚的继承层次和广泛使用的泛型类型）。

## SqlBuilder

SqlBuilder是一个SQL查询语句生成类库。由healthmarketscience提供。

它试图减轻在Java程序中生成SQL查询的麻烦。使用一种编程语言（Java）生成另一种语言（即SQL）的代码始终是一个挑战。总是存在以下问题：在字符串文字中转义字符，在正确的位置获取空格以及使括号匹配。而且，即使在调试和完全测试代码之后，它仍然非常脆弱。最细微的变化都会使事情失去平衡，并需要另一轮测试和调整[7]。

SqlBuilder通过将SQL语法包装在非常轻巧且易于使用的Java对象中来改变整个情况，该Java对象遵循“ builder”范例（类似于StringBuilder）。这将许多常见的SQL语法，运行时错误更改为Java编译时错误。

## 设计模式

### SPI

SPI ，全称为 Service Provider Interface，是一种服务发现机制。Java  SPI是为某个接口寻找服务实现的机制。有点类似IOC的思想，就是将装配的控制权移到程序之外，在模块化设计中这个机制尤其重要。

Java  SPI 的约定，当服务的提供者，提供了服务接口的一种实现之后，在jar包的META-INF/services/目录里同时创建一个以服务接口命名的文件。该文件里就是实现该服务接口的具体实现类。而当外部程序装配这个模块的时候，就能通过该jar包META-INF/services/里的配置文件找到具体的实现类名，并装载实例化，完成模块的注入。基于这样一个约定就能很好的找到服务接口的实现类，而不需要再代码里制定。jdk提供服务实现查找的一个工具类：java.util.ServiceLoader。

### Ioc

控制反转（Inversion of Control，英文缩写为IoC）是一个重要的面向对象编程的法则来削减计算机程序的耦合问题。 控制反转还有一个名字叫做依赖注入（Dependency Injection）。简称DI。

1996年，Michael Mattson在一篇有关探讨面向对象框架的文章中，首先提出了IOC 这个概念。IOC理论是指借助于“第三方”实现具有依赖关系的对象之间的解耦；其中第三方即指IOC容器。

传统应用程序都是由我们在类内部主动创建依赖对象，从而导致类与类之间高耦合，难于测试；有了IoC容器后，把创建和查找依赖对象的控制权交给了容器，由容器进行注入组合对象，所以对象与对象之间是松散耦合，这样也方便测试，利于功能复用，更重要的是使得程序的整个体系结构变得非常灵活。

IoC对编程带来的最大改变不是从代码上，而是从思想上，发生了“主从换位”的变化。应用程序原本是主体地位，可以主动获取资源。但是在IoC/DI思想中，应用程序就变成被动的了，被动的等待IoC容器来创建并注入它所需要的资源。

IoC很好的体现了面向对象设计法则之一—— 好莱坞法则：“别找我们，我们找你”；即由IoC容器帮对象找相应的依赖对象并注入，而不是由对象主动去找[8]。

### 责任链模式

责任链模式在面向对象程式设计里是一种软件设计模式，它包含了一些命令对象和一系列的处理对象[9]。每一个处理对象决定它能处理哪些命令对象，它也知道如何将它不能处理的命令对象传递给该链中的下一个处理对象。该模式还描述了往该处理链的末尾添加新的处理对象的方法。

# 需求分析与设计

## 业务需求

在日常生活中，用户发起请求到获得结果，主要经由以下流程：用户从页面发起请求，向业务服务器申请服务；业务服务器进行业务逻辑处理，并调用数据库；数据库接受请求，并返回数据；业务服务器将从数据库返回的数据进行页面渲染；最终呈现出用户的效果[10]。

在前后端分离的大环境下，为了使数据展现过程更加直接，并且提供更好的用户体验，业务服务被拆分成了前端服务和后端服务。其具体流程为：首先从浏览器发送AJAX请求，然后服务端接受该请求并返回JSON数据返回给浏览器，最后在浏览器中进行界面渲染。工作模式如图所示：

所以本文将采用业界比较流行的改性后的MVC模式，其如下图所示：

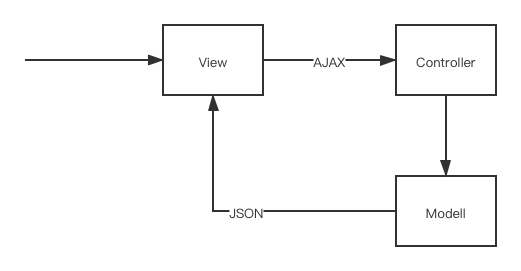


图3.1 MVC模式

## 功能需求

作为一个基于WEB框架，要具有以下功能：容器管理功能，请求响应功能和数据库操作功能。

### 容器管理功能

为了降低整体的耦合度，降低代码侵入性，将所有bean对象放入容器中进行管理。可以满足对bean对象的使用的需求。

其中主要包括容器管理和bean管理两个功能。

容器管理，是指管理包含IOC在内的bean容器。主要定义容器的功能，以及保留容器的可拓展性。

Bean管理，是指利用容器对bean的初始化，获取，释放整个生命周期进行管理的功能。

### 请求响应功能

请求响应功能主要包括以下功能：路由请求、json响应、传递参数以及拦截器。

路由请求是指系统api被调用时，能够根据urll中的路径地址，顺利的调用响应的方法，以满足调用者的需求。Json响应是指，将返回结果以json的数据格式进行返回。传递参数，是指获取前端通过API中传递的参数。拦截器是一个框架的基本功能，可以利用拦截器实现日志记录、事务提交等通用工作。

### sql功能

数据库操作主要包括：数据库连接，sql语句自动生成的功能。数据库连接是指能够根据配置，实现程序到数据库的成功连接。Sql语句生成是指通过方法名称，传入的对象以及部分条件，能够自动生成sql语句，使框架应用起来更加方便简洁。

## 架构设计

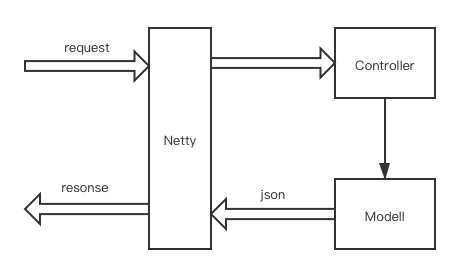


图3.2 架构设计

## 功能模块设计

在满足业务及功能需求的基础上，本文将框架划分为容器模块，请求响应模块，sql模块。其中，容器模块主要实现对容器的可插拔性设计，并且以IOC容器为默认容器。请求响应模块主要包括路由、json响应、传递参数以及拦截器。sql模块主要有数据库连接，自动生成sql语句。

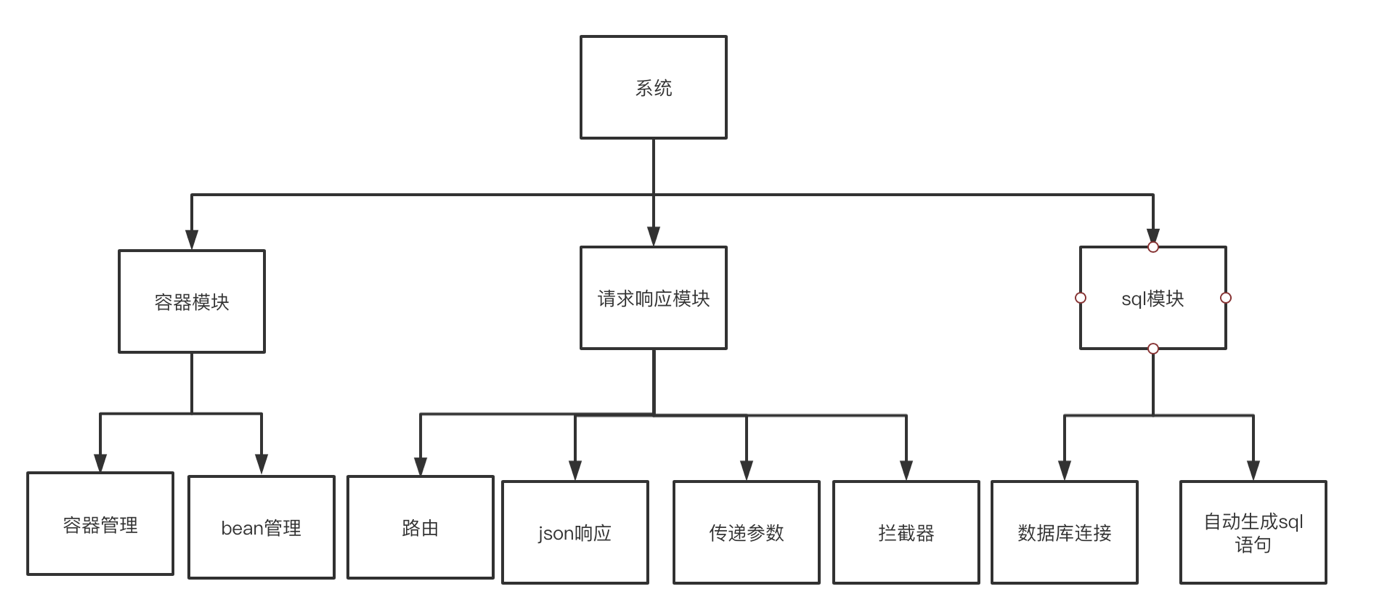


图3.3 整体功能模块

### 容器管理模块

为了降低整体的耦合度，降低代码侵入性，将所有bean对象放入容器中进行管理。可以满足对bean对象的使用的需求。

其中主要包括容器管理和bean管理两个功能。

容器管理，主要定义容器的功能，以及保留容器的可拓展性。要实现该功能，主要通过定义BeanFactory表示一个通用的接口，包含了注册容器、从容器中获取实例等方法。并具体实现了JupiterIOC容器。其次需要利用SPI来实现可插拔性，以实现由它来管理具体使用哪种 IOC 容器。可插拔性主要ClassScanner类的getBeanFactory方法在里面定义实现。

Bean管理，是指利用容器对bean的初始化，获取，释放整个生命周期进行管理的功能。主要在BeanManager里面实现该功能。

### 请求响应模块

1. **路由请求**

路由请求是指系统api被调用时，能够根据uil中的路径地址，顺利的调用响应的方法，以满足调用者的需求。

为了遵循一切从简的原则， 本文将采用基于注解的方式进行路由请求。首先定义注解，@Action为控制器，类似于Spring里面的@Controller。@Route负责具体的路由功能。

在介绍完注解后，将描述一下路由请求模块的流程。首先，扫描所有使用 @Action 注解的类，以获取控制器。其次，扫描所有使用 @Route 注解的方法，获取路由方法。然后，将路由方法路径和方法映射关系存入 Map 中，作为映射表。再尔，请求时根据 URL 去 Map 中查找这个关系。最后，根据查找到的具体类和方法，利用反射构建参数及方法调用。

为了完成以上流程，将分别定义类RouteScanner和RouteProcess完成上述功能。其中RouteScanner中的loadRouteMethods方法负责扫描所有使用注解@Route和@Action的类和方法，并将路由方法路径和方法映射到map中；routeMethod则负责实现格局url找到对应方法。RouteProcess主要实现根据查找到的具体类和方法利用反射构建参数及方法调用。

1. **Json响应**

Json响应是指，将返回结果以json的数据格式进行返回。在实现过程中，首先定义了WorkRes类，用来承接从数据库返回的数据。进而通过第三方开源工具fastjson将WorkRes中承载的数据转换为json格式。然后再将json放入到JupiterHttpResponse，进而完成json响应。

1. **传递参数**

传递参数，是指获取通过API中传递的参数。传递参数的方式可能有很多种，为了应对复杂的局面，将采用Map 方式的方式用来接收参数。因此定义了一个继承了 Map 接口通用的 Param 接口。在其中封装了几种基本类型的获取方式。在Dispatcher类中的buildParamMap方法中，实现对参数的封装接收。通过RouteProcess类中的parseRouteParameter方法，将参数跟方法绑定。

1. **拦截器**

拦截器是一个框架的基本功能，可以利用拦截器实现日志记录、事务提交等通用工作。因为该需求非常的符合对某一个对象、请求进行一系列处理的特征，所以，将采用责任链的模式实现拦截器。

首先是定义了 AbstractInterceptor的 抽象类最为责任链中的链，其内部定义了拦截器执行的方法，before和after方法，用来表达在拦截发生以前和拦截发生之后，拦截器需要执行的行为。同时定义了一个 InterceptProcess 的客户端，客户端提供一个执行入口以及添加责任链的入口即可。

### sql模块

数据库操作主要包括：数据库连接，sql语句自动生成的功能。

数据库连接是指能够根据配置，实现程序到数据库的成功连接。Sql语句生成是指通过方法名称，自动生成sql语句，更加简单便捷。

1. **数据库连接**

数据库连接因为不属于本文的主要研究设计，所以选择采用java ee的JDBC作为数据库连接的中间件。 但是并非照搬接收，将采用一些简单的封装使数据库连接更加优雅。

首先，为了之后引入更多数据库连接方式，将采用工厂模式获得连接。将JDBC作为默认连接，封装在DefaultConnect，其内部封装数据库驱动加载，以及数据库连接的功能。

然后，将定义一个SqlSession的对象，用来封装每次与数据库session的如用户名，密码，当前数据库等内容。为了之后预留预留其它类型的封装，将采用工厂模式，管理sqlsession。

1. **sql语句生成**

sql语句生成，将依赖于第三方开源工具sqlbuilder。利用第三方开源工具，将sql语句分为两类，一类是读操作，另一类是写操作。

读操作核心有两点，确定表名称，添加查询条件。写操作核心有两点，确定表明，增加操作的属性，传入输入值。

为了方便的实现以上功能点，将引入注解@TableName，@FieldName，@PrimaryId三个注解。在传入对象时，可以通过获取类的@TableName注解的值，获得与该类绑定的表名。通过@ FieldName可以获取与与属性绑定的数据库列名。通过@PrimaryId可以判断该属性是否为主键。对于注解的操作，统一封装在了ReflectTools中。

至此，基于以上条件，在DBHandler中可以完整实现自动生成sql语句的功能。

对于读操作，还需要查询条件才能够满足其正常功能。本文通过定义Condition类来满足其需求。

基于以上条件，在DQuery类中的buildSQL方法实现了自动自动生成sql语句。

至此，得到完整的sql语句后，依赖于java ee的JDBC实现对数据库的读写操作。

# 系统实现

## 框架开发环境

本框架面向Java生态，在开发过程中，采用的是业界主流版本JDK 8。其中第三方开源框架版本分别为：fastjson-1.2.25，netty-4.1.42-Final，slf4j-1.7.25，sqlbuilder-3.0.0。

## 容器模块实现

将按照容器管理模块的功能需求，以及设计，按照可拓展性的原则进行设计。

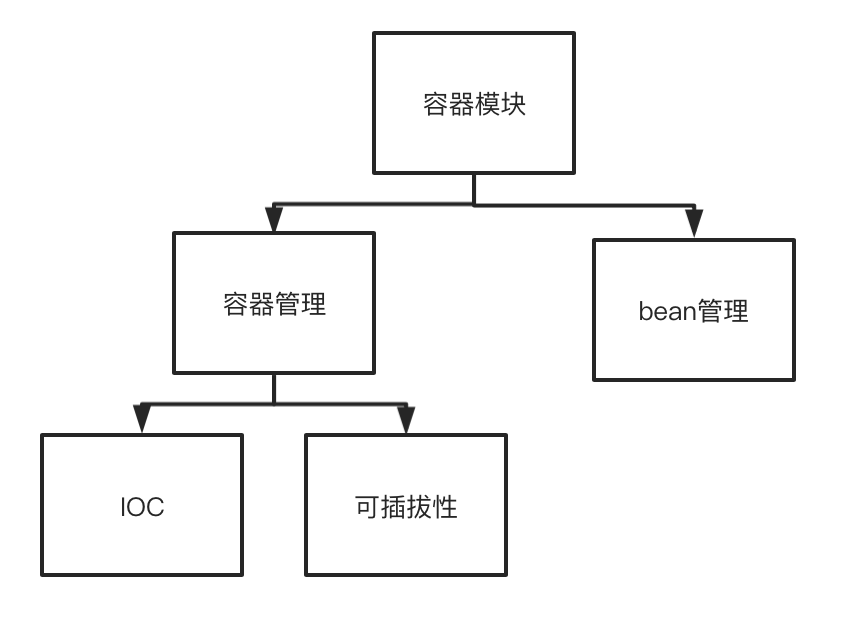


图4.1 容器模块功能图

### 容器管理

IOC功能，主要由BeanFactory和JupiterIOC三个类来实现其主要功能。其中，BeanFactory为接口，定义容器的功能。JupiterIOC为具IOC的一种以map为存储方式的具体实现。以下为具体实现方式。

**BeanFatory**用于管理bean对象的生命周期，包括bean的注册，获取，释放三个生命阶段。

public interface BeanFactory {  
 void register(Object var1);  
 Object getBean(String var1) throws Exception;  
 <T> T getBean(Class<T> var1) throws Exception;  
 void releaseBean();  
}

可插拔主要由ClassScanner的getBeanFactory方法里面通过SPI机制来实现其主要功能。主要通过调用java中原生的ServiceLoader扫描META-INF包下面的配置文件指定具体实现类，从而来实现SPI机制。通过META-INF.services下定义的配置文件，来实现具体的可插拔性。

public static BeanFactory getBeanFactory() {  
 ServiceLoader<BeanFactory> BeanFactories = ServiceLoader.load(BeanFactory.class);  
 return (BeanFactory)(BeanFactories.iterator().hasNext() ? (BeanFactory)BeanFactories.iterator().next() : new DefaultBean());  
}

容器管理的功能测试如下。

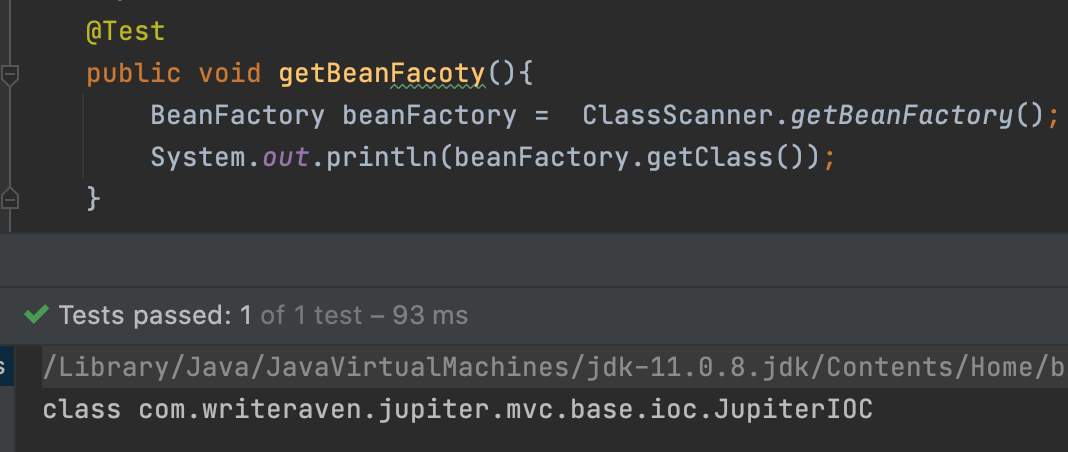


图4.2 容器模块功能测试效果图

### Bean管理

主要由BeanManager里面来实现其主要功能。BeanManager通过SPI机制获取IOC容器，然后通过包扫描的方式，将bean注册进入容器中，然后通过容器获取释放bean。以下为具体实现方式。

public final class BeanManager {  
 private static volatile BeanManager beanManager;  
 private static BeanFactory beanFactory;  
  
 public void initBeans(String packageName) throws Exception {  
 Map<String, Class<?>> beans = ClassScanner.getBeans(packageName);  
 beanFactory = ClassScanner.getBeanFactory();  
 Iterator var3 = beans.entrySet().iterator();  
 //遍历扫描到的类，注册到容器中

} }  
//省略getter and setter

}

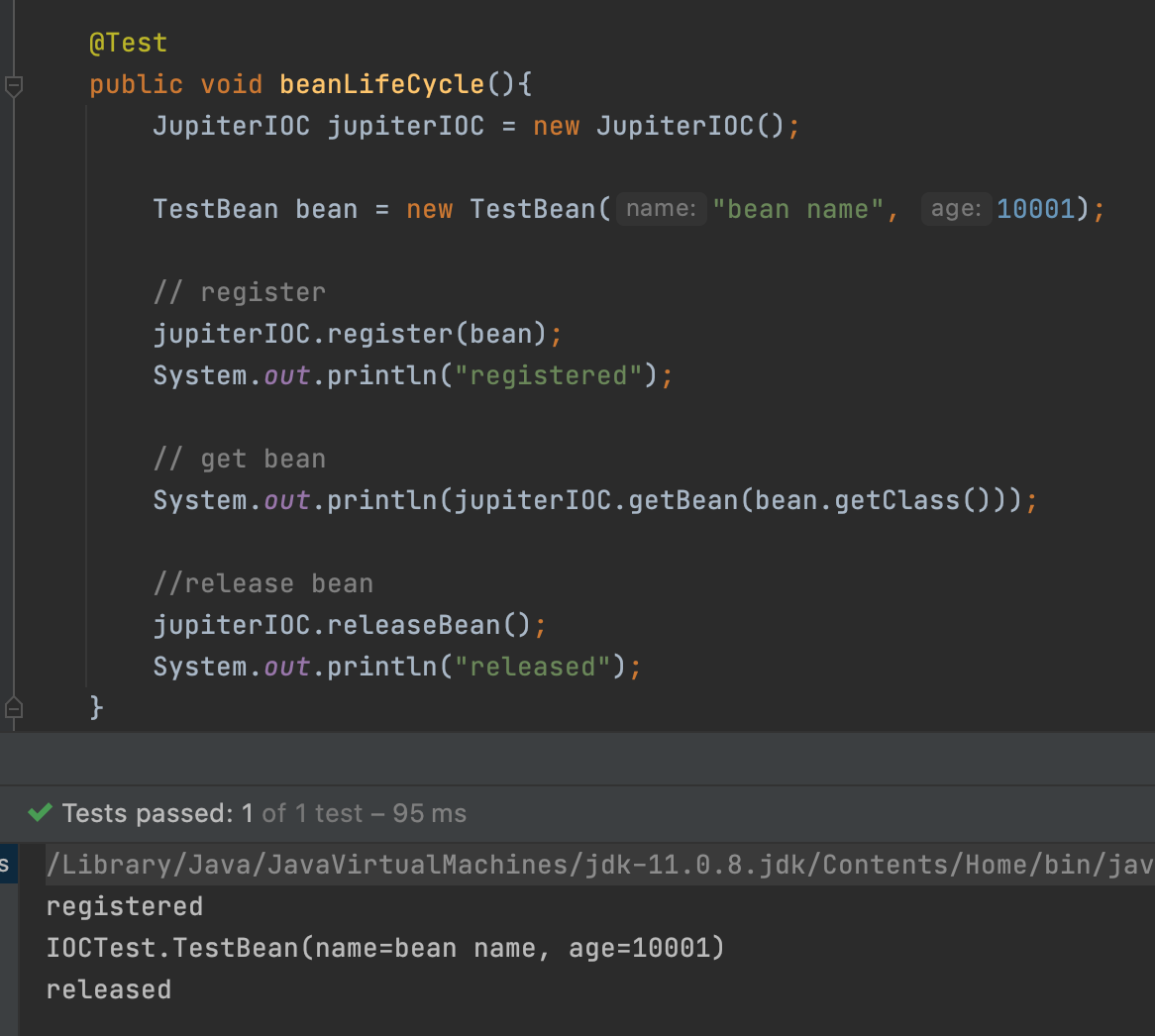
Bean管理模块的功能测试如下。

图4.3 Bean模块功能测试效果图

## 请求响应模块实现

### 路由请求

该功能主要由RouteScanner和RouteProcess两个类来实现其主要功能。

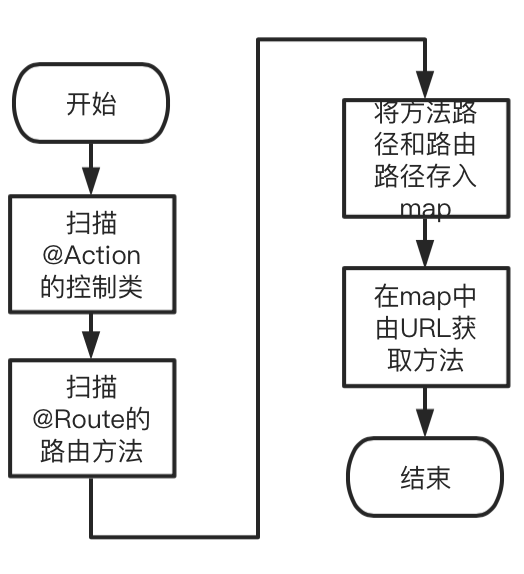


图4.4 路由请求流程图

通过RouteScanner.routeMethod方法实现扫描@Action的控制类和@Route的路由方法。在routeMethod中调用的loadRouteMethods方法主要实现了通过扫描包，获取到路由方法的功能。

public Method routeMethod(QueryStringDecoder queryStringDecoder) throws Exception {  
 if (routes == null) {  
 routes = new HashMap(16);  
 this.loadRouteMethods(this.appConfig.getRootPackageName());  
 }  
 boolean defaultResponse = this.defaultResponse(queryStringDecoder.path());  
 if (defaultResponse) {  
 return null;  
 } else {  
 Method method = (Method)routes.get(queryStringDecoder.path());  
 if (method == null) {  
 throw new JupiterException(StatusEnum.NOT\_FOUND);  
 } else {  
 return method;  
 }}}

获取路由方法测试如下。

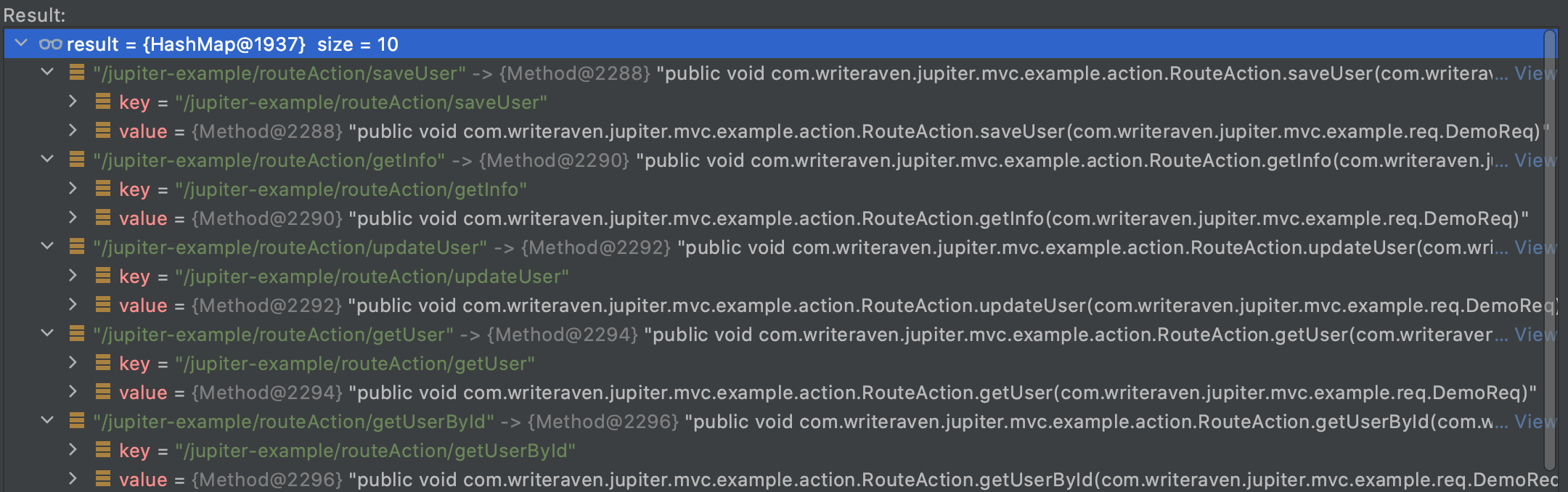


图4.5 获取路由方法测试效果图

### Json响应



图4.6 Json响应流程图

主要由WorkRes，JupiterHttpResponse和JupiterContext三个类来实现其主要功能。

其中，WorkRes定义了从数据库获取到的数据格式。JupiterHttpResponse定义了HttpReponse返回的格式。JupiterContext完成了将WorkRes转换为json的功能。

public final class JupiterContext {  
 private JupiterRequest request;  
 private JupiterResponse response;  
  
 public void json(WorkRes workRes) {  
 getResponse().setContentType("application/json; charset=UTF-8");  
 getResponse().setHttpContent(JSON.toJSONString(workRes));  
 }

//getter and setter  
}

json方法测试如下。

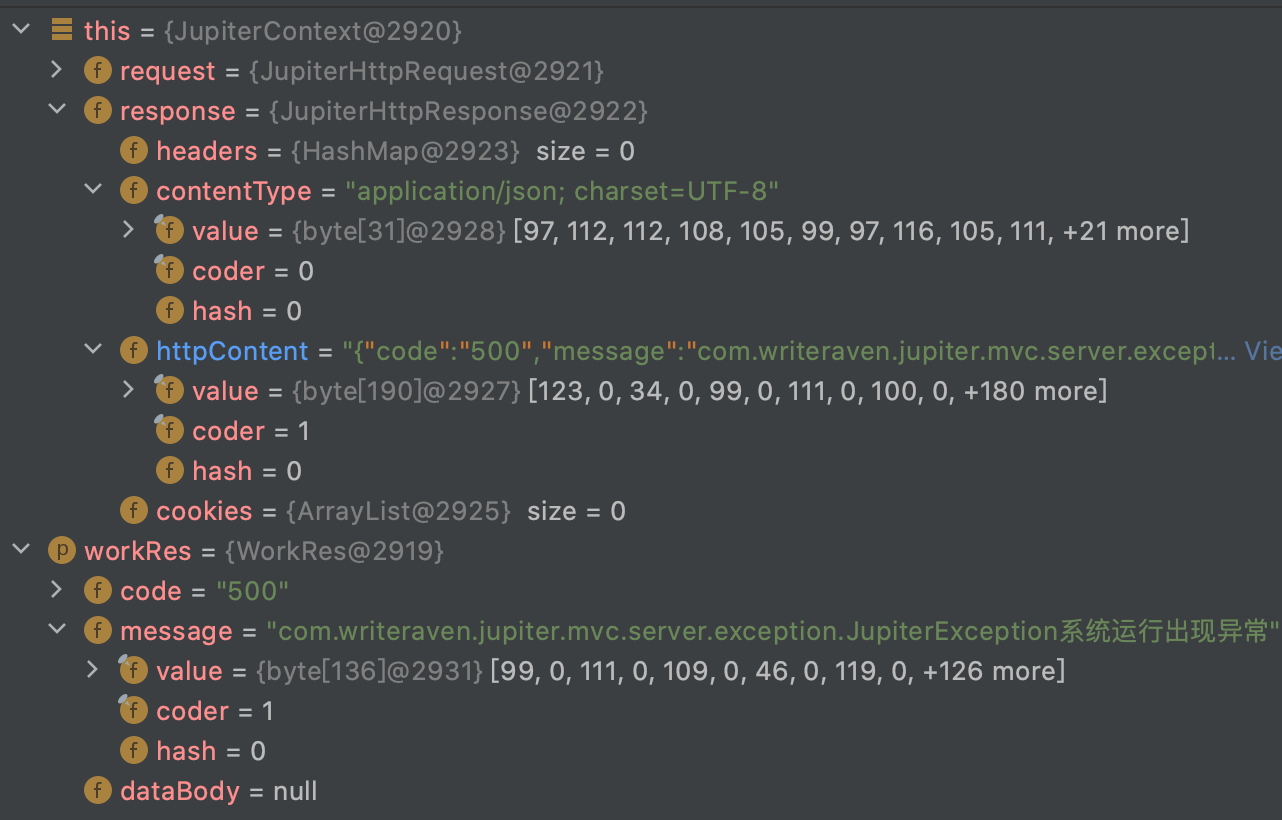


图4.7 json方法测试效果图

### 传递参数

主要由Param，Dispatcher.buildParamMap方法和RouteProcess类中的parseRouteParameter两个类来实现其主要功能。

其中，Param规范化了参数的类型。Dispatcher.buildParamMap方法通过获取Netty中传来的request，调用RouteProcess.parseRouteParameter方法解析构建param。通过RouteProcess解析Netty中queryStringDecoder传来的URL获取需要调用的路由方法方法。

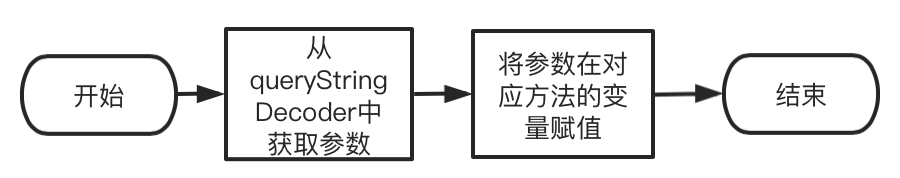


图4.8 传递参数流程图

以下为具体实现方式。

private Object[] parseRouteParameter(Method method, QueryStringDecoder queryStringDecoder) {  
 Object[] instances = new Object[parameterTypes.length];;  
 Class<?>[] parameterTypes = method.getParameterTypes();  
 for (int i = 0; i < instances.length; i++) {  
 //inject Jupiter context instance  
 if (parameterTypes[i] == JupiterContext.class) {  
 instances[i] = JupiterContext.getContext();  
 } else {  
 //inject custom pojo  
 Class<?> parameterType = parameterTypes[i];  
 Object instance = parameterType.getConstructor().newInstance();  
  
 Map<String, List<String>> parameters = queryStringDecoder.parameters();  
 // 省略遍历参数，并且将参数赋值给 参数容器对象  
 instances[i] = instance;  
 }  
 }  
 return instances;

}

解析参数方法测试如下。

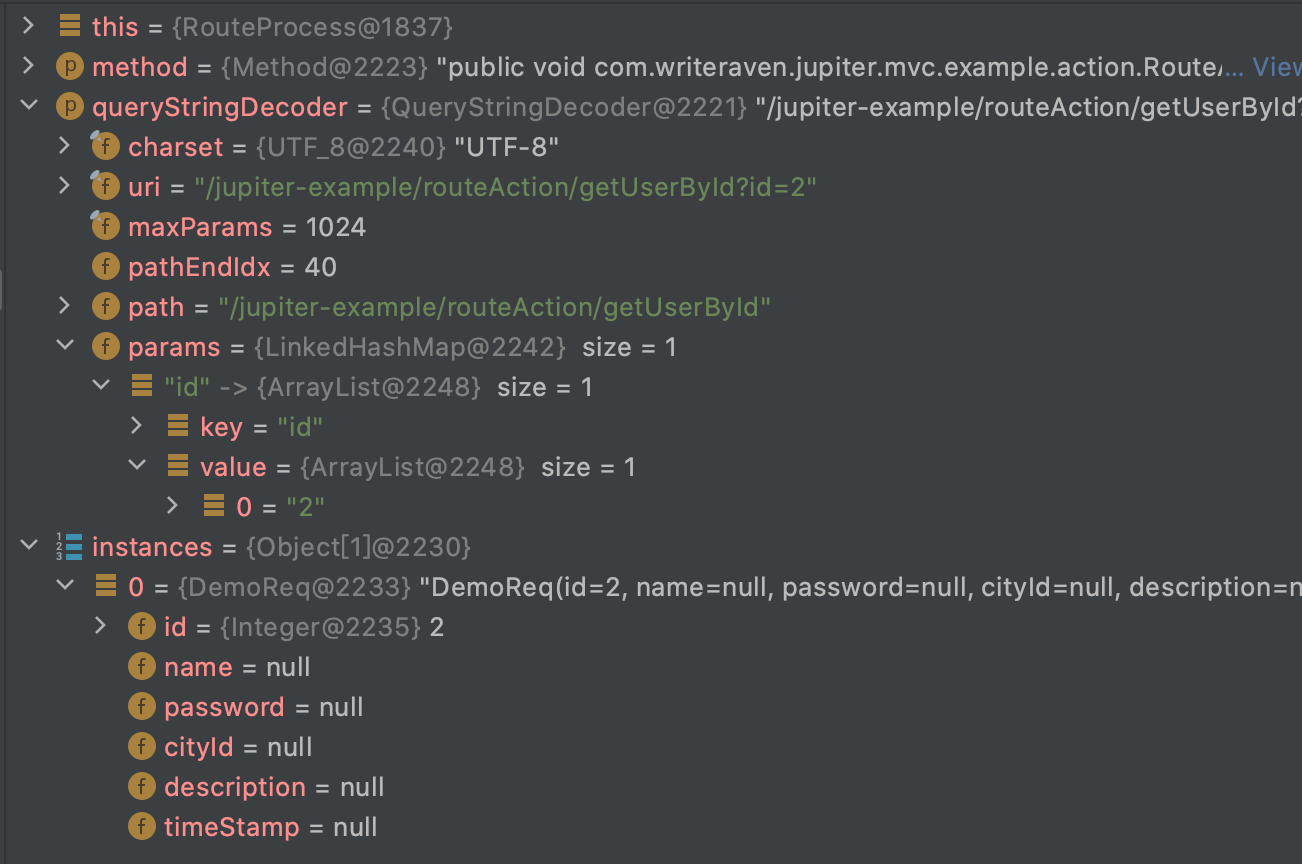


图4.9 解析参数方法测试效果图

### 拦截器

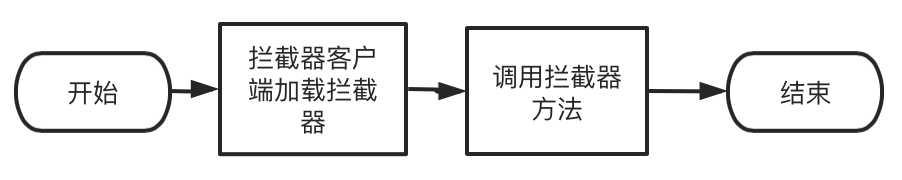


图4.10 拦截器流程图

主要由AbstractInterceptor和InterceptProcess两个类来实现其主要功能。

其中，AbstractInterceptor定义了拦截器的功能。InterceptProcess加载拦截器，并实现了其中的拦截前后的功能，以下为具体实现方式。

public class InterceptProcess {  
 // 单例模式

// 加载拦截器  
 public void loadInterceptors() throws Exception {  
 if (interceptors == null) {  
 interceptors = new ArrayList(10);  
 Map<Class<?>, Integer> JupiterInterceptor = ClassScanner.getJupiterInterceptor(this.appConfig.getRootPackageName());  
 Iterator var2 = JupiterInterceptor.entrySet().iterator();  
 while(var2.hasNext()) {  
 Entry<Class<?>, Integer> classEntry = (Entry)var2.next();  
 Class<?> interceptorClass = (Class)classEntry.getKey();  
 AbstractInterceptor interceptor = (AbstractInterceptor)interceptorClass.getConstructor().newInstance();  
 interceptor.setOrder((Integer)classEntry.getValue());  
 interceptors.add(interceptor);  
 }  
 Collections.sort(interceptors, new Comparator() {  
 public int compare(Object o1, Object o2) {  
 return ((AbstractInterceptor)o1).getOrder() <= ((AbstractInterceptor)o2).getOrder() ? 1 : 0;  
 }  
 });  
 }  
 }  
 // 实现AbstractInterceptor定义的processBefore和processAfter功能。此处省略。

}

加载拦截器方法测试如下。

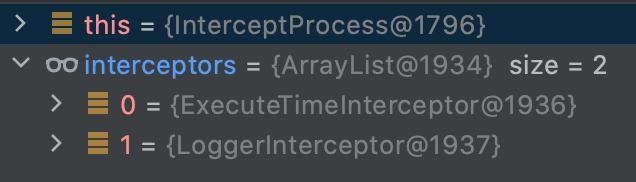


图4.11 加载拦截器测试效果图

## Sql模块实现

### 数据库连接

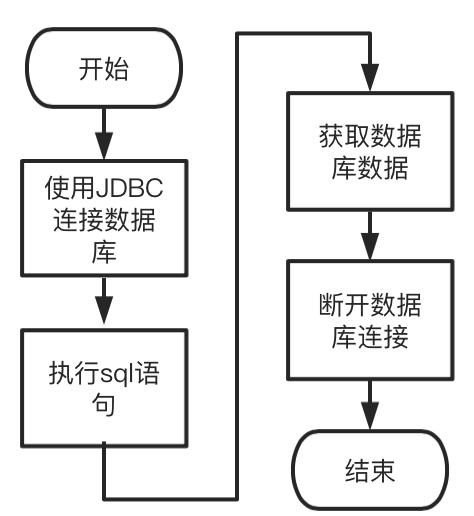


图4.12 数据库连接流程图

主要由SqlSession，SqlSessionFactory，ConnectFactory，ConnectFactory来实现其主要功能。

其中，SqlSession定义了数据库相关的信息，包括数据库名，密码，连接。并且还定义了即将操作的相关表。为了保证拓展性，采用工厂模式，引入了SqlSessionFactory类和ConnectFactory。ConnectFactory定义了一次连接的动作，ConnectFactory为默认使用JDBC的实现方式。默认连接具体实现方式。

public class DefaultConnection implements ConnectionFactory {  
  
 public Connection getConnection(SqlSession sqlSession) {  
 if (this.connection == null) {  
 try {  
 Class.forName("com.mysql.jdbc.Driver");  
 this.connection = DriverManager.getConnection(sqlSession.getUrl(), sqlSession.getUserName(), sqlSession.getPwd());  
 } catch (Exception var3) {  
 log.error("Exception", var3);  
 }  
 }  
 return this.connection;  
 }  
}

默认连接方法测试如下。

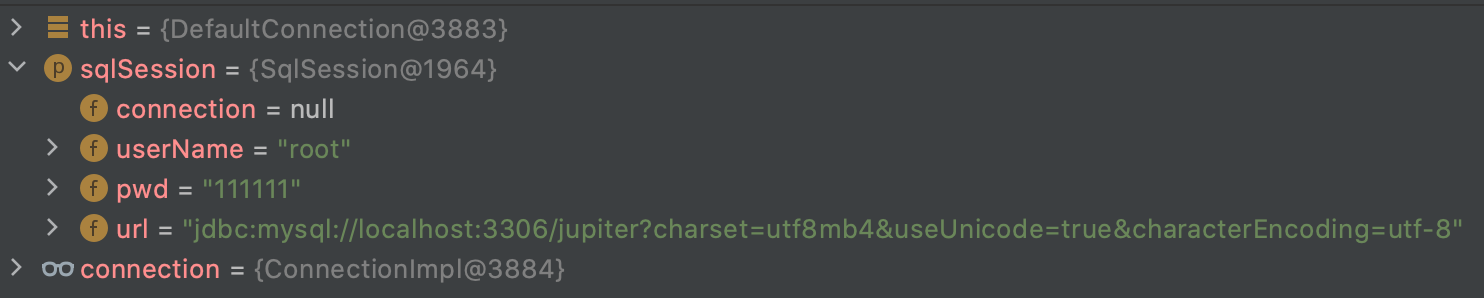


图4.13 默认连接方法测试效果图

### Sql语句生成

因为对数据库的操作主要分为读操作和写操作，所以分别讨论。

**1）读语句**

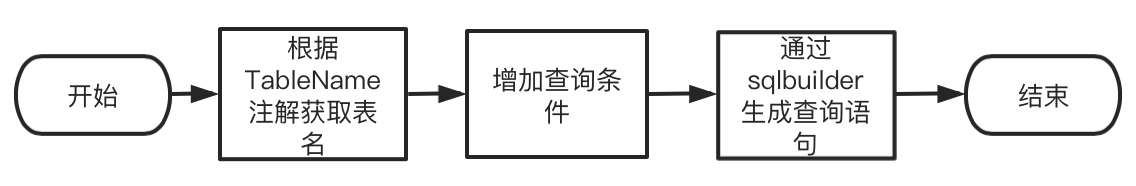


图4.14 自动生成读语句流程图

主要由DBQuery来实现以上流程。首先完成筛选条件；在实现过程中，采用获取全部加条件的方式，来定义方法。然后，调用DBQuery类中的buildSQL方法生成sql语句。具体实现方式如下。

private String buildSQL() {  
 SelectQuery selectQuery = new SelectQuery();  
 Field[] var2 = this.targetClass.getDeclaredFields();  
 int var3 = var2.length;  
 for(int var4 = 0; var4 < var3; ++var4) {  
 Field field = var2[var4];  
 String dbField = ReflectTools.getDbField(field);  
 DbColumn dbColumn = this.dbTable.addColumn(dbField);  
 selectQuery.addColumns(new Column[]{dbColumn});  
 this.columnMap.put(dbField, dbColumn);  
 }  
 Iterator var8 = this.conditions.iterator();  
 while(var8.hasNext()) {  
 Condition condition = (Condition)var8.next();  
 Filter filter = condition.getCondition();  
 DbColumn dbColumn = (DbColumn)this.columnMap.get(filter.getFiled());  
 selectQuery.addCondition(BinaryCondition.equalTo(dbColumn, filter.getValue()));  
 }  
 return ((SelectQuery)selectQuery.validate()).toString();  
 }  
}

生成查询sql方法测试如下。

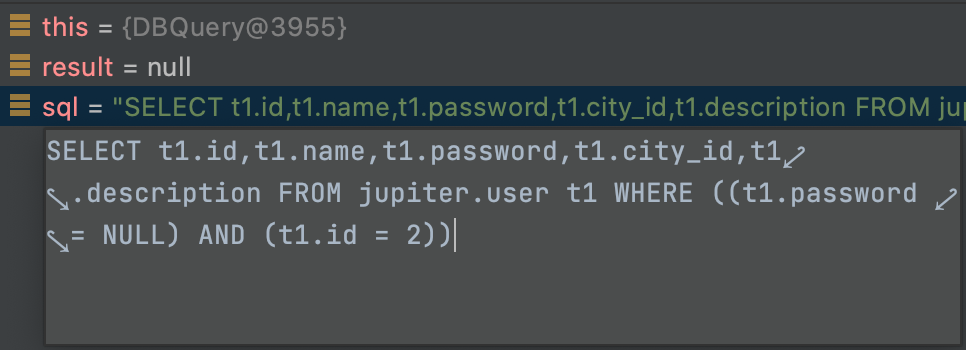


图4.15 生成查询sql方法测试效果图

**2）写语句**

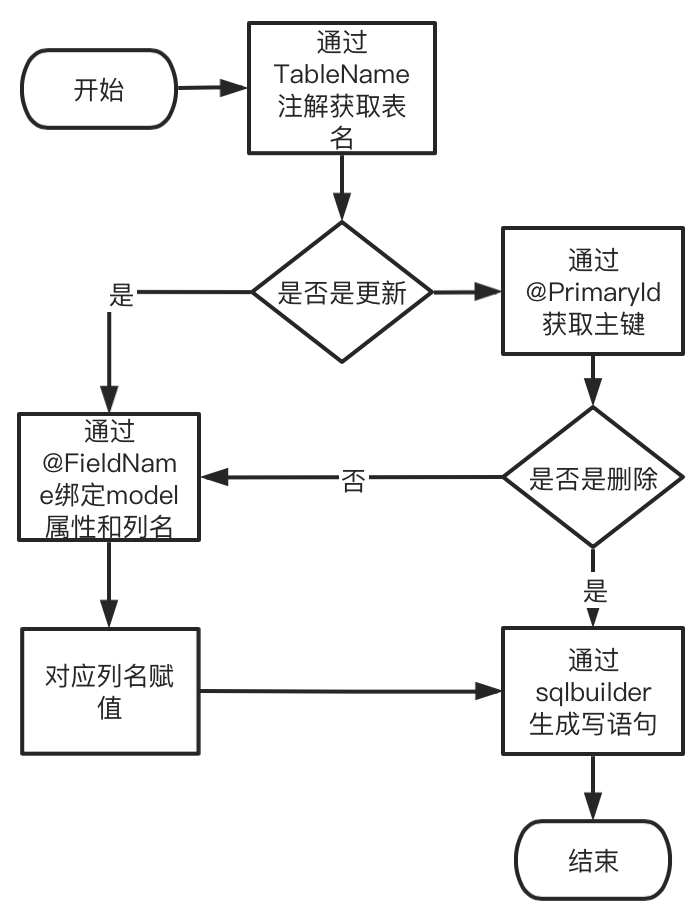


图4.16 自动生成写语句流程图

该功能主要由AbstractDBHandler，DBHandler，Conditions完成。其中，AbstractDBHandler定义了写操作的具体行为。DBHandler实现了AbstractDBHandler接口里定义的行为。将以更新操作为例，以下为具体实现方式如下。

private DbTable dbTable;  
 // 更新操作  
 @Override  
 public int update(Object obj) {  
 // 省略检验obj，省略获取dbTable  
 Map<DbColumn, Integer> primaryCondition = new HashMap<>(1);  
 UpdateQuery updateQuery = new UpdateQuery(dbTable);  
   
 for (Field field : obj.getClass().getDeclaredFields()) {  
 String dbField = ReflectTools.getDbField(field);  
 Object filedValue = Instance.getFiledValue(obj, field);  
 if (field.getAnnotation(PrimaryId.class) != null) {  
 primaryCondition.put(dbTable.addColumn(dbField), (Integer) filedValue);  
 } else {  
 DbColumn dbColumn = dbTable.addColumn(dbField);  
 updateQuery.addSetClause(dbColumn, filedValue);  
 }  
 }  
 // updateQuery中增加主键  
 for (Map.Entry<DbColumn, Integer> entry : primaryCondition.entrySet()) {  
 updateQuery.addCondition(BinaryCondition.equalTo(entry.getKey(), entry.getValue()));  
 }  
 //省略JDBC执行操作

return 0;  
 }

生成更新sql方法测试如下。

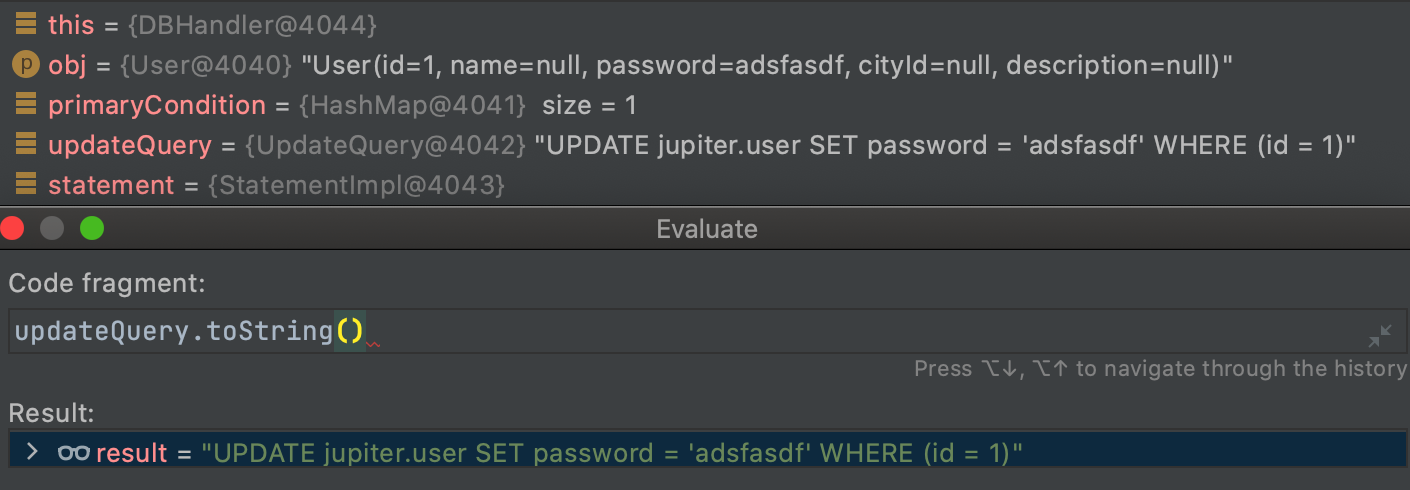


图4.17 生成更新sql方法测试效果图

# 框架功能测试

在该部分，将给予本框架创建一个实现基本增删改查的测试案例，用以检测框架的功能。因为本框架是一个面向后端的框架，所以将使用postman对其接口进行测试。

## 测试案例构建

本框架主要作用于后端，既要接收前端的api请求，又要将从数据库返回的数据以json的形式返回。

在本案例中，将构建一个简单的用户管理系统，该系统将完成基本的增删改查功能。

### 数据库设计

为了降低案例的复杂度，避免喧宾夺主，将采用一张user表支撑该用户管理系统的功能。该表表结构如下表所示。

表5.1 User表表结构

|  |  |  |
| --- | --- | --- |
| 字段 | 类型 | 标注 |
| id | Int | 主键，自增 |
| name | varchar(128) | 用户名,不能为空 |
| password | varchar(24) | 用户密码 |
| city\_id | int | 城市编号 |
| description | varchar(256) | 描述 |

默认输入了一下测试数据。在测试过程中，写操作完成与否，可以在数据库中验证。创建及插入sql语句如下：

create table user(id int primary key auto\_increment,  
 name varchar(128) not null,  
 password varchar(24),  
 city\_id int,  
 description varchar(256)  
);

**SQL:**  
INSERT INTO `user` VALUES (1, 'zhangsan', '123456', 1, 'hello, i\'m zhangsan');  
INSERT INTO `user` VALUES (2, 'lisi', '123456', 2, 'hi, i\'m lisi');  
INSERT INTO `user` VALUES (3, 'wangermazi', '123456', 2, 'kekeke, i\'m wangermazi');  
INSERT INTO `user` VALUES (4, 'elon musk', '123456', 100, 'i\'m elon, the dogelon\'s elon. not musk');  
INSERT INTO `user` VALUES (7, 'hello', '123456', NULL, 'world');  
INSERT INTO `user` VALUES (100, 'name', '123456', NULL, 'name && pawd are successfully.');

### 接口设计

表5.2 测试接口表

|  |  |  |
| --- | --- | --- |
| 接口 | 参数 | 备注 |
| /jupiter-example/routeAction/getUser | name string  password string | 通过name和password获取用户 |
| /jupiter-example/routeAction/getUserById | id int | 通过id获取用户 |
| /jupiter-example/routeAction/saveUser | name string  password string  cityId int  description string | 创建用户 |
| /jupiter-example/routeAction/updateUser | id int  name string  password string  cityId int  description string | 更新用户 |

## 功能测试

通过postman，对用户管理系统提供的api进行测试。

### 创建用户

通过postman访问http://127.0.0.1:5688/jupiter-example/routeAction/saveUser?name=蔡八斗，数据库成功插入该数据，测试用例通过。

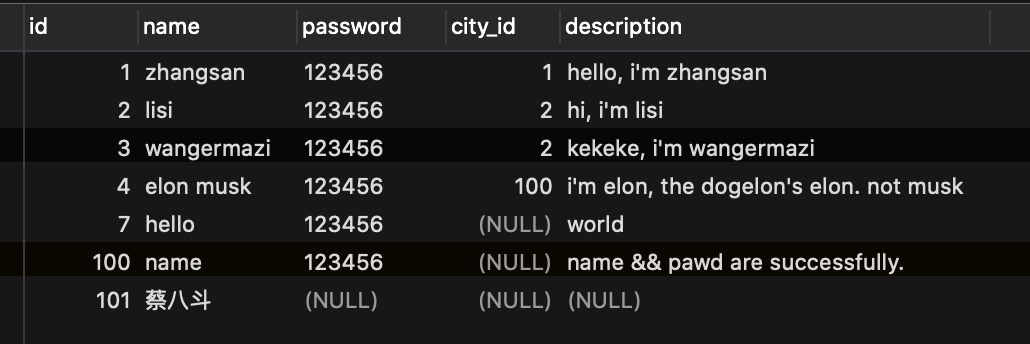


图5.1创建用户执行后User表内数据

### 更新用户

通过postman访问<http://127.0.0.1:5688/jupiter-example/routeAction/updateUser?id=1&password=adsfasdf>，数据库成功更新该条数据，测试用例通过。

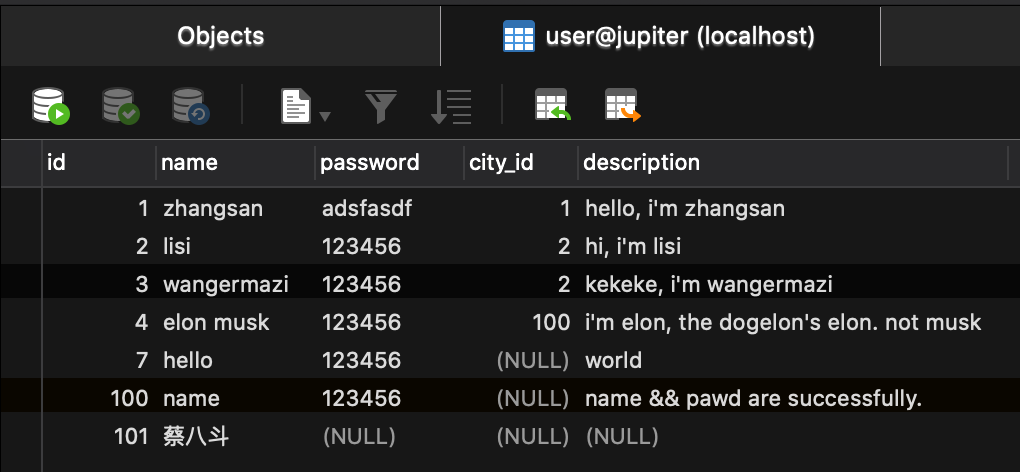


图5.2更新用户执行后User表内数据

### 查找用户

通过postman访问<http://127.0.0.1:5688/jupiterexample/routeAction/getUser?name=lisi&password=123456>，得到返回结果如下，获取到该用户的全部信息，测试用例通过。

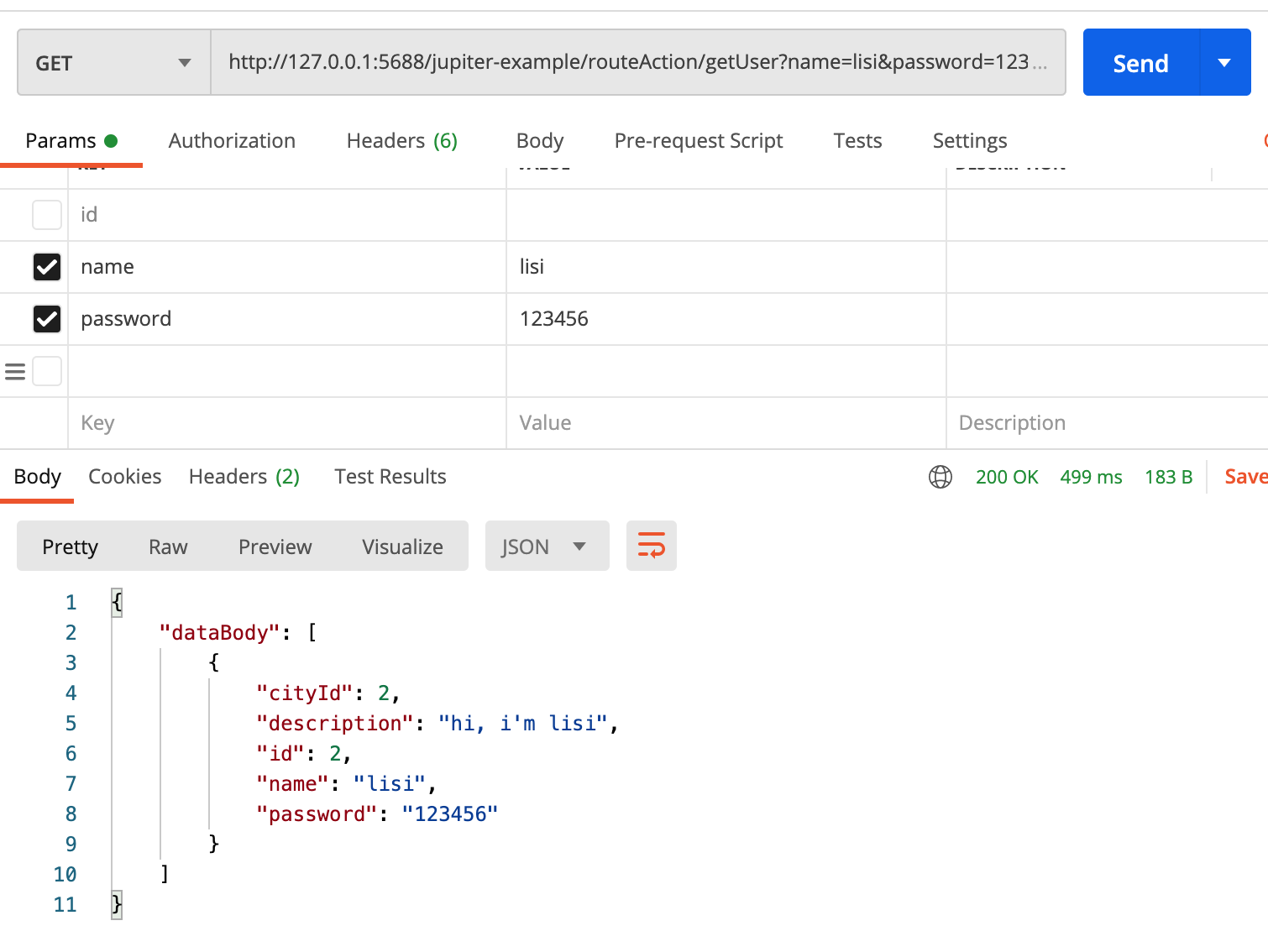


图5.3执行后返回的json数据

# 总结与展望

本文在前面几个章节，详细的阐述了对基于Netty的MVC WEB框架的研究。本章将对本次论文的工作内容进行总结。此外，针对研究中的考虑不周，还将作进一步的展望。

## 总结

Netty作为当今Java生态中网络编程框架的王者，被越来越多的向NIO转型的系统接受。MVC WEB框架作为笔者日常生活中接触最多的框架模式，发现很多MVC WEB框架依旧使用BIO模式。特别是针对于中小系统的框架。于是乎，笔者将基于Netty的MVC WEB框架作为毕业课题。本文主要内容如下：

1）介绍了MVC WEB框架的研究背景及意义，同时，介绍了行业研究现状。

2）介绍了实现期望MVC WEB框架需要依赖的第三方开源技术，以及会用到的设计理念和设计模式。

3）从业务和功能角度进行了需求分析，并从功能角度完成模块设计，并对各个模块的功能进行了阐述

4）根据功能模块设计，利用第三方开源技术对MVC WEB框架进行了实现。

5）基于开发的框架，构建了一个简易的系统，通过该系统，对框架的核心功能进行测试。

## 展望

目前市场上MVC WEB框架功能十分强大，特别是BIO的框架。本人鉴于时间精力有限，只能够基于Netty完成MVC WEB框架的核心功能。通过进一步对比分析，发现还有诸多不足：

1）虽然引入了Netty 为基础，实现了框架的NIO，因为没有添加锁机制，所以不能够保证并发安全。

2）因为数据库连接依旧依赖于JDBC，所以没有避免频繁创建数据库连接导致资源浪费的缺点。

3）因本人时间精力有限，框架抽象并不太完美，代码冗余相对不少。

# 结束语（致谢）

作为大学的毕业论文，很高心选择并完成了基于Netty的MVC WEB框架的设计与实现这一课题。宝剑锋从磨砺出，虽然其中路途比较坎坷，但不仅完成了一个毕业论文，更加深了笔者对于领域知识更深入的理解。同时，也增加了笔者对于很多前辈的钦佩。

在大学期间，从一个人只身远赴浙江读书，感受南北文化差异，城乡差异；再到独自一人赴罗马尼亚作交换生，沉浸语言障碍和文化差异。再到后来的找工作实习。期间，彷徨迷茫总伴我身边，内心的贪心算法与全局最优不断斗争。所幸遇到了风趣的老师，性格迥异的同学，以及跟我臭味相投的小朋友，在整个大学期间，轮番给我以启发和帮助。

而在论文期间，离不开老师同学朋友的支持陪伴。感谢他们的督促监督。

最后，感谢一直在我背后默默付出，鼎力支持我的父母。希望我爱的人们，幸福平安。

# 参考文献

[1] 李林锋.Netty权威指南(第二版)[M].北京:电子工业出版社,2015:24-25

[2] 夏斐.基于Netty的消息中间件的研究与实现[D].电子科技大学.2018

[4] 孔海峰.分布式RPC框架的设计与实现[D].华中科技大学. 2018

[5] 金志国,李炜.基于Netty的HTTP客户端的设计与实现[J]. 电信工程技术与标准 化. 2014

[6] 阿喜达.基于JVM的Netty网络通信性能测评[J].信息通信. 2015 (11)

[7] 朱启明.基于IOC容器的工作流引擎的研究与设计[D].国防科学技术大学. 2007

[8] 徐超国.基于领域建模和IoC/AoP的轻量级框架的设计与实现[D].华中科技大 学. 2007

[9] Description of sqlBuilder.<https://openhms.sourceforge.io/sqlbuilder/>

[10] The Spring Team.spring 5 docs. <https://spring.io/projects/spring-framework>

[11] Dr. Tejinder Singh .JAVA WEB DESIGN FRAMEWORKS- REVIEW OF JAVA FRAMEWORKS FOR WEB APPLICATIONS.