用友
yonyou

用友云平台
yonyou cloud platform

互联网开发培训教程
iuap 开发入门

2019 年

版权

目录

1 课堂目标

本节课讲结合案例学习 iuap 开发框架，具体目标：

1、 结合案例学习 iuap 前端框架 tinper-react 基本代码结构、基础组件介绍；

2、 学习 iuap 后台技术框架、项目结构、代码开发框架；

3、 完成 Helloworld 案例

当访问前端页面地址：http://127.0.0.1:3000/iuap_train_example/helloworld#/ 点击按钮弹出 hello world 提示

图 1：案例效果示意图

案例中提示信息“Hello world！”是从后台查询的数据中得出的，调用的后台服务为 http://127.0.0.1:8180/iuap_pap_quickstart/hello_world/。

本案例主旨在于帮助大家熟悉前后端基本代码框架，同时练习了前后台服务对接过程。

2 环境说明

培训中学习的案例为前后端分离的方式开发业务系统。

- 前端使用 tinper-react 框架开发前端页面；
- 后端使用 iuap 后端开发框架，开发后台服务；
- 开发者开发的业务功能，如果需要添加业务组件相关功能，如：编码规则、流程、权限等，需要使用 iuap 业务组件，同时与应用平台对接。

下面描述了在开发模式下，使用前端代理模式下配置的多个业务系统的调用关系。实际业务部署可以选择 nginx 代理的方式。

在前端业务节点上调用不同的服务，会代理到对应的后台服务地址上。具体配置请参见前端环境说明。

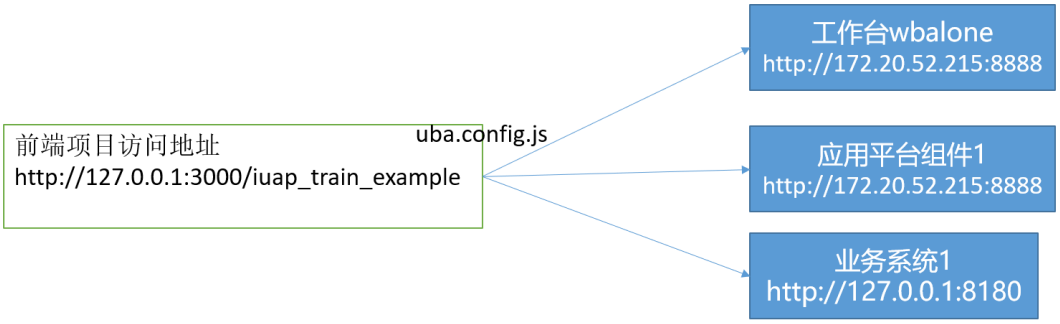

图 2：前端项目代理到不同后台服务

3 前端开发

3.1 前端环境准备

（1） 代理环境配置

在 uba.config.js 中，需要代理访问的后端地址，本案例中需要代理两处地址，一是应用平台地址，二是后台自身的业务服务器地址，代理方式，配置 uba.config.js 中的 proxyConfig，一是配置地址，而是在 router 字段中配置后台服务名

代理配置

```
const proxyConfig = [
  {
    enable: true,
    headers: {
      // 这是之前网页的地址，从中可以看到当前请求页面的链接。
      "Referer": "http://10.190.252.42:80"
    },
    // context，如果不配置，默认就是代理全部。
    router: [
      '/wbalone',     '/iuap-saas-message-center/',     '/iuap-saas-filesystem-service/',
'/eiap-plus/', '/newref/', '/print_service/', '/iuap-print/'
    ],
    url: 'http://10.190.252.42:80'
  },
  {
```

```
    enable: true,
    headers: {
      // 这是之前网页的地址，从中可以看到当前请求页面的链接。
      "Referer": "10.190.252.42:80"
    },
    // context，如果不配置，默认就是代理全部。
    router: [
      '/iuap_walsin_demo'   //配置项目后台业务服务器
    ],
    url: 'http://10.190.252.42:80'
  }
];
```

（2） 安装模块依赖包

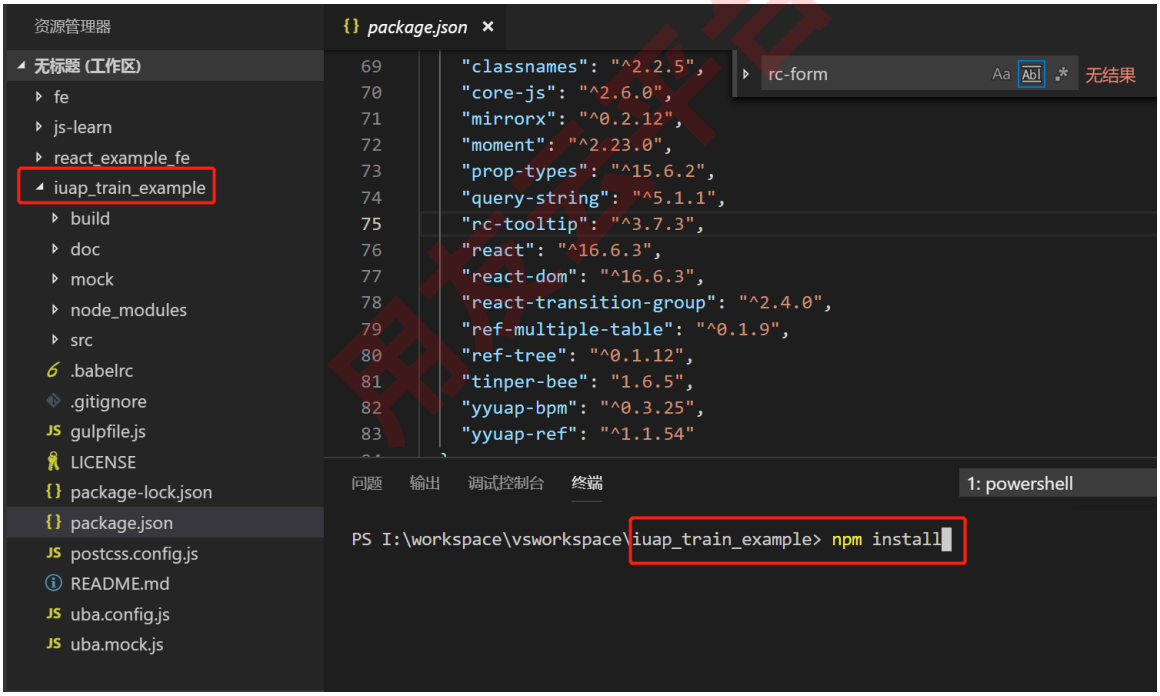

图 3：安装前端依赖包

在项目根目录下运行 npm install

（3） 项目启动

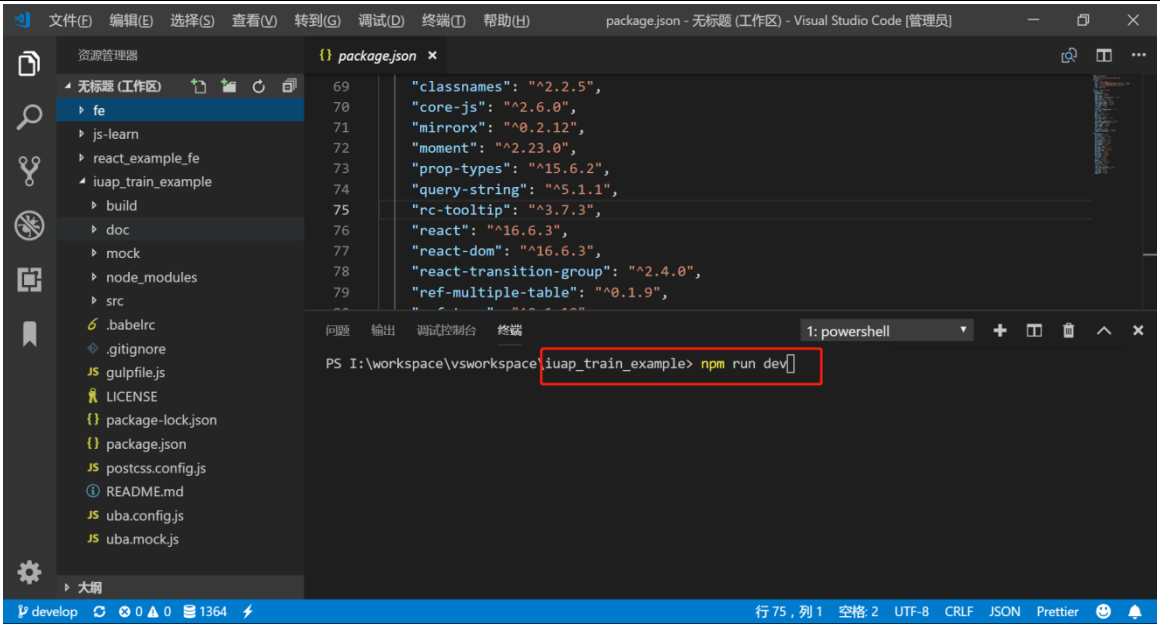

图 4：启动项目

项目根目录下运行 npm run dev，启动成功后提示如下：

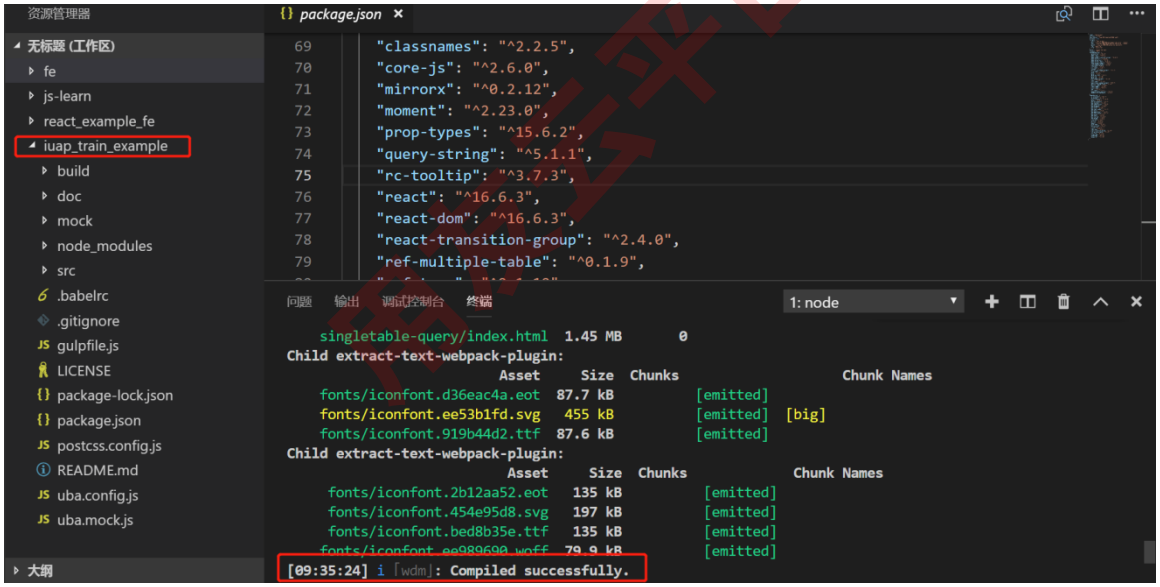

图 5：启动成功页面

（4） 登录应用平台

访问 http://127.0.0.1:3000/wbalone/pages/login/login.html 地址，效果如下。

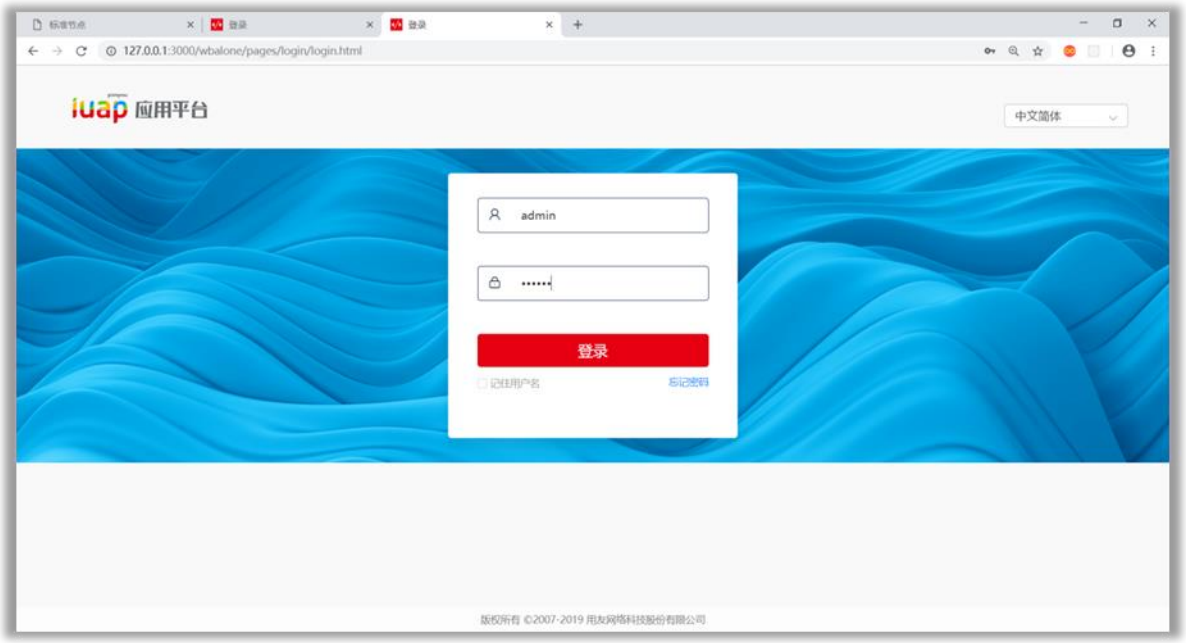

图 6：应用平台登录页面

用户名/密码为 admin/123qwe

（5） 访问页面路由地址

http://127.0.0.1:3000/iuap_train_example/standard#/，如果出现以下页面表示路由配置成功。

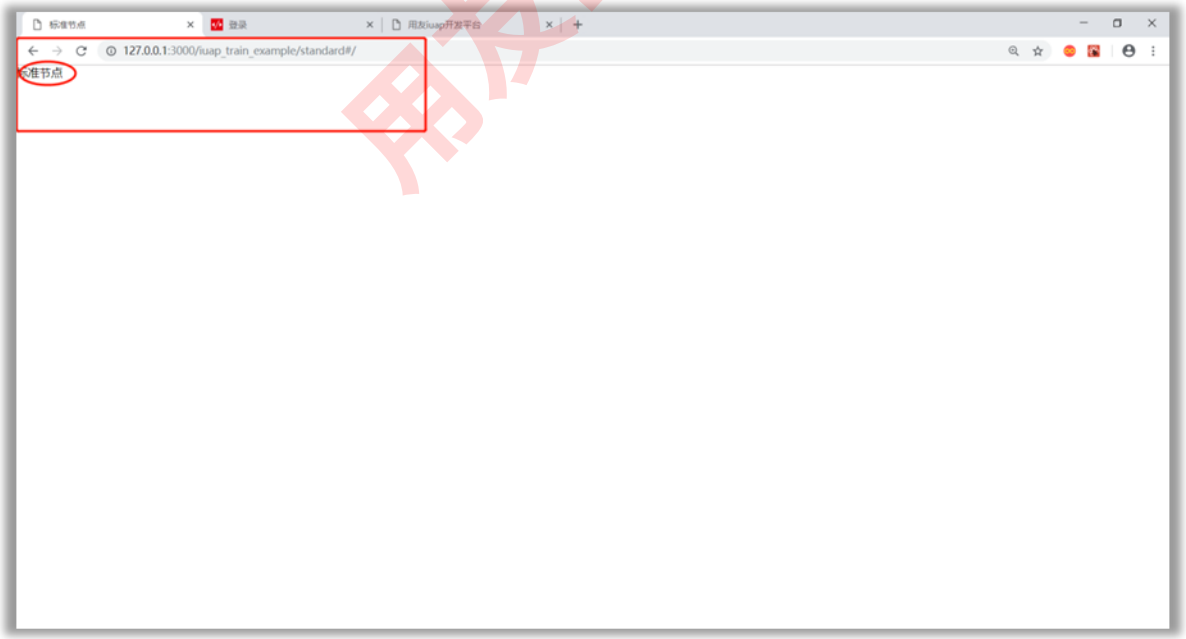

图 7：标准节点效果图

下面根据复制给出的 standard 节点，修改为 helloworld 节点并添加功能。最终效果如下

图 8：最终效果图

3.2 前端节点代码结构

Todo 脚手架项目地址

前端文件功能说明：

节点主要按照视图 UI 层、数据模型层、容器层、服务请求层、通用工具层、路由配置层，其中

① **视图 UI 层** 用于设计视图中的 UI 组件，视图 UI 层放置于节点目录 components 文件夹下

② **路由配置层** 用于配置页面访问路径（通过输入路由地址，展示相应的组件）、路由配置层位于节点 routes 文件夹

③ **数据模型层** 用于配置数据模型，供 UI 层使用。位于 model.js 中

④ **容器层** 容器层用于连接数据模型层与视图 UI 层 位于 container.js 中

⑤ **服务请求层** 用于定义具体的请求结构及完成最终请求 位于 service.js 中

⑥ **通用工具类** 用于定义通用的处理方法 位于 src/utils/ 文件夹中

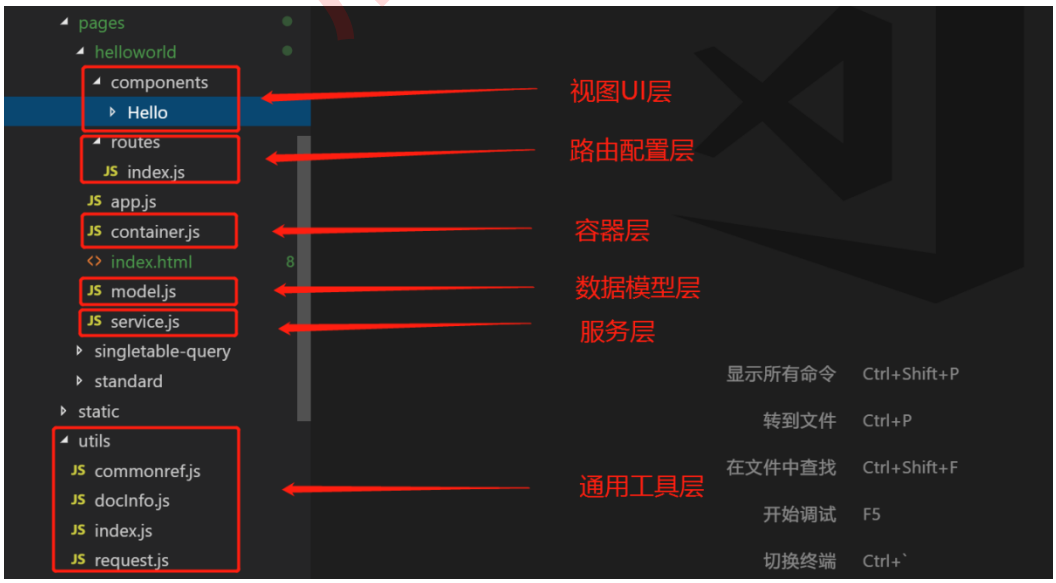

图 9：节点结构

3.3 helloworld 前台开发

3.3.1 创建应用节点

在\src\pages 创建如下节点。

（1） 复制案例中提供的 standard 节点

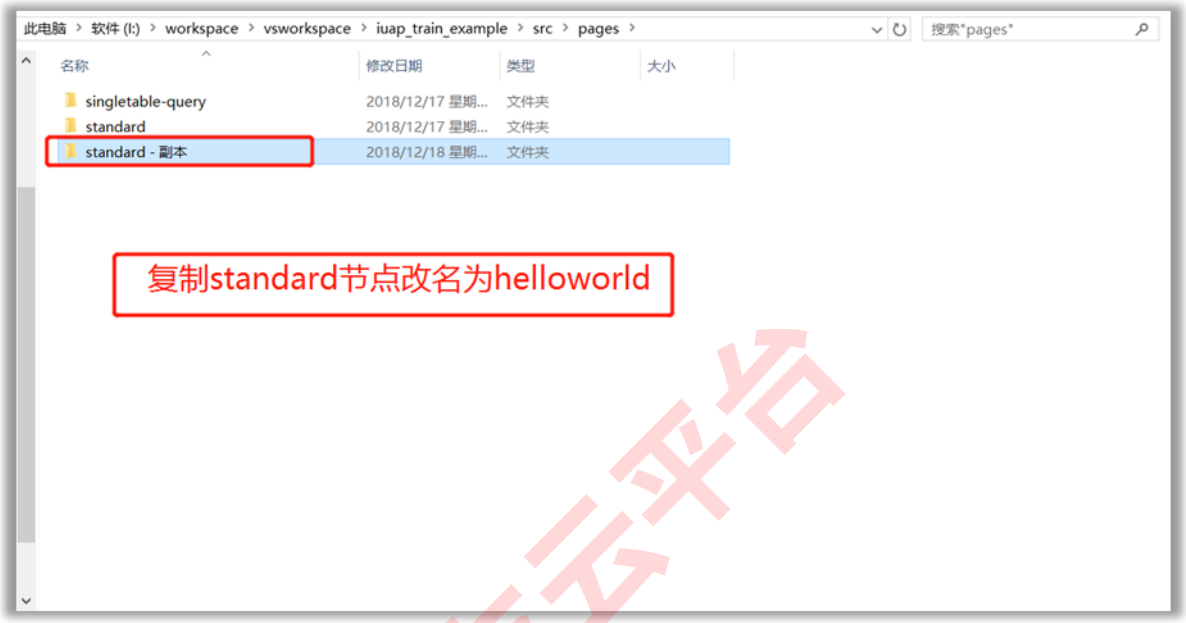

图 10：复制标准节点

修改完成后，编辑器中的节点名称会自动修改。

（2） 在 index.html 中修改节点名称

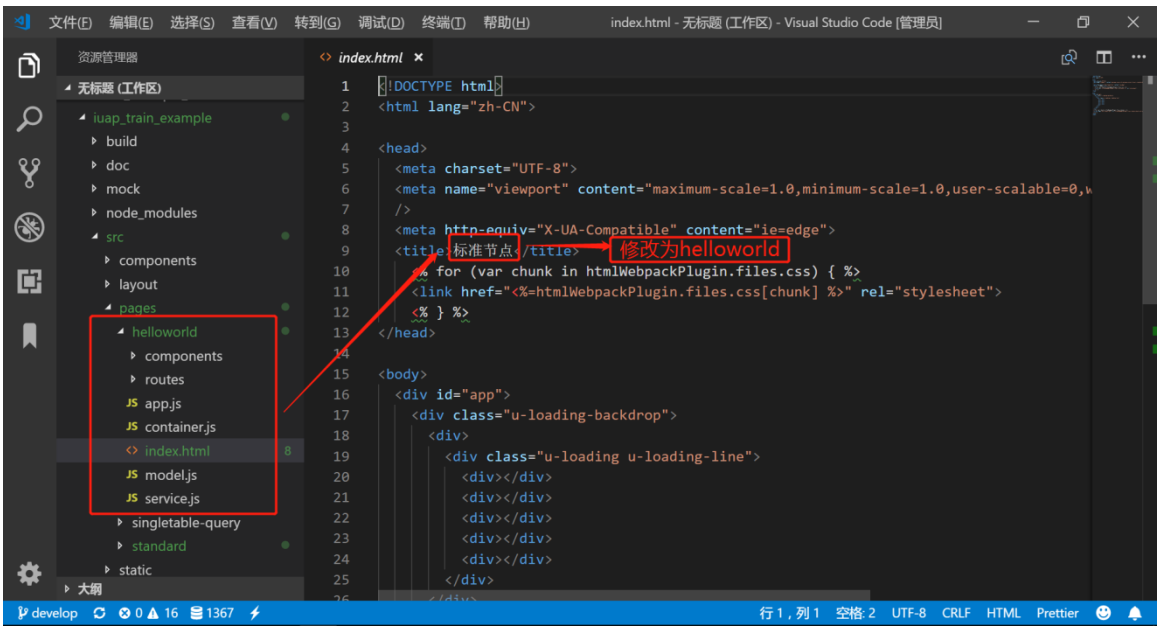

图 11：修改节点 title 值

（3） 修改组件名

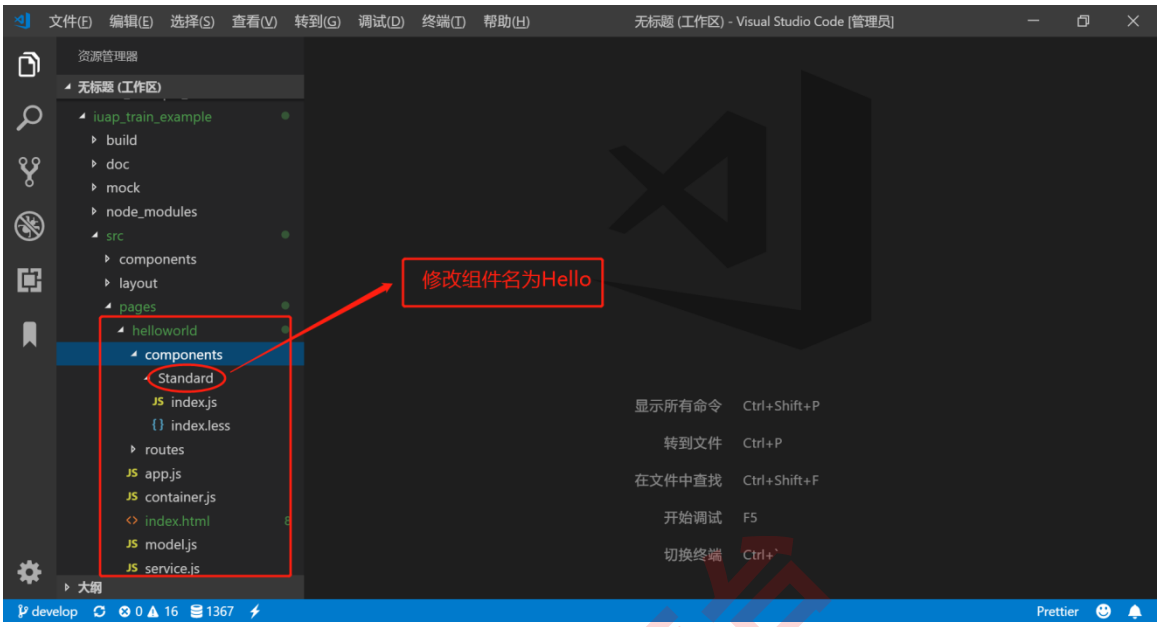

图 12：修改组件名

（4） 数据模型层名称修改

在 model.js 中修改 name 字段

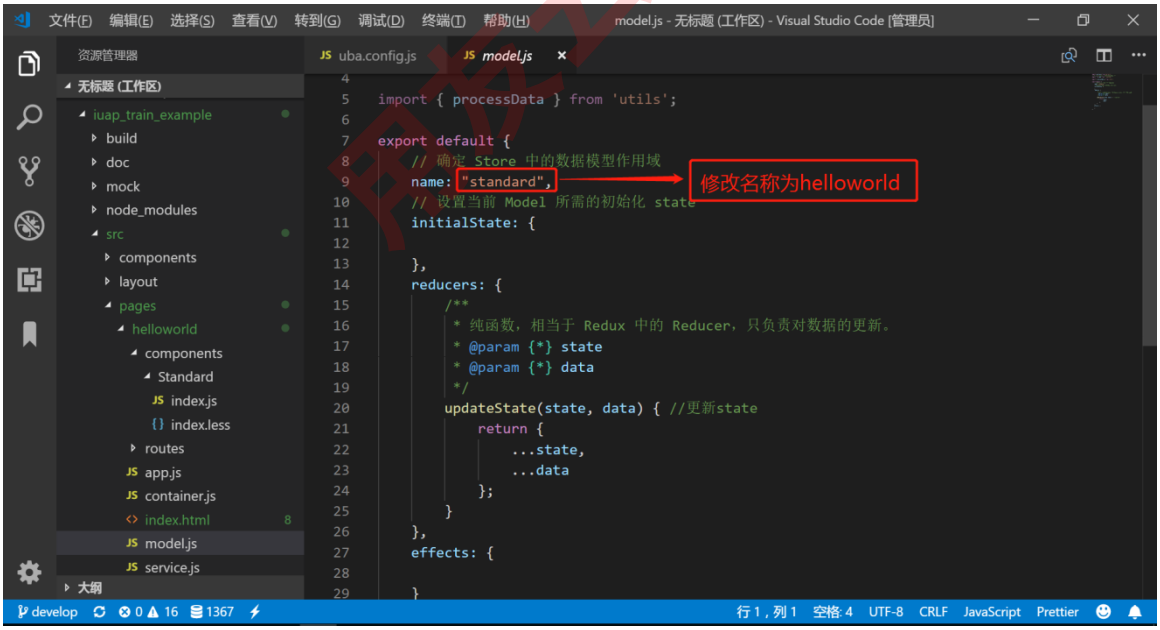

图 13：修改 model 名称

（5） 修改容器层 container.js 中导入的组件名

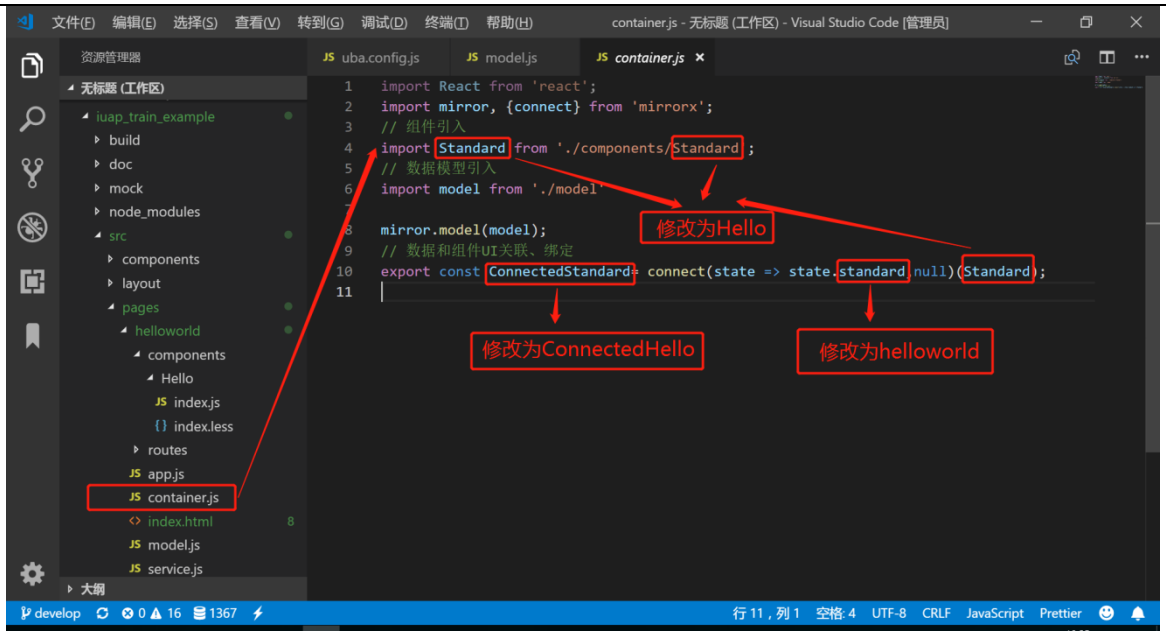

图 14：修改容器层内容

（6） 路由配置层

在 helloworld 节点下的 routes/index.js 中修改路由

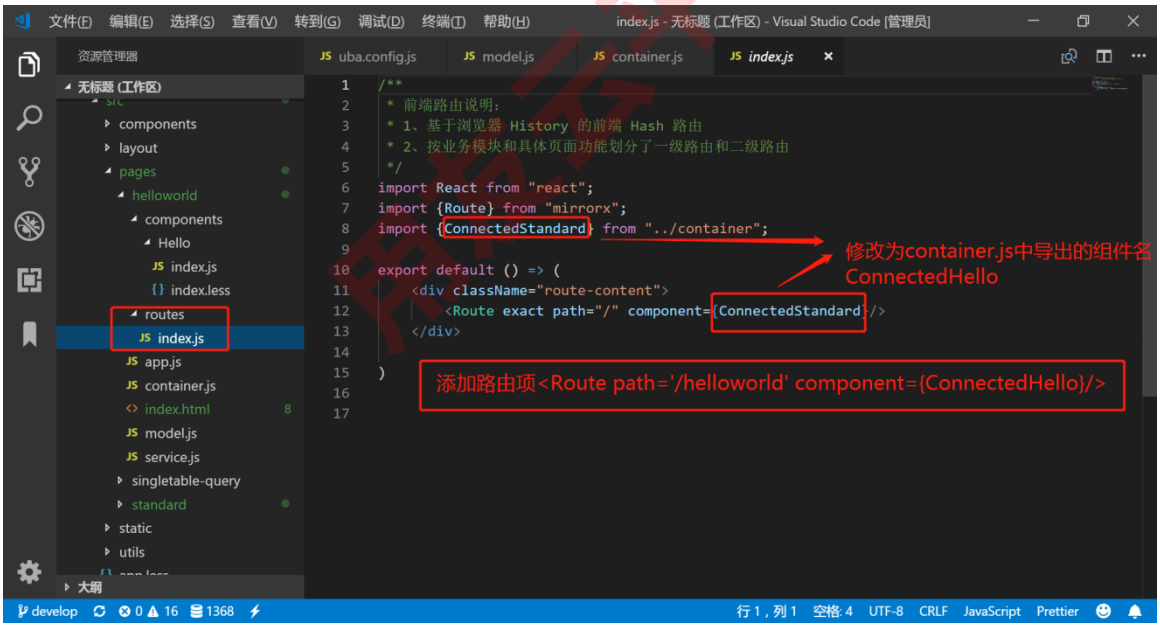

图 15：修改路由

（7） 访问路由地址，效果如下

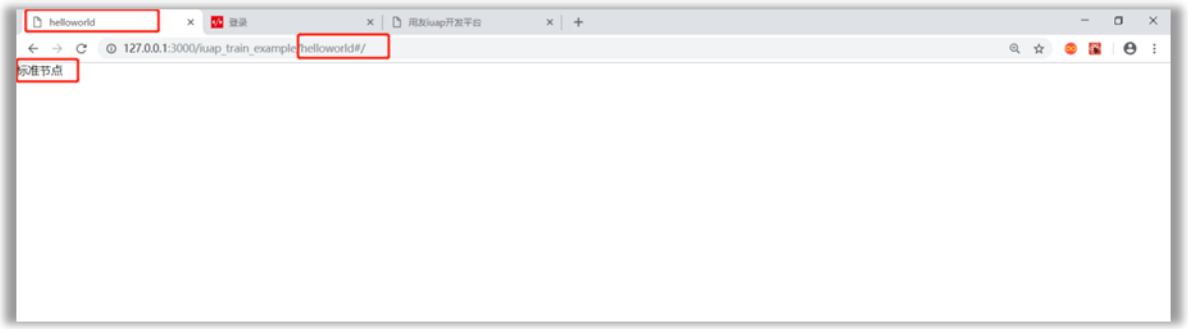

图 16：路由修改验证

3.3.2 视图 UI 层

（1）js 的创建

编辑 Hello 组件中的 index.js 文件，添加标黄部分代码如下：

```
import React, { Component } from 'react';
import { actions } from "mirrorx";
import {Button} from 'tinper-bee';

import './index.less'

class Hello extends Component {
    constructor(props) {
        super(props);
        this.state = {    };
    }

    handleClick = async () => {
        await actions. helloworld.loadData();
        alert(this.props. helloMsg);
    }

    render() {
        return (
            <div>
                <Button  className="mt20  ml20"  colors="primary"
onClick={ this.handleClick }>点击测试</Button>
```

```
                </div>
            );
        }
}

export default Hello;
```

（2）样式的添加

index.less 中的添加按钮间距，代码如下

```
.mt20 {
    margin-top: 20px;
}

.ml20 {
    margin-left: 20px;
}
```

3.3.3 数据模型层

在节点的 model.js 中定义数据模型，这里的数据模型为组件共享的状态及方法。

在 model.js 中调用请求并更新 state 状态，添加如下标黄部分

定义 mode 名及服务调用方法

```
import {actions} from "mirrorx";
// 引入 services，如不需要接口请求可不写
import * as api from "./service";

import { processData } from 'utils';

export default {
    // 确定 Store 中的数据模型作用域
    name: "helloworld",
    // 设置当前 Model 所需的初始化 state
```

```
    initialState: {
        helloMsg:'',
    },
    reducers: {
        /**
         * 纯函数，相当于 Redux 中的 Reducer，只负责对数据的更新。
         * @param {*} state
         * @param {*} data
         */
        updateState(state, data) { //更新 state
            return {
                ...state,
                ...data
            };
        }
    },
    effects: {
        /**
         * 按钮测试数据
         * @param {*} param
         * @param {*} getState
         */
        async loadData(param, getState) {

            let res = processData(await api.getData(param));
            console.log("res",res);
            if (res) {
                actions.helloworld.updateState({
                    helloMsg: res.content[0].name
                });
            }
        },

    }
};
```

3.3.4 服务请求层

服务请求层位于 service.js 中，在 service.js 中定义如下请求，请求中引入了通用工具层中的 request 中封装的函数

```
import request from "utils/request";

//定义接口地址
const URL = {
    "GET_DATA":  `${GROBAL_HTTP_CTX}/hello_world/list`

}

export const getData = (params) => {

    return request(URL.GET_DATA, {
        method: "get",
        param: params
    });
}
```

图 17：定义 URL 及请求方法

utils/request 地址如下：

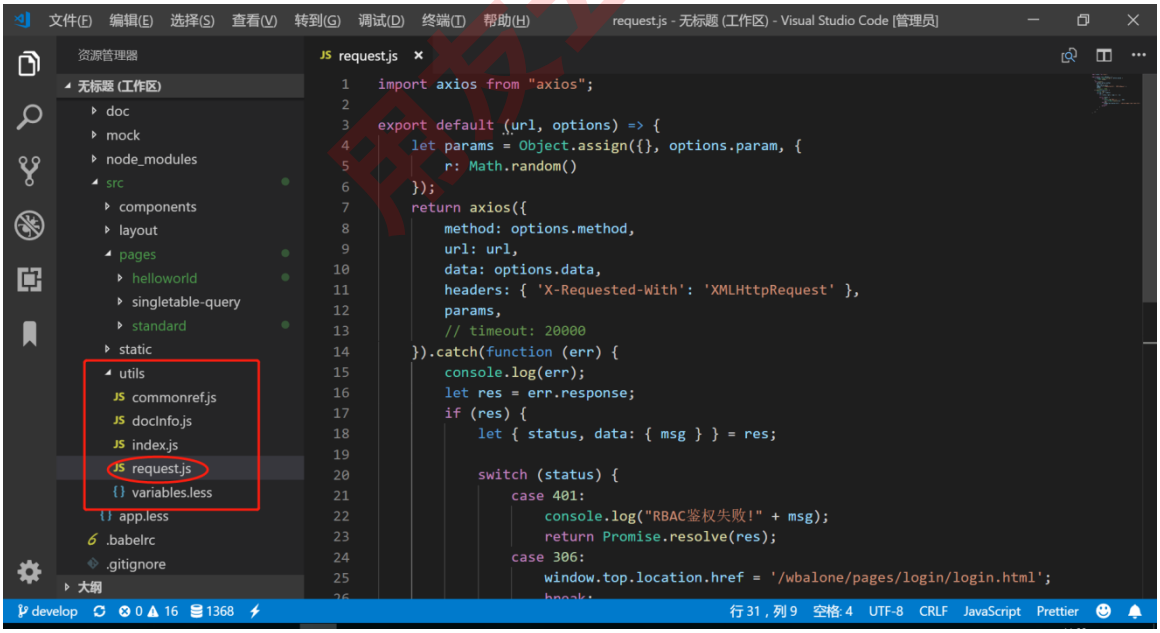

图 18：request 请求封装

3.4 运行调试

3.4.1 安装插件

在下发的插件中包括 React-Developer-Tools.crx、Redux DevTools，自己下载的话，chrome 浏览器下载.crx 后缀的文件，火狐下载.xpi 后缀的文件。

在 Chrome 浏览器中访问 chrome://extensions/

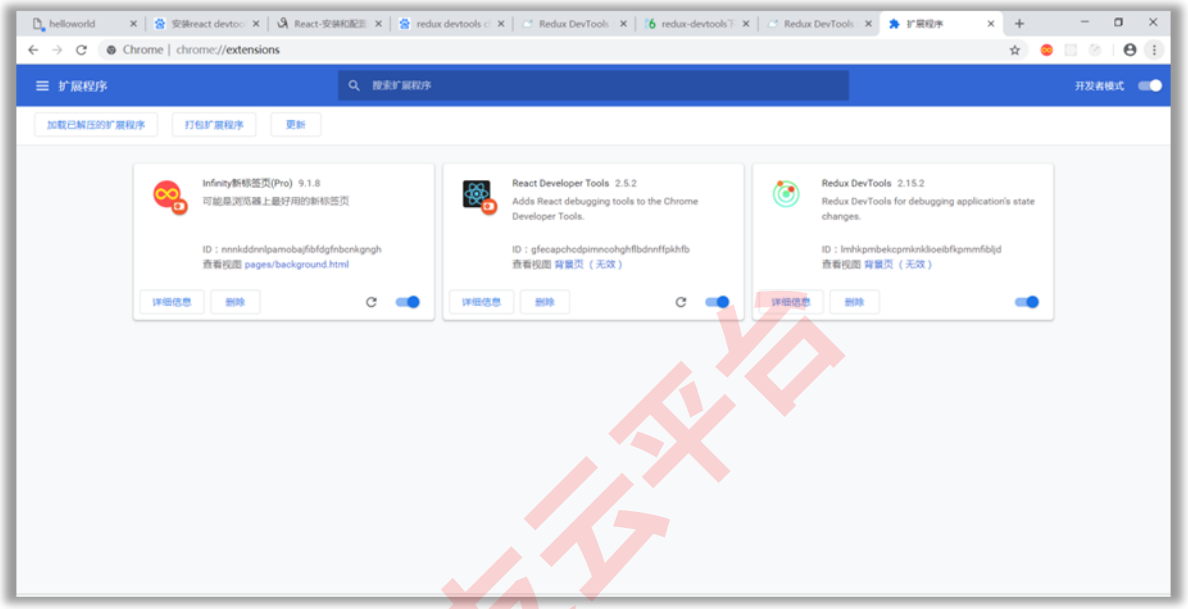

图 19：chrome 扩展程序页面

然后将插件拖动到扩展页面进行安装。

3.4.2 断点调试

按 F12 或者 Fn+F12 打开调试页面，如果当前页面为 react 页面，会出现 react 插件调试页面。

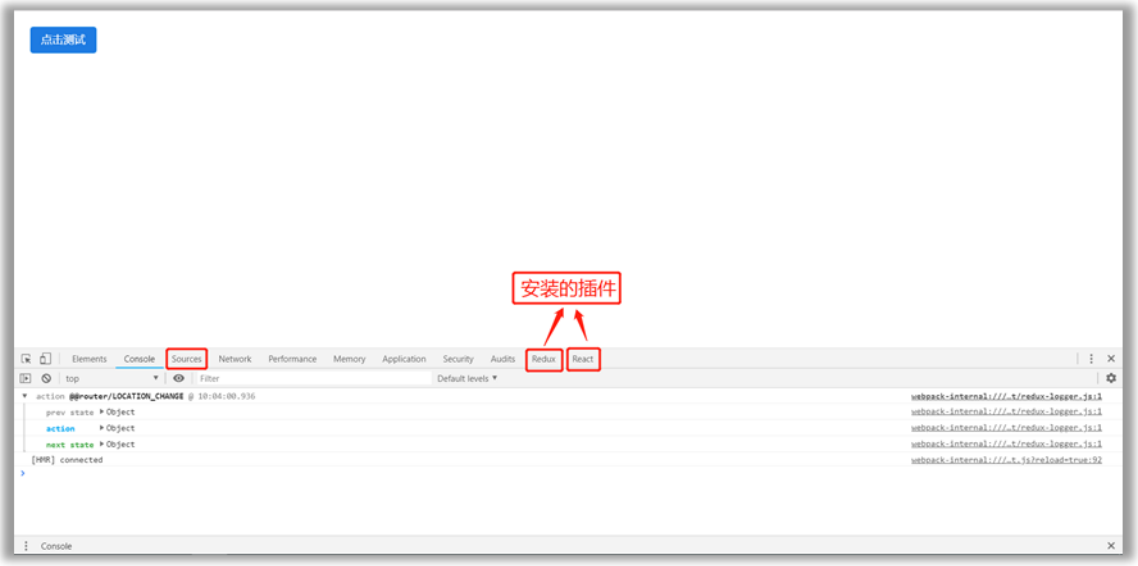

图 20：插件选项

(1) sources 页签

sources 页签下可找到页面源码

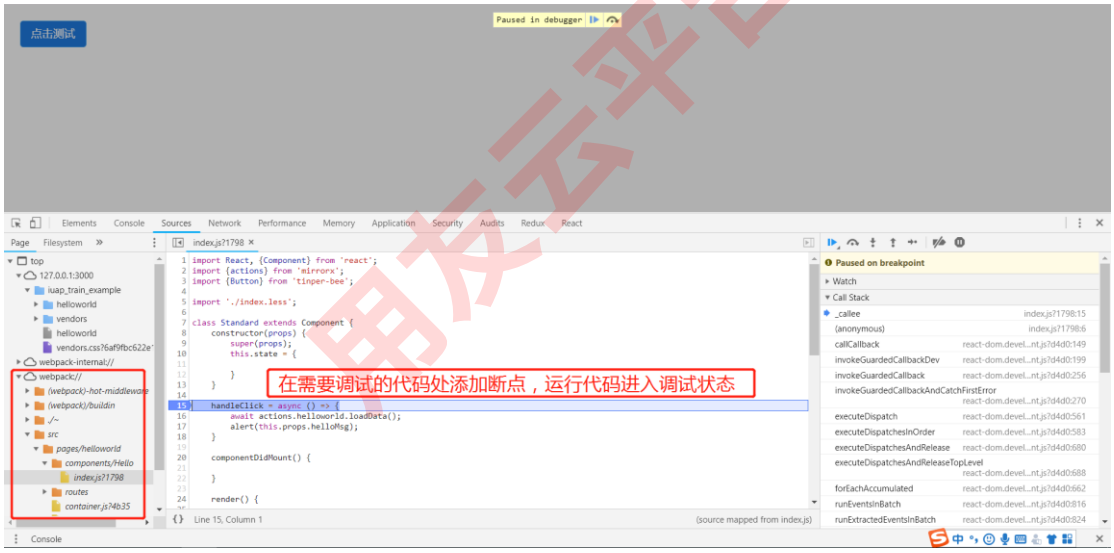

图 21：sources 页面调试

(2) redux 调试

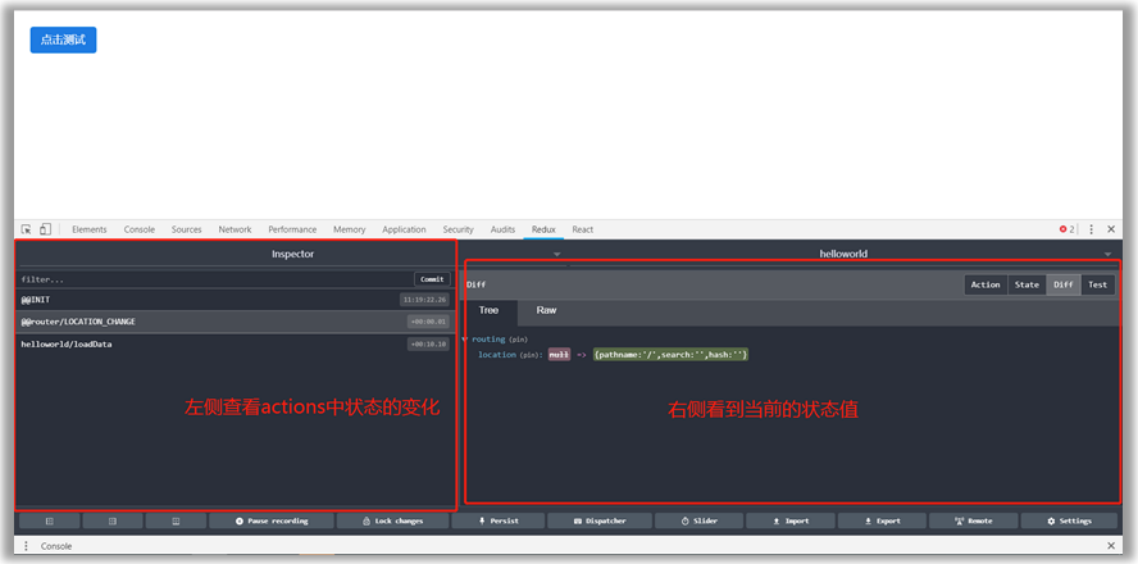

图 22：redux 调试

(3) react 页签

点击这个标签就可以看到当前应用的结构，过 React Developer Tools 我们可以很方便地看到各个组件之间的嵌套关系以及每个组件的事件、属性、状态等信息。右侧显示组件的 props 和 state

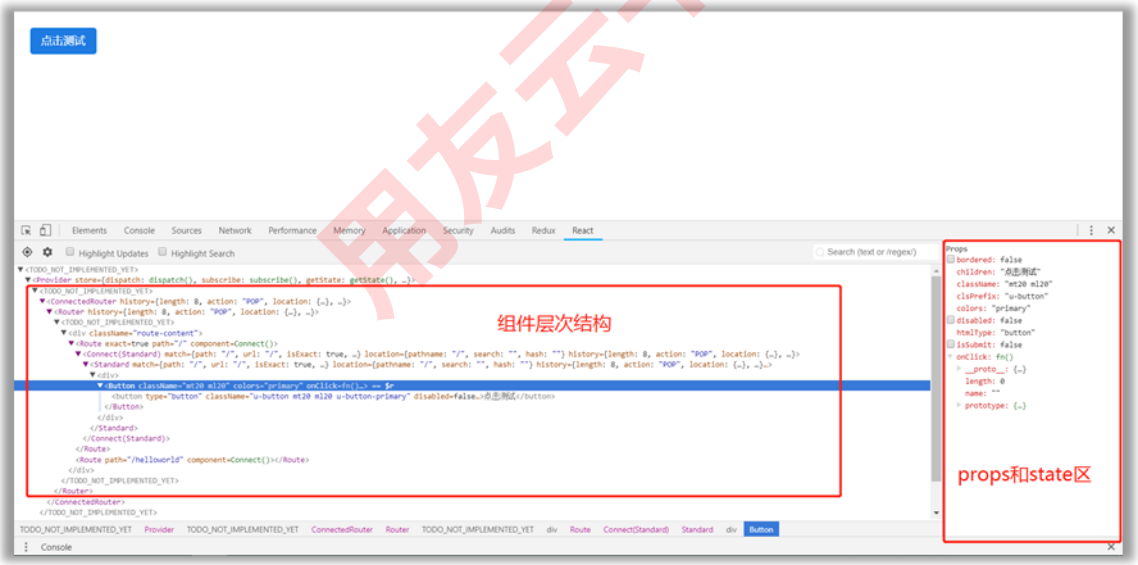

图 23：react dev tools 调试

4 服务器端开发

课程目标：学习后台基础代码结构、了解持久层代码开发过程。

4.1 后端项目准备

后台项目为标准的 maven 项目工程

图 24：项目结构

项目关键配置项解析如下

1、 pom.xml

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>com.yonyou.iuap.pap.base</groupId>
            <artifactId>pap_base_comp_bom</artifactId>
            <version>${pap_base_bom.version}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
```

管理应用平台及开发框架依赖包版本

```
<dependency>
    <groupId>com.yonyou.iuap.pap</groupId>
    <artifactId>iuap-pap-starter</artifactId>
</dependency>
```

配置应用平台及开发框架依赖包

2、 项目基本配置 web.xml

iuap 平台集成了 Spring 框架进行组件的配置和管理，以及 Spring MVC 作为后端 MVC 框架，更方便业务开发者进行开发使用。

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance
                http://www.springmodules.org/schema/cache/springmodules-cache.xsd
                http://www.springmodules.org/schema/cache/springmodules-ehcache.xsd"
         xmlns="http://java.sun.com/xml/ns/javaee"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"

    metadata-complete="true" version="3.0">
    <display-name>iuap_pap_quickstart</display-name>
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>
            classpath*:/applicationContext.xml,
            classpath*:/applicationContext-cache.xml,
            classpath*:/applicationContext-persistence.xml,
            classpath*:/applicationContext-shiro.xml,
            classpath*:/applicationContext-http.xml
        </param-value>
    </context-param>
```

主Spring配置文件

按业务功能划分的配置文件

Druid 连接池启用 web 监控统计功能

```
<!-- 连接池 启用Web 监控统计功能-->
<filter>
    <filter-name>DruidWebStatFilter</filter-name>
    <filter-class>com.alibaba.druid.support.http.WebStatFilter</filter-class>
    <init-param>
        <param-name>exclusions</param-name >
        <param-value>*.js,*.gif,*.jpg,*.png,*.css,*.ico,/druid/*</param-value>
    </init-param>
    <init-param>
        <param-name>profileEnable</param-name>
        <param-value>true</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>DruidWebStatFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
<servlet>
    <servlet-name>DruidStatView</servlet-name>
    <servlet-class>com.alibaba.druid.support.http.StatViewServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>DruidStatView</servlet-name>
    <url-pattern>/druid/*</url-pattern>
</servlet-mapping>
```

3、 Spring 集成 applicationContext.xml

```
<bean id="propertyConfigurer"
    class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
    <property name="locations">
        <list>
            <value>classpath*:/application.properties</value>
        </list>
    </property>
    <property name="systemPropertiesMode" value="2"></property>
</bean>
```

加载配置信息

配置注解扫描包位置

```
<!-- 使用annotation 自动注册bean, 并保证@Required、@Autowired的属性被注入 -->
<context:component-scan
    base-package="com.yonyou.iuap">
    <context:exclude-filter type="annotation"
        expression="org.springframework.stereotype.Controller" />
    <context:exclude-filter type="annotation"
        expression="org.springframework.web.bind.annotation.ControllerAdvice" />
</context:component-scan>
```

4、 Spring-mvc 集成配置文件

配置 Controller 的扫描路径，配置的服务才可以被访问

```
<!-- 自动扫描且只扫描@Controller -->
<context:component-scan base-package="com.yonyou.iuap,com.yonyou.uap.ieop"
    use-default-filters="false">
    <context:include-filter type="annotation"
        expression="org.springframework.stereotype.Controller" />
    <context:include-filter type="annotation"
        expression="org.springframework.web.bind.annotation.ControllerAdvice" />
</context:component-scan>
```

5、 持久层配置文件 applicationContext-persistence.xml

```
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="dataSource"/>
    <property name="configLocation" value="classpath:/sqlMapConfig.xml"></property>
--      自动扫描entity目录, 省掉Configuration.xml里的手工配置 -->
    <property name="typeAliasesPackage" value="com.yonyou.iuap.**.entity"/>
--      显式指定Mapper文件位置 -->
    <property name="mapperLocations">
--          切换数据库类型后，需要修改此处的配置文件，使用对应的数据库类型下的mapper文件 -->
        <array>
            <value>classpath*:/mybatis/oracle/*.xml</value>
            <!-- 示例中用到 -->
            <value>classpath*:/mybatis/mssql/*.xml</value>
            <!-- pap_base_comp_ref-jar中用到 -->
            <value>classpath*:/mybatis/mysql/*.xml</value>
            <!-- baseservice-jar中用到 -->
            <value>classpath*:/mybatis/sql/*.xml</value>
            <value>classpath*:/mybatis/sql/**/**/*.xml</value>
            <value>classpath*:/mybatis/sql/**/*.xml</value>
        </array>
    </property>
--      如果想用iuap的分页插件，可以放开下面注释，oracle或mysql，postgresql数据库则使用下面配置， -->
    <property name="plugins">
        <array>
            <bean id="paginationInterceptor"
                  class="com.yonyou.iuap.mybatis.plugins.PaginationInterceptor">
                <property name="properties">
                    <props>
--                      修改数据库类型后，dbms的属性需要修改，如oracle、postgresql、mybatis -->
                        <prop key="dbms">${jdbc.type}</prop>
                        <prop key="sqlRegex">.*selectAllByPage</prop>
                    </props>
                </property>
            </bean>
        </array>
```

entity扫描包

mapper文件位置

iuap分页插件

持久层 mapper 扫描包路径

```
<!-- 扫描basePackage下所有以@MyBatisRepository标识的 接口-->
<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
    <property name="basePackage" value="com.yonyou.iuap.**.dao"/>
    <property name="sqlSessionFactoryBeanName" value="sqlSessionFactory"/>
</bean>
```

4.2 Helloworld 后台开发思路

后端代码分为几层：实体、DAO、Service、Controller，下面的中描述了手工编写后台代码时的开发步骤。

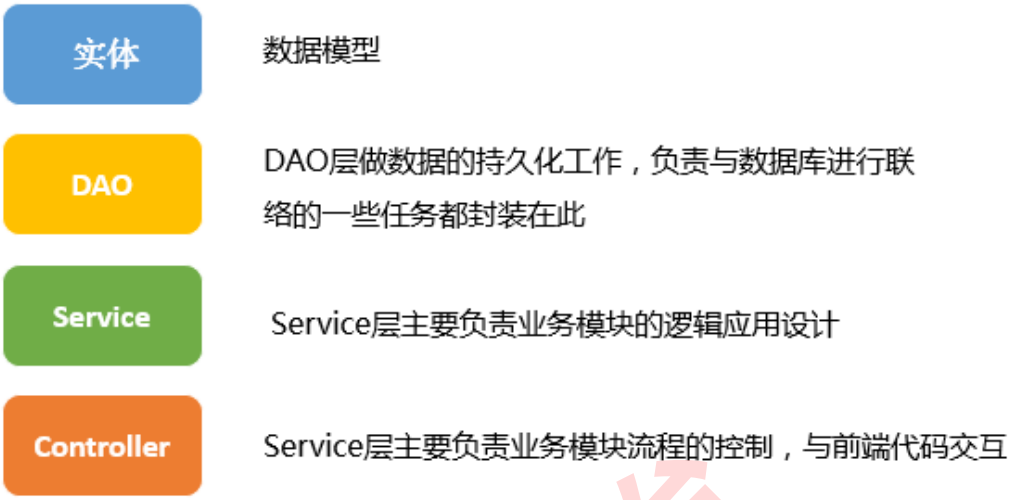

图 25：后端开发代码结构

业务分析：

1、 Helloworld 节点只需要需要实现查询功能，查询功能及字段简单，不需要其他特性。

2、 需要的后台查询服务 URL 以及前端返回格式如下所示：

前端所需的后台服务：实现 get 接口/iuap_pap_quickstart/hello_world/list

输入参数为空

返回结果是：

```
    "success": "success",
    "message": null,
    "detailMsg": {
        "data": {
            "content": [
                {
                    "id": "111",
                    "createTime": null,
                    "createUser": null,
                    "lastModified": null,
                    "lastModifyUser": null,
                    "ts": null,
                    "newTs": "2018-07-28 18:11:53 722",
                    "dr": 0,
                    "name": "HelloWorld!"
                }
            ],
            "last": true,
            "totalPages": 1,
            "totalElements": 1,
            "firstPage": true,
            "lastPage": true,
            "number": 0,
            "size": 10,
            "sort": null,
            "numberOfElements": 1,
            "first": true
        }
    }
}
```


3、代码结构

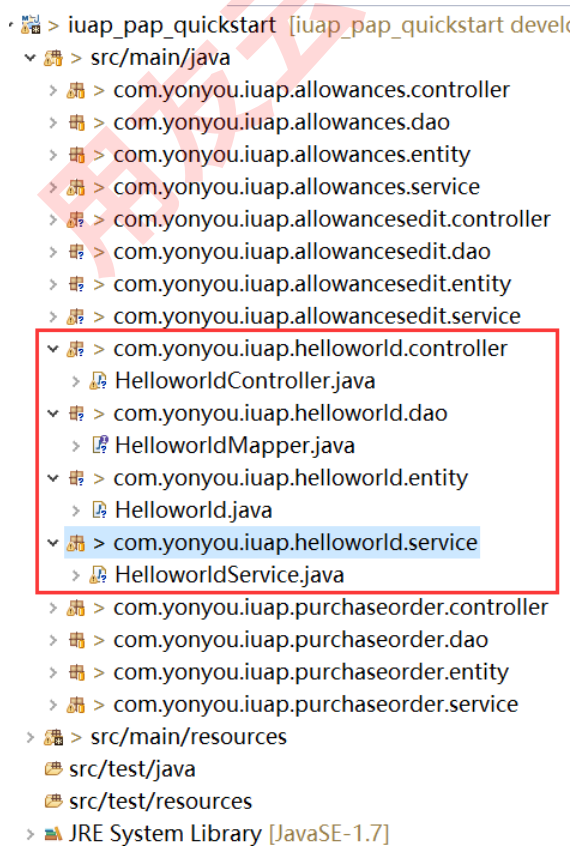

图 26：Helloworld 后台代码结构

4、 后台代码类图

helloworld 只需要实现简单查询，不需要其他的特性扩展，所以后台代码继承的 baseservice 框架类图关系如下：

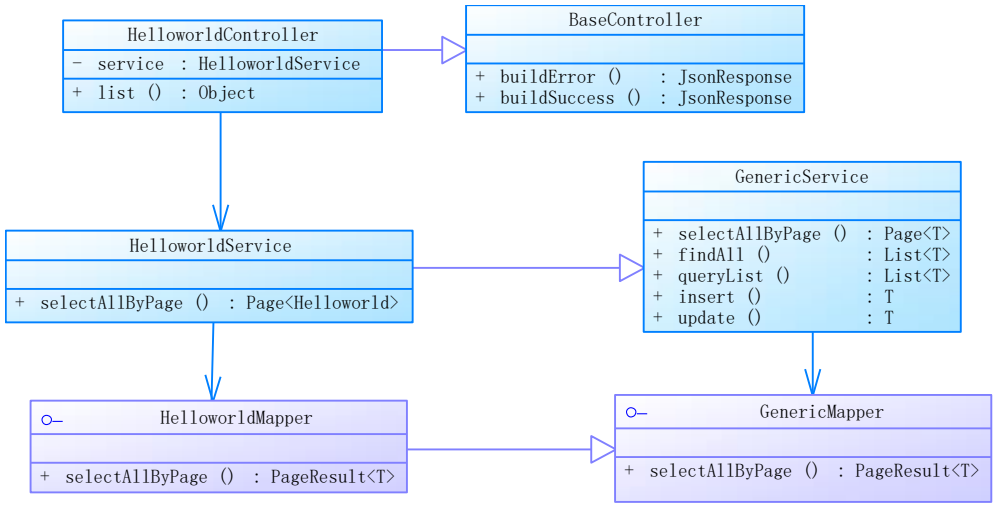

图 27：baseservice 框架类图

4.3 Helloworld 开发步骤

4.3.1 创建数据库表

数据库中已有 iuapd_helloworld 表格，包含的字段有 id、name，其他字段参照建表规范创建

```
CREATE TABLE [walsindemo].[IUAPD_HELLOWORLD] (
[ID] varchar(64) NOT NULL ,
[CODE] varchar(50) NULL ,
[NAME] varchar(50) NULL ,
[CREATE_TIME] varchar(64) NULL ,
[CREATE_USER] varchar(64) NULL ,
[LAST_MODIFIED] varchar(64) NULL ,
[LAST_MODIFY_USER] varchar(64) NULL ,
[TS] varchar(64) NULL ,
[DR] decimal(11) NULL
)
```

图 28：iuapd_helloworld 数据库表

4.3.2 创建实体层 entity

entity 层存放的是数据对象，创建 Helloworld 是对应 iuapd_helloworld 单表的实体，该实体对象只有两个属性 id、name，对象需要继承基类 AbsDrModel

```
package com.yonyou.iuap.helloworld.entity;

import java.io.Serializable;
import javax.persistence.Column;
import javax.persistence.Id;
import javax.persistence.Table;
import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import com.yonyou.iuap.baseservice.entity.AbsDrModel;

@JsonIgnoreProperties(ignoreUnknown = true)
@Table(name = "iuapd_helloworld")
public class Helloworld extends AbsDrModel {

    @Id
    @Column(name="id")
    protected String id;//ID

    @Override
    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    @Override
    public void setId(Serializable id){
        this.id= id.toString();
        super.id = id;
    }
```

```
    @Column(name="name")
    private String name; //展示数据

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }

}
```

【注解说明】：

1.@JsonIgnoreProperties(ignoreUnknown = true)

当 json 反序列化到对象中时，如果 json 中缺少对象中的属性时，忽略缺少的属性。

2.@Table(name = " iuapd_helloworld ")

这个注解是表示声明此对象映射到数据库的数据表，通过它可以为实体指定表(talbe)的名字。

3.@Id 声明此属性为主键

5.@Column(name="id")声明该属性与数据库字段的映射关系

4.3.3 创建持久层 dao

在 HelloworldMapper 接口中只需要继承父类接口 GenericMapper 即可，同时 HelloworldMapper 需要添加@MyBatisRepository 注解。

```
package com.yonyou.iuap.helloworld.dao;

import com.yonyou.iuap.baseservice.persistence.mybatis.mapper.GenericMapper;
import com.yonyou.iuap.helloworld.entity.Helloworld;
import com.yonyou.iuap.mybatis.anotation.MyBatisRepository;



@MyBatisRepository
public interface HelloworldMapper extends GenericMapper<Helloworld> {
```

```
}
```

GenericMapper 接口主要实现是增删改查方法

```
public abstract interface GenericMapper<T extends Model>
{
  @MethodMapper(type=SqlCommandType.SELECT)
  public abstract PageResult<T> selectAllByPage(@Param("page") PageRequest paramPageReq

  @MethodMapper(type=SqlCommandType.SELECT)
  public abstract List<T> queryList(@Param("condition") Map<String, Object> paramMap);

  @MethodMapper(type=SqlCommandType.SELECT)
  public abstract List<Map<String, Object>> queryListByMap(@Param("condition") Map<Stri

  @MethodMapper(type=SqlCommandType.INSERT)
  public abstract int insert(T paramT);

  @MethodMapper(type=SqlCommandType.UPDATE)
  public abstract int update(T paramT);

  @MethodMapper(type=SqlCommandType.DELETE)
  public abstract int delete(@Param("condition") Map<String, Object> paramMap);
}
```

4.3.4 创建服务层 service

创建 HelloworldService 继承 GenericExService，添加@Service 注解，标识此类为 Service 服务层代码；

GenericService 是后台框架提供的通用持久层方法，直接继承调用即可

```
package com.yonyou.iuap.helloworld.service;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import com.yonyou.iuap.baseservice.service.GenericService;
import com.yonyou.iuap.helloworld.dao.HelloworldMapper;
import com.yonyou.iuap.helloworld.entity.Helloworld;


@Service
public class HelloworldService extends GenericService<Helloworld>{

    private HelloworldMapper helloworldMapper;

    @Autowired
    public void setHelloworldMapper(HelloworldMapper helloworldMapper) {
        this.helloworldMapper = helloworldMapper;
        super.setGenericMapper(helloworldMapper);
```

```
    }

}
```

@Autowired 注释，它可以对类成员变量、方法及构造函数进行标注，完成自动装配的工作。通过 @Autowired 的使用来消除 set ， get 方法。

@Autowired 可以在变量上注解或者是在 set 方法上注解，写在成员变量上就是 spring 通过读取 xml 配置返回一个 bean；只写在 set 方法上就是 spring 通过读取 xml 配置返回一个 bean，然后将其注入你的成员变量

这里通过@Autowired 标签在 set 方法注入 helloworldMapper 对象。

4.3.5 创建 web 层 controller

1、 创建 HelloworldController 继承 com.yonyou.iuap.base.web.BaseController

2、 注入 com.yonyou.iuap.helloworld.service.HelloworldService

3、 实现 list（获取单表数据）方法，调用 Service 的方法并返回数据。返回的数据调用 buildSuccess 方法返回

```
package com.yonyou.iuap.helloworld.controller;

import java.util.HashMap;
import java.util.Map;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;
import com.yonyou.iuap.base.web.BaseController;
import com.yonyou.iuap.helloworld.entity.Helloworld;
import com.yonyou.iuap.helloworld.service.HelloworldService;
import com.yonyou.iuap.mvc.type.SearchParams;

@Controller
@RequestMapping(value="/hello_world")
```

```
public class HelloworldController extends BaseController{

    private Logger logger = LoggerFactory.getLogger(HelloworldController.class);

    private HelloworldService service;

    @Autowired
    public void setService(HelloworldService service) {
        this.service = service;
    }

    @RequestMapping({"/list"})
    @ResponseBody
    public Object list(PageRequest pageRequest, SearchParams searchParams) {
        Page<Helloworld> page = this.service.selectAllByPage(pageRequest, searchParams);
        return buildSuccess(page);
    }
}
```

关于 Controller 类上注解说明如下

@Controller 标识服务控制类

@RequestMapping(value="/list")配置地址映射

4.3.6 创建 sql 映射文件

在 spring 配置文件中 applicationContext-persistence.xml，配置扫描 dao 层注解，增删改查 sql 会在启动服务的时候就会自动生成。sql 只需要写 selectAllByPage，Helloworld 中字段简单，可以不用复写分页方法。直接使用框架自动生成的代码。

HelloworldMapper.xml 文件

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd" >
<mapper namespace="com.yonyou.iuap.helloworld.dao.HelloworldMapper">

  <resultMap id="BaseResultMap" type="com.yonyou.iuap.helloworld.entity.Helloworld">
```

```
        <id column="id" jdbcType="VARCHAR" property="id" />
        <result column="name"  property="name" />
        <result column="dr" property="dr" />
        <result column="ts" property="ts" />
        <result column="last_modified" property="lastModified" />
        <result column="last_modify_user" property="lastModifyUser" />
        <result column="create_time" property="createTime" />
        <result column="create_user" property="createUser" />
    </resultMap>

</mapper>
```

SQL 常见标签

1) mapper 持久层类
2) resultMap：映射管理器，是 Mybatis 中最强大的工具，使用其可以进行实体类之间的关系，并管理结果和实体类间的映射关系。

   需要配置的属性：<resultMap id=" " type=" "></resutlMap>

   - id=" ">>>表示这个映射管理器的唯一标识，外部通过该值引用；
   - type = " ">>> 表示需要映射的实体类；

```
<mapper namespace="com.yonyou.iuap.helloworld.dao.HelloworldMapper">

    <resultMap id="BaseResultMap" type="com.yonyou.iuap.helloworld.entity.Helloworld">
        <id column="id" jdbcType="VARCHAR" property="id" />
        <result column="name"  property="name" />
        <result column="dr" property="dr" />
        <result column="ts" property="ts" />
        <result column="last_modified" property="lastModified" />
        <result column="last_modify_user" property="lastModifyUser" />
        <result column="create_time" property="createTime" />
        <result column="create_user" property="createUser" />
    </resultMap>
```

主键配置： <id column = " " property= " " />

- <id>标签指的是：结果集中结果唯一的列【column】 和 实体属性【property】的映射关系，注意：<id>标签管理的列未必是主键列，需要根据具体需求指定；

普通列值配置： <result column= " " property=" " />

- <result>标签指的是：结果集中普通列【column】 和 实体属性【property】的映射关系；

5 应用效果

此时已经开发完毕，启动前后台项目的服务后，访问

http://127.0.0.1:3000/iuap_train_example/helloworld#/

点击「点击测试」按钮弹出 hello world 提示。

图 29：helloworld 应用效果