课程目标

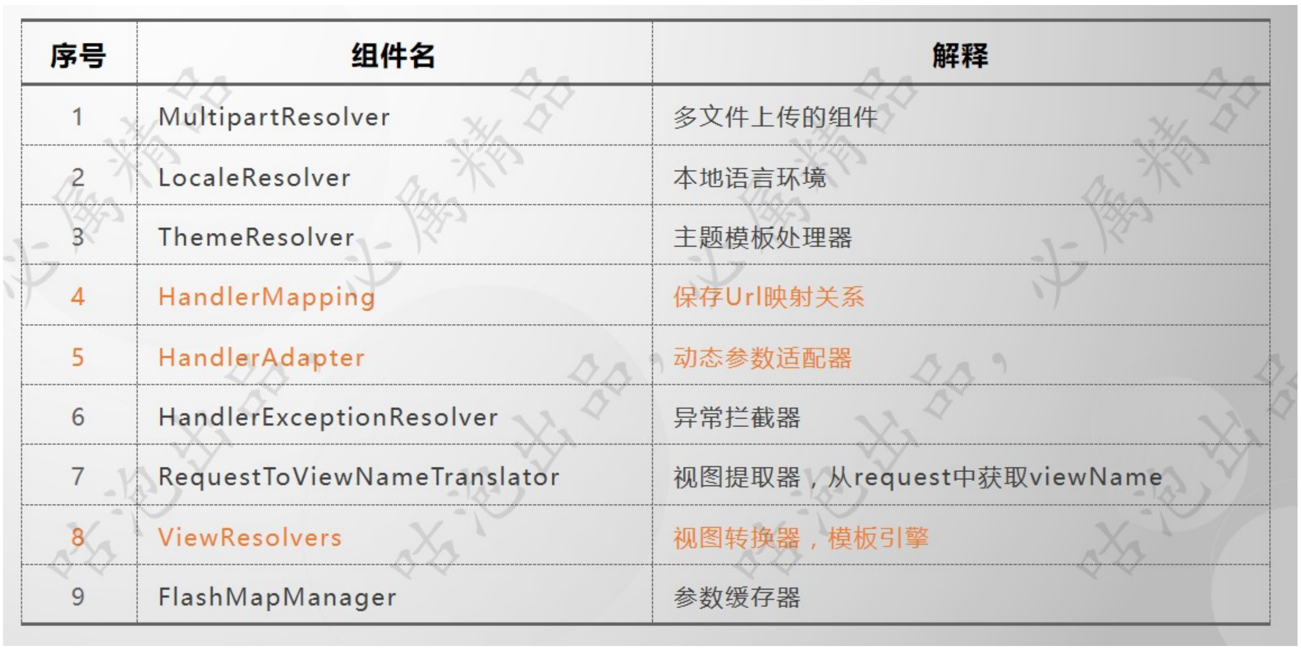
1、高仿真手写Spring MVC。
2、搭建基本框架，满足核心功能。

内容定位

在完全掌握Spring系统结构、实现原理，在理解设计模式的基础上，自己动手写一个高仿真版本的Spring框架，以达到透彻理解Spring的目的，感受作者创作意图。

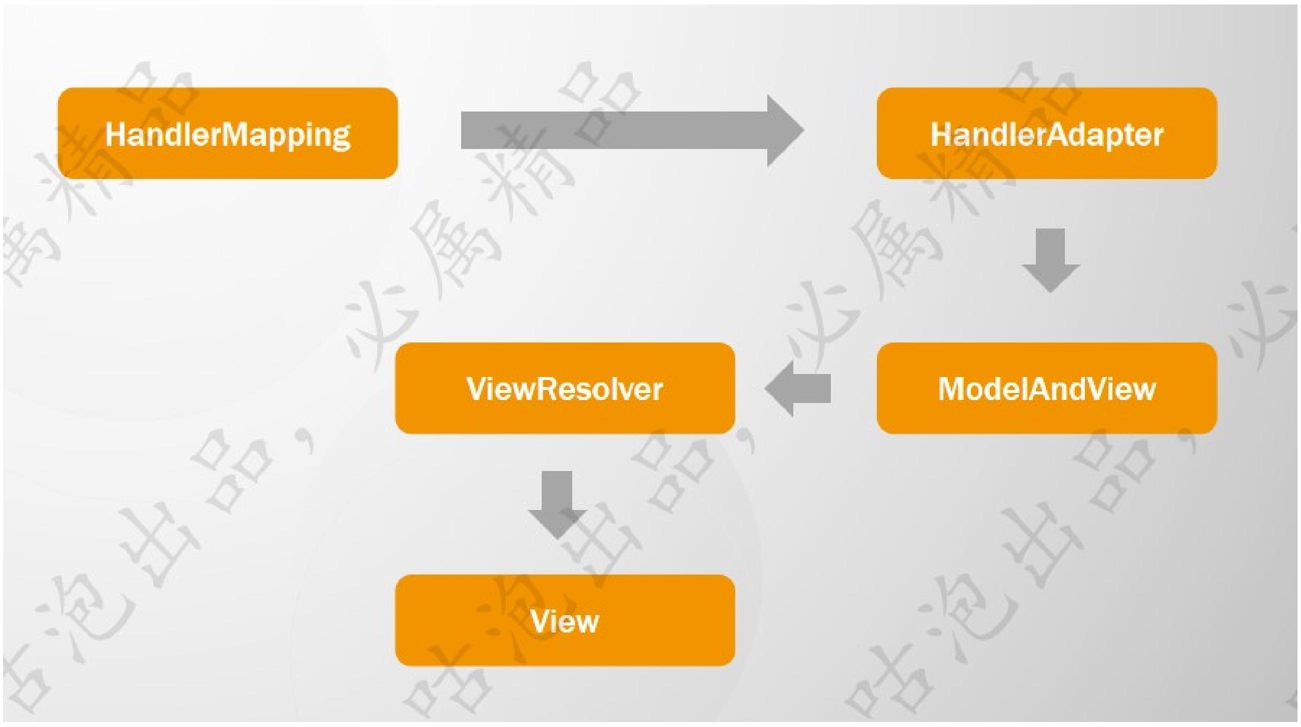
1. MVC顶层设计

MVC九大组件



序号	组件名	解释
1	MultipartResolver	多文件上传的组件
2	LocaleResolver	本地语言环境
3	ThemeResolver	主题模板处理器
4	HandlerMapping	保存Url映射关系
5	HandlerAdapter	动态参数适配器
6	HandlerExceptionResolver	异常拦截器
7	RequestToViewNameTranslator	视图提取器，从request中获取viewName
8	ViewResolvers	视图转换器，模板引擎
9	FlashMapManager	参数缓存器

Spring MVC 核心组件执行流程



MyDispatcherServlet 请求调度

```java
package cn.sitedev.spring.framework.webmvc.servlet;

import cn.sitedev.spring.framework.annotation.MyController;
import cn.sitedev.spring.framework.annotation.MyRequestMapping;
import cn.sitedev.spring.framework.context.MyApplicationContext;

import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.File;
import java.io.IOException;
import java.lang.reflect.Method;
import java.util.*;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class MyDispatcherServlet extends HttpServlet {
    private static final String CONTEXT_CONFIG_LOCATION = "contextConfigLocation";

    private MyApplicationContext context;

    private List<MyHandlerMapping> handlerMappings = new ArrayList<>();

    private Map<MyHandlerMapping, MyHandlerAdapter> handlerAdapters = new HashMap<>();

    private List<MyViewResolver> viewResolvers = new ArrayList<>();

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws Servl
        this.doPost(req, resp);
    }

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws Serv
        try {
            this.doDispatch(req, resp);
        } catch (Exception e) {
            try {
                MyModelAndView mav500 = new MyModelAndView("500");
                Map<String, Object> model500 = new HashMap<>();
                model500.put("detail", e.getMessage());
                model500.put("stackTrace", Arrays.toString(e.getStackTrace()));
                mav500.setModel(model500);
                processDispatchResult(req, resp, mav500);
            } catch (Exception exception) {
                exception.printStackTrace();
                resp.getWriter().write("500 Exception, Details:\r\n" + Arrays.toString(
            }
        }
    }

    private void doDispatch(HttpServletRequest request, HttpServletResponse response) t
        // 1. 通过从request中拿到URL，去匹配一个HandlerMapping
        MyHandlerMapping handler = getHandler(request);

        if (handler == null) {
            processDispatchResult(request, response, new MyModelAndView("404"));
            return;
        }

        // 2. 准备调用前的参数
        MyHandlerAdapter handlerAdapter = getHandlerAdapter(handler);

        // 3. 真正的调用方法，返回ModelAndView存储了要传给页面的值，和页面模板的名称
        MyModelAndView modelAndView = handlerAdapter.handle(request, response, handler)

        // 这一步才是真正的输出
        processDispatchResult(request, response, modelAndView);
    }

    private void processDispatchResult(HttpServletRequest request, HttpServletResponse
                                       MyModelAndView modelAndView) throws Exception {
        // 把给我的ModelAndView变成HTML, OutputStream, json, freemarker, velocity
        if (modelAndView == null) {
            return;
        }
        // 如果ModelAndView不为null, 怎么办?
        if (this.viewResolvers.isEmpty()) {
            return;
        }

        for (MyViewResolver viewResolver : this.viewResolvers) {
            MyView view = viewResolver.resolveViewName(modelAndView.getViewName(), null
            view.render(modelAndView.getModel(), request, response);
            return;
        }
    }

    private MyHandlerAdapter getHandlerAdapter(MyHandlerMapping handler) {
        if (this.handlerAdapters.isEmpty()) {
            return null;
        }
        MyHandlerAdapter handlerAdapter = this.handlerAdapters.get(handler);
        if (handlerAdapter.supports(handler)) {
            return handlerAdapter;
        }
        return null;
    }

    private MyHandlerMapping getHandler(HttpServletRequest request) {
        if (this.handlerMappings.isEmpty()) {
            return null;
        }
        String url = request.getRequestURI();
        String contextPath = request.getContextPath();
        url = url.replace(contextPath, "").replaceAll("/+", "/");

        for (MyHandlerMapping handlerMapping : this.handlerMappings) {
            try {
                Matcher matcher = handlerMapping.getPattern().matcher(url);
                // 如果没有匹配上继续下一个匹配
                if (!matcher.matches()) {
                    continue;
                }
                return handlerMapping;
            } catch (Exception e) {
                throw e;
            }
        }
        return null;
    }

    @Override
    public void init(ServletConfig config) throws ServletException {
        // 1. 初始化ApplicationContext
        context = new MyApplicationContext(config.getInitParameter(CONTEXT_CONFIG_LOCAT
        // 2. 初始化Spring MVC 九大组件
        initStrategies(context);
    }

    // 初始化策略
    private void initStrategies(MyApplicationContext context) {
        // handlerMapping, 必须实现
        initHandlerMappings(context);
        // 初始化参数适配器, 必须实现
        initHandlerAdapters(context);
        // 初始化视图转换器, 必须实现
        initViewResolvers(context);
    }

    private void initViewResolvers(MyApplicationContext context) {
        // 拿到模板的存放目录
        String templateRoot = context.getConfig().getProperty("templateRoot");
        String templateRootPath =
                this.getClass().getClassLoader().getResource(templateRoot).getFile();

        File templateRootDir = new File(templateRootPath);
        String[] templates = templateRootDir.list();
        for (int i = 0; i < templates.length; i++) {
            // 这里主要为了兼容多模板, 所以模仿Spring用List保存
            // 这里代码简化了, 其实只要一个模板就可以搞定
            // 只是为了仿真，所以还是搞了个List
            this.viewResolvers.add(new MyViewResolver(templateRoot));
        }
    }

    private void initHandlerAdapters(MyApplicationContext context) {
        // 把一个request请求变成handler，参数都是字符串，自动配到handler中的形参
        // 可想而知，它要拿到HandlerMapping才能干活
        // 就意味着，有几个HandlerMapping就有几个HandlerAdapter
        for (MyHandlerMapping handlerMapping : this.handlerMappings) {
            this.handlerAdapters.put(handlerMapping, new MyHandlerAdapter());
        }
    }

    private void initHandlerMappings(MyApplicationContext context) {
        String[] beanNames = context.getBeanDefinitionNames();

        try {
            for (String beanName : beanNames) {
                Object controller = context.getBean(beanName);

                Class<?> clazz = controller.getClass();

                if (!clazz.isAnnotationPresent(MyController.class)) {
                    continue;
                }

                String baseUrl = "";
                // 获取Controller的url配置
                if (clazz.isAnnotationPresent(MyRequestMapping.class)) {
                    MyRequestMapping requestMapping = clazz.getAnnotation(MyRequestMapp
                    baseUrl = requestMapping.value();
                }

                // 获取Method的url配置
                Method[] methods = clazz.getMethods();
                for (Method method : methods) {
                    // 没有加RequestMapping注解的直接忽略
                    if (!method.isAnnotationPresent(MyRequestMapping.class)) {
                        continue;
                    }
                    // 映射URL
                    MyRequestMapping requestMapping = method.getAnnotation(MyRequestMap

                    String regex = ("/" + baseUrl + "/" + requestMapping.value().replac
                            , ".*")).replaceAll("/+", "/");

                    Pattern pattern = Pattern.compile(regex );

                    this.handlerMappings.add(new MyHandlerMapping(pattern, controller,
                    System.out.println("Mapped " + regex + ", " + method);
                }

            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

MyHandlerMapping 请求映射

```java
package cn.sitedev.spring.framework.webmvc.servlet;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.lang.reflect.Method;
import java.util.regex.Pattern;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class MyHandlerMapping {
    // URL的正则匹配
    private Pattern pattern;
    // 保存方法对应的实例
    private Object controller;
    // 保存映射的方法
    private Method method;
}
```

```java
package cn.sitedev.spring.framework.webmvc.servlet;

import cn.sitedev.spring.framework.annotation.MyRequestParam;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.lang.annotation.Annotation;
import java.lang.reflect.InvocationTargetException;
import java.util.Arrays;
import java.util.HashMap;
import java.util.Map;

public class MyHandlerAdapter {
    public boolean supports(Object handler) {
        return handler instanceof MyHandlerMapping;
    }

    public MyModelAndView handle(HttpServletRequest request, HttpServletResponse respor
                                 Object handler) throws InvocationTargetException,
            IllegalAccessException {
        MyHandlerMapping handlerMapping = (MyHandlerMapping) handler;

        // 把 方法的形参列表和request的参数列表所在顺序一一对应
        Map<String, Integer> paramIndexMapping = new HashMap<>();

        // 提取方法中加了注解的参数
        // 把方法上的注解拿到，得到的是一个二维数组
        // 因为一个参数可以有多个注解，而一个方法又有多个参数
        Annotation[][] pa = handlerMapping.getMethod().getParameterAnnotations();
        for (int i = 0; i < pa.length; i++) {
            for (Annotation a : pa[i]) {
                if (a instanceof MyRequestParam) {
                    String paramName = ((MyRequestParam) a).value();
                    if (!"".equals(paramName.trim())) {
                        paramIndexMapping.put(paramName, i);
                    }
                }
            }
        }

        // 提取方法中的request和response参数
        for (int i = 0; i < paramsTypes.length; i++) {
            Class<?> type = paramsTypes[i];
            if (type == HttpServletRequest.class || type == HttpServletResponse.class)
                paramIndexMapping.put(type.getName(), i);
            }
        }

        // 获得方法的形参列表
        Map<String, String[]> params = request.getParameterMap();

        // 实参列表
        Object[] paramValues = new Object[paramsTypes.length];

        for (Map.Entry<String, String[]> param : params.entrySet()) {
            String value =
                    Arrays.toString(param.getValue()).replaceAll("\\[|\\]", "").replace
                            ",");
            if (!paramIndexMapping.containsKey(param.getKey())) {
                continue;
            }
            int index = paramIndexMapping.get(param.getKey());
            paramValues[index] = caseStringValue(value, paramsTypes[index]);
        }

        if (paramIndexMapping.containsKey(HttpServletRequest.class.getName())) {
            int reqIndex = paramIndexMapping.get(HttpServletRequest.class.getName());
            paramValues[reqIndex] = request;
        }

        if (paramIndexMapping.containsKey(HttpServletResponse.class.getName())) {
            int respIndex = paramIndexMapping.get(HttpServletResponse.class.getName());
            paramValues[respIndex] = response;
        }

        Object result = handlerMapping.getMethod().invoke(handlerMapping.getController(
                paramValues);
        if (result == null || result instanceof Void) {
            return null;
        }

        boolean isModelAndView = handlerMapping.getMethod().getReturnType() == MyModelA
        if (isModelAndView) {
            return (MyModelAndView) result;
        }
        return null;
    }

    private Object caseStringValue(String value, Class<?> paramsType) {
        if (String.class == paramsType) {
            return value;
        }
        // 如果还有double或者其他类型，继续加if
        // 这时候，我们应该想到策略模式
        // 在这里暂时不实现
        if (Integer.class == paramsType) {
            return Integer.valueOf(value);
        }
        if (Double.class == paramsType) {
            return Double.valueOf(value);
        }
        if (value != null) {
            return value;
        }
        return null;
    }
}
```

MyModelAndView 页面数据封装

```java
package cn.sitedev.spring.framework.webmvc.servlet;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.util.Map;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class MyModelAndView {
    private String viewName;
    private Map<String, ?> model;

    public MyModelAndView(String viewName) {
        this.viewName = viewName;
    }
}
```

MyViewResolver 视图解析器

```java
package cn.sitedev.spring.framework.webmvc.servlet;

import java.io.File;
import java.util.Locale;

public class MyViewResolver {
    private static final String DEFAULT_TEMPLATE_SUFFIX = ".html";

    private File templateRootDir;

    public MyViewResolver(String templateRoot) {
        String templateRootPath =
                this.getClass().getClassLoader().getResource(templateRoot).getFile();
        this.templateRootDir = new File(templateRootPath);
    }

    public MyView resolveViewName(String viewName, Locale locale) throws Exception {
        if (viewName == null || "".equals(viewName.trim())) {
            return null;
        }
        viewName = viewName.endsWith(DEFAULT_TEMPLATE_SUFFIX) ? viewName :
                (viewName + DEFAULT_TEMPLATE_SUFFIX);
        File templateFile = new File((templateRootDir.getPath() + "/" + viewName).repla
                , "/"));
        return new MyView(templateFile);
    }
}
```

MyView 自定义模板引擎

```java
package cn.sitedev.spring.framework.webmvc.servlet;

```

```java
import lombok.Data;
import lombok.NoArgsConstructor;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.File;
import java.io.RandomAccessFile;
import java.util.Map;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class MyView {
    private File viewFile;

    public void render(Map<String, ?> model, HttpServletRequest request,
                       HttpServletResponse response) throws Exception {
        StringBuffer stringBuffer = new StringBuffer();

        RandomAccessFile randomAccessFile = new RandomAccessFile(this.viewFile, "r");
        String line = null;
        while ((line = randomAccessFile.readLine()) != null) {
            line = new String(line.getBytes("ISO-8859-1"), "UTF-8");
            Pattern pattern = Pattern.compile("￥\\{[^\\}]+\\}", Pattern.CASE_INSENSITI
            Matcher matcher = pattern.matcher(line);
            while (matcher.find()) {
                String paramName = matcher.group();
                paramName = paramName.replaceAll("￥\\{|\\}", "");
                Object paramValue = model.get(paramName);
                if (paramValue == null) {
                    continue;
                }
                line = matcher.replaceFirst(makeStringForRegExp(paramValue.toString()))
                matcher = pattern.matcher(line);
            }
            stringBuffer.append(line);
        }
        response.setCharacterEncoding("UTF-8");
        response.getWriter().write(stringBuffer.toString());

    }

    // 处理特殊字符
    private String makeStringForRegExp(String str) {
        return str
                .replace("\\","\\\\").replace("*", "\\*")
                .replace("+","\\+").replace("|","\\|")
                .replace("{","\\{").replace("}","\\}")
                .replace("(", "\\(").replace(")","\\)")
                .replace("^", "\\^").replace("$","\\$")
                .replace("[","\\[").replace("]","\\]")
                .replace("?","\\?").replace(",","\\,")
                .replace(".","\\.").replace("&","\\&");
    }
}
```

2. 业务代码实现

IQueryService 查询业务接口定义

```java
package cn.sitedev.demo.service;

/**
 * 查询业务
 */
public interface IQueryService {
    // 查询
    String query(String name);
}
```

QueryService 查询处理业务逻辑

```java
package cn.sitedev.demo.service.impl;

import cn.sitedev.demo.service.IQueryService;
import cn.sitedev.spring.framework.annotation.MyService;

import java.text.SimpleDateFormat;
import java.util.Date;

```

```java
/**
 * 查询业务
 */
@MyService
public class QueryService implements IQueryService {
    @Override
    public String query(String name) {
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        String time = sdf.format(new Date());
        String json = "{ name: \"" + name + "\", time: \"" + time + "\"}";

        System.out.println("这是在业务方法中打印的:" + json);
        return json;
    }
}
```

IModifyService增、删、改业务接口定义

```java
package cn.sitedev.demo.service;

/**
 * 增删改业务
 */
public interface IModifyService {
    // 增加
    String add(String name, String addr);

    // 修改
    String edit(String id, String name);

    // 删除
    String remove(Integer id);
}
```

ModifyService增、删、改业务逻辑实现

```java
package cn.sitedev.demo.service.impl;

import cn.sitedev.demo.service.IModifyService;
```

```java
/**
 * 增删改业务
 */
public class ModifyService implements IModifyService {
    @Override
    public String add(String name, String addr) {
        return "ModifyService add: name = " + name + ", addr = " + addr;
    }

    @Override
    public String edit(String id, String name) {
        return "ModifyService edit: id = " + id + ", name = " + name;
    }

    @Override
    public String remove(Integer id) {
        return "ModifyService remove: id = " + id;
    }
}
```

MyAction数据逻辑处理

```java
package cn.sitedev.demo.action;

import cn.sitedev.demo.service.IModifyService;
import cn.sitedev.demo.service.IQueryService;
import cn.sitedev.spring.framework.annotation.MyAutowired;
import cn.sitedev.spring.framework.annotation.MyController;
import cn.sitedev.spring.framework.annotation.MyRequestMapping;
import cn.sitedev.spring.framework.annotation.MyRequestParam;
import cn.sitedev.spring.framework.webmvc.servlet.MyModelAndView;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

/**
 * 公布接口url
 */
@MyController
@MyRequestMapping("/web")
```

```java
public class MyAction {
    @MyAutowired
    private IQueryService queryService;

    @MyAutowired
    private IModifyService modifyService;

    @MyRequestMapping("/query.json")
    public MyModelAndView query(HttpServletRequest request, HttpServletResponse respons
                                @MyRequestParam("name") String name) {
        String result = queryService.query(name);
        return out(response, result);
    }

    @MyRequestMapping("/add*.json")
    public MyModelAndView add(HttpServletRequest request, HttpServletResponse response,
                              @MyRequestParam("name") String name,
                              @MyRequestParam("addr") String addr) {
        String result = modifyService.add(name, addr);
        return out(response, result);
    }

    @MyRequestMapping("/remove.json")
    public MyModelAndView remve(HttpServletRequest request, HttpServletResponse respons
                                @MyRequestParam("id") Integer id) {
        String result = modifyService.remove(id);
        return out(response, result);
    }

    @MyRequestMapping("edit.json")
    public MyModelAndView edit(HttpServletRequest request, HttpServletResponse response
                               @MyRequestParam("id") Integer id,
                               @MyRequestParam("name") String name) {
        String result = modifyService.edit(id, name);
        return out(response, result);
    }

    private MyModelAndView out(HttpServletResponse response, String str) {
        try {
            response.getWriter().write(str);
        } catch (IOException e) {
            e.printStackTrace();
        }
        return null;
    }
}
```

PageAction页面逻辑处理

```
package cn.sitedev.demo.action;

import cn.sitedev.demo.service.IQueryService;
import cn.sitedev.spring.framework.annotation.MyAutowired;
import cn.sitedev.spring.framework.annotation.MyController;
import cn.sitedev.spring.framework.annotation.MyRequestMapping;
import cn.sitedev.spring.framework.annotation.MyRequestParam;
import cn.sitedev.spring.framework.webmvc.servlet.MyModelAndView;

import java.util.HashMap;
import java.util.Map;

/**
 * 公布接口url
 */
@MyController
@MyRequestMapping("/")
public class PageAction {
    @MyAutowired
    private IQueryService queryService;

    @MyRequestMapping("/first.html")
    public MyModelAndView query(@MyRequestParam("teacher") String teacher) {
        String result = queryService.query(teacher);
        Map<String, Object> model = new HashMap<>();
        model.put("teacher", teacher);
        model.put("data", result);
        model.put("token", "123456");
        return new MyModelAndView("first.html", model);
    }
}
```

3. 定制模板页面

first.html 动态展示页

```html
<!DOCTYPE html>
<html lang="zh-cn">
<head>
    <meta charset="utf-8">
    <title>咕泡学院SpringMVC模板引擎演示</title>
</head>
<center>
    <h1>大家好，我是￥{teacher}老师<br/>欢迎大家一起来探索Spring的世界</h1>
    <h3>Hello,My name is ￥{teacher}</h3>
    <div>￥{data}</div>
    Token值：￥{token}
</center>
</html>
```

404.html页面未找到提示

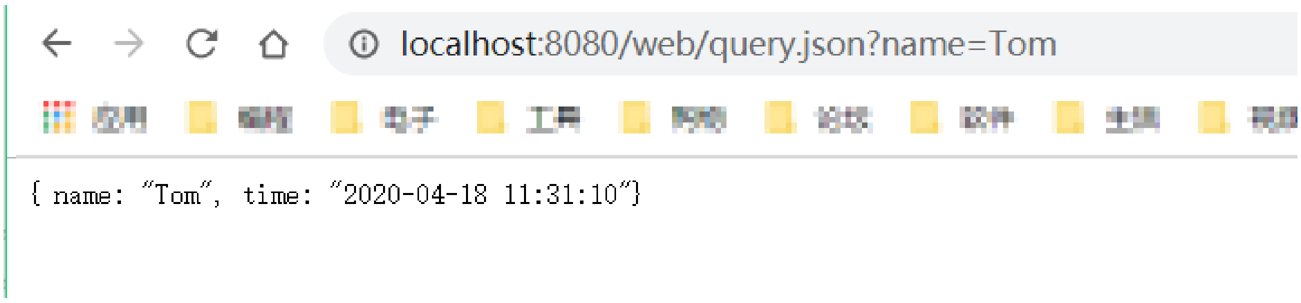
```html
<!DOCTYPE html>
<html lang="zh-cn">
<head>
    <meta charset="utf-8">
    <title>页面去火星了</title>
</head>
<body>
    <font size='25' color='red'>404 Not Found</font><br/><font color='green'><i>Copyrig
</body>
</html>
```

500.html 错误提示页面

```html
<!DOCTYPE html>
<html lang="zh-cn">
<head>
    <meta charset="utf-8">
    <title>服务器好像累了</title>
</head>
<body>
    <font size='25' color='blue'>500 服务器好像有点累了，需要休息一下</font><br/>
    <b>Message:￥{detail}</b><br/>
    <b>StackTrace:￥{stackTrace}</b><br/>
    <font color='green'><i>Copyright@GupaoEDU</i></font>
</body>
</html>
```

4. 运行效果演示

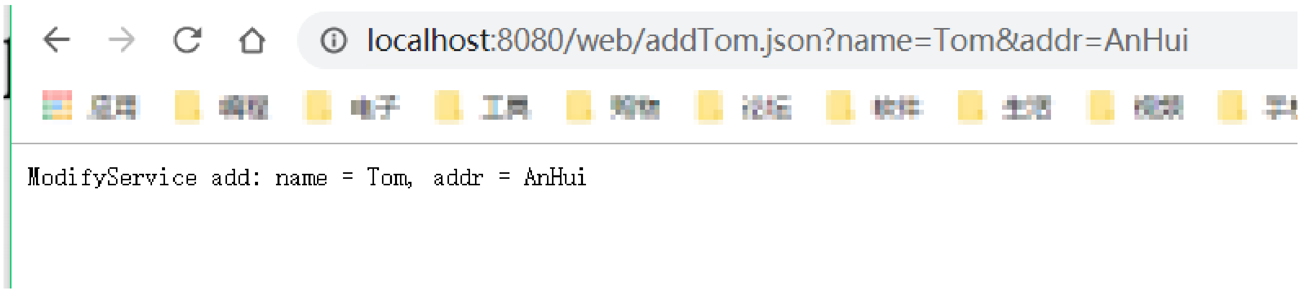
/query.json



localhost:8080/web/query.json?name=Tom

```
{ name: "Tom", time: "2020-04-18 11:31:10"}
```

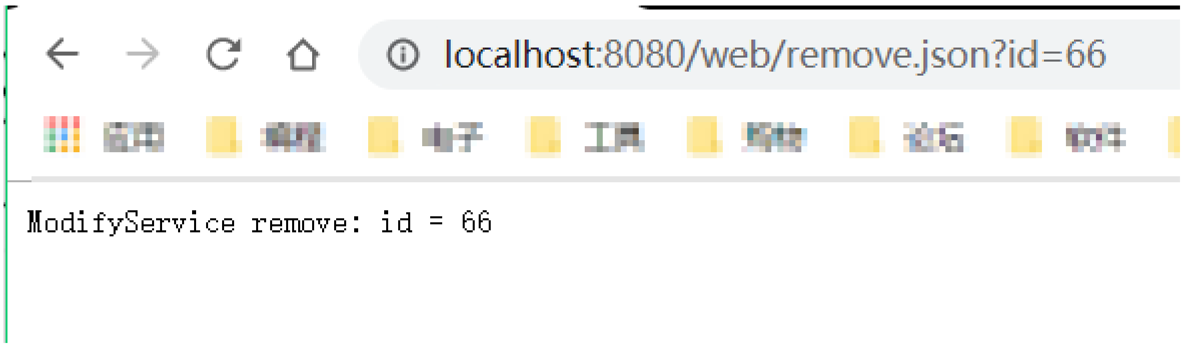
/add*.json



localhost:8080/web/addTom.json?name=Tom&addr=AnHui

```
ModifyService add: name = Tom, addr = AnHui
```

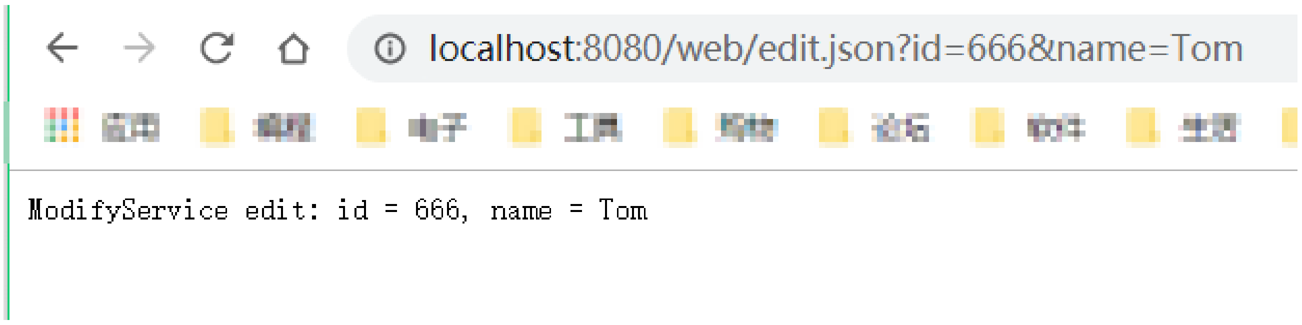
/remove.json



localhost:8080/web/remove.json?id=66

```
ModifyService remove: id = 66
```

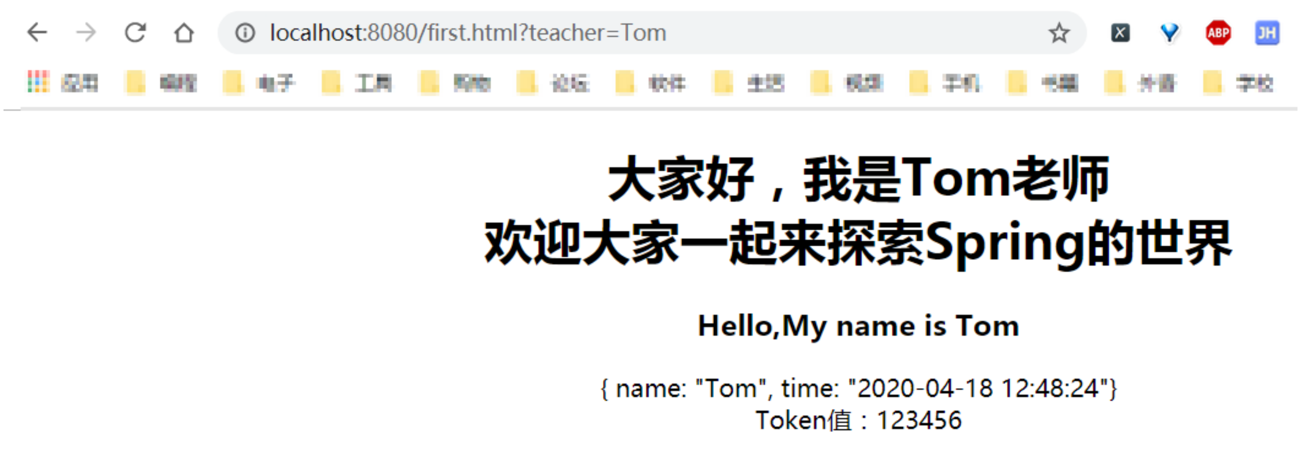
/edit.json



localhost:8080/web/edit.json?id=666&name=Tom

```
ModifyService edit: id = 666, name = Tom
```

first.html



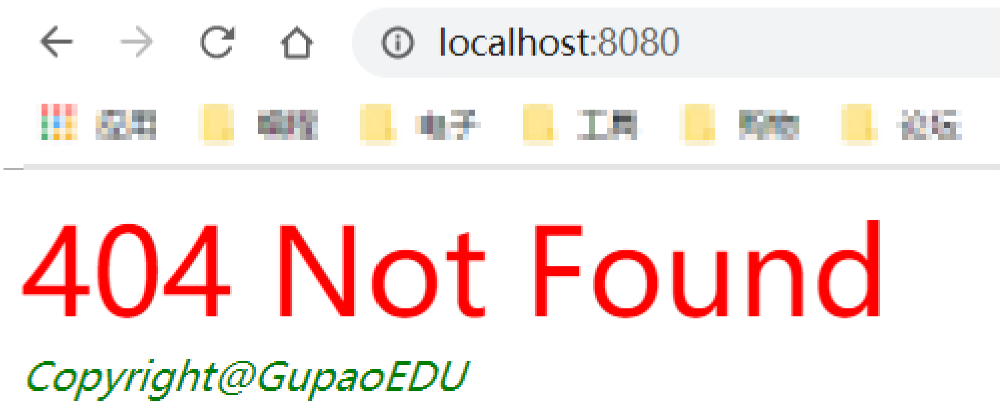
localhost:8080/first.html?teacher=Tom

大家好，我是Tom老师
欢迎大家一起来探索Spring的世界

Hello,My name is Tom

{ name: "Tom", time: "2020-04-18 12:48:24"}
Token值：123456

404页面

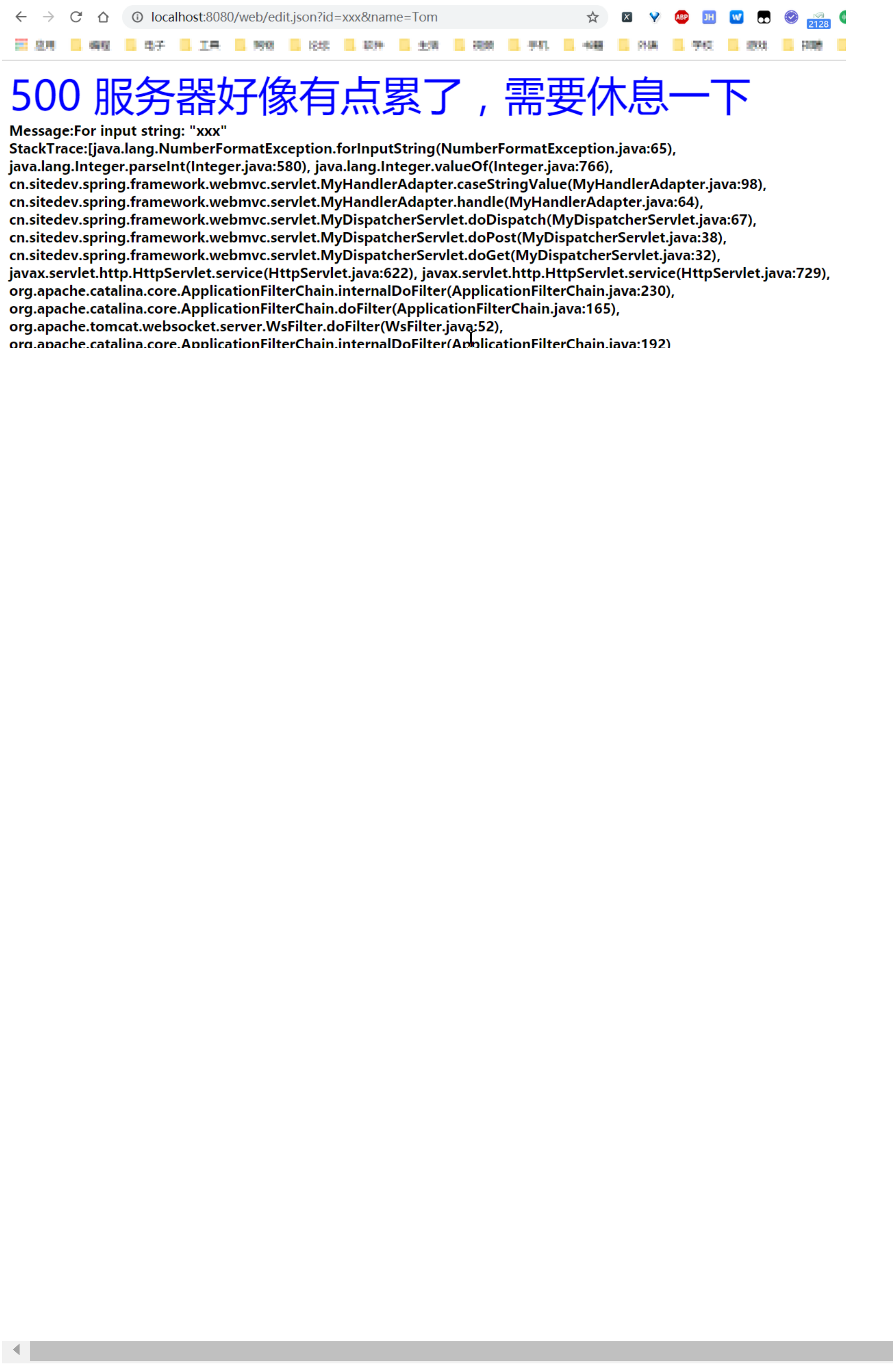


localhost:8080

404 Not Found

Copyright@GupaoEDU

500页面



localhost:8080/web/edit.json?id=xxx&name=Tom

500 服务器好像有点累了，需要休息一下

```
Message:For input string: "xxx"
StackTrace:[java.lang.NumberFormatException.forInputString(NumberFormatException.java:65),
java.lang.Integer.parseInt(Integer.java:580), java.lang.Integer.valueOf(Integer.java:766),
cn.sitedev.spring.framework.webmvc.servlet.MyHandlerAdapter.caseStringValue(MyHandlerAdapter.java:98),
cn.sitedev.spring.framework.webmvc.servlet.MyHandlerAdapter.handle(MyHandlerAdapter.java:64),
cn.sitedev.spring.framework.webmvc.servlet.MyDispatcherServlet.doDispatch(MyDispatcherServlet.java:67),
cn.sitedev.spring.framework.webmvc.servlet.MyDispatcherServlet.doPost(MyDispatcherServlet.java:38),
cn.sitedev.spring.framework.webmvc.servlet.MyDispatcherServlet.doGet(MyDispatcherServlet.java:32),
javax.servlet.http.HttpServlet.service(HttpServlet.java:622), javax.servlet.http.HttpServlet.service(HttpServlet.java:729),
org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:230),
org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:165),
org.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter.java:52),
org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:192)
```