JavaScript速通

使用 window.alert() 写入警告框

window.alert("Hello JavaScript"); //浏览器弹出警告框

使用 document.write() 写入 HTML 输出

document.write("Hello JavaScript"); //写入HTML,在浏览器展示

使用 console.log() 写入浏览器控制台

console.log("Hello JavaScript"); //写入浏览器控制台

var：声明变量，全局作用域/函数作用域，允许重复声明

let：声明变量，块级作用域，不允许重复声明

const：声明常量，一旦声明，常量的值不能改变

== 会进行类型转换，=== 不会进行类型转换

var arr = new Array(1,2,3,4);

|  |  |
| --- | --- |
| forEach() | 遍历数组中的每个有值的元素，并调用一次传入的函数 |
| push() | 将新元素添加到数组的末尾，并返回新的长度。 |
| splice() | 从数组中删除元素。 |

JSON

var user = {

    name:"Tom",

    age:20,

    gender:"male",

    eat: function(){

        alert("用膳~");

    }

};

history：对 History 对象的只读引用。

location：用于窗口或框架的 Location 对象

navigator：对 Navigator 对象的只读引用

alert()：显示带有一段消息和一个确认按钮的警告框

confirm()：显示带有一段消息以及确认按钮和取消按钮的对话框

setInterval()：按照指定的周期（以毫秒计）来调用函数或计算表达式

setTimeout()：在指定的毫秒数后调用函数或计算表达式

根据id属性值获取，返回单个Element对象

var h1 = document.getElementById('h1');

根据标签名称获取，返回Element对象数组

var divs = document.getElementsByTagName('div');

根据name属性值获取，返回Element对象数组

var hobbys = document.getElementsByName('hobby');

根据class属性值获取，返回Element对象数组

|  |  |
| --- | --- |
| onclick | 鼠标单击事件 |
| onblur | 元素失去焦点 |
| onfocus | 元素获得焦点 |
| onload | 某个页面或图像被完成加载 |
| onsubmit | 当表单提交时触发该事件 |
| onkeydown | 某个键盘的键被按下 |
| onmouseover | 鼠标被移到某元素之上 |
| onmouseout | 鼠标从某元素移开 |

var clss = document.getElementsByClassName('cls');

通过 HTML标签中的事件属性进行绑定

函数事件绑定

<input type="button" onclick="on()" value="按钮1">

<script>

    function on(){

        alert('我被点击了!');

    }

</script>

通过 DOM 元素属性绑定

<input type="button" id="btn" value="按钮2">

<script>

    document.getElementById('btn').onclick=function(){

        alert('我被点击了!');

    }

</script>

Vue

<script>

  new Vue({

    el: "#app",

     data: {

        url: "https://www.itcast.cn"

     }

  })

</script>

|  |  |
| --- | --- |
| v-bind | 为HTML标签绑定属性值，如设置 href , css样式等 |

<a v-bind:href="url">传智教育</a> <a :href="url">传智教育</a>

|  |  |
| --- | --- |
| v-model | 在表单元素上创建双向数据绑定 |

<input type="text" v-model="url"> //url的数据通过表单进行修改，//那前面绑定的数据也会跟着改,改的是js上的url

<script>

    new Vue({

        el: "#app",

        data: {

//...

        },

        methods: {

            handle:function(){

                alert('我被点击了');

            }

        },

    })

</script>

|  |  |
| --- | --- |
| v-on | 为HTML标签绑定事件 |

<input type="button" value="按钮" v-on:click="handle()">

<input type="button" value="按钮" @click="handle()">

click就是用来替换的

|  |  |
| --- | --- |
| v-if | 条件性的渲染某元素，判定为true时渲染,否则不渲染 |
| v-else-if |
| v-else |

年龄{{age}},经判定为:

<span v-if="age <= 35">年轻人</span>

<span v-else-if="age > 35 && age < 60">中年人</span>

<span v-else>老年人</span>

|  |  |
| --- | --- |
| v-show | 根据条件展示某元素，区别在于切换的是display属性的值 |

年龄{{age}},经判定为:

<span v-show="age <= 35">年轻人</span>

data: {

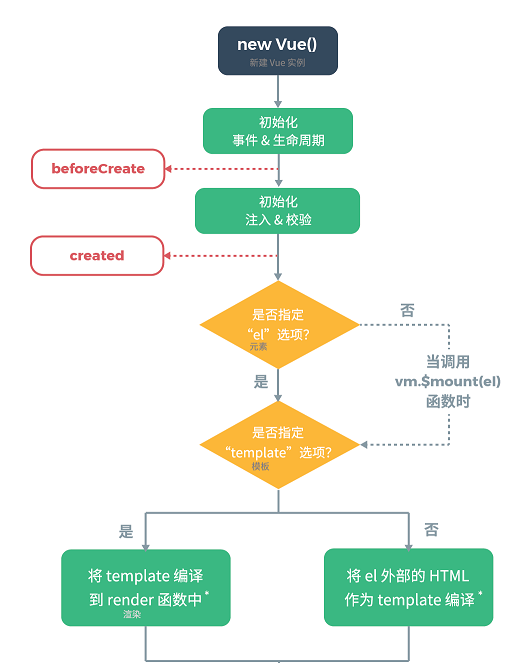
addrs: ['北京','上海','广州','深圳','成都','杭州']

},

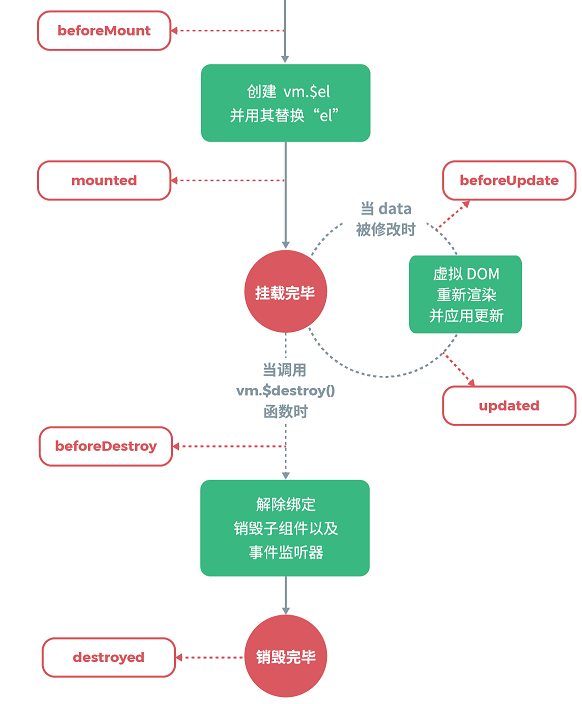
|  |  |
| --- | --- |
| v-for | 列表渲染，遍历容器的元素或者对象的属性 |

<div v-for="addr in addrs">{{addr}}</div>

<div v-for="(addr,index) in addrs">{{index + 1}} : {{addr}}</div>



[[



Axios

axios({

}).then(result => {

})

1.引入Axios的js文件

<script src="js/axios-0.18.0.js"></script>

1. 使用Axios发送请求，并获取响应结果

axios({

    method: "get",

url: "http://yapi.smart-xwork.cn/mock/169327/emp/list"

//在后面直接加参数？那种形式的，下面是成功回调函数

}).then((result) => {

    console.log(result.data);

});

axios({

    method: "post",

    url: "http://yapi.smart-xwork.cn/mock/169327/emp/deleteById",

data: "id=1"

//下面是成功回调函数

}).then((result) => {

    console.log(result.data);

});

在vue中用axios要导包

在当前项目下npm install axios

import axios from 'axios' 直接写在script里就行

import axios from 'axios'

mounted() {

axios.get("http://yapi.smart-xwork.cn/mock/169327/emp/list").then((result) => {

this.tableData = result.data.data;

});

}

Vue-Element

初始化vue项目:vue create vue-project 查看图形化界面:vue ui

装插件:npm i element-ui -S

应该有”template”,”script”,”style”三块

<template>

//模板区，cv就完了

</template>

<script>

//JS区，控制模板的数据来源和行为

//样式区，用以补属性

import axios from 'axios' //导包的

export default { //传值的

data() {

return {

tableData: [],

searchForm: {

name: "",

gender: ""

}

}

},

methods: { //定义函数的，和上面cv区联动

onsubmit: function() {

alert("提交信息")

}

},

mounted() {

axios.get("http://yapi.smart-xwork.cn/mock/169327/emp/list").then((result) => {

this.tableData = result.data.data;

});

}

}

</script>

<style>

</style>

修改端口(vue.config.js) 配置文件

const {

defineConfig

} = require('@vue/cli-service')

module.exports = defineConfig({

transpileDependencies: true,

devServer: {

port: 8000,//修改接口

}

})

引入ElementUI组件库{(main.js) [入口文件，初始化Vue实例并使用需要的插件](https://blog.csdn.net/m0_62823653/article/details/124865264" \t "https://www.bing.com/_blank)}

import ElementUI from 'element-ui';

import 'element-ui/lib/theme-chalk/index.css';

Vue.use(ElementUI);

Vue路由(说白了就是路径)

记得检查router是否存在(npm install [vue-router@3.5.1)](mailto:vue-router@3.5.1))

VueRouter：路由器类，根据路由请求在路由视图中动态渲染选中的组件

<router-link>：请求链接组件，浏览器会解析成<a>

<router-view>：动态视图组件，用来渲染展示与路由路径对应的组件

router下的js

import Vue from 'vue'

import VueRouter from 'vue-router'

Vue.use(VueRouter)

const routes = [{

path: '/emp', //表示路由路径

name: 'emp', //表示路由名称

component: () => import('../views/tilas/EmpView.vue')

//表示路由组件,import导入的路径要../开头

},

{

path: '/',

redirect: '/dept’ //弄个重定向

}

]

//创建Vue Router实例，传入routes参数，创建一个路由实例

const router = new VueRouter({

routes

})

//导出路由实例，让其他组件可以使用该路由实例进行路由跳转。

export default router

页面中修改(以性别从0，1变为男女为例)

<el-table-column label="性别" width="140">

<template slot-scope="scope">

<!-- slot-scope 用于定义插槽的作用域 -->

<!-- scope包含了当前行数据的信息，包括了该行中每个字段的值 -->

<!-- scope.row 表示当前行的数据对象，可以通过 scope.row 访问到当前行中的所有字段 -->

{{scope.row.gender===1?'男':'女'}}

</template>

</el-table-column>

App.vue内

<template>

<div>

<!--一个页面就这么写，跟你命名的一个样-->

<element-view></element-view>

<!--用于直接导入路由-->

<router-view></router-view>

</div>

</template>

Maven

添加依赖 //https://mvnrepository.com

<dependencies>

<dependency>  
 在里面写

</dependency>  
</dependencies>

依赖的传递性

直接依赖:在当前项目中通过依赖配置建立的依赖关系

间接依赖:被依赖的资源如果依赖其他资源，当前项目间接依赖其他资源

排除依赖:排除依赖指主动断开依赖的资源，被排除的资源无需指定版本。

<dependency>  
 <groupId>com.itheima</groupId>  
 <artifactId>maven-projectB</artifactId>  
 <version>1.0-SNAPSHOT</version>

<!--一去除版本-->  
 <exclusions>  
 <exclusion>  
 <groupId>junit</groupId>  
 <artifactId>junit</artifactId>  
 </exclusion>  
 </exclusions>  
</dependency>

生命周期

每套生命周期包含一些阶段，阶段是有顺序的，后面的阶段依赖于前面的阶段

mvn +

清理 clean：移除上一次构建生成的文件

编译 compile：编译项目源代码

测试 test：使用合适的单元测试框架运行测试(junit)

打包 package：将编译后的文件打包，如：jar、war等

安装 install：安装项目到本地仓库

HTTP协议

基于TCP协议：面向连接，安全

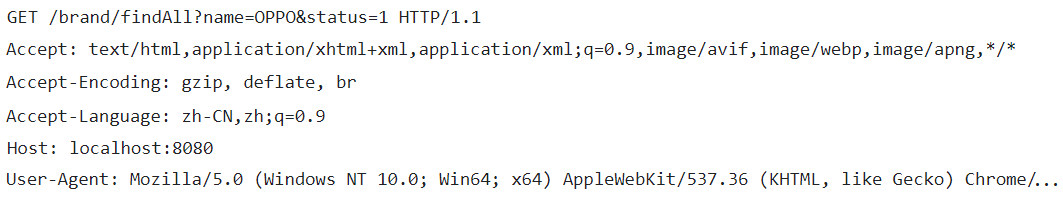
基于请求-响应模型的：一次请求对应一次响应

HTTP协议是无状态的协议：对于事务处理没有记忆能力。每次请求-响应都是独立的。

缺点：多次请求间不能共享数据。

优点：速度快

HTTP-请求格式



请求行：请求数据第一行

(请求方式、资源路径、协议)

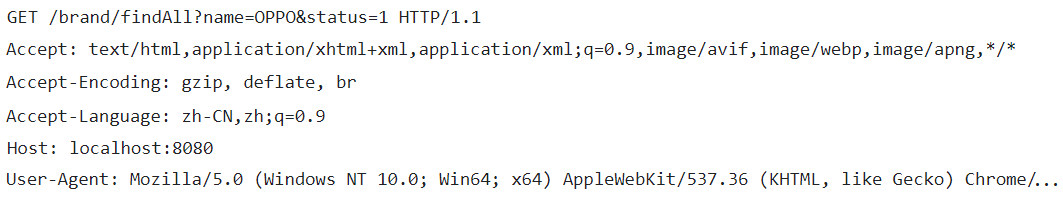
请求头：第二行开始，格式key：value

请求行：请求数据第一行 (请求方式、资源路径、协议)

请求头：第二行开始，格式key：value



|  |  |
| --- | --- |
| Host | 请求的主机名 |
| User-Agent | 浏览器版本，例如Chrome浏览器的标识类似Mozilla/5.0 ... Chrome/79，IE浏览器的标识类似 |
| Accept | 表示浏览器能接收的资源类型，如text/\*，image/\*或者\*/\*表示所有； |
| Accept-Language | 表示浏览器偏好的语言，服务器可以据此返回不同语言的网页； |
| Accept-Encoding | 表示浏览器可以支持的压缩类型，例如gzip, deflate等。 |
| Content-Type | 请求主体的数据类型。 |
| Content-Length | 请求主体的大小（单位：字节1）。 |



请求体：POST请求，存放请求参数

请求头：第二行开始，格式key：value

请求行：请求数据第一行

(请求方式、资源路径、协议)



请求方式-GET: 请求参数在请求行中，没有请求体，如：/brand/findAll?name=OPPO&status=1。GET请求大小是有限制的。

请求方式-POST: 请求参数在请求体中，POST请求大小是没有限制的。

HTTP-响应协议

响应头：第二行开始，格式key：value

响应行：响应数据第一行(协议、状态码、描述)

响应体：最后一部分，存放响应数据

HTTP-协议解析

感觉没啥卵用

请求响应：

请求（HttpServletRequest）：获取请求数据

响应（HttpServletResponse）：设置响应数据

BS架构：Browser/Server，浏览器/服务器架构模式。客户端只需要浏览器，应用程序的逻辑和数据都存储在服务端。

CS架构：Client/Server，客户端/服务器架构模式。

请求

获取请求参数

<http://localhost:8080/simpleParam?name=Tom&age=10>

通过HttpServletRequest 对象手动获取请求参数

@RequestMapping("/simpleParam")  
public String simpleParam(HttpServletRequest request){  
 String name = request.getParameter("name");  
 String ageStr = request.getParameter("age");

int age = Integer.parseInt(ageStr);  
 System.out.println(name+" : "+age);  
 return "OK";  
}

SpringBoot方式

@RequestMapping("/simpleParam")  
public String simpleParam(String name , Integer age){  
 System.out.println(name+" : "+age);  
 return "OK";  
}

@RequestMapping("/simpleParam")

//要是前后端传参数据名字不匹配那就取别名@RequestParam  
public String simpleParam(@RequestParam(name = "name") String username , Integer age){  
 System.out.println(username + " : " + age);  
 return "OK";  
}

PS: @RequestParam中的required属性默认为true，代表该请求参数必须传递，如果不传递将报错。 如果该参数是可选的，可以将required属性设置为false。

目前常用(请求参数名与形参对象属性名相同用POJO类接收)

http://localhost:8080/simplePojo?name=Tom&age=10

简单实体对象：请求参数名与形参对象属性名相同，定义POJO接收即可

public class User {  
 private String name;  
 private Integer age;

}

@RequestMapping("/simplePojo")  
public String simplePojo(User user){  
 System.out.println(user);  
 return "OK";  
}

复杂一点的就按层次结构

http:/localhost:8080/complexPojo?name=Tom&age=10&address.province=beijing&address.city=beijing

public class User { public class Address {  
 private String name; private String province;  
 private Integer age; private String city;  
 private Address address; }  
}

@RequestMapping("/complexPojo")  
public String complexPojo(User user){  
 System.out.println(user);  
 return "OK";  
}

数组集合参数

请求参数名与形参数组名称相同且请求参数为多个，用数组类型形接收参数

<http://localhost:8080/arrayParamPhobby=game&hobby=java>

@RequestMapping("/arrayParam")  
public String arrayParam(String[] hobby){  
 System.out.println(Arrays.toString(hobby));  
 return "OK";  
}

要是要用list接那就@RequestParam

@RequestMapping("/arrayParam")  
public String listParam(@RequestParam List<String> hobby){  
 System.out.println(hobby);  
 return "OK";  
}

日期参数@DateTimeFormat+pattern

http:/localhost:8080/dateParam?updateTime=2022-12-1210:05:45

@RequestMapping("/dateParam")  
public String dateParam(@DateTimeFormat(pattern = "yyyy-MM-dd HH:mm:ss") LocalDateTime updateTime){  
 System.out.println(updateTime);  
 return "OK";  
}

JSON 参数

跟array，list差不多，就多个@RequestBody

路径参数@PathVariable

通过请求URL直接传递参数，使用{…}来标识该路径参数

http://localhost:8080/path/1/Tom

@RequestMapping("/path/{id}/{name}")  
public String pathParam2(@PathVariable Integer id, @PathVariable String name){  
 System.out.println(id+ " : " +name);  
 return "OK";  
}

响应

定义一个JSON类来接收大把的响应

public class Result {

//响应码，1 代表成功; 0 代表失败  
 private Integer code;

//提示信息  
 private String msg;

//返回的数据  
 private Object data;

//…….  
}

三层架构

controller：控制层，接收前端发送的请求，对请求进行处理，并响应数据。

service：业务逻辑层，处理具体的业务逻辑。

dao：数据访问层(Data Access Object)（持久层），负责数据访问操作，包括数据的增、删、改、查。

MySQL

基础语法

SQL语句可以单行或多行书写，以分号结尾。

SQL语句可以使用空格/缩进来增强语句的可读性。

MySQL数据库的SQL语句不区分大小写。

注释：

单行注释：-- 注释内容 或 # 注释内容(MySQL特有)

多行注释： /\* 注释内容 \*/

|  |  |  |
| --- | --- | --- |
| **分类** | **全称** | **说明** |
| DDL | Data Definition Language | 数据定义语言，用来定义数据库对象(数据库，表，字段) |
| DML | Data Manipulation Language | 数据操作语言，用来对数据库表中的数据进行增删改 |
| DQL | Data Query Language | 数据查询语言，用来查询数据库中表的记录 |
| DCL | Data Control Language | 数据控制语言，用来创建数据库用户、控制数据库的访问权限 |

DDL（数据库操作，高级货）

查询所有数据库：show databases; | 使用数据库：use 数据库名 ;

查询当前数据库：select database(); |

创建数据库：create database [ if not exists ] 数据库名 ;

删除数据库：drop database [ if exists ] 数据库名 ;

ps: database替换成 schema 如：create schema db01;(这才常用吧)

建表懂的都懂

create table 表名(

字段1 字段类型 [ 约束 ] [ comment 字段1注释 ] ,

......

字段n 字段类型 [ 约束 ] [ comment 字段n注释 ]

) [ comment 表注释 ] ;

查表 有可视化界面真的有必要么

查询当前数据库所有表：show tables;

查询表结构：desc 表名;

查询建表语句：show create table 表名;

改表 同上

添加字段：alter table 表名 add 字段名 类型(长度) [comment 注释] [约束];

修改字段类型：alter table 表名 modify 字段名 新数据类型(长度);

修改字段名和字段类型：alter table 表名 change 旧字段名 新字段名 类型 (长度) [comment 注释] [约束];

删除字段：alter table 表名 drop column 字段名;

修改表名： rename table 表名 to 新表名;

删表

删除表：drop table [ if exists ] 表名;

增删改查

增 insert

指定字段添加数据：insert into 表名 (字段名1, 字段名2) values (值1, 值2);

全部字段添加数据：insert into 表名 values (值1, 值2, ...);

批量添加数据（指定字段）：insert into 表名 (字段名1, 字段名2) values (值1, 值2), (值1, 值2);

批量添加数据（全部字段）：insert into 表名 values (值1, 值2, ...), (值1, 值2, ...);

Ps：插入数据时，指定的字段顺序需要与值的顺序是一一对应的。

字符串和日期型数据应该包含在引号中。

插入的数据大小，应该在字段的规定范围内。

改 update

修改数据：update 表名 set 字段名1 = 值1 , 字段名2 = 值2 , .... [ where 条件 ] ;

Ps:修改语句的条件可以有，也可以没有，如果没有条件，则会修改整张表的所有数据。

删 delete

删除数据：delete from 表名 [ where 条件 ];

查 select

select

字段列表

from

表名列表

where 条件查询（where）

条件列表

group by 分组查询（group by）

分组字段列表

having

分组后条件列表

order by 排序查询（order by）

排序字段列表

Limit 分页查询（limit）

分页参数

查询多个字段：select 字段1, 字段2, 字段3 from 表名;

查询所有字段（通配符）：select \* from 表名;

设置别名：select 字段1 [ as 别名1 ] , 字段2 [ as 别名2 ] from 表名;

去除重复记录：select distinct 字段列表 from 表名;

条件列表条件补充

|  |  |
| --- | --- |
| between ... and ... | 在某个范围之内(含最小、最大值) |
| in(...) | 在in之后的列表中的值，多选一 |
| like  占位符 | 模糊匹配(\_匹配单个字符, %匹配任意个字符) |
| is null | 是null |

Mybatis

1.SQL建表

2.导坐标(maven)

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis</artifactId>

<version>3.5.5</version>

</dependency>

<!--junit 单元测试-->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13</version>

<scope>test</scope>

</dependency>

<!--slf4j-->

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>1.7.21</version>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-log4j12</artifactId>

<version>1.7.21</version>

</dependency>

3.写mybaties核心配置文件

(mybatis.xml){

<?xml version="1.0" encoding="UTF-8" ?>

<!DOCTYPE configuration

PUBLIC "-//mybatis.org//DTD Config 3.0//EN"

"http://mybatis.org/dtd/mybatis-3-config.dtd">

<configuration>

<environments default="development">

<environment id="development">

<transactionManager type="JDBC"/>

<dataSource type="POOLED">

<property name="driver" value="${driver}"/>

<property name="url" value="${url}"/>

<property name="username" value="${username}"/>

<property name="password" value="${password}"/>

</dataSource>

</environment>

</environments>

<mappers>

<!--加载SQL映射文件 类路径-->

<mapper resource="org/mybatis/example/BlogMapper.xml"/>

</mappers>

</configuration>

}

4.映射SQL文件

//映射文件（UserMapper）{

<?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">

<!--namespace命名空间-->

<mapper namespace="test">

<!--id 取名称 resultType 返回类型 在User类里面就包装成User，填路径-->

<select id="selectBlog" resultType="Blog">

select

\*

from

Blog

</select>

</mapper>

}

5.编码

1.定义POJO类（新开一个User,下面都写在User里）

元素跟SQL里的表名相同//alt+鼠标左 Integer

get+set+toString//快捷生成

2.加载核心配置文件，获取SqlSessinFactory对象

String resource = "org/mybatis/example/mybatis-config.xml";

InputStream inputStream = Resources.getResourceAsStream(resource);

SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);

3.获取SqlSessin对象，执行SQL语句

SqlSessioin sqlSession = sqlSessionFactory.openSession();

list<User> users = sqlSession.selectList("test.selectBlog");//看上面的路径

Mapper代理

1.定义SQL映射同名的Mapper接口（一个Mapper在Java里，另一个在resources里）

2.设置SQL映射文件的namespace属性为Mapper接口全限定名（Java里Mapper的路径）

3.在Mapper接口中定义方法，方法名和SQL映射文件中sql语句的id相同，保证参数类型和返回值相同

SqlSessioin sqlSession = sqlSessionFactory.openSession();

list<User> users = sqlSession.selectList("test.selectBlog");//看上面的路径

改为

UserMapper userMapper = sqlSession.getMapper(UserMapper.class);

List<User> users = userMapper.selectBlog"();

增删改查

1.查

1.1查所有

//（select \* from Blog + 无参数 + list集合对象（返回值））

//1.编写接口方法 Mapper接口

//2.写SQL语句，SQL映射文件

<select id="selectAll" resultType="Blog">

select \* select id , user\_name as userName //给数据库表名取别名和mapper中一样就能找

from Blog

</select>

在外面定义sql片段（取别名）

<sql id = "brand\_column">

id , user\_name as userName

</sql>

<select id="selectAll" resultType="Blog">

select <include refid = "brand\_column"/>

from Blog

</select>

<!--\*\*\*\*resultMap 做映射\*\*\*\*-->

<resultMap id="UserEntityMap" type="com.example.demo.entity.User">

<!--将表的字段和类的属性名不一致的进行匹配指定,名称一致的也可以指定,但没必要

但是,在定义映射规则时无论主键名称是否一致都不能省

column属性:表示表中的字段名称

property属性:表示类中的属性名称-->

<result column="user\_name " property="userName"></result>

</resultMap>

<select id="selectAll" resultType="Blog">

select \*

from Blog

</select>

<select id="selectById" resultType="Blog">

select \*

from Blog

where id = #{};

</select>

1.2查详细

//（select \* from Blog where \*\*\*\*\*\* + 参数 + 对象（返回值 单个）)

//1.编写接口方法 Mapper接口

//接口要传值 Blog seletcById(int id);

//2.写SQL语句，SQL映射文件

<select id="selectById" resultType="Blog"> <!--parameterType = "int" 设置参数类型-->

select \*

from Blog <!--动态查表${tableName}-->

where id = #{id}; <!--${}也能用，但有SQL注入 &lt ===<(小于)

<![CDATA[

<

]]>

-->

</select>

1.3多条件查询

//（select \* from Blog + 所有查询条件 + list集合对象（返回值））

//1.编写接口方法 Mapper接口

List<Blog> selectByCondition(@Param("status") int status @Param("ComName"));

//封装一下

List<Blog> selectByCondition(Blog blog);

List<Blog> selectByCondition(Map map);

//2.写SQL语句，SQL映射文件

<select id="selectByCondition" resultType="Blog">

select \*

from Blog

where

status = #{status} <!--模糊查询要在java中自己加-->

and ComName like #{ComName};

</select>

1.4动态查询（if choose trim foreach）

<select id="selectByCondition" resultType="Blog">

select \*

from Blog

<where> <!--在外面套个where-->

<!--where 1 = 1 <!--添加恒等式1=1，给下面所有内容加上and-->-->

<if test = "status != null">

and status = #{status}

</if>

<if test = "ComName != null and ComName != ' ' ">

and ComName like #{ComName};

</if>

</where>

</select>

2.增

//（除id外参数 + void（返回值））

//1.编写接口方法 Mapper接口

void insert(Blog blog);

//2.写SQL语句，SQL映射文件

<!--不能及时获取id加这个指向id useGeneratedKeys="true" keyProperty="id"-->

<insert id="insert" useGeneratedKeys="true" keyProperty="id">

insert into Blog (username,`password`,salt,phone,email,gender,avatar)

values (#{username},#{password},#{salt},#{phone},#{email},#{gender},#{avatar});

</insert>

3.改

3.1改全字段

//（所有参数 + void（返回值））

//1.编写接口方法 Mapper接口

void update(Blog blog);

//2.写SQL语句，SQL映射文件

<update id = update>

update Blog

set

username = #{username},

password = #{password}

where id = #{id};

</update>

3.2修改动态字段

<update id = update>

update Blog

<set> <!--set标签-->

<if test = “username != null and username != ' ' ">

username = #{username},

</if>

<if test = ”password != null and password != ' ' ">

password = #{password},

</if>

where id = #{id};

</set>

</update>

4.删(一般采取逻辑删除)

4.1删一个

//1.编写接口方法 Mapper接口

void delete(int id);

//2.写SQL语句，SQL映射文件

<delete id ="delete">

delete from Blog where id = #{id};

</delete>

4.2删一把

//1.编写接口方法 Mapper接口

void deleteById(@Param("ids")int [] ids);

//2.写SQL语句，SQL映射文件

<delete id ="deleteById">

delete from Blog

where id in

<foreach collection ="ids" item = "id" separator="," open="(" close=")">

#{id}

</foreach>

</delete>

5.传参（POJO,Map,Collection,List,Array）

<!--最上面的给我用这个-->

<?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="org.mybatis.example.BlogMapper">

<select id="selectBlog" resultType="Blog">

select \* from Blog where id = #{id}

</select>

</mapper>

<update id="UpdateInfoByUid">

update t\_user

set <if test="phone!=null">phone = #{phone},</if>

<if test="email!=null">email = #{email},</if>

<if test="gender!=null">gender = #{gender},</if>

modified\_user=#{modifiedUser},

modified\_time=#{modifiedTime}

where uid=#{uid}

</update>