

MySQL基础

今日目标：

- 完成MySQL的安装及登陆基本操作
- 能通过SQL对数据库进行CRUD
- 能通过SQL对表进行CRUD
- 能通过SQL对数据进行CRUD

1. 数据库相关概念

以前我们做系统，数据持久化的存储采用的是文件存储。存储到文件中可以达到系统关闭数据不会丢失的效果，当然文件存储也有它的弊端。

假设在文件中存储以下的数据：

1	姓名	年龄	性别	住址
2	张三	23	男	北京西三旗
3	李四	24	女	北京西二旗
4	王五	25	男	西安软件新城

现要修改李四这条数据的性别数据改为男，我们现学习的IO技术可以通过将所有的数据读取到内存中，然后进行修改再存到该文件中。通过这种方式操作存在很大问题，现在只有三条数据，如果文件中存储1T的数据，那么就会发现内存根本就存储不下了。

现需要既能持久化存储数据，也要能避免上述问题的技术使用在我们的系统中。数据库就是这样的一门技术。

1.1 数据库

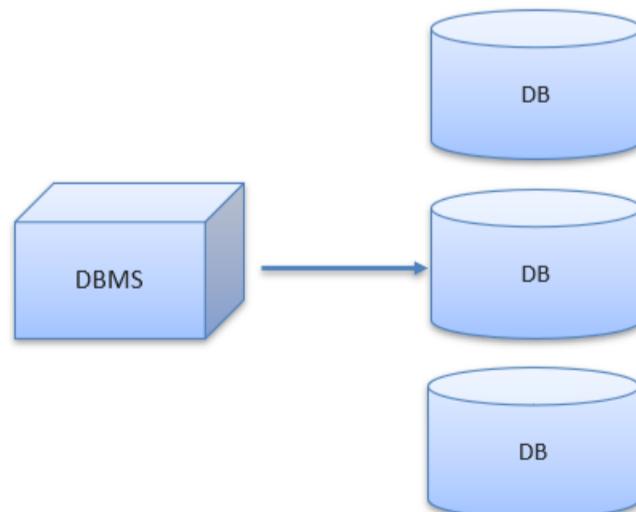
- **存储和管理数据的仓库，数据是有组织的进行存储。**
- 数据库英文名是 DataBase，简称DB。

数据库就是将数据存储在硬盘上，可以达到持久化存储的效果。那又是如何解决上述问题的？使用数据库管理系统。

1.2 数据库管理系统

- **管理数据库的大型软件**
- 英文： DataBase Management System，简称 DBMS

在电脑上安装了数据库管理系统后，就可以通过数据库管理系统创建数据库来存储数据，也可以通过该系统对数据库中的数据进行数据的增删改查相关的操作。我们平时说的MySQL数据库其实是MySQL数据库管理系统。



通过上面的描述，大家应该已经知道了 **数据库管理系统** 和 **数据库** 的关系。那么有哪些常见的数据库管理系统呢？

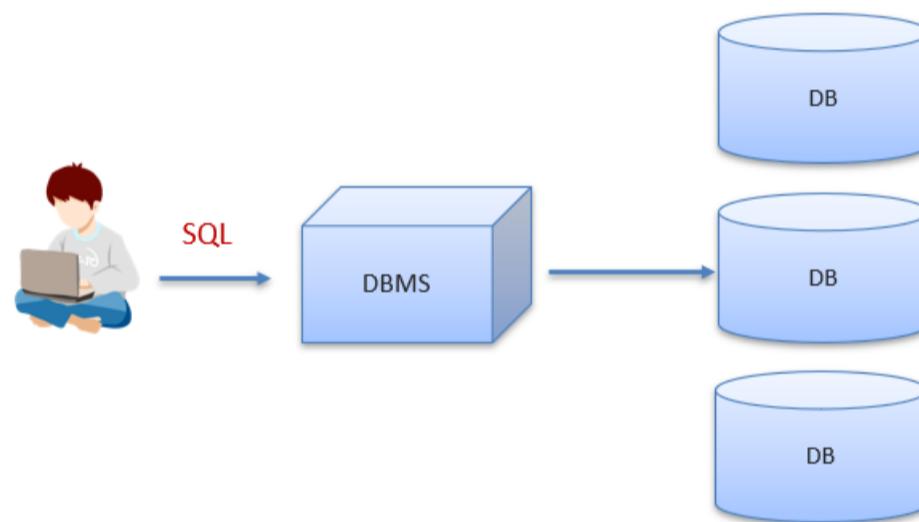
1.3 常见的数据库管理系统

Rank			DBMS	Database Model	Score		
Apr 2018	Mar 2018	Apr 2017			Apr 2018	Mar 2018	Apr 2017
1.	1.	1.	Oracle +	Relational DBMS	1289.79	+0.18	-112.21
2.	2.	2.	MySQL +	Relational DBMS	1226.40	-2.46	-138.22
3.	3.	3.	Microsoft SQL Server +	Relational DBMS	1095.51	-9.28	-109.26
4.	4.	4.	PostgreSQL +	Relational DBMS	395.47	-3.88	+33.69
5.	5.	5.	MongoDB +	Document store	341.41	+0.89	+15.98
6.	6.	6.	DB2 +	Relational DBMS	188.95	+2.28	+2.29
7.	7.	7.	Microsoft Access	Relational DBMS	132.22	+0.27	+4.04
8.	↑ 9.	↑ 11.	Elasticsearch +	Search engine	131.36	+2.81	+25.69
9.	↓ 8.	9.	Redis +	Key-value store	130.11	-1.12	+15.75
10.	10.	↓ 8.	Cassandra +	Wide column store	119.09	-4.40	-7.10
11.	11.	↓ 10.	SQLite +	Relational DBMS	115.99	+1.17	+2.19

接下来对上面列举的数据库管理系统进行简单的介绍：

- Oracle：收费的大型数据库，Oracle公司的产品
- MySQL：开源免费的中小型数据库。后来Sun公司收购了MySQL，而Sun公司又被Oracle收购
- SQL Server：MicroSoft公司收费的中型的数据库。C#、.net等语言常使用
- PostgreSQL：开源免费中小型的数据库
- DB2：IBM公司的大型收费数据库产品
- SQLite：嵌入式的微型数据库。如：作为Android内置数据库
- MariaDB：开源免费中小型的数据库

我们课程上学习的是MySQL数据库管理系统，PostgreSQL在一些公司也有使用，此时大家肯定会想以后在公司中如果使用我们没有学习过的PostgreSQL数据库管理系统怎么办？这点大家大可不必担心，如下图所示：



我们可以通过数据库管理系统操作数据库，对数据库中的数据进行增删改查操作，而怎么样让用户跟数据库管理系统打交道呢？就可以通过一门编程语言（SQL）来实现。

1.4 SQL

- 英文：Structured Query Language，简称SQL，结构化查询语言
- 操作关系型数据库的编程语言
- 定义操作所有关系型数据库的统一标准，可以使用SQL操作所有的关系型数据库管理系统，以后工作中如果使用到了其他的数据库管理系统，也同样的使用SQL来操作。

2, MySQL

2.1 MySQL安装

安装环境:Win10 64位 软件版本:MySQL 5.7.24 解压版

2.1.1 下载

<https://downloads.mysql.com/archives/community/>

点开上面的链接就能看到如下界面：

④ MySQL Product Archives

◀ MySQL Community Server (Archived Versions)

Please note that these are old versions. New releases will have recent bug fixes and features!
To download the latest release of MySQL Community Server, please visit [MySQL Downloads](#).

Product Version: **5.7.24**

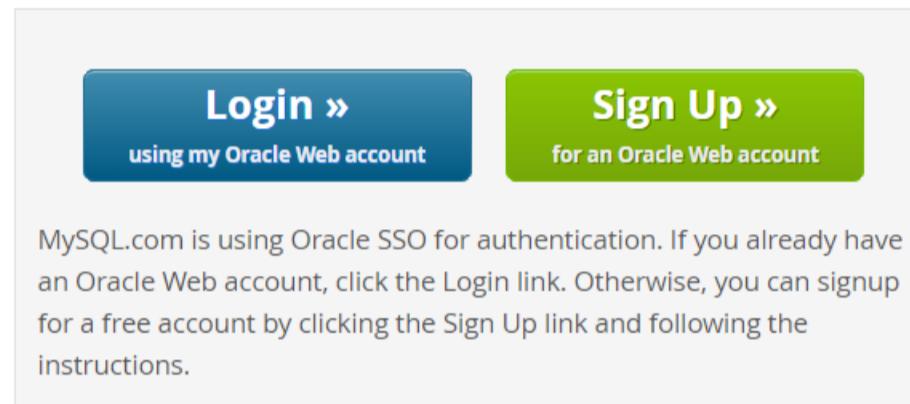
Operating System: **Microsoft Windows**

OS Version: **All**

Windows (x86, 32-bit), ZIP Archive	Oct 4, 2018	308.7M	Download
(mysql-5.7.24-win32.zip)			MD5: daed322d15a1251069dded4419768ade Signature
Windows (x86, 64-bit), ZIP Archive	Oct 4, 2018	321.1M	Download
(mysql-5.7.24-winx64.zip)			MD5: 44d30639a3310a4f60dc5bdf757749ef Signature
Windows (x86, 32-bit), ZIP Archive	Oct 4, 2018	376.4M	Download
Debug Binaries & Test Suite (mysql-5.7.24-win32-debug-test.zip)			MD5: 89d1c61b3ba783eb303fb563c0d8895 Signature
Windows (x86, 64-bit), ZIP Archive	Oct 4, 2018	386.0M	Download
Debug Binaries & Test Suite (mysql-5.7.24-winx64-debug-test.zip)			MD5: 2617862726cce9e0031c8a5b32c75f8b Signature

选择选择和自己系统位数相对应的版本点击右边的 **Download**，此时会进到另一个页面，同样在接近页面底部的地方找到如下图所示的位置：

- Comment in the MySQL Documentation



不用理会上面的登录和注册按钮，直接点击 **No thanks, just start my download.** 就可以下载。

 mysql-5.7.24-win32.zip
 mysql-5.7.24-winx64.zip

2.1.2 安装(解压)

下载完成后我们得到的是一个压缩包，将其解压，我们就可以得到MySQL 5.7.24的软件本体了(就是一个文件夹)，我们可以把它放在你想安装的位置。

磁盘 (D:) > software > mysql-5.7.24-winx64			
名称	修改日期	类型	大小
bin	2018/10/4 14:53	文件夹	
docs	2018/10/4 14:53	文件夹	
include	2018/10/4 14:53	文件夹	
lib	2018/10/4 14:53	文件夹	
share	2018/10/4 14:53	文件夹	
COPYING	2018/10/4 13:48	文件	18 KB
README	2018/10/4 13:48	文件	3 KB

2.2 MySQL卸载

如果你想卸载MySQL，也很简单。

右键开始菜单，选择命令提示符(管理员)，打开黑框。

1. 敲入 `net stop mysql`，回车。

```
1 | net stop mysql
```

管理员: 命令提示符
Microsoft Windows [版本 10.0.17763.134]
(c) 2018 Microsoft Corporation。保留所有权利。
C:\Windows\system32>net stop mysql
MySQL 服务正在停止。
MySQL 服务已成功停止。
C:\Windows\system32>

2. 再敲入`mysqld -remove mysql`, 回车。

```
1 | mysqld -remove mysql
```

C:\Windows\system32>mysqld -remove mysql
Service successfully removed.
C:\Windows\system32>

3. 最后删除MySQL目录及相关的环境变量。

至此, MySQL卸载完成!

2.3 MySQL配置

2.3.1 添加环境变量

环境变量里面有很多选项, 这里我们只用到 Path 这个参数。为什么在初始化的开始要添加环境变量呢? 在黑框(即CMD)中输入一个可执行程序的名字, Windows会先在环境变量中的 Path 所指的路径中寻找一遍, 如果找到了就直接执行, 没找到就在当前工作目录找, 如果还没找到, 就报错。我们添加环境变量的目的就是能够在任意一个黑框直接调用MySQL中的相关程序而不用总是修改工作目录, 大大简化了操作。

右键 `此电脑` → `属性`, 点击 `高级系统设置`

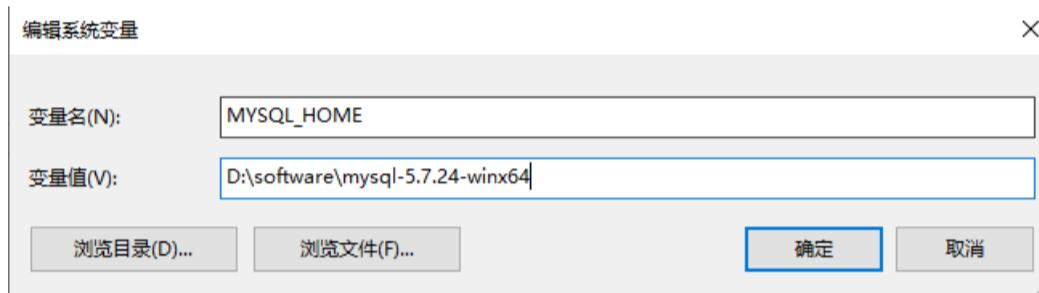


点击 `环境变量`

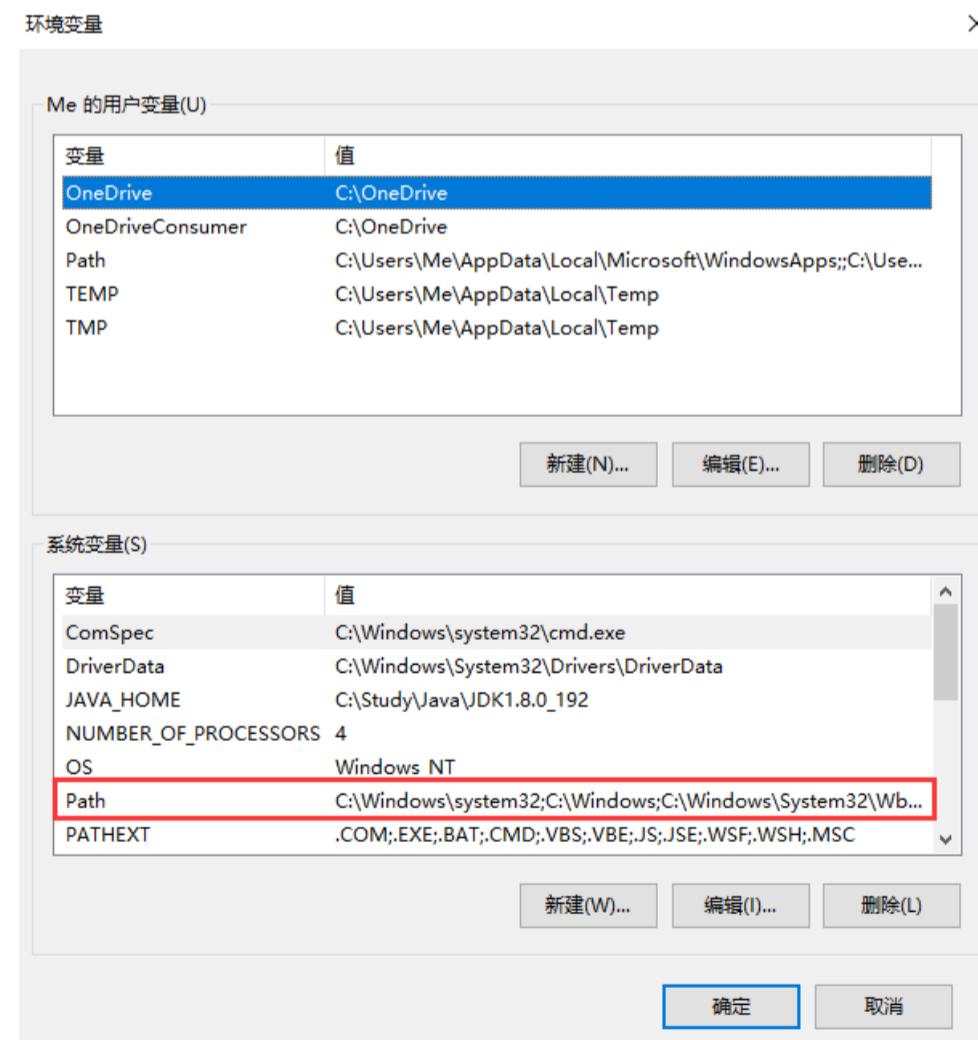
系统属性



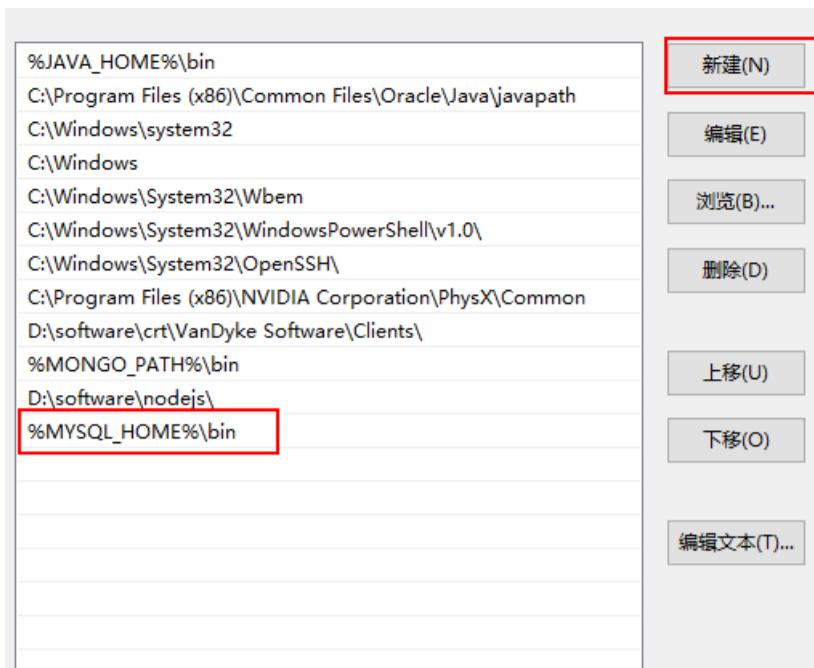
在 系统变量 中新建 MySQL_HOME



在 系统变量 中找到并双击 Path



点击 新建



最后点击确定。

如何验证是否添加成功？

右键开始菜单(就是屏幕左下角)，选择命令提示符(管理员)，打开黑框，敲入 mysql，回车。如果提示 Can't connect to MySQL server on 'localhost' 则证明添加成功；如果提示 mysql 不是内部或外部命令，也不是可运行的程序或批处理文件则表示添加失败，请重新检查步骤并重试。

2.3.2 新建配置文件

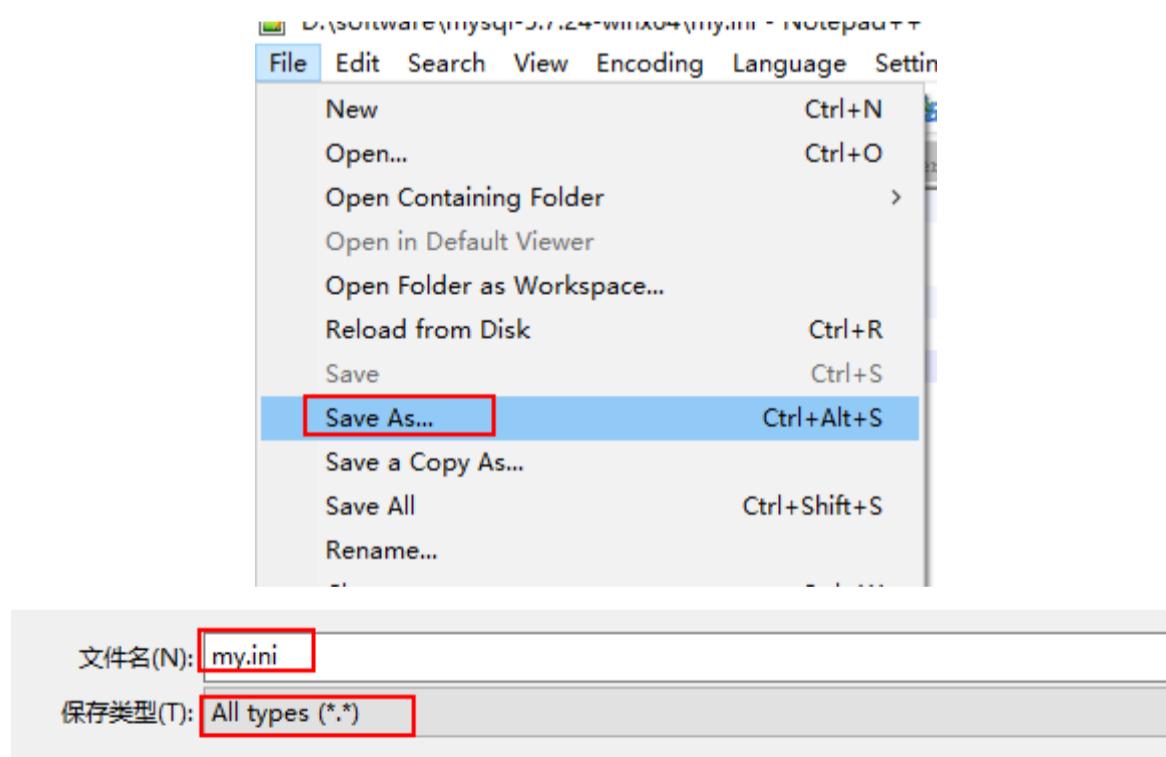
新建一个文本文件，内容如下：

```

1 [mysql]
2 default-character-set=utf8
3
4 [mysqld]
5 character-set-server=utf8
6 default-storage-engine=INNODB
7 sql_mode=STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION

```

把上面的文本文件另存为，在保存类型里选所有文件 (*.*)，文件名叫 my.ini，存放的路径为MySQL的根目录(例如我的是 D:\software\mysql-5.7.24-winx64，根据自己的MySQL目录位置修改)。



上面代码意思就是配置数据库的默认编码集为utf-8和默认存储引擎为INNODB。

2.3.3 初始化MySQL

在刚才的黑框中敲入 mysql --initialize-insecure，回车，稍微等待一会，如果出现没有出现报错信息(如下图)则证明 data 目录初始化没有问题，此时再查看 MySQL 目录下已经有 data 目录生成。

```
1 mysql --initialize-insecure
```

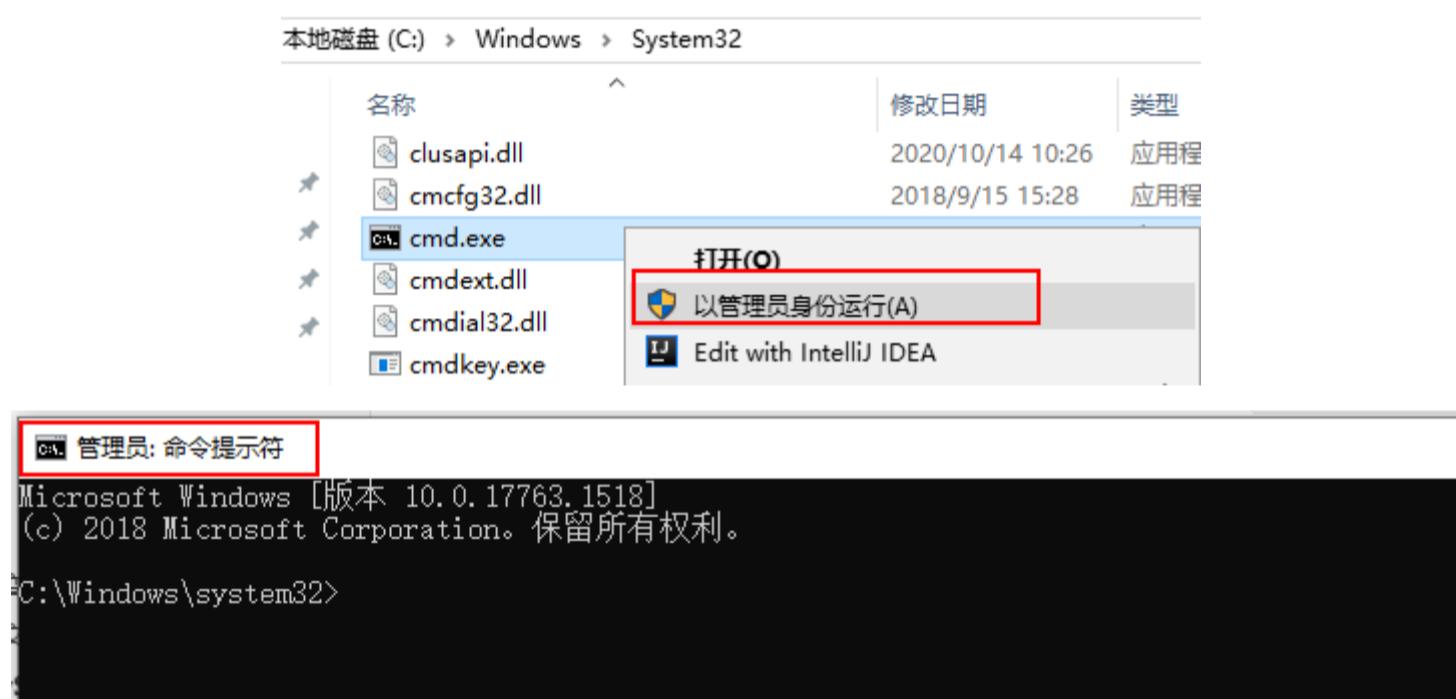
```
管理员: 命令提示符
Microsoft Windows [版本 10.0.17763.1518]
(c) 2018 Microsoft Corporation。保留所有权利。

C:\Windows\system32>mysqld --initialize-insecure
C:\Windows\system32>
```

tips：如果出现如下错误

```
C:\Users\itcast>mysqld --initialize-insecure
mysqld: Can't create directory 'C:\Program Files\MySQL\MySQL Server 5.7\data\' (Errcode: 13 - Permission denied)
2020-11-09T05:57:38.064222Z 0 [Warning] TIMESTAMP with implicit DEFAULT value is deprecated. Please use --explicit_defaults_for_timestamp server option (see documentation for more details).
2020-11-09T05:57:38.070727Z 0 [ERROR] Aborting
```

是由于权限不足导致的，去 `c:\windows\System32` 下以管理员方式运行 cmd.exe



2.3.4 注册MySQL服务

在黑框里敲入 `mysqld -install`，回车。

```
1 | mysqld -install
```

```
C:\Windows\system32>mysqld -install
Service successfully installed.

C:\Windows\system32>
```

现在你的计算机上已经安装好了MySQL服务了。

MySQL服务器

2.3.5 启动MySQL服务

在黑框里敲入 `net start mysql`，回车。

```
1 | net start mysql // 启动mysql服务
2 |
3 | net stop mysql // 停止mysql服务
```

```
C:\Windows\system32>net start mysql  
MySQL 服务正在启动。  
MySQL 服务已经启动成功。
```

2.3.6 修改默认账户密码

在黑框里敲入`mysqladmin -u root password 1234`, 这里的`1234`就是指默认管理员(即root账户)的密码, 可以自行修改成你喜欢的。

```
1 | mysqladmin -u root password 1234
```

```
C:\Windows\system32>mysqladmin -u root password 1234  
mysqladmin: [Warning] Using a password on the command line interface can be insecure.  
Warning: Since password will be sent to server in plain text, use ssl connection to ensure  
password safety.
```

至此, MySQL 5.7 解压版安装完毕!

2.4 MySQL登陆和退出

2.4.1 登陆

右键开始菜单, 选择命令提示符, 打开黑框。在黑框中输入, `mysql -uroot -p1234`, 回车, 出现下图且左下角为`mysql>`, 则登录成功。

```
1 | mysql -uroot -p1234
```

```
命令提示符 - mysql -uroot -p1234  
Microsoft Windows [版本 10.0.17763.134]  
(c) 2018 Microsoft Corporation。保留所有权利。  
  
C:\Users\Me>mysql -uroot -p1234  
mysql: [Warning] Using a password on the command line interface can be insecure.  
Welcome to the MySQL monitor. Commands end with ; or \g.  
Your MySQL connection id is 3  
Server version: 5.7.24 MySQL Community Server (GPL)  
  
Copyright (c) 2000, 2018, Oracle and/or its affiliates. All rights reserved.  
  
Oracle is a registered trademark of Oracle Corporation and/or its  
affiliates. Other names may be trademarks of their respective  
owners.  
  
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.  
mysql>
```

到这里你就可以开始你的MySQL之旅了!

登陆参数:

```
1 | mysql -u用户名 -p密码 -h要连接的mysql服务器的ip地址(默认127.0.0.1) -P端口号(默认3306)
```

2.4.2 退出

退出mysql:

```
1 | exit  
2 | quit
```

2.5 MySQL数据模型

关系型数据库：

关系型数据库是建立在关系模型基础上的数据库，简单说，关系型数据库是由多张能互相连接的二维表组成的数据库

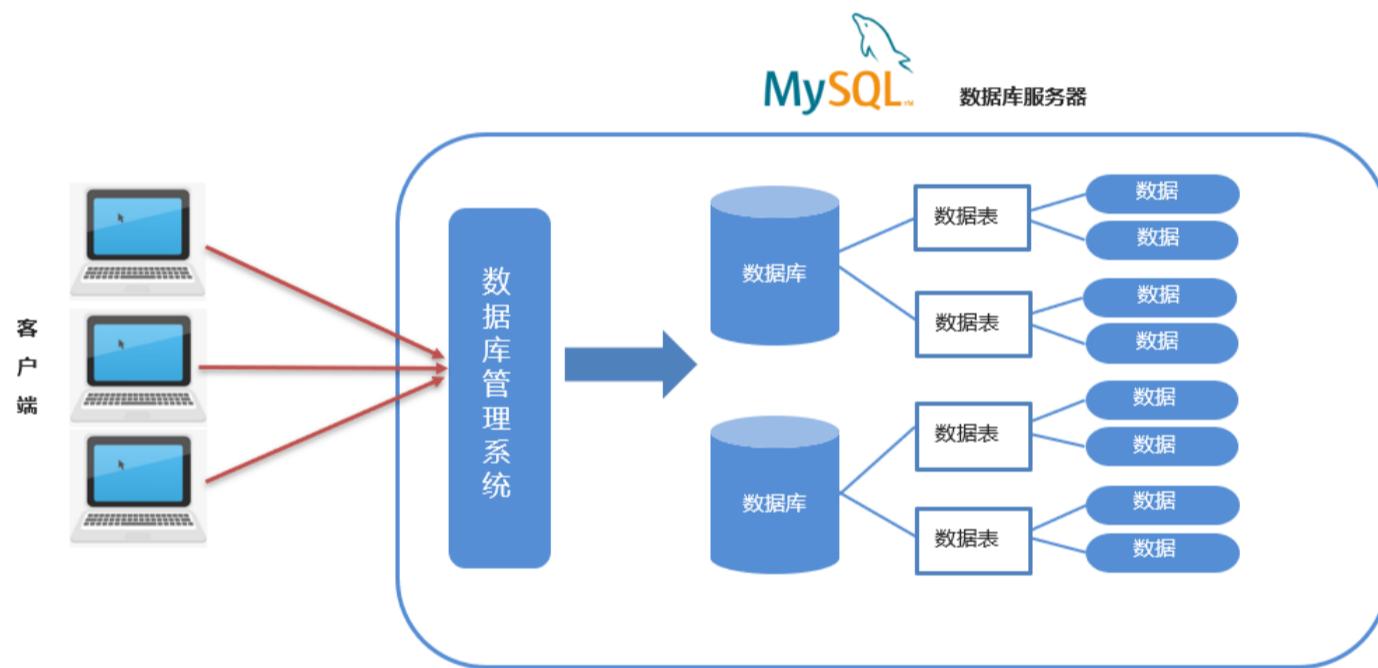
如下图，`订单信息表` 和 `客户信息表` 都是有行有列二维表我们将这样的称为关系型数据库。

订单信息表						客户信息表			
订单编号	订单项目	负责人	业务员	订单数量	客户编号	客户编号	客户名称	所属公司	联系方式
001	挖掘机	刘明	李东明	1台	1	1	李聪	五一建设	13253661015
002	冲击钻	李刚	霍新峰	8个	2	2	刘新明	个体经营	13285746958
003	铲车	郭新一	艾美丽	2辆	1				

接下来看关系型数据库的优点：

- 都是使用表结构，格式一致，易于维护。
- 使用通用的 SQL 语言操作，使用方便，可用于复杂查询。
 - 关系型数据库都可以通过SQL进行操作，所以使用方便。
 - 复杂查询。现在需要查询001号订单数据，我们可以看到该订单是1号客户的订单，而1号订单是李聪这个客户。以后也可以在一张表中进行统计分析等操作。
- 数据存储在磁盘中，安全。

数据模型：



如上图，我们通过客户端可以通过数据库管理系统创建数据库，在数据库中创建表，在表中添加数据。创建的每一个数据库对应到磁盘上都是一个文件夹。比如可以通过SQL语句创建一个数据库（数据库名称为db1），语句如下。该语句咱们后面会学习。

```
mysql> create database db1;
```

我们可以在数据库安装目录下的data目录下看到多了一个 `db1` 的文件夹。所以，在MySQL中一个数据库对应到磁盘上的一个文件夹。

而一个数据库下可以创建多张表，我们到MySQL中自带的mysql数据库的文件夹目录下：



而上图中右边的 `db.frm` 是表文件，`db.MYD` 是数据文件，通过这两个文件就可以查询到数据展示成二维表的效果。

小结：

- MySQL中可以创建多个数据库，每个数据库对应到磁盘上的一个文件夹
- 在每个数据库中可以创建多个表，每张都对应到磁盘上一个 frm 文件
- 每张表可以存储多条数据，数据会被存储到磁盘中 MYD 文件中

3, SQL概述

了解了数据模型后，接下来我们就学习SQL语句，通过SQL语句对数据库、表、数据进行增删改查操作。

3.1 SQL简介

- 英文：Structured Query Language，简称 SQL
- 结构化查询语言，一门操作关系型数据库的编程语言
- 定义操作所有关系型数据库的统一标准
- 对于同一个需求，每一种数据库操作的方式可能会存在一些不一样的地方，我们称为“方言”

3.2 通用语法

- SQL 语句可以单行或多行书写，以分号结尾。

```
mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| db1 |
| mysql |
| performance_schema |
| sys |
+-----+
5 rows in set (0.00 sec)
```

如上，以分号结尾才是一个完整的sql语句。

- MySQL 数据库的 SQL 语句不区分大小写，关键字建议使用大写。

同样的一条sql语句写成下图的样子，一样可以运行处结果。

```
mysql> Show DataBases;
+-----+
| Database |
+-----+
| information_schema |
| db1 |
| mysql |
| performance_schema |
| sys |
+-----+
5 rows in set (0.00 sec)
```

- 注释

- 单行注释: -- 注释内容 或 #注释内容(MySQL 特有)

```
mysql> Show DataBases;-- 查询所有数据库名称
+-----+
| Database |
+-----+
| information_schema |
| db1 |
| mysql |
| performance_schema |
| sys |
+-----+
5 rows in set (0.00 sec)
```

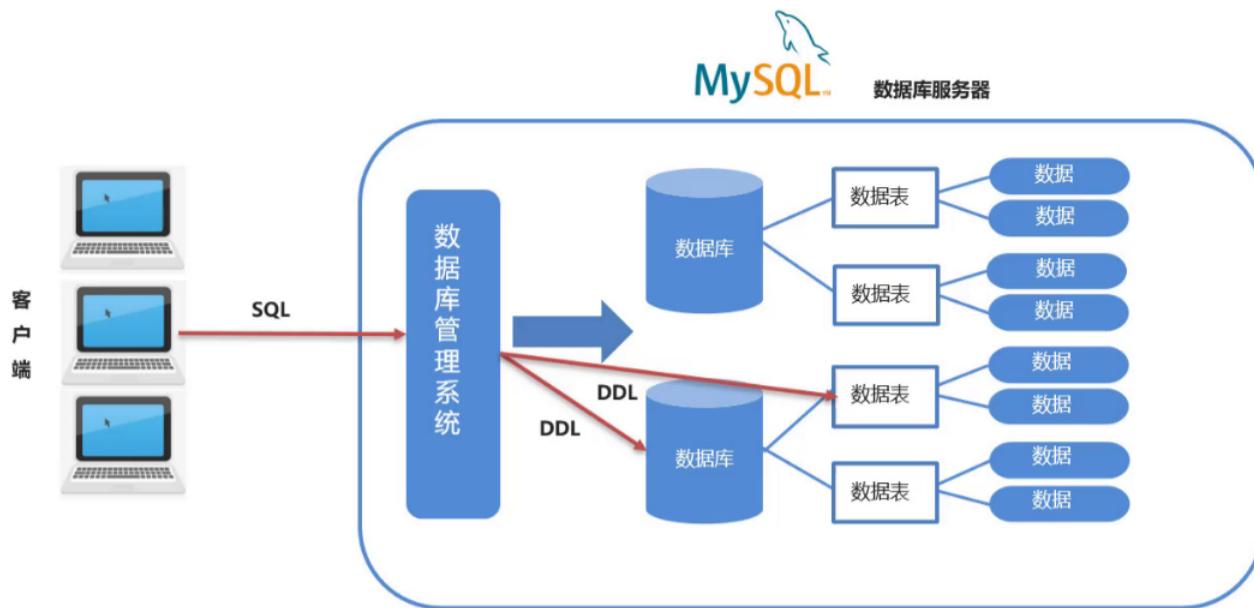
```
mysql> show databases; #查询所有数据库名称
+-----+
| Database |
+-----+
| information_schema |
| db1 |
| mysql |
| performance_schema |
| sys |
+-----+
5 rows in set (0.00 sec)
```

注意： 使用-- 添加单行注释时，--后面一定要加空格，而#没有要求。

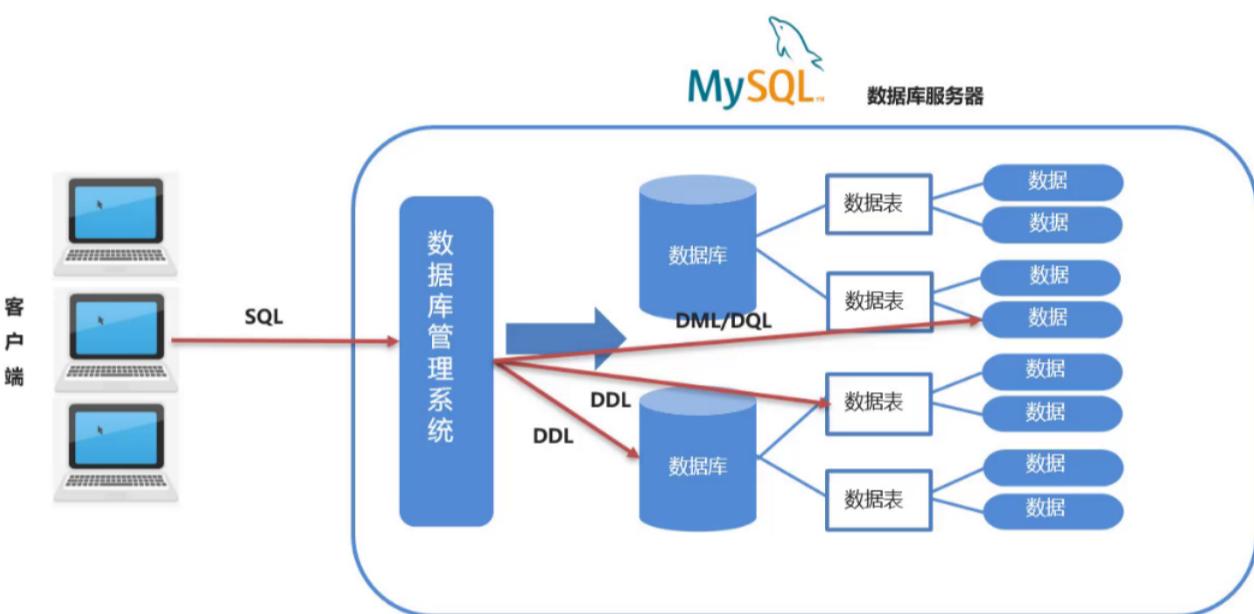
◦ 多行注释: /* 注释 */

3.3 SQL分类

- DDL(Data Definition Language) : 数据定义语言, 用来定义数据库对象: 数据库, 表, 列等
DDL简单理解就是用来操作数据库, 表等



- DML(Data Manipulation Language) 数据操作语言, 用来对数据库中表的数据进行增删改
DML简单理解就对表中数据进行增删改



- DQL(Data Query Language) 数据查询语言, 用来查询数据库中表的记录(数据)
DQL简单理解就是对数据进行查询操作。从数据库表中查询到我们想要的数据。
- DCL(Data Control Language) 数据控制语言, 用来定义数据库的访问权限和安全级别, 及创建用户
DML简单理解就是对数据库进行权限控制。比如我让某一个数据库表只能让某一个用户进行操作等。

注意: 以后我们最常操作的是 DML 和 DQL , 因为我们开发中最常操作的就是数据。

4, DDL:操作数据库

我们先来学习DDL来操作数据库。而操作数据库主要就是对数据库的增删查操作。

4.1 查询

查询所有的数据库

```
1 | SHOW DATABASES;
```

运行上面语句效果如下:

```
mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
| sys |
+-----+
4 rows in set (0.00 sec)
```

上述查询到的是的这些数据库是mysql安装好自带的数据库，我们以后不要操作这些数据库。

4.2 创建数据库

- **创建数据库：**

```
1 | CREATE DATABASE 数据库名称;
```

运行语句效果如下：

```
mysql> create database db1;
Query OK, 1 row affected (0.00 sec)

mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| db1 |
| mysql |
| performance_schema |
| sys |
+-----+
5 rows in set (0.00 sec)
```

而在创建数据库的时候，我并不知道db1数据库有没有创建，直接再次创建名为db1的数据库就会出现错误。

```
mysql> create database db1;
ERROR 1007 (HY000): Can't create database 'db1' ; database exists
```

为了避免上面的错误，在创建数据库的时候先做判断，如果不存在再创建。

- **创建数据库(判断，如果不存在则创建)**

```
1 | CREATE DATABASE IF NOT EXISTS 数据库名称;
```

运行语句效果如下：

```
mysql> create database if not exists db1;
Query OK, 1 row affected, 1 warning (0.00 sec)

mysql> create database if not exists db2;
Query OK, 1 row affected (0.00 sec)

mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| db1 |
| db2 |
| mysql |
| performance_schema |
| sys |
+-----+
6 rows in set (0.00 sec)
```

从上面的效果可以看到虽然db1数据库已经存在，再创建db1也没有报错，而创建db2数据库则创建成功。

4.3 删除数据库

- **删除数据库**

```
1 | DROP DATABASE 数据库名称;
```

- **删除数据库(判断，如果存在则删除)**

```
1 | DROP DATABASE IF EXISTS 数据库名称;
```

运行语句效果如下：

```
mysql> drop database db2;
ERROR 1008 (HY000): Can't drop database 'db2' ; database doesn't exist

mysql> drop database if exists db2;
Query OK, 0 rows affected, 1 warning (0.00 sec)
```

4.4 使用数据库

数据库创建好了，要在数据库中创建表，得先明确在哪儿个数据库中操作，此时就需要使用数据库。

- 使用数据库

```
1 | USE 数据库名称;
```

- 查看当前使用的数据库

```
1 | SELECT DATABASE();
```

运行语句效果如下：

```
mysql> use db1;
Database changed
mysql> select database();
+-----+
| database() |
+-----+
| db1 |
+-----+
1 row in set (0.00 sec)
```

5. DDL:操作表

操作表也就是对表进行增 (Create) 删 (Retrieve) 改 (Update) 查 (Delete) 。

5.1 查询表

- 查询当前数据库下所有表名称

```
1 | SHOW TABLES;
```

我们创建的数据库中没有任何表，因此我们进入mysql自带的mysql数据库，执行上述语句查看

```
mysql> use mysql;
Database changed
mysql> show tables;
```

- 查询表结构

```
1 | DESC 表名称;
```

查看mysql数据库中func表的结构，运行语句如下：

```
mysql> desc func;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| name  | char(64) | NO | PRI | 0 | 
| ret   | tinyint(1) | NO |    |    | 
| dl    | char(128) | NO |    |    | 
| type  | enum('function','aggregate') | NO |    | NULL | 
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)
```

5.2 创建表

- 创建表

```
1 | CREATE TABLE 表名 (
2 |     字段名1 数据类型1,
3 |     字段名2 数据类型2,
4 |     ...
5 |     字段名n 数据类型n
6 | );
7 |
```

注意：最后一行末尾，不能加逗号

知道了创建表的语句，那么我们创建创建如下结构的表

tb_user

id	username	password

```
1 create table tb_user (
2     id int,
3     username varchar(20),
4     password varchar(32)
5 );
```

运行语句如下：

```
mysql> create table tb_user(
    -> id int,
    -> username varchar(20),
    -> password varchar(32)
    -> );
Query OK, 0 rows affected (0.02 sec)

mysql> show tables;
+-----+
| Tables_in_db1 |
+-----+
| tb_user |
+-----+
1 row in set (0.00 sec)

mysql> desc tb_user;
+-----+-----+-----+-----+-----+-----+
| Field | Type   | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| id    | int(11) | YES  |     | NULL    |       |
| username | varchar(20) | YES  |     | NULL    |       |
| password | varchar(32) | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

5.3 数据类型

MySQL 支持多种类型，可以分为三类：

- 数值

```
1 tinyint : 小整数型，占一个字节
2 int : 大整数类型，占四个字节
3 eg : age int
4 double : 浮点类型
5 使用格式：字段名 double(总长度,小数点后保留的位数)
6 eg : score double(5,2)
```

- 日期

```
1 date : 日期值。只包含年月日
2 eg : birthday date :
3 datetime : 混合日期和时间值。包含年月日时分秒
```

- 字符串

```
1 char : 定长字符串。
2 优点：存储性能高
3 缺点：浪费空间
4 eg : name char(10) 如果存储的数据字符个数不足10个，也会占10个的空间
5 varchar : 变长字符串。
6 优点：节约空间
7 缺点：存储性能底
8 eg : name varchar(10) 如果存储的数据字符个数不足10个，那就数据字符个数是几就占几个的空间
```

注意：其他类型参考资料中的《MySQL数据类型.xlsx》

案例：

- 1 需求：设计一张学生表，请注重数据类型、长度的合理性
- 2 1. 编号
- 3 2. 姓名，姓名最长不超过10个汉字
- 4 3. 性别，因为取值只有两种可能，因此最多一个汉字
- 5 4. 生日，取值为年月日
- 6 5. 入学成绩，小数点后保留两位
- 7 6. 邮件地址，最大长度不超过 64
- 8 7. 家庭联系电话，不一定是手机号码，可能会出现 - 等字符
- 9 8. 学生状态（用数字表示，正常、休学、毕业...）

语句设计如下：

```
1 create table student (
2   id int,
3   name varchar(10),
4   gender char(1),
5   birthday date,
6   score double(5,2),
7   email varchar(15),
8   tel varchar(15),
9   status tinyint
10 );
```

5.4 删除表

- **删除表**

```
1 DROP TABLE 表名;
```

- **删除表时判断表是否存在**

```
1 DROP TABLE IF EXISTS 表名;
```

运行语句效果如下：

```
mysql> show tables;
+-----+
| Tables_in_db1 |
+-----+
| student       |
+-----+
1 row in set (0.00 sec)

mysql> drop table tb_user;
ERROR 1051 (42S02): Unknown table 'db1.tb_user'
mysql> drop table if exists tb_user;
Query OK, 0 rows affected, 1 warning (0.00 sec)
```

5.5 修改表

- **修改表名**

```
1 ALTER TABLE 表名 RENAME TO 新的表名;
2
3 -- 将表名student修改为stu
4 alter table student rename to stu;
```

- **添加一列**

```
1 ALTER TABLE 表名 ADD 列名 数据类型;
2
3 -- 给stu表添加一列address, 该字段类型是varchar(50)
4 alter table stu add address varchar(50);
```

- 修改数据类型

```
1 ALTER TABLE 表名 MODIFY 列名 新数据类型;
2
3 -- 将stu表中的address字段的类型改为 char(50)
4 alter table stu modify address char(50);
```

- 修改列名和数据类型

```
1 ALTER TABLE 表名 CHANGE 列名 新列名 新数据类型;
2
3 -- 将stu表中的address字段名改为 addr, 类型改为varchar(50)
4 alter table stu change address addr varchar(50);
```

- 删除列

```
1 ALTER TABLE 表名 DROP 列名;
2
3 -- 将stu表中的addr字段 删除
4 alter table stu drop addr;
```

6, navicat使用

通过上面的学习，我们发现在命令行中写sql语句特别不方便，尤其是编写创建表的语句，我们只能在记事本上写好后直接复制到命令行进行执行。那么有没有刚好的工具提供给我们进行使用呢？有。

6.1 navicat概述

- Navicat for MySQL 是管理和开发 MySQL 或 MariaDB 的理想解决方案。
- 这套全面的前端工具为数据库管理、开发和维护提供了一款直观而强大的图形界面。
- 官网：<http://www.navicat.com.cn>

6.2 navicat安装

参考：资料\navicat安装包\navicat_mysql_x86\navicat安装步骤.md

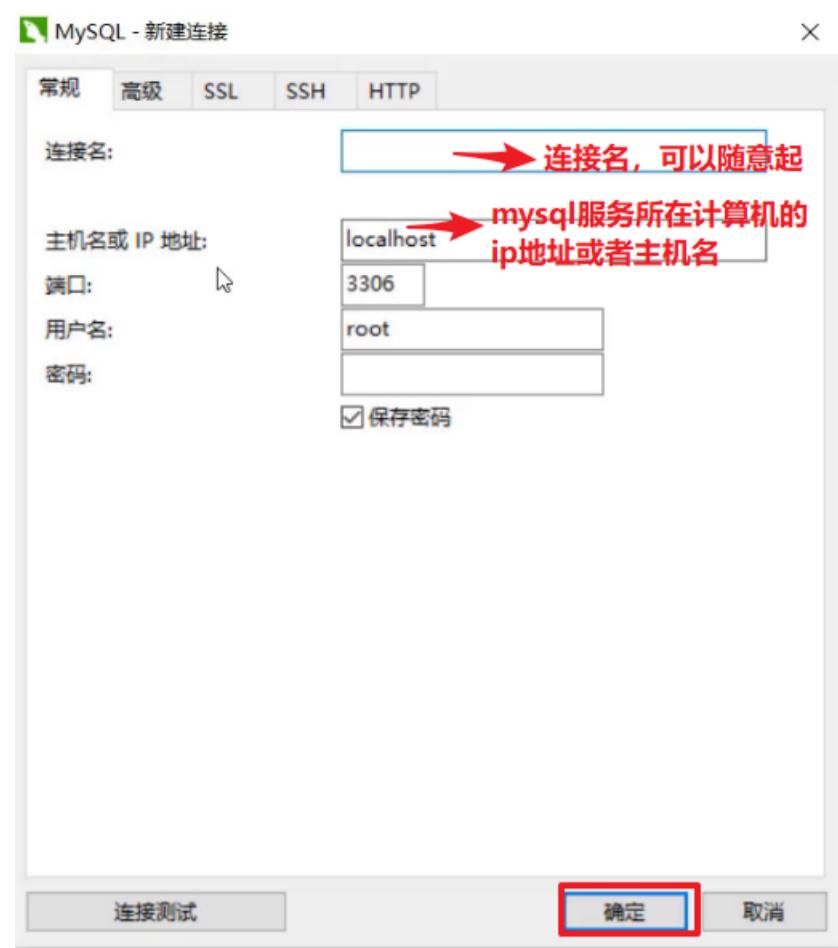
6.3 navicat使用

6.3.1 建立和mysql服务的连接

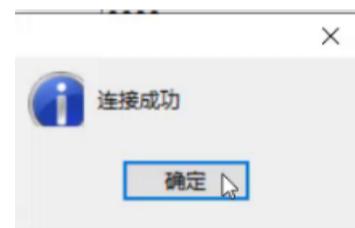
第一步：点击连接，选择MySQL



第二步：填写连接数据库必要的信息



以上操作没有问题就会出现如下图所示界面：



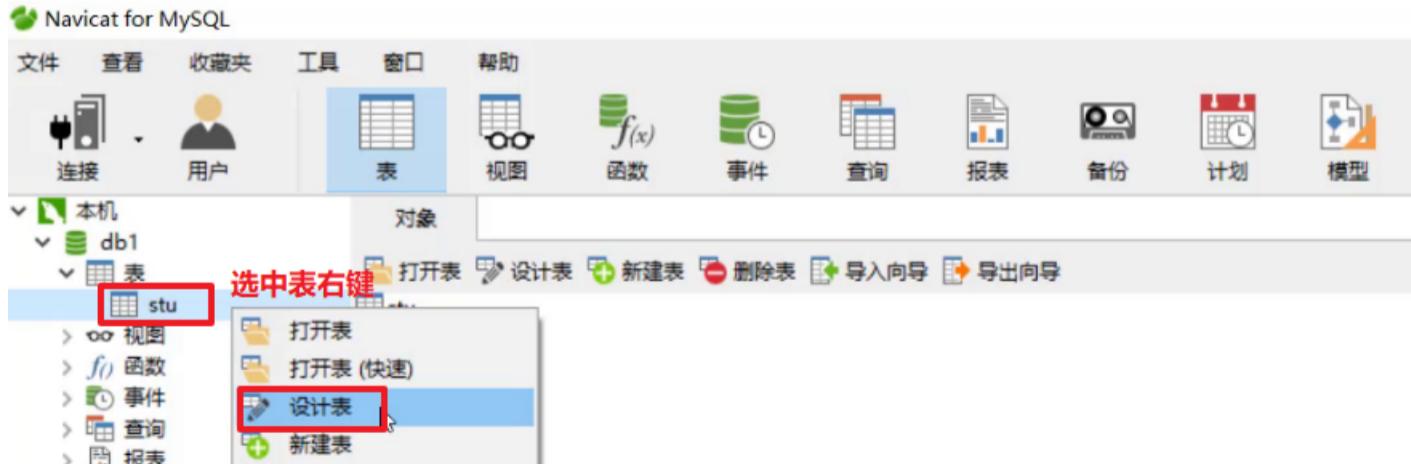
6.3.2 操作

连接成功后就能看到如下图界面：



- 修改表结构

通过下图操作修改表结构：



点击了设计表后即出现如下图所示界面，在图中红框中直接修改字段名，类型等信息：

名	类型	长度	小数点	不是 null
id	int	11	0	<input type="checkbox"/>
name	varchar	10	0	<input type="checkbox"/>
gender	char	1	0	<input type="checkbox"/>
birthday	date	0	0	<input type="checkbox"/>
score	double	5	2	<input type="checkbox"/>
email	varchar	64	0	<input type="checkbox"/>
tel	varchar	15	0	<input type="checkbox"/>
status	tinyint	4	0	<input type="checkbox"/>

- 编写SQL语句并执行

按照如下图所示进行操作即可书写SQL语句并执行sql语句。

7, DML

DML主要是对数据进行增 (insert) 删 (delete) 改 (update) 操作。

7.1 添加数据

- 给指定列添加数据

```
1 | INSERT INTO 表名(列名1,列名2,...) VALUES(值1,值2,...);
```

- 给全部列添加数据

```
1 | INSERT INTO 表名 VALUES(值1,值2,...);
```

- 批量添加数据

```
1 | INSERT INTO 表名(列名1,列名2,...) VALUES(值1,值2,...),(值1,值2,...),(值1,值2,...)...;
```

```
2 | INSERT INTO 表名 VALUES(值1,值2,...),(值1,值2,...),(值1,值2,...)...;
```

- 练习

为了演示以下的增删改操作是否操作成功，故先将查询所有数据的语句介绍给大家：

```
1 | select * from stu;
```

```

1 -- 给指定列添加数据
2 INSERT INTO stu (id, NAME) VALUES (1, '张三');
3 -- 给所有列添加数据, 列名的列表可以省略
4 INSERT INTO stu (id,NAME,sex,birthday,score,email,tel,STATUS) VALUES (2,'李四','男','1999-11-
11',88.88,'lisi@itcast.cn','13888888888',1);
5
6 INSERT INTO stu VALUES (2,'李四','男','1999-11-11',88.88,'lisi@itcast.cn','13888888888',1);
7
8 -- 批量添加数据
9 INSERT INTO stu VALUES
10 (2,'李四','男','1999-11-11',88.88,'lisi@itcast.cn','13888888888',1),
11 (2,'李四','男','1999-11-11',88.88,'lisi@itcast.cn','13888888888',1),
12 (2,'李四','男','1999-11-11',88.88,'lisi@itcast.cn','13888888888',1);

```

7.2 修改数据

- 修改表数据

```
1 UPDATE 表名 SET 列名1=值1,列名2=值2,... [WHERE 条件] ;
```

注意:

- 修改语句中如果不加条件, 则将所有数据都修改!
- 像上面的语句中的中括号, 表示在写sql语句中可以省略这部分

- 练习

- 将张三的性别改为女

```
1 update stu set sex = '女' where name = '张三';
```

- 将张三的生日改为 1999-12-12 分数改为99.99

```
1 update stu set birthday = '1999-12-12', score = 99.99 where name = '张三';
```

- 注意: 如果update语句没有加where条件, 则会将表中所有数据全部修改!

```
1 update stu set sex = '女';
```

上面语句的执行完后查询到的结果是:

信息								
	id	name	sex	birthday	score	email	tel	status
▶	1	张三	女	1999-12-12	99.99	(Null)	(Null)	(Null)
	2	李四	女	1999-11-11	88.88	lisi@itcast.cn	13888888888	1
	2	李四	女	1999-11-11	88.88	lisi@itcast.cn	13888888888	1
	2	李四	女	1999-11-11	88.88	lisi@itcast.cn	13888888888	1

7.3 删除数据

- 删除数据

```
1 DELETE FROM 表名 [WHERE 条件] ;
```

- 练习

```

1 -- 删除张三记录
2 delete from stu where name = '张三';
3
4 -- 删除stu表中所有的数据
5 delete from stu;

```

8, DQL

下面是黑马程序员展示试题库数据的页面

The screenshot shows a web-based application interface for managing a test library. At the top, there's a navigation bar with links for '首页' (Home), '试题库' (Test Library) (which is active and highlighted in blue), '教学管理' (Teaching Management), '教师空间' (Teacher Space), '教学资源' (Teaching Resources) (which is currently selected and has a blue underline), and 'BI'. On the right side of the header, there are dropdown menus for '角色' (Role) set to '课程管理' (Course Management) and '陈长宏' (Chen Changhong). Below the header, there's a sidebar on the left with icons for '试题库' (Test Library), '试卷库' (Exam Paper Library), '媒体库' (Media Library), and '课程库' (Course Library). The main content area displays a table of test questions. The table columns include: 试题编号 (Test Question ID), 使用场景 (Usage Scenario), 试题类型 (Test Type), 所属学科 (Subject), 题型 (Question Type), 题干 (Question Statement), 难度 (Difficulty), 变形题数量 (Number of变形 Questions), 引用次数 (Number of References), 使用次数 (Number of Usages), 入库时间 (入库 Time), 状态 (Status), and 操作 (Operations). There are 7 rows of data in the table, each representing a different test question with various details like subject (JavaEE), type (Operational Problem, Code Problem, etc.), and difficulty levels (1星 to 5星).

页面上展示的数据肯定是在数据库中的试题库表中进行存储，而我们需要将数据库中的数据查询出来并展示在页面给用户看。上图中的是最基本的查询效果，那么数据库其实是很多的，不可能在将所有的数据在一页进行全部展示，而页面上会有分页展示的效果，如下：



当然上图中的难度字段当我们点击也可以实现排序查询操作。从这个例子我们就可以看出，对于数据库的查询时灵活多变的，需要根据具体的需求来实现，而数据库查询操作也是最重要的操作，所以此部分需要大家重点掌握。

接下来我们先介绍查询的完整语法：

```

1 SELECT
2   字段列表
3 FROM
4   表名列表
5 WHERE
6   条件列表
7 GROUP BY
8   分组字段
9 HAVING
10  分组后条件
11 ORDER BY
12  排序字段
13 LIMIT
14  分页限定

```

为了给大家演示查询的语句，我们需要先准备表及一些数据：

```

1 -- 删除stu表
2 drop table if exists stu;
3
4
5 -- 创建stu表
6 CREATE TABLE stu (
7   id int, -- 编号
8   name varchar(20), -- 姓名
9   age int, -- 年龄
10  sex varchar(5), -- 性别

```

```

11 address varchar(100), -- 地址
12 math double(5,2), -- 数学成绩
13 english double(5,2), -- 英语成绩
14 hire_date date -- 入学时间
15 );
16
17 -- 添加数据
18 INSERT INTO stu(id,NAME,age,sex,address,math,english,hire_date)
19 VALUES
20 (1,'马运',55,'男','杭州',66,78,'1995-09-01'),
21 (2,'马花疼',45,'女','深圳',98,87,'1998-09-01'),
22 (3,'马斯克',55,'男','香港',56,77,'1999-09-02'),
23 (4,'柳白',20,'女','湖南',76,65,'1997-09-05'),
24 (5,'柳青',20,'男','湖南',86,NULL,'1998-09-01'),
25 (6,'刘德华',57,'男','香港',99,99,'1998-09-01'),
26 (7,'张学友',22,'女','香港',99,99,'1998-09-01'),
27 (8,'德玛西亚',18,'男','南京',56,65,'1994-09-02');

```

接下来咱们从最基本的查询语句开始学起。

8.1 基础查询

8.1.1 语法

- 查询多个字段

```

1 | SELECT 字段列表 FROM 表名;
2 | SELECT * FROM 表名; -- 查询所有数据

```

- 去除重复记录

```
1 | SELECT DISTINCT 字段列表 FROM 表名;
```

- 起别名

```
1 | AS: AS 也可以省略
```

8.1.2 练习

- 查询name、age两列

```
1 | select name,age from stu;
```

- 查询所有列的数据，列名的列表可以使用*替代

```
1 | select * from stu;
```

上面语句中的*不建议大家使用，因为在这写*不方便我们阅读sql语句。我们写字段列表的话，可以添加注释对每一个字段进行说明

```

SELECT
    NAME, -- 姓名
    age -- 年龄
FROM
    stu;

```

而在上课期间为了简约课程的时间，老师很多地方都会写*。

- 查询地址信息

```
1 | select address from stu;
```

执行上面语句结果如下：

信息	结果1	概况	状态
address			
	杭州		
	深圳		
	香港		
	湖南		
▶	湖南		
	香港		
	香港		
	南京		

从上面的结果我们可以看到有重复的数据，我们也可以使用 `distinct` 关键字去重重复数据。

- 去除重复记录

```
1 | select distinct address from stu;
```

- 查询姓名、数学成绩、英语成绩。并通过as给math和english起别名（as关键字可以省略）

```
1 | select name,math as 数学成绩,english as 英文成绩 from stu;
2 | select name,math 数学成绩,english 英文成绩 from stu;
```

8.2 条件查询

8.2.1 语法

```
1 | SELECT 字段列表 FROM 表名 WHERE 条件列表;
```

- 条件

条件列表可以使用以下运算符

符号	功能
>	大于
<	小于
>=	大于等于
<=	小于等于
=	等于
<> 或 !=	不等于
BETWEEN ... AND ...	在某个范围之内(都包含)
IN(...)	多选一
LIKE 占位符	模糊查询 _单个任意字符 %多个任意字符
IS NULL	是NULL
IS NOT NULL	不是NULL
AND 或 &&	并且
OR 或	或者
NOT 或 !	非, 不是

8.2.2 条件查询练习

- 查询年龄大于20岁的学员信息

```
1 | select * from stu where age > 20;
```

- 查询年龄大于等于20岁的学员信息

```
1 | select * from stu where age >= 20;
```

- 查询年龄大于等于20岁 并且 年龄 小于等于 30岁的学员信息

```
1 select * from stu where age >= 20 && age <= 30;
2 select * from stu where age >= 20 and age <= 30;
```

上面语句中 `&&` 和 `and` 都表示并且的意思。建议使用 `and`。

也可以使用 `between ... and` 来实现上面需求

```
1 select * from stu where age BETWEEN 20 and 30;
```

- 查询入学日期在'1998-09-01' 到 '1999-09-01' 之间的学员信息

```
1 select * from stu where hire_date BETWEEN '1998-09-01' and '1999-09-01';
```

- 查询年龄等于18岁的学员信息

```
1 select * from stu where age = 18;
```

- 查询年龄不等于18岁的学员信息

```
1 select * from stu where age != 18;
2 select * from stu where age <> 18;
```

- 查询年龄等于18岁 或者 年龄等于20岁 或者 年龄等于22岁的学员信息

```
1 select * from stu where age = 18 or age = 20 or age = 22;
2 select * from stu where age in (18,20 ,22);
```

- 查询英语成绩为 null 的学员信息

null 值的比较不能使用 `=` 或者 `!=`。需要使用 `is` 或者 `is not`

```
1 select * from stu where english = null; -- 这个语句是不行的
2 select * from stu where english is null;
3 select * from stu where english is not null;
```

8.2.3 模糊查询练习

模糊查询使用 `like` 关键字，可以使用通配符进行占位：

(1) `_` : 代表单个任意字符

(2) `%` : 代表任意个数字符

- 查询姓'马'的学员信息

```
1 select * from stu where name like '马%';
```

- 查询第二个字是'花'的学员信息

```
1 select * from stu where name like '_花%';
```

- 查询名字中包含 '德' 的学员信息

```
1 select * from stu where name like '%德%';
```

8.3 排序查询

8.3.1 语法

```
1 SELECT 字段列表 FROM 表名 ORDER BY 排序字段名1 [排序方式1], 排序字段名2 [排序方式2] ...;
```

上述语句中的排序方式有两种，分别是：

- ASC : 升序排列 (默认值)
- DESC : 降序排列

注意：如果有多个排序条件，当前边的条件值一样时，才会根据第二条件进行排序

8.3.2 练习

- 查询学生信息，按照年龄升序排列

```
1 | select * from stu order by age ;
```

- 查询学生信息，按照数学成绩降序排列

```
1 | select * from stu order by math desc ;
```

- 查询学生信息，按照数学成绩降序排列，如果数学成绩一样，再按照英语成绩升序排列

```
1 | select * from stu order by math desc , english asc ;
```

8.4 聚合函数

8.4.1 概念

将一列数据作为一个整体，进行纵向计算。

如何理解呢？假设有如下表

id	name	age	sex	address	math	english	hire_date
1	马运	55	男	杭州	66	78	1995-09-01
2	马花疼	45	女	深圳	98	87	1998-09-01
3	马斯克	55	男	香港	56	77	1999-09-02
4	柳白	20	女	湖南	76	65	1997-09-05
5	柳青	20	男	湖南	86	(Null)	1998-09-01
6	刘德花	57	男	香港	99	99	1998-09-01
7	张学右	22	女	香港	99	99	1998-09-01
8	德玛西亚	18	男	南京	56	65	1994-09-02

现有一需求让我们求表中所有数据的数学成绩的总和。这就是对math字段进行纵向求和。

8.4.2 聚合函数分类

函数名	功能
count(列名)	统计数量 (一般选用不为null的列)
max(列名)	最大值
min(列名)	最小值
sum(列名)	求和
avg(列名)	平均值

8.4.3 聚合函数语法

```
1 | SELECT 聚合函数名(列名) FROM 表;
```

注意：null 值不参与所有聚合函数运算

8.4.4 练习

- 统计班级一共有多少个学生

```
1 | select count(id) from stu;
2 | select count(english) from stu;
```

上面语句根据某个字段进行统计，如果该字段某一行的值为null的话，将不会被统计。所以可以在count(*)来实现。* 表示所有字段数据，一行中也不可能所有的数据都为null，所以建议使用 count(*)

```
1 | select count(*) from stu;
```

- 查询数学成绩的最高分

```
1 | select max(math) from stu;
```

- 查询数学成绩的最低分

```
1 | select min(math) from stu;
```

- 查询数学成绩的总分

```
1 | select sum(math) from stu;
```

- 查询数学成绩的平均分

```
1 | select avg(math) from stu;
```

- 查询英语成绩的最低分

```
1 | select min(english) from stu;
```

8.5 分组查询

8.5.1 语法

```
1 | SELECT 字段列表 FROM 表名 [WHERE 分组前条件限定] GROUP BY 分组字段名 [HAVING 分组后条件过滤];
```

注意：分组之后，查询的字段为聚合函数和分组字段，查询其他字段无任何意义

8.5.2 练习

- 查询男同学和女同学各自的数学平均分

```
1 | select sex, avg(math) from stu group by sex;
```

注意：分组之后，查询的字段为聚合函数和分组字段，查询其他字段无任何意义

```
1 | select name, sex, avg(math) from stu group by sex; -- 这里查询name字段就没有任何意义
```

- 查询男同学和女同学各自的数学平均分，以及各自人数

```
1 | select sex, avg(math), count(*) from stu group by sex;
```

- 查询男同学和女同学各自的数学平均分，以及各自人数，要求：分数低于70分的不参与分组

```
1 | select sex, avg(math), count(*) from stu where math > 70 group by sex;
```

- 查询男同学和女同学各自的数学平均分，以及各自人数，要求：分数低于70分的不参与分组，分组之后人数大于2个的

```
1 | select sex, avg(math), count(*) from stu where math > 70 group by sex having count(*) > 2;
```

where 和 having 区别：

- 执行时机不一样：where 是分组之前进行限定，不满足where条件，则不参与分组，而having是分组之后对结果进行过滤。
- 可判断的条件不一样：where 不能对聚合函数进行判断，having 可以。

8.6 分页查询

如下图所示，大家在很多网站都见过类似的效果，如京东、百度、淘宝等。分页查询是将数据一页一页的展示给用户看，用户也可以通过点击查看下一页的数据。

12	072269	阶段考试	原题	JavaEE	单选题	关于Docker中容器
13	072268	阶段考试	原题	JavaEE	单选题	下列关于RocketMC
14	072267	阶段考试	原题	JavaEE	单选题	对于SpringCloud的
15	072266	阶段考试	原题	JavaEE	单选题	下列哪个不是Rocke
15 ▼ ⏪ ⏴ 第 1 共801页 ⏵ ⏪ ⏴						

接下来我们先说分页查询的语法。

8.6.1 语法

```
1 | SELECT 字段列表 FROM 表名 LIMIT 起始索引 , 查询条目数;
```

注意：上述语句中的起始索引是从0开始

8.6.2 练习

- 从0开始查询，查询3条数据

```
1 | select * from stu limit 0 , 3;
```

- 每页显示3条数据，查询第1页数据

```
1 | select * from stu limit 0 , 3;
```

- 每页显示3条数据，查询第2页数据

```
1 | select * from stu limit 3 , 3;
```

- 每页显示3条数据，查询第3页数据

```
1 | select * from stu limit 6 , 3;
```

从上面的练习推导出起始索引计算公式：

```
1 | 起始索引 = (当前页码 - 1) * 每页显示的条数
```

mysql高级

今日目标

- 掌握约束的使用
- 掌握表关系及建表原则
- 重点掌握多表查询操作
- 掌握事务操作

1. 约束

id	name	age	sex	address	math	english	hire_date
1	马运	55	男	杭州	-5	78	1995-09-01
(Null)	马花疼	45	女	深圳	98	87	1998-09-01
1	马斯克	55	男	香港	56	77	1999-09-02
1	柳白	3000	女	湖南	76	65	1997-09-05
5	柳青	20	男	湖南	86	(Null)	1998-09-01
6	刘德花	57	男	香港	99	99	1998-09-01
7	张学右	22	女	香港	99	99	1998-09-01
8	德玛西亚	18	男	南京	56	65	1994-09-02

上面表中可以看到表中数据存在一些问题：

- id 列一般用以表示数据的唯一性，而上述表中的id为1的有三条数据，并且 马花疼 没有id进行标示
- 柳白 这条数据的age列的数据是3000，而人也不可能活到3000岁
- 马运 这条数据的math数学成绩是-5，而数学学得再不好也不可能出现负分
- 柳青 这条数据的english列（英文成绩）值为null，而成绩即使没考也得是0分

针对上述数据问题，我们就可以从数据库层面在添加数据的时候进行限制，这个就是约束。

1.1 概念

- 约束是作用于表中列上的规则，用于限制加入表的数据

例如：我们可以给id列加约束，让其值不能重复，不能为null值。

- 约束的存在保证了数据库中数据的正确性、有效性和完整性

添加约束可以在添加数据的时候就限制不正确的数据，年龄是3000，数学成绩是-5分这样无效的数据，进而保障数据的完整性。

1.2 分类

- 非空约束：关键字是 NOT NULL**

保证列中所有的数据不能有null值。

例如：id列在添加 马花疼 这条数据时就不能添加成功。

- 唯一约束：关键字是 UNIQUE**

保证列中所有数据各不相同。

例如：id列中三条数据的值都是1，这样的数据在添加时是绝对不允许的。

- 主键约束：关键字是 PRIMARY KEY**

主键是一行数据的唯一标识，要求非空且唯一。一般我们都会给每张表添加一个主键列用来唯一标识数据。

例如：上图表中id就可以作为主键，来标识每条数据。那么这样就要求数据中id的值不能重复，不能为null值。

- 检查约束：关键字是 CHECK**

保证列中的值满足某一条件。

例如：我们可以给age列添加一个范围，最低年龄可以设置为1，最大年龄就可以设置为300，这样的数据才更合理些。

注意：MySQL不支持检查约束。

这样是不是就没办法保证年龄在指定的范围内了？从数据库层面不能保证，以后可以在java代码中进行限制，一样也可以实现要求。

- **默认约束：关键字是 DEFAULT**

保存数据时，未指定值则采用默认值。

例如：我们在给english列添加该约束，指定默认值是0，这样在添加数据时没有指定具体值时就会采用默认给定的0。

- **外键约束：关键字是 FOREIGN KEY**

外键用来让两个表的数据之间建立链接，保证数据的一致性和完整性。

外键约束现在可能还不太好理解，后面我们会重点进行讲解。

1.3 非空约束

- 概念

非空约束用于保证列中所有数据不能有NULL值

- 语法

- 添加约束

```
1 -- 创建表时添加非空约束
2 CREATE TABLE 表名(
3     列名 数据类型 NOT NULL,
4     ...
5 );
6
```

```
1 -- 建完表后添加非空约束
2 ALTER TABLE 表名 MODIFY 字段名 数据类型 NOT NULL;
```

- 删除约束

```
1 ALTER TABLE 表名 MODIFY 字段名 数据类型;
```

1.4 唯一约束

- 概念

唯一约束用于保证列中所有数据各不相同

- 语法

- 添加约束

```
1 -- 创建表时添加唯一约束
2 CREATE TABLE 表名(
3     列名 数据类型 UNIQUE [AUTO_INCREMENT],
4     -- AUTO_INCREMENT: 当不指定值时自动增长
5     ...
6 );
7 CREATE TABLE 表名(
8     列名 数据类型,
9     ...
10    [CONSTRAINT] [约束名称] UNIQUE(列名)
11 );
```

```
1 -- 建完表后添加唯一约束
2 ALTER TABLE 表名 MODIFY 字段名 数据类型 UNIQUE;
```

- 删除约束

```
1 ALTER TABLE 表名 DROP INDEX 字段名;
```

1.5 主键约束

- 概念

主键是一行数据的唯一标识，要求非空且唯一

一张表只能有一个主键

- 语法

- 添加约束

```
1 -- 创建表时添加主键约束
2 CREATE TABLE 表名(
3     列名 数据类型 PRIMARY KEY [AUTO_INCREMENT],
4     ...
5 );
6 CREATE TABLE 表名(
7     列名 数据类型,
8     [CONSTRAINT] [约束名称] PRIMARY KEY(列名)
9 );
10
```

```
1 -- 建完表后添加主键约束
2 ALTER TABLE 表名 ADD PRIMARY KEY(字段名);
```

- 删除约束

```
1 ALTER TABLE 表名 DROP PRIMARY KEY;
```

1.6 默认约束

- 概念

保存数据时，未指定值则采用默认值

- 语法

- 添加约束

```
1 -- 创建表时添加默认约束
2 CREATE TABLE 表名(
3     列名 数据类型 DEFAULT 默认值,
4     ...
5 );
```

```
1 -- 建完表后添加默认约束
2 ALTER TABLE 表名 ALTER 列名 SET DEFAULT 默认值;
```

- 删除约束

```
1 ALTER TABLE 表名 ALTER 列名 DROP DEFAULT;
```

1.7 约束练习

根据需求，为表添加合适的约束

```
1 -- 员工表
2 CREATE TABLE emp (
3     id INT,    -- 员工id，主键且自增长
4     ename VARCHAR(50), -- 员工姓名，非空且唯一
5     joindate DATE, -- 入职日期，非空
6     salary DOUBLE(7,2), -- 工资，非空
7     bonus DOUBLE(7,2) -- 奖金，如果没有将近默认为0
8 );
```

上面一定给出了具体的要求，我们可以根据要求创建这张表，并为每一列添加对应的约束。建表语句如下：

```
1 DROP TABLE IF EXISTS emp;
2
3 -- 员工表
4 CREATE TABLE emp (
5   id INT PRIMARY KEY, -- 员工id, 主键且自增长
6   ename VARCHAR(50) NOT NULL UNIQUE, -- 员工姓名, 非空并且唯一
7   joindate DATE NOT NULL, -- 入职日期, 非空
8   salary DOUBLE(7,2) NOT NULL, -- 工资, 非空
9   bonus DOUBLE(7,2) DEFAULT 0 -- 奖金, 如果没有奖金默认为0
10 );
```

通过上面语句可以创建带有约束的 `emp` 表，约束能不能发挥作用呢。接下来我们一一进行验证，先添加一条没有问题的数据

```
1 | INSERT INTO emp(id,ename,joindate,salary,bonus) values(1,'张三','1999-11-11',8800,5000);
```

- 验证主键约束，非空且唯一

```
1 | INSERT INTO emp(id,ename,joindate,salary,bonus) values(null,'张三','1999-11-11',8800,5000);
```

执行结果如下：

```
信息 概况 状态
[SQL]-- 演示主键约束: 非空且唯一
INSERT INTO emp(id,ename,joindate,salary,bonus) values(null,'张三','1999-11-11',8800,5000);
[Err] 1048 - Column 'id' cannot be null
```

从上面的结果可以看到，字段 `id` 不能为null。那我们重新添加一条数据，如下：

```
1 | INSERT INTO emp(id,ename,joindate,salary,bonus) values(1,'张三','1999-11-11',8800,5000);
```

执行结果如下：

```
信息 概况 状态
[SQL]INSERT INTO emp(id,ename,joindate,salary,bonus) values(1,'张三','1999-11-11',8800,5000);
[Err] 1062 - Duplicate entry '1' for key 'PRIMARY'
```

从上面结果可以看到，1这个值重复了。所以主键约束是用来限制数据非空且唯一的。那我们再添加一条符合要求的数据

```
1 | INSERT INTO emp(id,ename,joindate,salary,bonus) values(2,'李四','1999-11-11',8800,5000);
```

执行结果如下：

```
信息 概况 状态
[SQL]INSERT INTO emp(id,ename,joindate,salary,bonus) values(2,'李四','1999-11-11',8800,5000);
受影响的行: 1
时间: 0.009s
```

- 验证非空约束

```
1 | INSERT INTO emp(id,ename,joindate,salary,bonus) values(3,null,'1999-11-11',8800,5000);
```

执行结果如下：

```
信息 概况 状态
[SQL]-- 演示非空约束
INSERT INTO emp(id,ename,joindate,salary,bonus) values(3,null,'1999-11-11',8800,5000);
[Err] 1048 - Column 'ename' cannot be null
```

从上面结果可以看到，`ename` 字段的非空约束生效了。

- 验证唯一约束

```
1 | INSERT INTO emp(id,ename,joindate,salary,bonus) values(3,'李四','1999-11-11',8800,5000);
```

执行结果如下：

```
信息 概况 状态
[SQL]-- 演示唯一约束
INSERT INTO emp(id,ename,joindate,salary,bonus) values(3,'李四','1999-11-11',8800,5000);
[Err] 1062 - Duplicate entry '李四' for key 'ename'
```

从上面结果可以看到，`ename` 字段的唯一约束生效了。

- 验证默认约束

```
1 | INSERT INTO emp(id,ename,joindate,salary) values(3,'王五','1999-11-11',8800);
```

执行完上面语句后查询表中数据，如下图可以看到王五这条数据的bonus列就有了默认值0。

信息 结果1 概况 状态				
id	ename	joindate	salary	bonus
1	张三	1999-11-11	8800	5000
2	李四	1999-11-11	8800	5000
3	王五	1999-11-11	8800	0

注意：默认约束只有在不给值时才会采用默认值。如果给了null，那值就是null值。

如下：

```
1 | INSERT INTO emp(id,ename,joindate,salary,bonus) values(4,'赵六','1999-11-11',8800,null);
```

执行完上面语句后查询表中数据，如下图可以看到赵六这条数据的bonus列的值是null。

信息 结果1 概况 状态				
id	ename	joindate	salary	bonus
1	张三	1999-11-11	8800	5000
2	李四	1999-11-11	8800	5000
3	王五	1999-11-11	8800	0
4	赵六	1999-11-11	8800	(Null)

- 验证自动增长：auto_increment 当列是数字类型 并且唯一约束

重新创建 emp 表，并给id列添加自动增长

```
1 | -- 员工表
2 | CREATE TABLE emp (
3 |   id INT PRIMARY KEY auto_increment, -- 员工id, 主键且自增长
4 |   ename VARCHAR(50) NOT NULL UNIQUE, -- 员工姓名, 非空并且唯一
5 |   joindate DATE NOT NULL , -- 入职日期, 非空
6 |   salary DOUBLE(7,2) NOT NULL , -- 工资, 非空
7 |   bonus DOUBLE(7,2) DEFAULT 0 -- 奖金, 如果没有奖金默认为0
8 | );
```

接下来给emp添加数据，分别验证不给id列添加值以及给id列添加null值，id列的值会不会自动增长：

```
1 | INSERT INTO emp(ename,joindate,salary,bonus) values('赵六','1999-11-11',8800,null);
2 | INSERT INTO emp(id,ename,joindate,salary,bonus) values(null,'赵六2','1999-11-11',8800,null);
3 | INSERT INTO emp(id,ename,joindate,salary,bonus) values(null,'赵六3','1999-11-11',8800,null);
```

1.8 外键约束

1.8.1 概述

外键用来让两个表的数据之间建立链接，保证数据的一致性和完整性。

如何理解上面的概念呢？如下图有两张表，员工表和部门表：

emp 员工表				dept 部门表		
id	name	age	dep_id	id	dep_name	addr
1	张三	20	1	1	研发部	广州
2	李四	20	1	2	销售部	深圳
3	王五	20	1			
4	赵六	20	2			
5	孙七	22	2			
6	周八	18	2			

员工表中的dep_id字段是部门表的id字段关联，也就是说1号学生张三属于1号部门研发部的员工。现在我要删除1号部门，就会出现错误的数据（员工表中属于1号部门的数据）。而我们上面说的两张表的关系只是我们认为它们有关系，此时需要通过外键让这两张表产生数据库层面的关系，这样你要删除部门表中的1号部门的数据将无法删除。

1.8.2 语法

- 添加外键约束

```
1 -- 创建表时添加外键约束
2 CREATE TABLE 表名(
3     列名 数据类型,
4     ...
5     [CONSTRAINT] [外键名称] FOREIGN KEY(外键列名) REFERENCES 主表(主表列名)
6 );
```

```
1 -- 建完表后添加外键约束
2 ALTER TABLE 表名 ADD CONSTRAINT 外键名称 FOREIGN KEY (外键字段名称) REFERENCES 主表名称(主表列名称);
```

- 删除外键约束

```
1 ALTER TABLE 表名 DROP FOREIGN KEY 外键名称;
```

1.8.3 练习

根据上述语法创建员工表和部门表，并添加上外键约束：

```
1 -- 删除表
2 DROP TABLE IF EXISTS emp;
3 DROP TABLE IF EXISTS dept;
4
5 -- 部门表
6 CREATE TABLE dept(
7     id int primary key auto_increment,
8     dep_name varchar(20),
9     addr varchar(20)
10 );
11 -- 员工表
12 CREATE TABLE emp(
13     id int primary key auto_increment,
14     name varchar(20),
15     age int,
16     dep_id int,
17
18     -- 添加外键 dep_id, 关联 dept 表的id主键
19     CONSTRAINT fk_emp_dept FOREIGN KEY(dep_id) REFERENCES dept(id)
20 );
```

添加数据

```
1 -- 添加 2 个部门
2 insert into dept(dep_name,addr) values
3 ('研发部', '广州'), ('销售部', '深圳');
4
5 -- 添加员工, dep_id 表示员工所在的部门
6 INSERT INTO emp (NAME, age, dep_id) VALUES
7 ('张三', 20, 1),
8 ('李四', 20, 1),
9 ('王五', 20, 1),
10 ('赵六', 20, 2),
11 ('孙七', 22, 2),
12 ('周八', 18, 2);
```

此时删除 研发部 这条数据，会发现无法删除。

删除外键

```
1 alter table emp drop FOREIGN key fk_emp_dept;
```

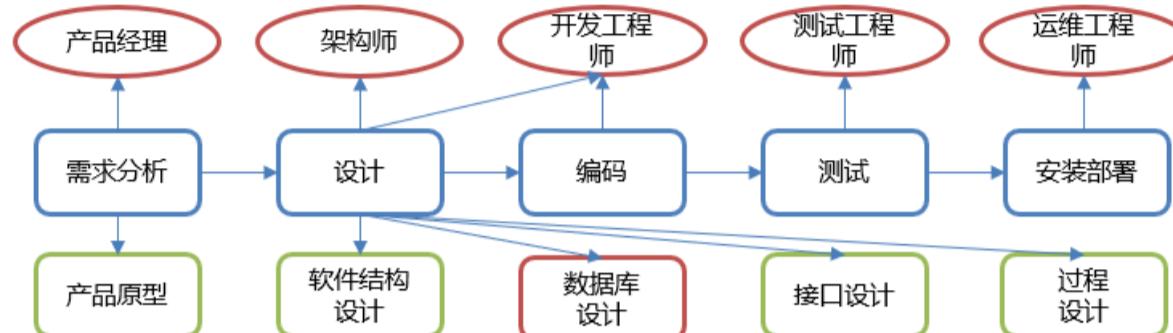
重新添加外键

```
1 | alter table emp add CONSTRAINT fk_emp_dept FOREIGN key(dep_id) REFERENCES dept(id);
```

2. 数据库设计

2.1 数据库设计简介

- 软件的研发步骤



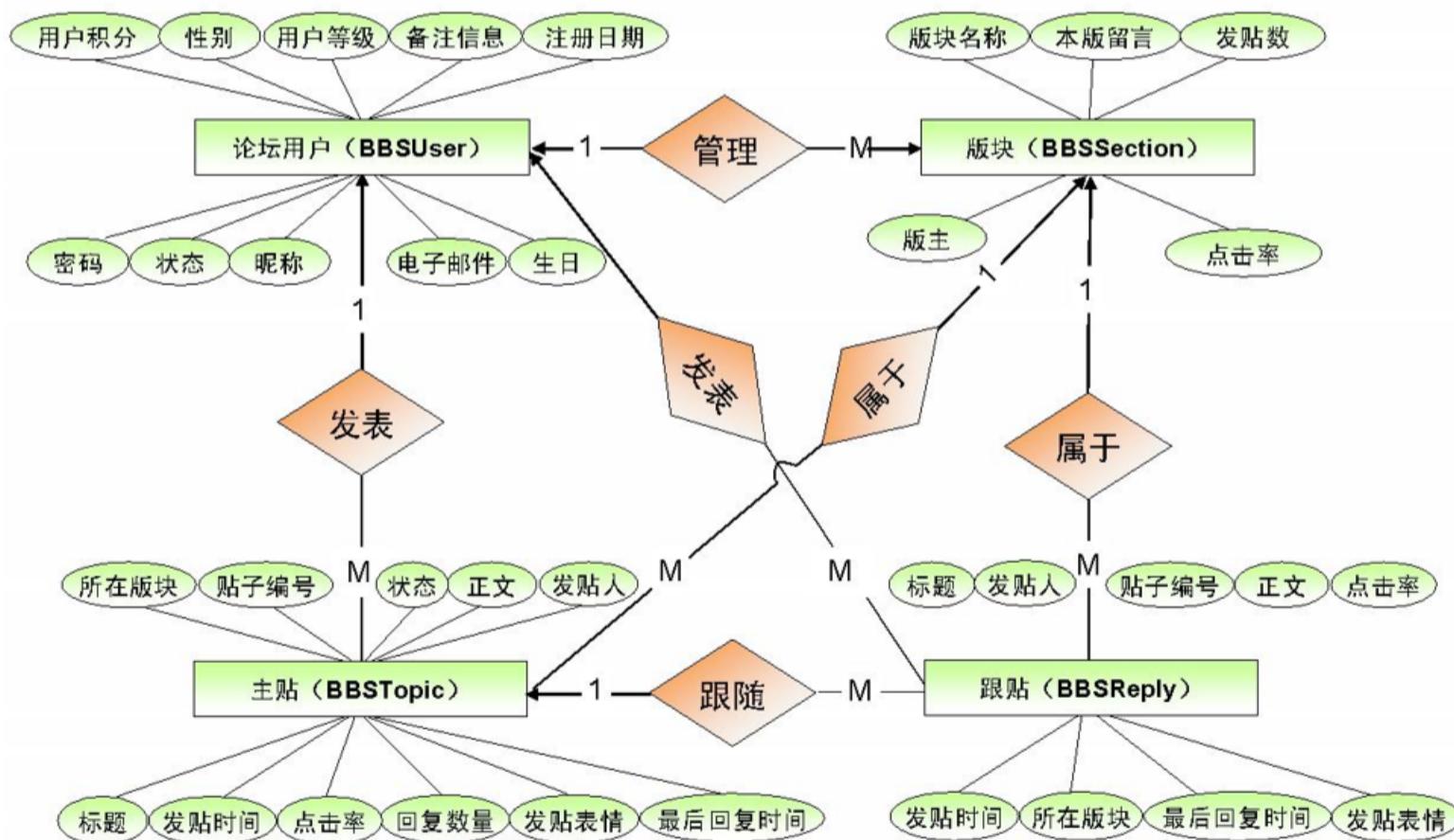
- 数据库设计概念

- 数据库设计就是根据业务系统的具体需求，结合我们所选用的DBMS，为这个业务系统构造出最优的数据存储模型。
- 建立数据库中的**表结构**以及**表与表之间的关联关系**的过程。
- 有哪些表？表里有哪些字段？表和表之间有什么关系？

- 数据库设计的步骤

- 需求分析（数据是什么？数据具有哪些属性？数据与属性的特点是什么）
- 逻辑分析（通过ER图对数据库进行逻辑建模，不需要考虑我们所选用的数据库管理系统）

如下图就是ER(Entity/Relation)图：

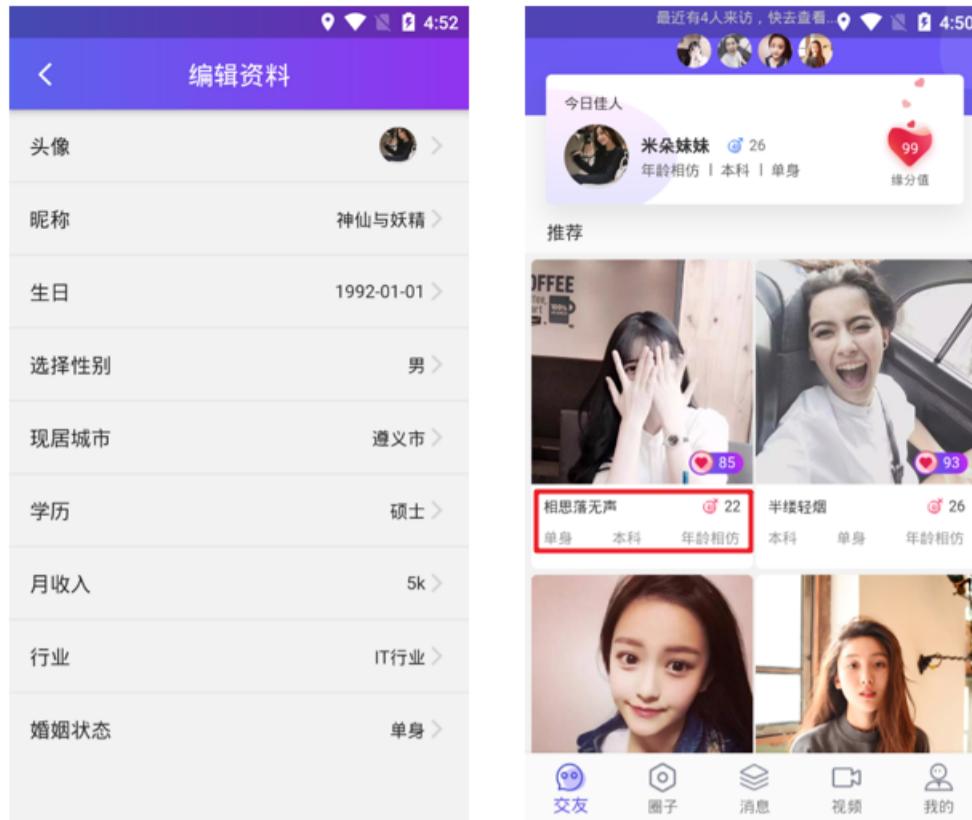


- 物理设计（根据数据库自身的特点把逻辑设计转换为物理设计）
- 维护设计（1.对新的需求进行建表；2.表优化）

- 表关系

- 一对多

- 如：用户 和 用户详情
- 一对多关系多用于表拆分，将一个实体中经常使用的字段放一张表，不经常使用的字段放另一张表，用于提升查询性能



上图左边是用户的详细信息，而我们真正在展示用户信息时最长用的则是上图右边红框所示，所以我们会将详细信息查分成两周那个表。

- 一对多

- 如：部门 和 员工
- 一个部门对应多个员工，一个员工对应一个部门。如下图：

emp 员工表				dept 部门表		
<u>id</u>	<u>name</u>	<u>age</u>	<u>dep_id</u>	<u>id</u>	<u>dep_name</u>	<u>addr</u>
1	张三	20	1	1	研发部	广州
2	李四	20	1	2	销售部	深圳
3	王五	20	1			
4	赵六	20	2			
5	孙七	22	2			
6	周八	18	2			

- 多对多

- 如：商品 和 订单
- 一个商品对应多个订单，一个订单包含多个商品。如下图：

填写并核对订单信息

收货人信息
新增收货地址

北京朝阳区四环到五环之间北京市朝阳区...

更多地址 ▾

支付方式
返回修改购物车

货到付款
在线支付
分期付款
公司转账
邮局汇款

配送方式
对应商品

京东快递
上门自提(荐)

配送时间：预计 5月16日[周六] 09:00-15:00 送达 修改

送货清单
返回修改购物车

商家：京东自营

	艾威博尔 (Everpower) 203301 高纯度铅锡合金制造 简装焊锡丝	¥7.90	x1
有货			
7天无理由退货			

	华为 Ascend P6 (P6-T00)黑色移 动3G手机	¥1199.00	x1
有货			
7天无理由退货			

发票信息
提交订单

普通发票（电子）
个人
明细
修改

应付总额：¥1206.90

2.2 表关系(一对多)

- 一对多
 - 如: 部门 和 员工
 - 一个部门对应多个员工, 一个员工对应一个部门。

- 实现方式

在多的一方建立外键, 指向一的一方的主键

- 案例

我们还是以 `员工表` 和 `部门表` 举例:

tb_emp 员工表 M			1 tb_dept 部门表		
id	name	age	id	name	addr
1	张三	23	1	财务部	北京
2	李四	24	2	市场部	上海
3	王五	25	3	研发部	成都

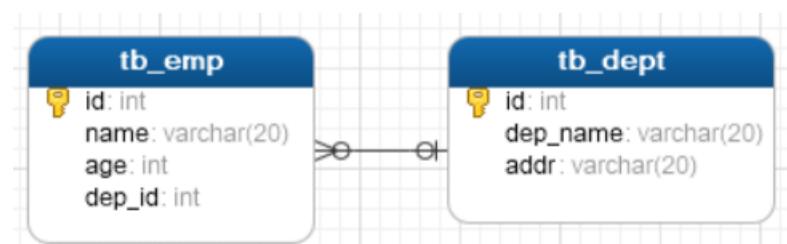
经过分析发现, 员工表属于多的一方, 而部门表属于一的一方, 此时我们会在员工表中添加一列 (`dep_id`) , 指向于部门表的主键 (`id`) :

tb_emp 员工表 M				1 tb_dept 部门表		
id	name	age	dep_id	id	name	addr
1	张三	23	1	1	财务部	北京
2	李四	24	1	2	市场部	上海
3	王五	25	2	3	研发部	成都

建表语句如下:

```
1 -- 删除表
2 DROP TABLE IF EXISTS tb_emp;
3 DROP TABLE IF EXISTS tb_dept;
4
5 -- 部门表
6 CREATE TABLE tb_dept(
7     id int primary key auto_increment,
8     dep_name varchar(20),
9     addr varchar(20)
10 );
11 -- 员工表
12 CREATE TABLE tb_emp(
13     id int primary key auto_increment,
14     name varchar(20),
15     age int,
16     dep_id int,
17
18     -- 添加外键 dep_id, 关联 dept 表的id主键
19     CONSTRAINT fk_emp_dept FOREIGN KEY(dep_id) REFERENCES tb_dept(id)
20 );
```

查看表结构模型图:



2.3 表关系(多对多)

- 多对多
 - 如: 商品 和 订单
 - 一个商品对应多个订单, 一个订单包含多个商品
- 实现方式

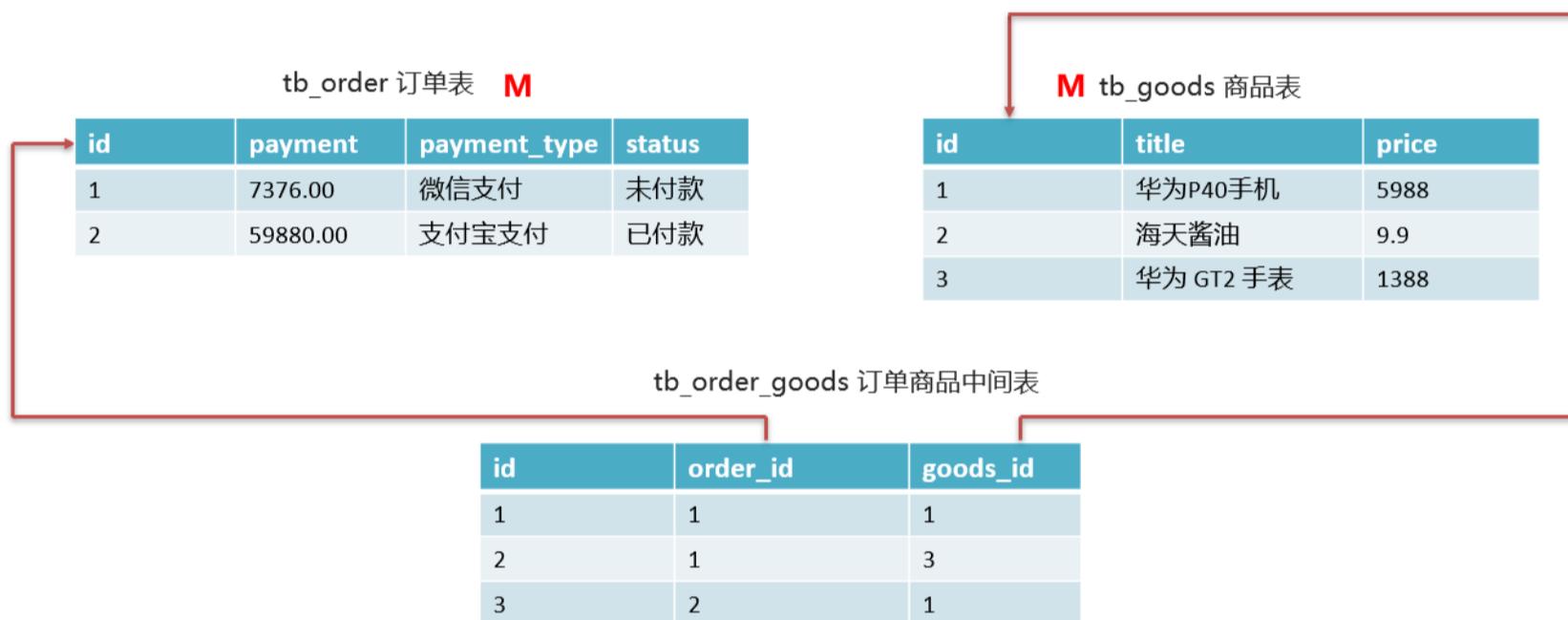
建立第三张中间表, 中间表至少包含两个外键, 分别关联两方主键

- 案例

我们以 `订单表` 和 `商品表` 举例:

tb_order 订单表 M				M tb_goods 商品表		
id	payment	payment_type	status	id	title	price
1	7376.00	微信支付	未付款	1	华为P40手机	5988
2	59880.00	支付宝支付	已付款	2	海天酱油	9.9
				3	华为 GT2 手表	1388

经过分析发现, 订单表和商品表都属于多的一方, 此时需要创建一个中间表, 在中间表中添加订单表的外键和商品表的外键指向两张表的主键:



建表语句如下:

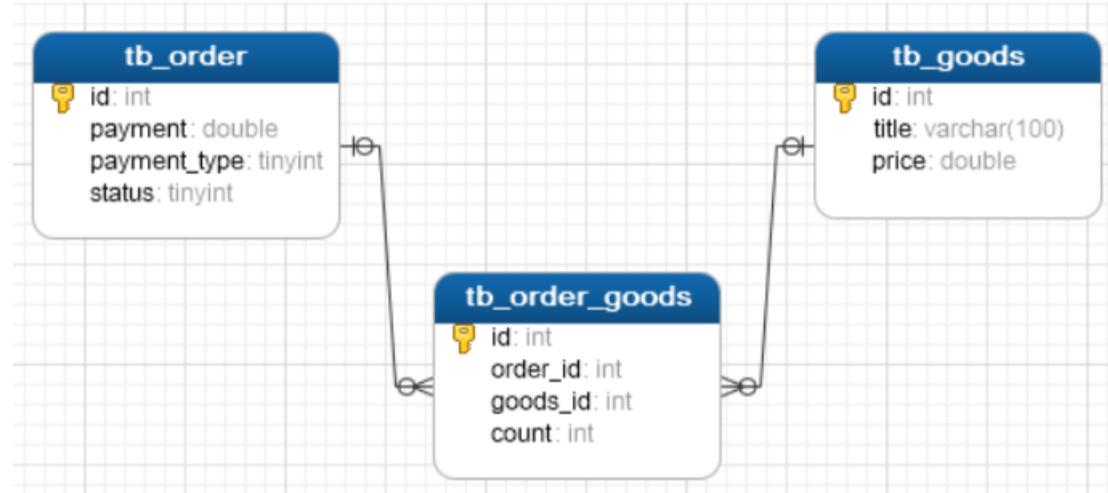
```
1 -- 删除表
2 DROP TABLE IF EXISTS tb_order_goods;
3 DROP TABLE IF EXISTS tb_order;
4 DROP TABLE IF EXISTS tb_goods;
5
6 -- 订单表
7 CREATE TABLE tb_order(
8     id int primary key auto_increment,
9     payment double(10,2),
10    payment_type TINYINT,
11    status TINYINT
12 );
13
14 -- 商品表
15 CREATE TABLE tb_goods(
16     id int primary key auto_increment,
17     title varchar(100),
18     price double(10,2)
19 );
20
21 -- 订单商品中间表
22 CREATE TABLE tb_order_goods(
23     id int primary key auto_increment,
24     order_id int,
25     goods_id int,
26     count int
27 );
```

```

28
29 -- 建完表后，添加外键
30 alter table tb_order_goods add CONSTRAINT fk_order_id FOREIGN key(order_id) REFERENCES
  tb_order(id);
31 alter table tb_order_goods add CONSTRAINT fk_goods_id FOREIGN key(goods_id) REFERENCES
  tb_goods(id);

```

查看表结构模型图：



2.4 表关系(一对)

- 一对
 - 如：用户 和 用户详情
 - 一对关系多用于表拆分，将一个实体中经常使用的字段放一张表，不经常使用的字段放另一张表，用于提升查询性能
- 实现方式

在任意一方加入外键，关联另一方主键，并且设置外键为唯一(UNIQUE)

- 案例

我们以 `用户表` 举例：

`tb_user` 用户表

<code>id</code>	<code>photo</code>	<code>nickname</code>	<code>age</code>	<code>gender</code>	<code>city</code>	<code>edu</code>	<code>income</code>	<code>status</code>	<code>desc</code>
1	a.jpg	一场梦	23	女	广州	硕士	3000	单身	...
2	b.png	风清扬	35	男	湖北	本科	30000	离异	...
3	c.jpg	赵云	41	男	河南	本科	40000	单身	...

而在真正使用过程中发现 `id`、`photo`、`nickname`、`age`、`gender` 字段比较常用，此时就可以将这张表拆分成两张表。

<code>tb_user</code> 用户表 1						1 <code>tb_user_desc</code> 用户详情表					
<code>id</code>	<code>photo</code>	<code>nickname</code>	<code>age</code>	<code>gender</code>	<code>desc_id</code>	<code>id</code>	<code>city</code>	<code>edu</code>	<code>income</code>	<code>status</code>	<code>desc</code>
1	a.jpg	一场梦	23	女	1	1	广州	硕士	3000	单身	...
2	b.png	风清扬	35	男	2	2	湖北	本科	30000	离异	...
3	c.jpg	赵云	41	男	3	3	河南	本科	40000	单身	...

建表语句如下：

```

1 create table tb_user_desc (
2     id int primary key auto_increment,
3     city varchar(20),
4     edu varchar(10),
5     income int,
6     status char(2),
7     des varchar(100)
8 );
9
10 create table tb_user (
11     id int primary key auto_increment,
12     photo varchar(100),
13     nickname varchar(50),

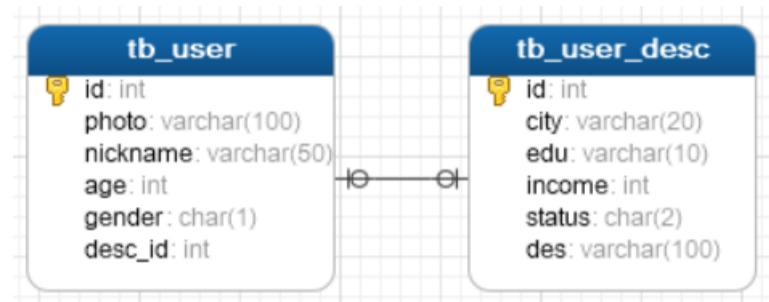
```

```

14     age int,
15     gender char(1),
16     desc_id int unique,
17     -- 添加外键
18     CONSTRAINT fk_user_desc FOREIGN KEY(desc_id) REFERENCES tb_user_desc(id)
19 );

```

查看表结构模型图：



2.5 数据库设计案例

根据下图设计表及表和表之间的关系：

我只在乎你

又名: 留聲經典復刻版
表演者: 邓丽君
流派: 流行
专辑类型: 专辑
介质: CD
发行时间: 1987-01-02
出版者: 环球
唱片数: 1
条形码: 4718622110798
其他版本: 我只在乎你 (全部)

9.4 ★★★★★ 2764人评价

5星 77.1%
4星 20.3%
3星 2.4%
2星 0.2%
1星 0.1%

想听 在听 听过 评价: ☆☆☆☆☆
写短评 写乐评 加入豆列 分享到 推荐

简介 ······
邓丽君在1987年推出的唱片专集《我只在乎你》中，有三首歌的词作者是“桃丽莎”。其实，桃丽莎即是邓丽君自己（英文名TERESA的中译）。根据我手中的资料，邓丽君作的词并不多，虽然她确曾向媒体表示“最大的心愿是出一张一脚踏的唱片”——即由自己包办下全部的词曲和制作，但是因意外去世而没能实现。但是，在此专集中竟有三首之多，不能不令人关注。大体上说，这三首歌具有两种风格，一为写实，一为浪漫。《非龙非影》以现代汉语与古汉语混合，歌词的意境悲凉，心态哀伤，而且隐含着非比寻常的寓意。笔者愿在此写出来就教于方家。

一般来说，邓丽君的歌词都是浅显清新易于理解的，因而更显得这首歌的另类。从字面上看，似乎是歌者在感叹自己感情上的坎坷和时光的飞逝，“龙”、“丽”二字也容易让人联想到与她有过一段情的成龙。其实，却是别有一番含意的，试解如下：
“把... (展开全部)

曲目 ······
01. 酒醉的探戈
02. 像故事般温柔
03. 命运之川
04. 爱人
05. 午夜微风
06. 夏日圣诞
07. 非龙非影
08. 不着痕迹
09. 心路过黄昏
10. 我只在乎你

短评 ······ (全部 1064 条)
热门 / 最新 / 好友
/ 我来说两句

不擅长 ★★★★★ 2010-10-24
我只在乎你 (个人特别~ http://www.xiami.com/album/B000)

五月之星 ★★★★★ 2007-10-30
按刘姐的说法，一张专辑有几首好听的歌就够了。当然要力荐，更何况好听的《我只在乎你》，

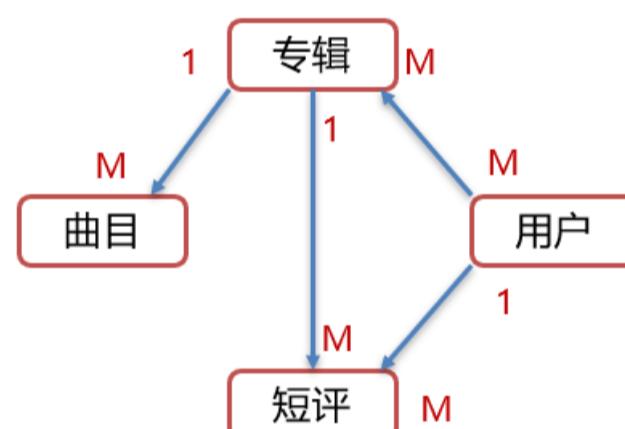
经过分析，我们分为 专辑表 | 曲目表 | 短评表 | 用户表 4张表。



一个专辑可以有多个曲目，一个曲目只能属于某一张专辑，所以专辑表和曲目表的关系是**一对多**。

一个专辑可以被多个用户进行评论，一个用户可以对多个专辑进行评论，所以专辑表和用户表的关系是**多对多**。

一个用户可以发多个短评，一个短评只能是某一个人发的，所以用户表和短评表的关系是**一对多**。



3. 多表查询

多表查询顾名思义就是从多张表中一次性的查询出我们想要的数据。我们通过具体的sql给他们演示，先准备环境

```

1 DROP TABLE IF EXISTS emp;
2 DROP TABLE IF EXISTS dept;
3
4
5 # 创建部门表
6 CREATE TABLE dept(
7     did INT PRIMARY KEY AUTO_INCREMENT,
8     dname VARCHAR(20)
9 );
10
11 # 创建员工表
12 CREATE TABLE emp (
13     id INT PRIMARY KEY AUTO_INCREMENT,
14     name VARCHAR(10),
15     gender CHAR(1), -- 性别
16     salary DOUBLE, -- 工资
17     join_date DATE, -- 入职日期
18     dep_id INT,
19     FOREIGN KEY (dep_id) REFERENCES dept(did) -- 外键，关联部门表(部门表的主键)

```

```

20 );
21 -- 添加部门数据
22 INSERT INTO dept (dNAME) VALUES ('研发部'),('市场部'),('财务部'),('销售部');
23 -- 添加员工数据
24 INSERT INTO emp(NAME,gender,salary,join_date,dep_id) VALUES
25 ('孙悟空','男',7200,'2013-02-24',1),
26 ('猪八戒','男',3600,'2010-12-02',2),
27 ('唐僧','男',9000,'2008-08-08',2),
28 ('白骨精','女',5000,'2015-10-07',3),
29 ('蜘蛛精','女',4500,'2011-03-14',1),
30 ('小白龙','男',2500,'2011-02-14',null);

```

执行下面的多表查询语句

```
1 | select * from emp , dept; -- 从emp和dept表中查询所有的字段数据
```

结果如下：

信息	结果1	概况	状态				
id	NAME	gender	salary	join_date	dep_id	did	dname
▶ 1	孙悟空	男	7200	2013-02-24		1	1 研发部
1	孙悟空	男	7200	2013-02-24		1	2 市场部
1	孙悟空	男	7200	2013-02-24		1	3 财务部
1	孙悟空	男	7200	2013-02-24		1	4 销售部
2	猪八戒	男	3600	2010-12-02		2	1 研发部
2	猪八戒	男	3600	2010-12-02		2	2 市场部
2	猪八戒	男	3600	2010-12-02		2	3 财务部

从上面的结果我们看到有一些无效的数据，如 孙悟空 这个员工属于1号部门，但也同时关联的2、3、4号部门。所以我们要通过限制员工表中的 `dep_id` 字段的值和部门表 `did` 字段的值相等来消除这些无效的数据，

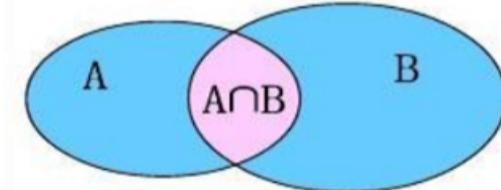
```
1 | select * from emp , dept where emp.dep_id = dept.did;
```

执行后结果如下：

信息	结果1	概况	状态				
id	NAME	gender	salary	join_date	dep_id	did	dname
1	孙悟空	男	7200	2013-02-24		1	研发部
5	蜘蛛精	女	4500	2011-03-14		1	研发部
2	猪八戒	男	3600	2010-12-02		2	市场部
3	唐僧	男	9000	2008-08-08		2	市场部
▶ 4	白骨精	女	5000	2015-10-07		3	财务部

上面语句就是连接查询，那么多表查询都有哪些呢？

- 连接查询



- 内连接查询：相当于查询AB交集数据
- 外连接查询
 - 左外连接查询：相当于查询A表所有数据和交集部门数据
 - 右外连接查询：相当于查询B表所有数据和交集部分数据
- 子查询

3.1 内连接查询

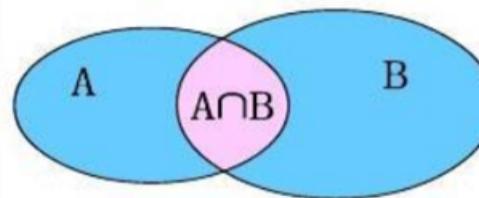
- 语法

```

1 -- 隐式内连接
2 SELECT 字段列表 FROM 表1,表2... WHERE 条件;
3
4 -- 显示内连接
5 SELECT 字段列表 FROM 表1 [INNER] JOIN 表2 ON 条件;

```

内连接相当于查询 A B 交集数据



- 案例

- 隐式内连接

```

1 SELECT
2   *
3 FROM
4   emp,
5   dept
6 WHERE
7   emp.dep_id = dept.did;

```

执行上述语句结果如下:

信息 结果1 概况 状态							
	id	NAME	gender	salary	join_date	dep_id	did dname
▶	1	孙悟空	男	7200	2013-02-24	1	1 研发部
	2	猪八戒	男	3600	2010-12-02	2	2 市场部
	3	唐僧	男	9000	2008-08-08	2	2 市场部
	4	白骨精	女	5000	2015-10-07	3	3 财务部
	5	蜘蛛精	女	4500	2011-03-14	1	1 研发部

- 查询 emp 的 name, gender, dept 表的 dname

```

1 SELECT
2   emp. NAME,
3   emp.gender,
4   dept.dname
5 FROM
6   emp,
7   dept
8 WHERE
9   emp.dep_id = dept.did;

```

执行语句结果如下:

信息 结果1 概况 状态		
	NAME	gender dname
▶	孙悟空	男 研发部
	猪八戒	男 市场部
	唐僧	男 市场部
	白骨精	女 财务部
	蜘蛛精	女 研发部

上面语句中使用表名指定字段所属有点麻烦, sql也支持给表指别名, 上述语句可以改进为

```

1 SELECT
2     t1. NAME,
3     t1.gender,
4     t2.dname
5 FROM
6     emp t1,
7     dept t2
8 WHERE
9     t1.dep_id = t2.did;

```

- 显式内连接

```

1 select * from emp inner join dept on emp.dep_id = dept.did;
2 -- 上面语句中的inner可以省略，可以书写为如下语句
3 select * from emp join dept on emp.dep_id = dept.did;

```

执行结果如下：

信息	结果1	概况	状态					
id	NAME	gender	salary	join_date	dep_id	did	dname	
1	孙悟空	男	7200	2013-02-24	1	1	研发部	
2	猪八戒	男	3600	2010-12-02	2	2	市场部	
3	唐僧	男	9000	2008-08-08	2	2	市场部	
4	白骨精	女	5000	2015-10-07	3	3	财务部	
5	蜘蛛精	女	4500	2011-03-14	1	1	研发部	

3.2 外连接查询

- 语法

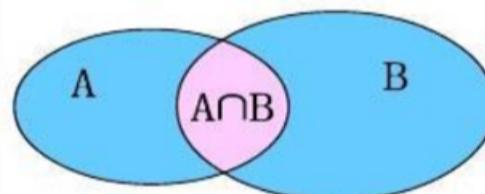
```

1 -- 左外连接
2 SELECT 字段列表 FROM 表1 LEFT [OUTER] JOIN 表2 ON 条件;
3
4 -- 右外连接
5 SELECT 字段列表 FROM 表1 RIGHT [OUTER] JOIN 表2 ON 条件;

```

左外连接：相当于查询A表所有数据和交集部分数据

右外连接：相当于查询B表所有数据和交集部分数据



- 案例

- 查询emp表所有数据和对应的部门信息（左外连接）

```

1 select * from emp left join dept on emp.dep_id = dept.did;

```

执行语句结果如下：

信息	结果1	概况	状态					
id	NAME	gender	salary	join_date	dep_id	did	dname	
1	孙悟空	男	7200	2013-02-24	1	1	研发部	
5	蜘蛛精	女	4500	2011-03-14	1	1	研发部	
2	猪八戒	男	3600	2010-12-02	2	2	市场部	
3	唐僧	男	9000	2008-08-08	2	2	市场部	
4	白骨精	女	5000	2015-10-07	3	3	财务部	
6	小白龙	男	2500	2011-02-14	(Null)	(Null)	(Null)	

结果显示查询到了左表（emp）中所有的数据及两张表能关联的数据。

- 查询dept表所有数据和对应的员工信息（右外连接）

```
1 | select * from emp right join dept on emp.dep_id = dept.did;
```

执行语句结果如下：

信息	结果1	概况	状态					
id	NAME	gender	salary	join_date	dep_id	did	dname	
1	孙悟空	男	7200	2013-02-24	1	1	研发部	
2	猪八戒	男	3600	2010-12-02	2	2	市场部	
3	唐僧	男	9000	2008-08-08	2	2	市场部	
4	白骨精	女	5000	2015-10-07	3	3	财务部	
5	蜘蛛精	女	4500	2011-03-14	1	1	研发部	
(Null)	(Null)	(Null)	(Null)	(Null)	(Null)	4	销售部	

结果显示查询到了右表 (dept) 中所有的数据及两张表能关联的数据。

要查询出部门表中所有的数据，也可以通过左外连接实现，只需要将两个表的位置进行互换：

```
1 | select * from dept left join emp on emp.dep_id = dept.did;
```

3.3 子查询

- 概念

查询中嵌套查询，称嵌套查询为子查询。

什么是查询中嵌套查询呢？我们通过一个例子来看：

需求：查询工资高于猪八戒的员工信息。

来实现这个需求，我们就可以通过二步实现，第一步：先查询出来猪八戒的工资

```
1 | select salary from emp where name = '猪八戒'
```

第二步：查询工资高于猪八戒的员工信息

```
1 | select * from emp where salary > 3600;
```

第二步中的3600可以通过第一步的sql查询出来，所以将3600用第一步的sql语句进行替换

```
1 | select * from emp where salary > (select salary from emp where name = '猪八戒');
```

这就是查询语句中嵌套查询语句。

- 子查询根据查询结果不同，作用不同

- 子查询语句结果是单行单列，子查询语句作为条件值，使用 = != > < 等进行条件判断
- 子查询语句结果是多行单列，子查询语句作为条件值，使用 in 等关键字进行条件判断
- 子查询语句结果是多行多列，子查询语句作为虚拟表

- 案例

- 查询 '财务部' 和 '市场部' 所有的员工信息

```
1 | -- 查询 '财务部' 或者 '市场部' 所有的员工的部门did
2 | select did from dept where dname = '财务部' or dname = '市场部';
3 |
4 | select * from emp where dep_id in (select did from dept where dname = '财务部' or dname =
  '市场部');
```

- 查询入职日期是 '2011-11-11' 之后的员工信息和部门信息

```
1 | -- 查询入职日期是 '2011-11-11' 之后的员工信息
2 | select * from emp where join_date > '2011-11-11' ;
3 | -- 将上面语句的结果作为虚拟表和dept表进行内连接查询
4 | select * from (select * from emp where join_date > '2011-11-11') t1, dept where
  t1.dep_id = dept.did;
```

3.4 案例

- 环境准备：

```
1 DROP TABLE IF EXISTS emp;
2 DROP TABLE IF EXISTS dept;
3 DROP TABLE IF EXISTS job;
4 DROP TABLE IF EXISTS salarygrade;
5
6 -- 部门表
7 CREATE TABLE dept (
8     did INT PRIMARY KEY PRIMARY KEY, -- 部门id
9     dname VARCHAR(50), -- 部门名称
10    loc VARCHAR(50) -- 部门所在地
11 );
12
13 -- 职务表, 职务名称, 职务描述
14 CREATE TABLE job (
15     id INT PRIMARY KEY,
16     jname VARCHAR(20),
17     description VARCHAR(50)
18 );
19
20 -- 员工表
21 CREATE TABLE emp (
22     id INT PRIMARY KEY, -- 员工id
23     ename VARCHAR(50), -- 员工姓名
24     job_id INT, -- 职务id
25     mgr INT, -- 上级领导
26     joindate DATE, -- 入职日期
27     salary DECIMAL(7,2), -- 工资
28     bonus DECIMAL(7,2), -- 奖金
29     dept_id INT, -- 所在部门编号
30     CONSTRAINT emp_jobid_ref_job_id_fk FOREIGN KEY (job_id) REFERENCES job (id),
31     CONSTRAINT emp_deptid_ref_dept_id_fk FOREIGN KEY (dept_id) REFERENCES dept (id)
32 );
33 -- 工资等级表
34 CREATE TABLE salarygrade (
35     grade INT PRIMARY KEY, -- 级别
36     losalary INT, -- 最低工资
37     hisalary INT -- 最高工资
38 );
39
40 -- 添加4个部门
41 INSERT INTO dept(did,dname,loc) VALUES
42 (10,'教研部','北京'),
43 (20,'学工部','上海'),
44 (30,'销售部','广州'),
45 (40,'财务部','深圳');
46
47 -- 添加4个职务
48 INSERT INTO job (id, jname, description) VALUES
49 (1, '董事长', '管理整个公司, 接单'),
50 (2, '经理', '管理部门员工'),
51 (3, '销售员', '向客人推销产品'),
52 (4, '文员', '使用办公软件');
53
54
55 -- 添加员工
56 INSERT INTO emp(id,ename,job_id,mgr,joindate,salary,bonus,dept_id) VALUES
57 (1001,'孙悟空',4,1004,'2000-12-17','8000.00',NULL,20),
58 (1002,'卢俊义',3,1006,'2001-02-20','16000.00','3000.00',30),
59 (1003,'林冲',3,1006,'2001-02-22','12500.00','5000.00',30),
60 (1004,'唐僧',2,1009,'2001-04-02','29750.00',NULL,20),
61 (1005,'李逵',4,1006,'2001-09-28','12500.00','14000.00',30),
```

```

62 (1006,'宋江',2,1009,'2001-05-01','28500.00',NULL,30),
63 (1007,'刘备',2,1009,'2001-09-01','24500.00',NULL,10),
64 (1008,'猪八戒',4,1004,'2007-04-19','30000.00',NULL,20),
65 (1009,'罗贯中',1,NULL,'2001-11-17','50000.00',NULL,10),
66 (1010,'吴用',3,1006,'2001-09-08','15000.00','0.00',30),
67 (1011,'沙僧',4,1004,'2007-05-23','11000.00',NULL,20),
68 (1012,'李逵',4,1006,'2001-12-03','9500.00',NULL,30),
69 (1013,'小白龙',4,1004,'2001-12-03','30000.00',NULL,20),
70 (1014,'关羽',4,1007,'2002-01-23','13000.00',NULL,10);

71
72
73 -- 添加5个工资等级
74 INSERT INTO salarygrade(grade,losalary,hisalary) VALUES
75 (1,7000,12000),
76 (2,12010,14000),
77 (3,14010,20000),
78 (4,20010,30000),
79 (5,30010,99990);

```

• 需求

1. 查询所有员工信息。查询员工编号，员工姓名，工资，职务名称，职务描述

```

1 /*
2      分析:
3          1. 员工编号，员工姓名，工资 信息在emp 员工表中
4          2. 职务名称，职务描述 信息在 job 职务表中
5          3. job 职务表 和 emp 员工表 是 一对多的关系 emp.job_id = job.id
6 */
7 -- 方式一：隐式内连接
8 SELECT
9     emp.id,
10    emp.ename,
11    emp.salary,
12    job.jname,
13    job.description
14 FROM
15    emp,
16    job
17 WHERE
18     emp.job_id = job.id;
19
20 -- 方式二：显式内连接
21 SELECT
22     emp.id,
23     emp.ename,
24     emp.salary,
25     job.jname,
26     job.description
27 FROM
28     emp
29 INNER JOIN job ON emp.job_id = job.id;

```

2. 查询员工编号，员工姓名，工资，职务名称，职务描述，部门名称，部门位置

```

1 /*
2      分析:
3          1. 员工编号，员工姓名，工资 信息在emp 员工表中
4          2. 职务名称，职务描述 信息在 job 职务表中
5          3. job 职务表 和 emp 员工表 是 一对多的关系 emp.job_id = job.id
6
7          4. 部门名称，部门位置 来自于 部门表 dept
8          5. dept 和 emp 一对多关系 dept.id = emp.dept_id
9 */
10
11 -- 方式一：隐式内连接

```

```

12 SELECT
13   emp.id,
14   emp.ename,
15   emp.salary,
16   job.jname,
17   job.description,
18   dept.dname,
19   dept.loc
20 FROM
21   emp,
22   job,
23   dept
24 WHERE
25   emp.job_id = job.id
26   and dept.id = emp.dept_id
27 ;
28
29 -- 方式二：显式内连接
30 SELECT
31   emp.id,
32   emp.ename,
33   emp.salary,
34   job.jname,
35   job.description,
36   dept.dname,
37   dept.loc
38 FROM
39   emp
40 INNER JOIN job ON emp.job_id = job.id
41 INNER JOIN dept ON dept.id = emp.dept_id

```

3. 查询员工姓名，工资，工资等级

```

1 /*
2 分析：
3   1. 员工姓名，工资 信息在emp 员工表中
4   2. 工资等级 信息在 salarygrade 工资等级表中
5   3. emp.salary >= salarygrade.losalary and emp.salary <= salarygrade.hisalary
6 */
7 SELECT
8   emp.ename,
9   emp.salary,
10  t2.*
11 FROM
12  emp,
13  salarygrade t2
14 WHERE
15  emp.salary >= t2.losalary
16 AND emp.salary <= t2.hisalary

```

4. 查询员工姓名，工资，职务名称，职务描述，部门名称，部门位置，工资等级

```

1 /*
2 分析：
3   1. 员工编号，员工姓名，工资 信息在emp 员工表中
4   2. 职务名称，职务描述 信息在 job 职务表中
5   3. job 职务表 和 emp 员工表 是一对多的关系 emp.job_id = job.id
6
7   4. 部门名称，部门位置 来自于 部门表 dept
8   5. dept 和 emp 一对多关系 dept.id = emp.dept_id
9   6. 工资等级 信息在 salarygrade 工资等级表中
10  7. emp.salary >= salarygrade.losalary and emp.salary <= salarygrade.hisalary
11 */
12 SELECT
13   emp.id,

```

```

14     emp.ename,
15     emp.salary,
16     job.jname,
17     job.description,
18     dept.dname,
19     dept.loc,
20     t2.grade
21 FROM
22     emp
23 INNER JOIN job ON emp.job_id = job.id
24 INNER JOIN dept ON dept.id = emp.dept_id
25 INNER JOIN salarygrade t2 ON emp.salary BETWEEN t2.losalary and t2.hisalary;

```

5. 查询出部门编号、部门名称、部门位置、部门人数

```

1  /*
2  分析:
3      1. 部门编号、部门名称、部门位置 来自于部门 dept 表
4      2. 部门人数: 在emp表中 按照dept_id 进行分组, 然后count(*)统计数量
5      3. 使用子查询, 让部门表和分组后的表进行内连接
6 */
7 -- 根据部门id分组查询每一个部门id和员工数
8 select dept_id, count(*) from emp group by dept_id;
9
10 SELECT
11     dept.id,
12     dept.dname,
13     dept.loc,
14     t1.count
15 FROM
16     dept,
17     (
18         SELECT
19             dept_id,
20             count(*) count
21         FROM
22             emp
23         GROUP BY
24             dept_id
25     ) t1
26 WHERE
27     dept.id = t1.dept_id

```

4, 事务

4.1 概述

数据库的事务 (Transaction) 是一种机制、一个操作序列，包含了**一组数据库操作命令**。

事务把所有的命令作为一个整体一起向系统提交或撤销操作请求，即这一组数据库命令**要么同时成功，要么同时失败**。

事务是一个不可分割的工作逻辑单元。

这些概念不好理解，接下来举例说明，如下图有一张表

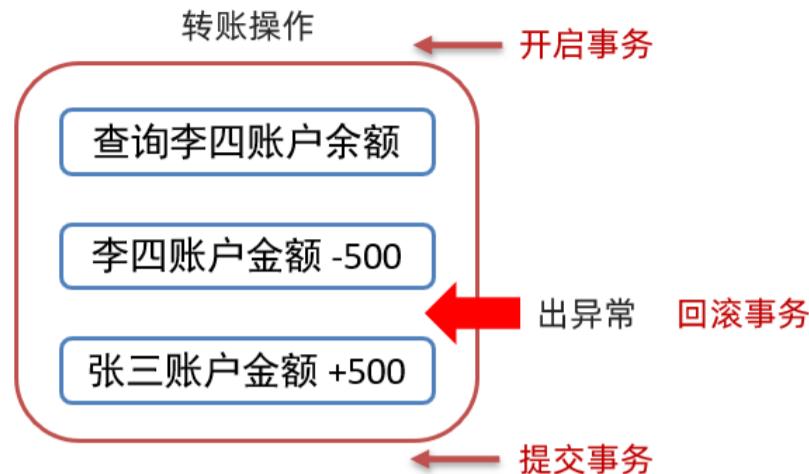
id	name	money
1	张三	1000
2	李四	500

张三和李四账户中各有100块钱，现李四需要转换500块钱给张三，具体的转账操作为

- 第一步：查询李四账户余额

- 第二步：从李四账户金额 -500
- 第三步：给张三账户金额 +500

现在假设在转账过程中第二步完成后出现了异常第三步没有执行，就会造成李四账户金额少了500，而张三金额并没有多500；这样的系统是有问题的。如果解决呢？使用事务可以解决上述问题。



从上图可以看到在转账前开启事务，如果出现了异常回滚事务，三步正常执行就提交事务，这样就可以完美解决问题。

4.2 语法

- 开启事务

```

1 START TRANSACTION;
2 或者
3 BEGIN;

```

- 提交事务

```
1 commit;
```

- 回滚事务

```
1 rollback;
```

4.3 代码验证

- 环境准备

```

1 DROP TABLE IF EXISTS account;
2
3 -- 创建账户表
4 CREATE TABLE account(
5     id int PRIMARY KEY auto_increment,
6     name varchar(10),
7     money double(10,2)
8 );
9
10 -- 添加数据
11 INSERT INTO account(name,money) values('张三',1000),('李四',1000);

```

- 不加事务演示问题

```

1 -- 转账操作
2 -- 1. 查询李四账户金额是否大于500
3
4 -- 2. 李四账户 -500
5 UPDATE account set money = money - 500 where name = '李四';
6
7 出现异常了... -- 此处不是注释，在整体执行时会出问题，后面的sql则不执行
8 -- 3. 张三账户 +500
9 UPDATE account set money = money + 500 where name = '张三';

```

整体执行结果肯定会出现问题，我们查询账户表中数据，发现李四账户少了500。

id	name	money
1	张三	1000
2	李四	500

- 添加事务sql如下：

```
1 -- 开启事务
2 BEGIN;
3 -- 转账操作
4 -- 1. 查询李四账户金额是否大于500
5
6 -- 2. 李四账户 -500
7 UPDATE account set money = money - 500 where name = '李四';
8
9 出现异常了... -- 此处不是注释，在整体执行时会出问题，后面的sql则不执行
10 -- 3. 张三账户 +500
11 UPDATE account set money = money + 500 where name = '张三';
12
13 -- 提交事务
14 COMMIT;
15
16 -- 回滚事务
17 ROLLBACK;
```

上面sql中的执行成功进选择执行提交事务，而出现问题则执行回滚事务的语句。以后我们肯定不可能这样操作，而是在java中进行操作，在java中可以抓取异常，没出现异常提交事务，出现异常回滚事务。

4.4 事务的四大特征

- 原子性 (Atomicity) : 事务是不可分割的最小操作单位，要么同时成功，要么同时失败
- 一致性 (Consistency) : 事务完成时，必须使所有的数据都保持一致状态
- 隔离性 (Isolation) : 多个事务之间，操作的可见性
- 持久性 (Durability) : 事务一旦提交或回滚，它对数据库中的数据的改变就是永久的

说明：

mysql中事务是自动提交的。

也就是说我们不添加事务执行sql语句，语句执行完毕会自动的提交事务。

可以通过下面语句查询默认提交方式：

```
1 SELECT @@autocommit;
```

查询到的结果是1 则表示自动提交，结果是0表示手动提交。当然也可以通过下面语句修改提交方式

```
1 set @@autocommit = 0;
```

JDBC

今日目标

- 掌握JDBC的CRUD
- 理解JDBC中各个对象的作用
- 掌握Druid的使用

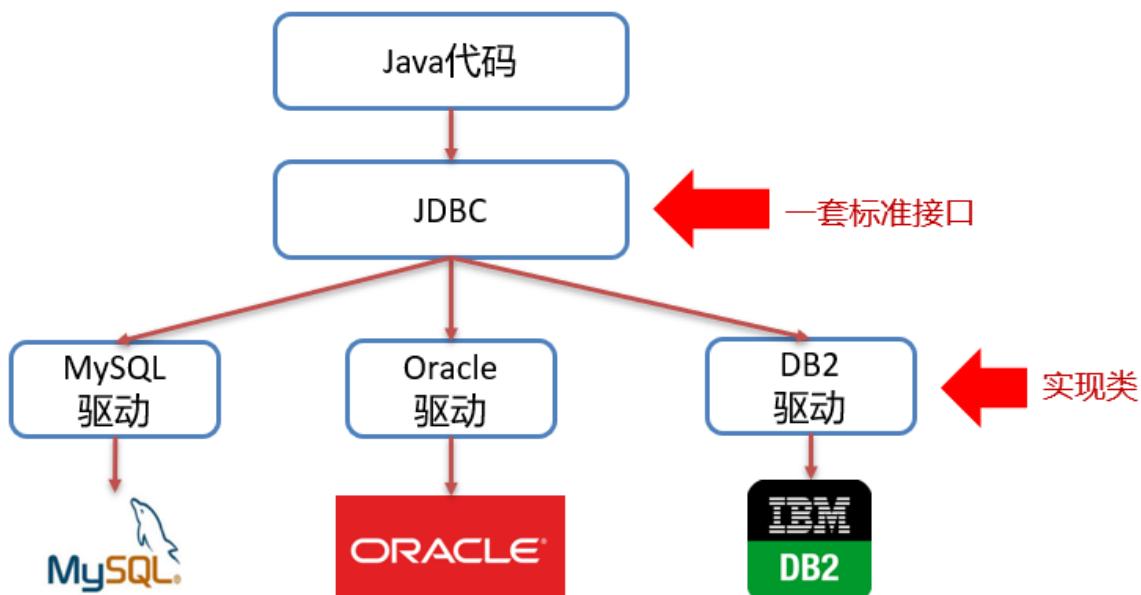
1. JDBC概述

在开发中我们使用的是java语言，那么势必要通过java语言操作数据库中的数据。这就是接下来要学习的JDBC。

1.1 JDBC概念

JDBC 就是使用Java语言操作关系型数据库的一套API

全称：(Java DataBase Connectivity) Java 数据库连接



我们开发的同一套Java代码是无法操作不同的关系型数据库，因为每一个关系型数据库的底层实现细节都不一样。如果这样，问题就很大了，在公司中可以在开发阶段使用的是MySQL数据库，而上线时公司最终选用oracle数据库，我们就需要对代码进行大批量修改，这显然并不是我们想看到的。我们要做到的是同一套Java代码操作不同的关系型数据库，而此时sun公司就指定了一套标准接口（JDBC），JDBC中定义了所有操作关系型数据库的规则。众所周知接口是无法直接使用的，我们需要使用接口的实现类，而这套实现类（称之为：驱动）就由各自的数据库厂商给出。

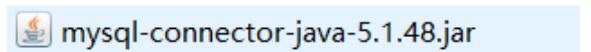
1.2 JDBC本质

- 官方（sun公司）定义的一套操作所有关系型数据库的规则，即接口
- 各个数据库厂商去实现这套接口，提供数据库驱动jar包
- 我们可以使用这套接口（JDBC）编程，真正执行的代码是驱动jar包中的实现类

1.3 JDBC好处

- 各数据库厂商使用相同的接口，Java代码不需要针对不同数据库分别开发
- 可随时替换底层数据库，访问数据库的Java代码基本不变

以后编写操作数据库的代码只需要面向JDBC（接口），操作哪几个关系型数据库就需要导入该数据库的驱动包，如需要操作MySQL数据库，就需要再项目中导入MySQL数据库的驱动包。如下图就是MySQL驱动包



mysql-connector-java-5.1.48.jar

2, JDBC快速入门

先来看看通过Java操作数据库的流程

Java代码



第一步：编写Java代码

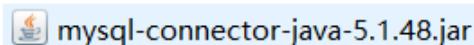
第二步：Java代码将SQL发送到MySQL服务端

第三步：MySQL服务端接收到SQL语句并执行该SQL语句

第四步：将SQL语句执行的结果返回给Java代码

2.1 编写代码步骤

- 创建工程，导入驱动jar包



- 注册驱动

```
Class.forName("com.mysql.jdbc.Driver");
```

- 获取连接

```
Connection conn =  
DriverManager.getConnection(url, username,  
password);
```

Java代码需要发送SQL给MySQL服务端，就需要先建立连接

- 定义SQL语句

```
String sql = "update...";
```

- 获取执行SQL对象

执行SQL语句需要SQL执行对象，而这个执行对象就是 Statement对象

```
Statement stmt = conn.createStatement();
```

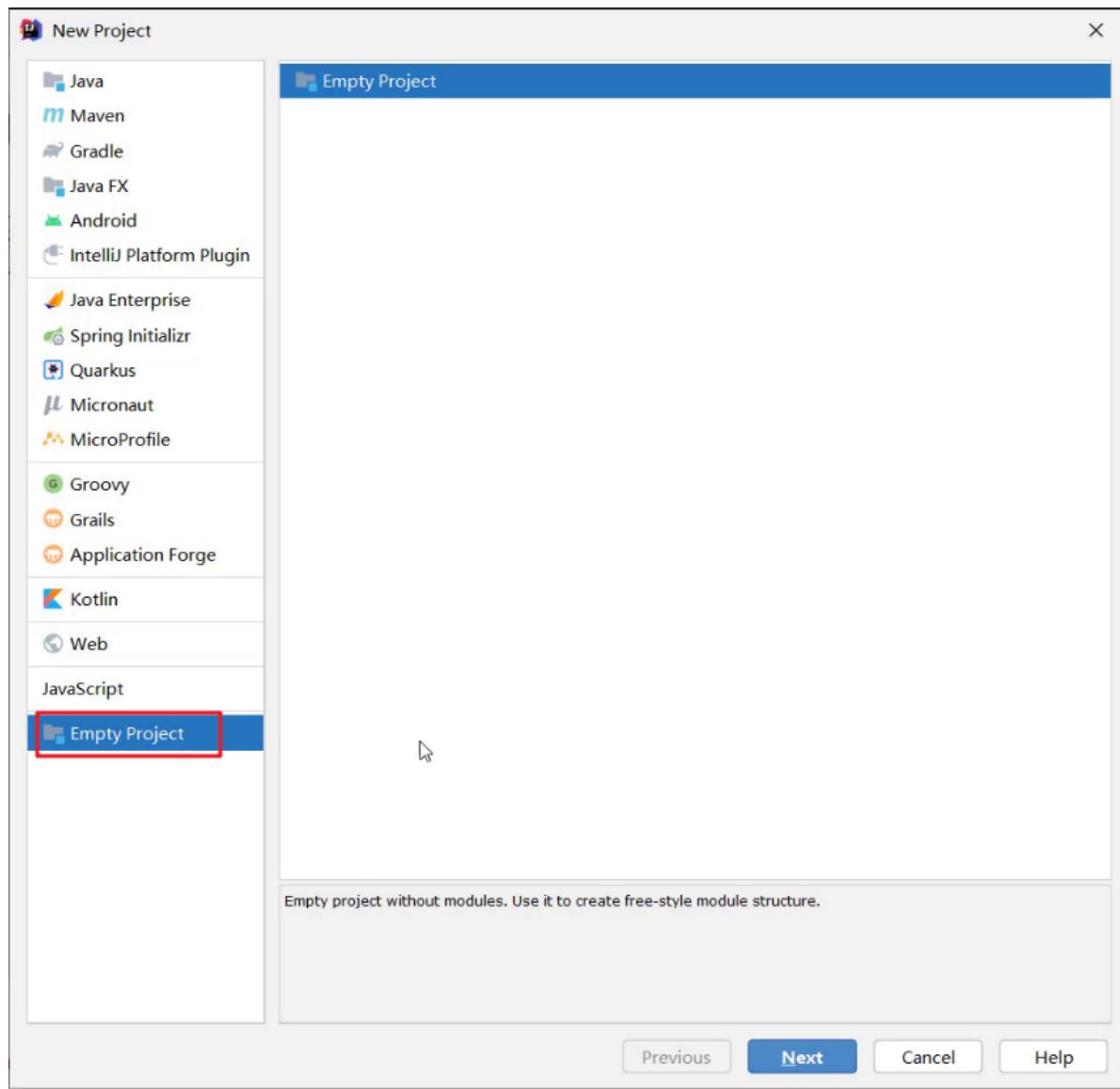
- 执行SQL

```
stmt.executeUpdate(sql);
```

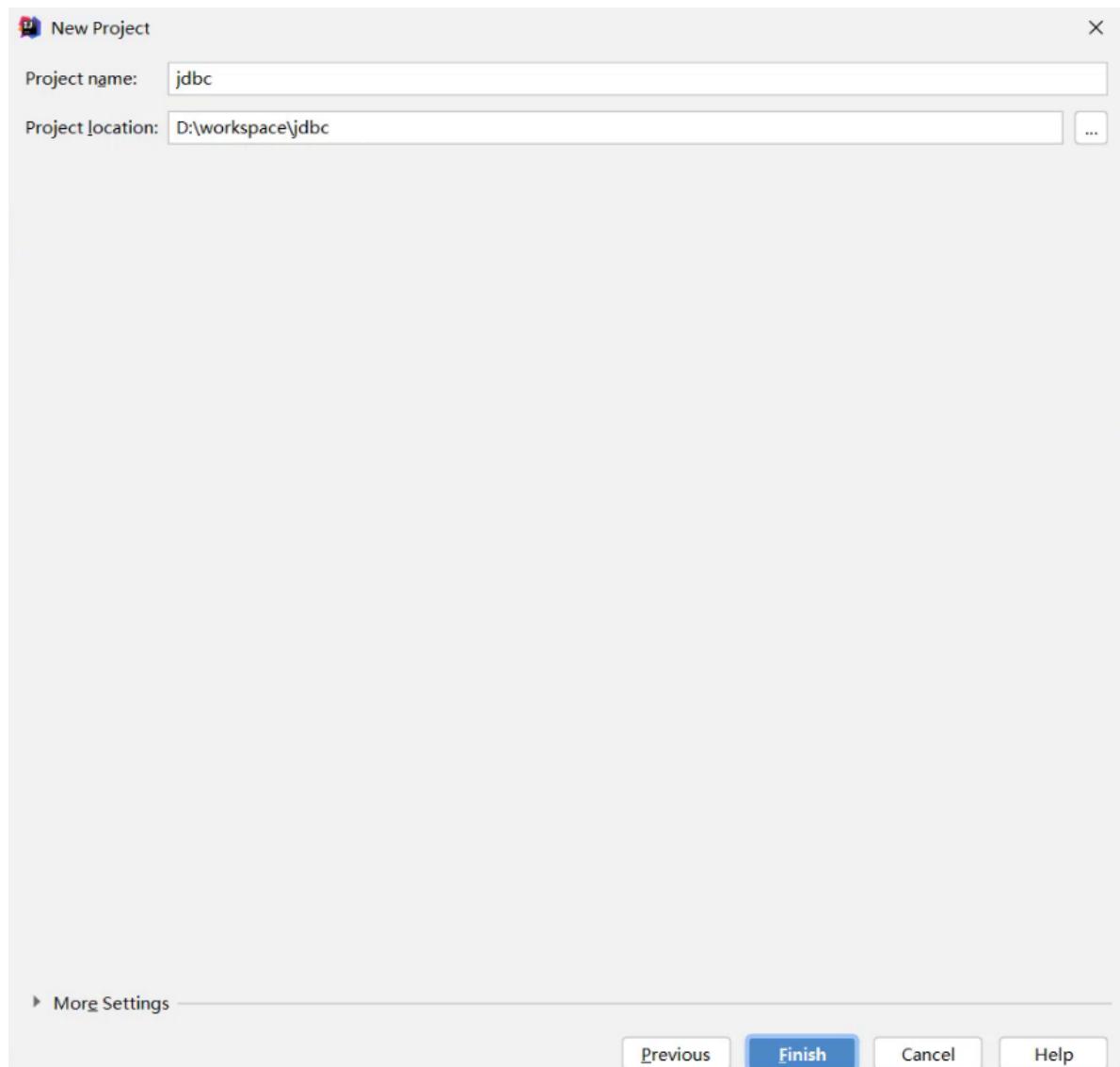
- 处理返回结果
- 释放资源

2.2 具体操作

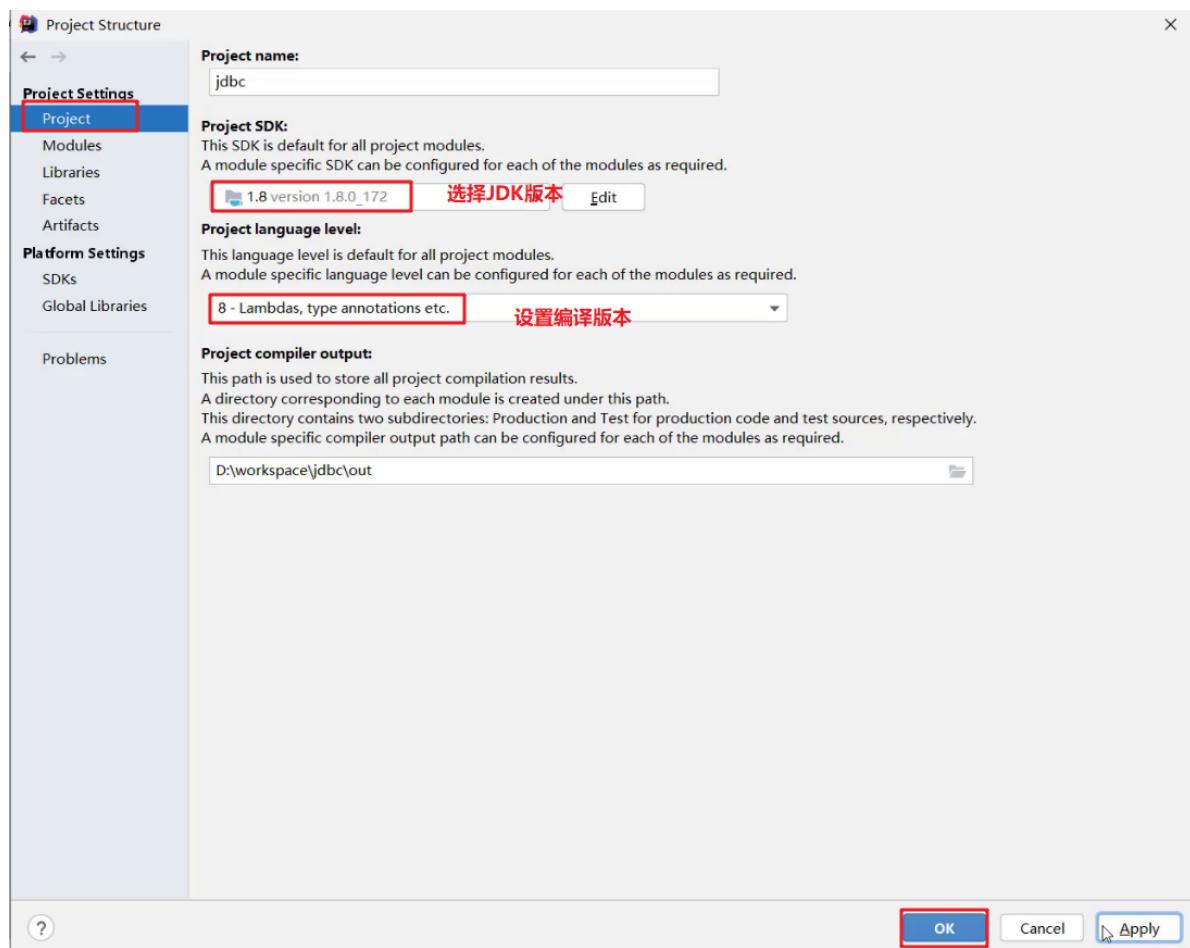
- 创建新的空的项目



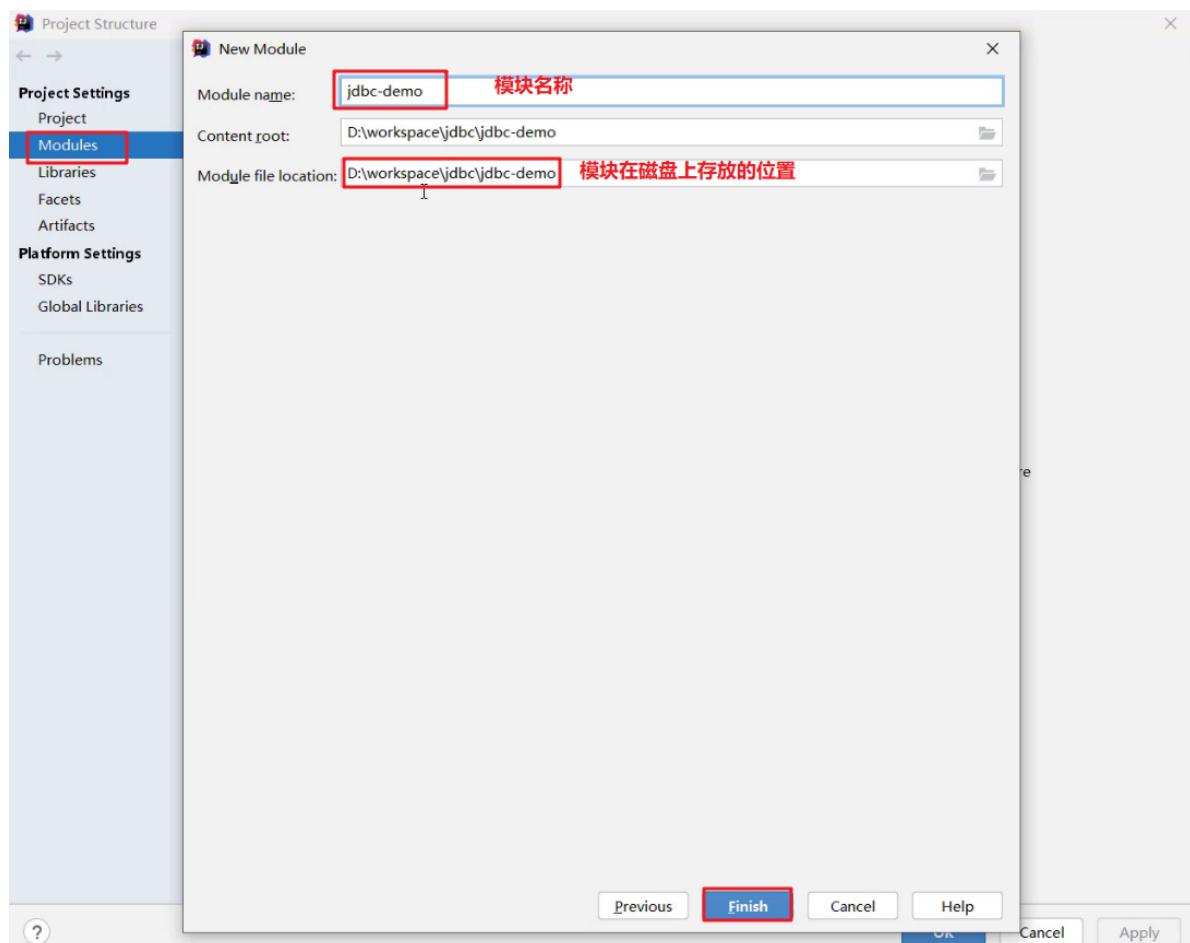
- 定义项目的名称，并指定位置



- 对项目进行设置，JDK版本、编译版本

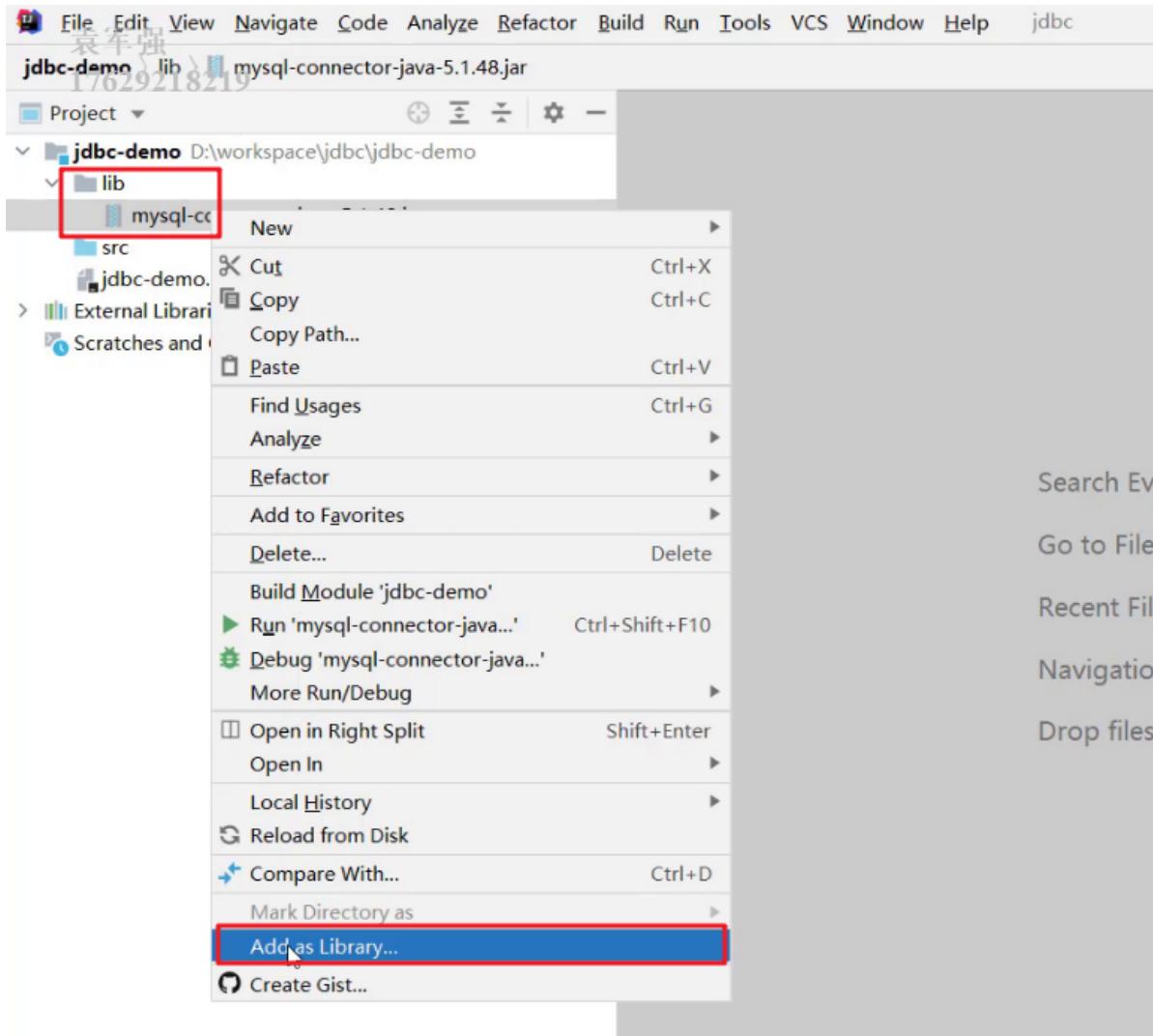


- 创建模块，指定模块的名称及位置



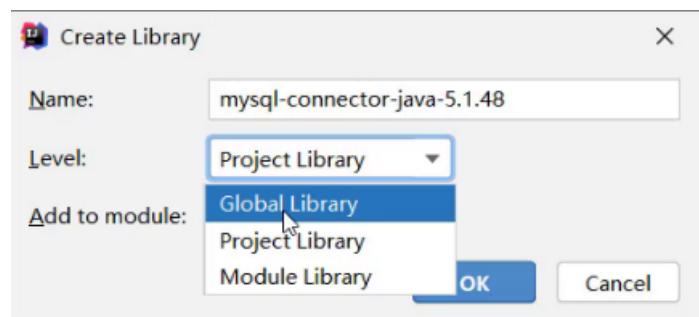
- 导入驱动包

将mysql的驱动包放在模块下的lib目录（随意命名）下，并将该jar包添加为库文件

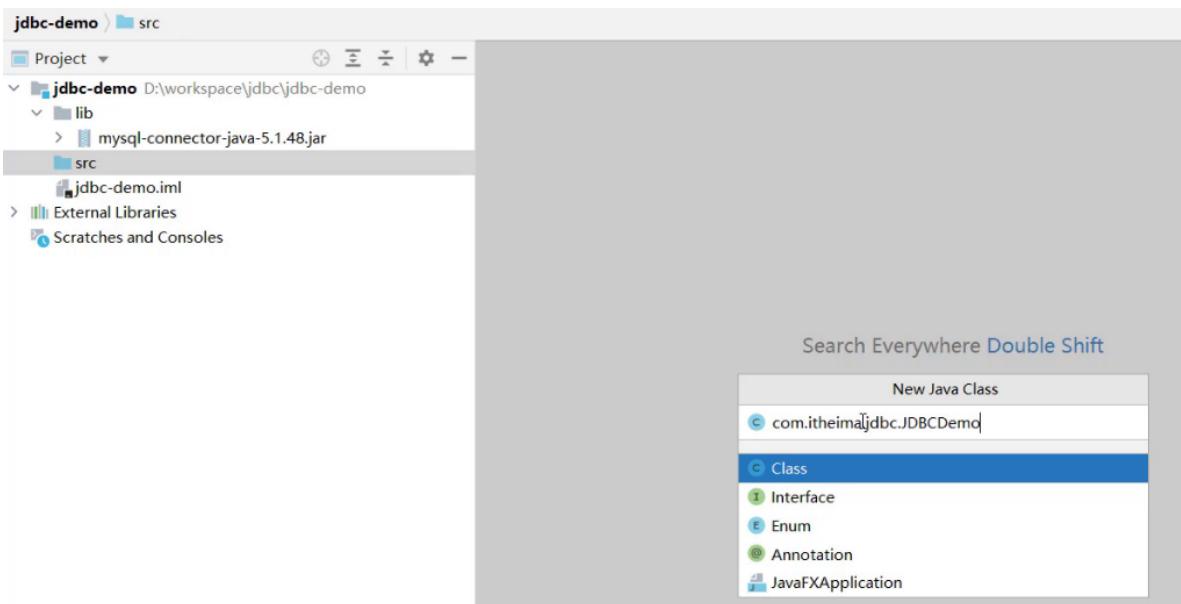


- 在添加为库文件的时候，有如下三个选项

- Global Library : 全局有效
- Project Library : 项目有效
- Module Library : 模块有效



- 在src下创建类



- 编写代码如下

```
/**  
 * JDBC快速入门  
 */  
public class JDBCdemo {  
  
    public static void main(String[] args) throws  
Exception {  
        //1. 注册驱动  
        //Class.forName("com.mysql.jdbc.Driver");  
        //2. 获取连接  
        String url =  
"jdbc:mysql://127.0.0.1:3306/db1";  
        String username = "root";  
        String password = "1234";  
        Connection conn =  
DriverManager.getConnection(url, username,  
password);  
        //3. 定义sql  
        String sql = "update account set money =  
2000 where id = 1";  
        //4. 获取执行sql的对象 Statement  
        Statement stmt = conn.createStatement();  
        //5. 执行sql
```

```
    int count = stmt.executeUpdate(sql); //受影响  
的行数  
    //6. 处理结果  
    System.out.println(count);  
    //7. 释放资源  
    stmt.close();  
    conn.close();  
}  
}
```

3, JDBC API详解

3.1 DriverManager

DriverManager (驱动管理类) 作用:

- 注册驱动

```
static void registerDriver(Driver driver) 使用 DriverManager注册给定的驱动程序。
```

registerDriver方法是用于注册驱动的，但是我们之前做的入门案例并不是这样写的。而是如下实现

```
Class.forName("com.mysql.jdbc.Driver");
```

我们查询MySQL提供的Driver类，看它是如何实现的，源码如下：

```
static {  
    try {  
        DriverManager.registerDriver(new Driver());  
    } catch (SQLException var1) {  
        throw new RuntimeException("Can't register driver!");  
    }  
}
```

在该类中的静态代码块中已经执行了 `DriverManager` 对象的 `registerDriver()` 方法进行驱动的注册了，那么我们只需要加载 `Driver` 类，该静态代码块就会执行。而 `Class.forName("com.mysql.jdbc.Driver");` 就可以加载 `Driver` 类。

==提示： ==

- MySQL 5之后的驱动包，可以省略注册驱动的步骤
- 自动加载jar包中META-INF/services/java.sql.Driver文件中的驱动类

- 获取数据库连接

```
static Connection getConnection(String url, String user, String password) 尝试建立与给定数据库URL的连接。
```

参数说明：

- url： 连接路径

语法： `jdbc:mysql://ip地址(域名):端口号/数据库名称?参数键值对1&参数键值对2...`

示例： `jdbc:mysql://127.0.0.1:3306/db1`

==细节： ==

- 如果连接的是本机mysql服务器，并且mysql服务默认端口是3306，则url可以简写为：`jdbc:mysql:///数据库名称?参数键值对`
- 配置 `useSSL=false` 参数，禁用安全连接方式，解决警告提示

- user： 用户名
- password： 密码

3.2 Connection

Connection (数据库连接对象) 作用:

- 获取执行 SQL 的对象
- 管理事务

3.2.1 获取执行对象

- 普通执行SQL对象

```
Statement createStatement()
```

入门案例中就是通过该方法获取的执行对象。

- 预编译SQL的执行SQL对象：防止SQL注入

```
PreparedStatement prepareStatement(sql)
```

通过这种方式获取的 `PreparedStatement` SQL语句执行对象是我们一会重点要进行讲解的，它可以防止SQL注入。

- 执行存储过程的对象

```
CallableStatement prepareCall(sql)
```

通过这种方式获取的 `CallableStatement` 执行对象是用来执行存储过程的，而存储过程在MySQL中不常用，所以这个我们将不进行讲解。

3.2.2 事务管理

先回顾一下MySQL事务管理的操作:

- 开启事务 : BEGIN; 或者 START TRANSACTION;
- 提交事务 : COMMIT;
- 回滚事务 : ROLLBACK;

MySQL默认是自动提交事务

接下来学习JDBC事务管理的方法。

Connection接口中定义了3个对应的方法：

- 开启事务

void	setAutoCommit(boolean autoCommit)	将此连接的自动提交模式设置为给定状态。
------	-----------------------------------	---------------------

参与autoCommit 表示是否自动提交事务，true表示自动提交事务，false表示手动提交事务。而开启事务需要将该参数设为为false。

- 提交事务

void	commit()	使自上次提交/回滚以来所做的所有更改成为永久更改，并释放此 Connection对象当前持有的所有数据库锁。
------	----------	--

- 回滚事务

void	rollback()	撤消当前事务中所做的所有更改，并释放此 Connection对象当前持有的所有数据库锁。
------	------------	--

具体代码实现如下：

```
/*
 * JDBC API 详解: Connection
 */
public class JDBCDemo3_Connection {

    public static void main(String[] args) throws
Exception {
        //1. 注册驱动
        //Class.forName("com.mysql.jdbc.Driver");
        //2. 获取连接：如果连接的是本机mysql并且端口是默认
        //的 3306 可以简化书写
        String url = "jdbc:mysql:////db1?
useSSL=false";
        String username = "root";
        String password = "1234";
        Connection conn =
DriverManager.getConnection(url, username,
password);
        //3. 定义sql
```

```
String sql1 = "update account set money =  
3000 where id = 1";  
String sql2 = "update account set money =  
3000 where id = 2";  
//4. 获取执行sql的对象 Statement  
Statement stmt = conn.createStatement();
```

try {
 // =====开启事务=====
 conn.setAutoCommit(false);
 //5. 执行sql
 int count1 = stmt.executeUpdate(sql1); //受影响的行数

//6. 处理结果
System.out.println(count1);
int i = 3/0;
//5. 执行sql
int count2 = stmt.executeUpdate(sql2); //受影响的行数

//6. 处理结果
System.out.println(count2);

// =====提交事务=====
//程序运行到此处，说明没有出现任何问题，则需求提交事务

conn.commit();
} catch (Exception e) {
 // =====回滚事务=====
 //程序在出现异常时会执行到这个地方，此时就需要回滚事务
 conn.rollback();
 e.printStackTrace();
}

//7. 释放资源
stmt.close();
conn.close();

```
    }  
}
```

3.3 Statement

3.3.1 概述

Statement对象的作用就是用来执行SQL语句。而针对不同类型的SQL语句使用的方法也不一样。

- 执行DDL、DML语句

```
int executeUpdate(String sql)
```

执行给定的SQL语句，这可能是 INSERT, UPDATE, 或 DELETE语句，或者不返回任何内容，如SQL DDL语句的SQL语句。

- 执行DQL语句

```
ResultSet executeQuery(String sql)
```

执行给定的SQL语句，该语句返回单个 ResultSet对象。

该方法涉及到了 `ResultSet` 对象，而这个对象我们还没有学习，一会再重点讲解。

3.3.2 代码实现

- 执行DML语句

```
/**  
 * 执行DML语句  
 * @throws Exception  
 */  
@Test  
public void testDML() throws Exception {  
    //1. 注册驱动  
    //Class.forName("com.mysql.jdbc.Driver");  
    //2. 获取连接：如果连接的是本机mysql并且端口是默认的  
    //3306 可以简化书写  
    String url = "jdbc:mysql:///db1?  
useSSL=false";  
    String username = "root";
```

```

String password = "1234";
Connection conn =
DriverManager.getConnection(url, username,
password);
//3. 定义sql
String sql = "update account set money = 3000
where id = 1";
//4. 获取执行sql的对象 Statement
Statement stmt = conn.createStatement();
//5. 执行sql
int count = stmt.executeUpdate(sql); //执行完
DML语句，受影响的行数
//6. 处理结果
//System.out.println(count);
if(count > 0){
    System.out.println("修改成功~");
} else{
    System.out.println("修改失败~");
}
//7. 释放资源
stmt.close();
conn.close();
}

```

- 执行DDL语句

```

/**
 * 执行DDL语句
 * @throws Exception
 */
@Test
public void testDDL() throws Exception {
    //1. 注册驱动
    //Class.forName("com.mysql.jdbc.Driver");
    //2. 获取连接：如果连接的是本机mysql并且端口是默认的
    //3306 可以简化书写
}

```

```
String url = "jdbc:mysql://db1?  
useSSL=false";  
String username = "root";  
String password = "1234";  
Connection conn =  
DriverManager.getConnection(url, username,  
password);  
//3. 定义sql  
String sql = "drop database db2";  
//4. 获取执行sql的对象 Statement  
Statement stmt = conn.createStatement();  
//5. 执行sql  
int count = stmt.executeUpdate(sql); //执行完  
DDL语句，可能是0  
//6. 处理结果  
System.out.println(count);  
  
//7. 释放资源  
stmt.close();  
conn.close();  
}
```

注意：

- 以后开发很少使用java代码操作DDL语句

3.4 ResultSet

3.4.1 概述

ResultSet (结果集对象) 作用：

- ==封装了SQL查询语句的结果。==

而执行了DQL语句后就会返回该对象，对应执行DQL语句的方法如下：

```
ResultSet executeQuery(sql): 执行DQL 语句，返回  
ResultSet 对象
```

那么我们就需要从 `ResultSet` 对象中获取我们想要的数据。

`ResultSet` 对象提供了操作查询结果数据的方法，如下：

```
boolean next()
```

- 将光标从当前位置向前移动一行
- 判断当前行是否为有效行

方法返回值说明：

- true : 有效航，当前行有数据
- false : 无效行，当前行没有数据

```
xxx getXxx(参数): 获取数据
```

- xxx : 数据类型；如： int getInt(参数) ; String getString(参数)
- 参数
 - int类型的参数：列的编号，从1开始
 - String类型的参数：列的名称

如下图为执行SQL语句后的结果

id	name	money
1	张三	3000
2	李四	1000
4	王五	3000

一开始光标指定于第一行前，如图所示红色箭头指向于表头行。当我们调用了 `next()` 方法后，光标就下移到第一行数据，并且方法返回true，此时就可以通过 `getInt("id")` 获取当前行id字段的值，也可以通过 `getString("name")` 获取当前行name字段的值。如果想获取下一行的数据，继续调用 `next()` 方法，以此类推。

3.4.2 代码实现

```
/**  
 * 执行DQL  
 * @throws Exception  
 */  
  
@Test  
public void testResultSet() throws Exception {  
    //1. 注册驱动  
    //Class.forName("com.mysql.jdbc.Driver");  
    //2. 获取连接: 如果连接的是本机mysql并且端口是默认的  
    //3306 可以简化书写  
    String url = "jdbc:mysql:////db1?useSSL=false";  
    String username = "root";  
    String password = "1234";  
    Connection conn =  
        DriverManager.getConnection(url, username,  
        password);  
    //3. 定义sql  
    String sql = "select * from account";  
    //4. 获取statement对象  
    Statement stmt = conn.createStatement();  
    //5. 执行sql  
    ResultSet rs = stmt.executeQuery(sql);  
    //6. 处理结果, 遍历rs中的所有数据  
    /* // 6.1 光标向下移动一行, 并且判断当前行是否有数据  
        while (rs.next()) {  
            //6.2 获取数据 getXXX()  
            int id = rs.getInt(1);  
            String name = rs.getString(2);  
            double money = rs.getDouble(3);  
  
            System.out.println(id);  
            System.out.println(name);  
            System.out.println(money);  
  
            System.out.println("-----");  
        }  
    */  
}
```

```

    }*/
```

// 6.1 光标向下移动一行，并且判断当前行是否有数据

```

while (rs.next()){
    //6.2 获取数据 getXXX()
    int id = rs.getInt("id");
    String name = rs.getString("name");
    double money = rs.getDouble("money");

    System.out.println(id);
    System.out.println(name);
    System.out.println(money);

    System.out.println("-----");
}
```

//7. 释放资源

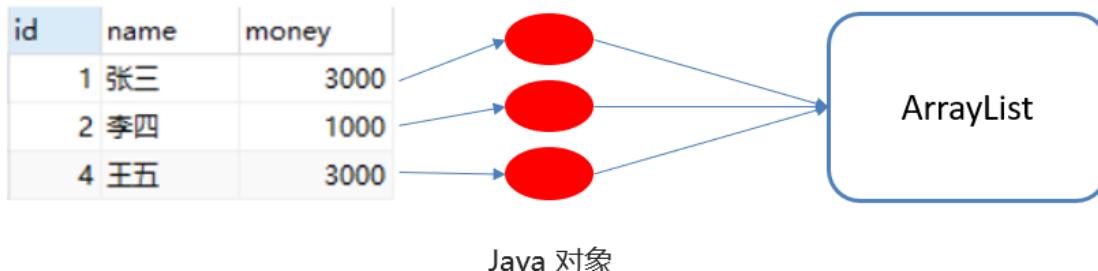
```

rs.close();
stmt.close();
conn.close();
}

```

3.5 案例

- 需求：查询account账户表数据，封装为Account对象中，并且存储到ArrayList集合中



- 代码实现

```
/**
```

```
* 查询account账户表数据，封装为Account对象中，并且存储  
到ArrayList集合中  
    * 1. 定义实体类Account  
    * 2. 查询数据，封装到Account对象中  
    * 3. 将Account对象存入ArrayList集合中  
*/  
  
@Test  
public void testResultSet2() throws Exception {  
    //1. 注册驱动  
    //Class.forName("com.mysql.jdbc.Driver");  
    //2. 获取连接：如果连接的是本机mysql并且端口是默认的  
    3306 可以简化书写  
    String url = "jdbc:mysql://db1?  
useSSL=false";  
    String username = "root";  
    String password = "1234";  
    Connection conn =  
    DriverManager.getConnection(url, username,  
    password);  
  
    //3. 定义sql  
    String sql = "select * from account";  
  
    //4. 获取statement对象  
    Statement stmt = conn.createStatement();  
  
    //5. 执行sql  
    ResultSet rs = stmt.executeQuery(sql);  
  
    // 创建集合  
    List<Account> list = new ArrayList<>();  
  
    // 6.1 光标向下移动一行，并且判断当前行是否有数据  
    while (rs.next()) {  
        Account account = new Account();  
  
        //6.2 获取数据 getXXX()  
    }  
}
```

```
int id = rs.getInt("id");
String name = rs.getString("name");
double money = rs.getDouble("money");

//赋值
account.setId(id);
account.setName(name);
account.setMoney(money);

// 存入集合
list.add(account);
}

System.out.println(list);

//7. 释放资源
rs.close();
stmt.close();
conn.close();
}
```

3.6 PreparedStatement

PreparedStatement作用：

- 预编译SQL语句并执行：预防SQL注入问题

对上面的作用中SQL注入问题大家肯定不理解。那我们先对SQL注入进行说明。

3.6.1 SQL注入

SQL注入是通过操作输入来修改事先定义好的SQL语句，用以达到执行代码对服务器进行攻击的方法。

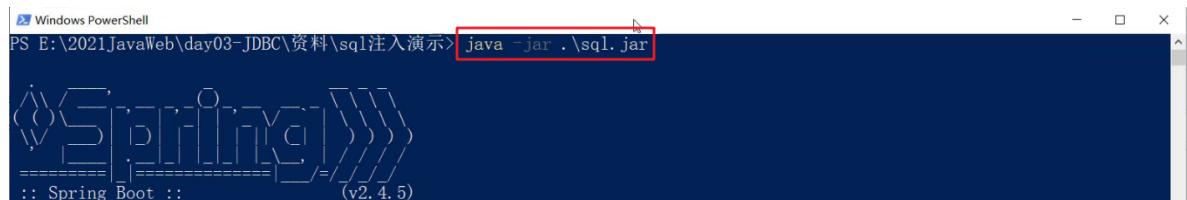
在今天资料下的 day03-JDBC\资料\2. sql注入演示 中修改 application.properties 文件中的用户名和密码，文件内容如下：

```
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/test?
useSSL=false&useUnicode=true&characterEncoding=UTF-8
spring.datasource.username=root
spring.datasource.password=1234
```

在MySQL中创建名为 test 的数据库

```
create database test;
```

在命令提示符中运行今天资料下的 day03-JDBC\资料\2. sql注入演示\sql.jar 这个jar包。



此时我们就能在数据库中看到user表

A screenshot of MySQL Workbench. On the left, the database tree shows '本机' (Localhost) with schemas 'db1', 'information_schema', 'mysql', 'performance_schema', 'sys', and 'test'. Under 'test', there are tables 'product' and 'user'. The 'user' table is selected and highlighted with a red box. On the right, the '对象' (Objects) tab is selected, showing a table named '无标题 @db1 (本机) - 查询'. The 'user' table is displayed with the following data:

	id	username	password
	1	zhangsan	123
	2	lisi	123

接下来在浏览器的地址栏输入 localhost:8080/login.html 就能看到如下页面



我们就可以在如上图中输入用户名和密码进行登陆。用户名和密码输入正确就登陆成功，跳转到首页。用户名和密码输入错误则给出错误提示，如下图



但是我可以通过输入一些特殊的字符登陆到首页。

用户名随意写，密码写成 '`' or '1' = '1`



这就是SQL注入漏洞，也是很危险的。当然现在市面上的系统都不会存在这种问题了，所以大家也不要尝试用这种方式去试其他的系统。

那么该如何解决呢？这里就可以将SQL执行对象 `Statement` 换成 `PreparedStatement` 对象。

3.6.2 代码模拟SQL注入问题

```
@Test
public void testLogin() throws Exception {
    //2. 获取连接：如果连接的是本机mysql并且端口是默认的
    //3306 可以简化书写
    String url = "jdbc:mysql:////db1?useSSL=false";
    String username = "root";
    String password = "1234";
    Connection conn =
    DriverManager.getConnection(url, username,
    password);

    // 接收用户输入 用户名和密码
    String name = "sjdljf1d";
    String pwd = "' or '1' = '1";
```

```
String sql = "select * from tb_user where  
username = '" + name + "' and password = '" + pwd + "'";  
// 获取stmt对象  
Statement stmt = conn.createStatement();  
// 执行sql  
ResultSet rs = stmt.executeQuery(sql);  
// 判断登录是否成功  
if(rs.next()) {  
    System.out.println("登录成功~");  
} else {  
    System.out.println("登录失败~");  
}  
  
//7. 释放资源  
rs.close();  
stmt.close();  
conn.close();  
}
```

上面代码是将用户名和密码拼接到sql语句中，拼接后的sql语句如下

```
select * from tb_user where username = 'sjd1jf1d'  
and password = ''or '1' = '1'
```

从上面语句可以看出条件 `username = 'sjd1jf1d' and password = ''` 不管是否满足，而 `or` 后面的 `'1' = '1'` 是始终满足的，最终条件是成立的，就可以正常的进行登陆了。

接下来我们来学习PreparedStatement对象.

3.6.3 PreparedStatement概述

PreparedStatement作用：

- 预编译SQL语句并执行：预防SQL注入问题
- 获取 PreparedStatement 对象

```
// SQL语句中的参数值，使用? 占位符替代
String sql = "select * from user where username =
? and password = ?";
// 通过Connection对象获取，并传入对应的sql语句
PreparedStatement pstmt =
conn.prepareStatement(sql);
```

- 设置参数值

上面的sql语句中参数使用?进行占位，在之前之前肯定要设置这些? 的值。

PreparedStatement对象：setXxx(参数1, 参数2)：给?赋值

- Xxx：数据类型；如.setInt (参数1, 参数2)
- 参数：
 - 参数1：?的位置编号，从1开始
 - 参数2：?的值

- 执行SQL语句

executeUpdate(); 执行DDL语句和DML语句

executeQuery(); 执行DQL语句

==注意：==

- 调用这两个方法时不需要传递SQL语句，因为获取SQL语句执行对象时已经对SQL语句进行预编译了。

3.6.4 使用PreparedStatement改进

```
@Test
public void testPreparedStatement() throws
Exception {
    //2. 获取连接：如果连接的是本机mysql并且端口是默认的
    //3306 可以简化书写
```

```
String url = "jdbc:mysql://db1?useSSL=false";
String username = "root";
String password = "1234";
Connection conn =
DriverManager.getConnection(url, username,
password);

// 接收用户输入 用户名和密码
String name = "zhangsan";
String pwd = '' or '1' = '1';

// 定义sql
String sql = "select * from tb_user where
username = ? and password = ?";
// 获取pstmt对象
PreparedStatement pstmt =
conn.prepareStatement(sql);
// 设置? 的值
pstmt.setString(1, name);
pstmt.setString(2, pwd);
// 执行sql
ResultSet rs = pstmt.executeQuery();
// 判断登录是否成功
if(rs.next()){
    System.out.println("登录成功~");
} else{
    System.out.println("登录失败~");
}
//7. 释放资源
rs.close();
pstmt.close();
conn.close();
}
```

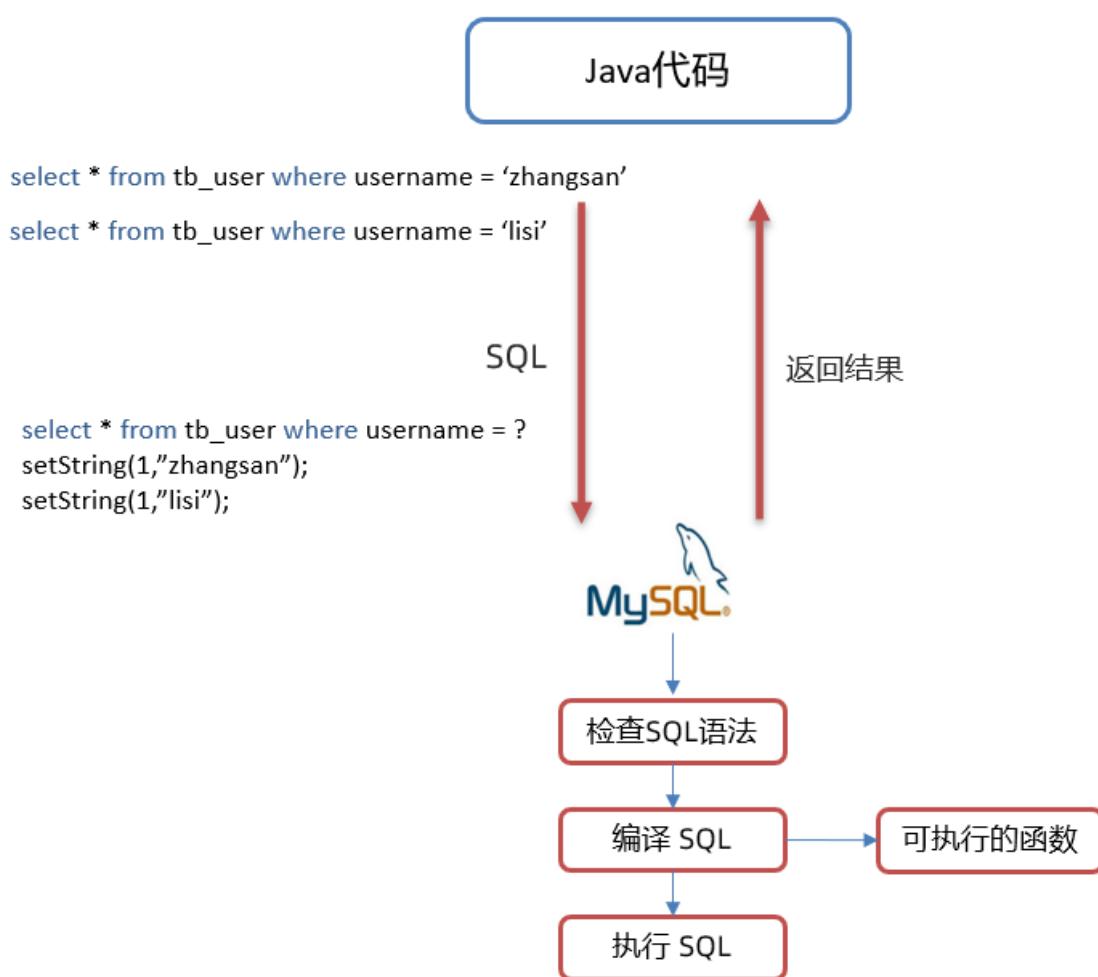
执行上面语句就可以发现不会出现SQL注入漏洞问题了。那么PreparedStatement又是如何解决的呢？它是将特殊字符进行了转义，转义的SQL如下：

```
select * from tb_user where username = 'sjd1jf1d'  
and password = '\'or \'1\' = '1'
```

3.6.5 PreparedStatement原理

PreparedStatement 好处:

- 预编译SQL，性能更高
- 防止SQL注入：==将敏感字符进行转义==



Java代码操作数据库流程如图所示:

- 将sql语句发送到MySQL服务器端
- MySQL服务端会对sql语句进行如下操作
 - 检查SQL语句

检查SQL语句的语法是否正确。

- 编译SQL语句。将SQL语句编译成可执行的函数。

检查SQL和编译SQL花费的时间比执行SQL的时间还要长。如果我们只是重新设置参数，那么检查SQL语句和编译SQL语句将不需要重复执行。这样就提高了性能。

- 执行SQL语句

接下来我们通过查询日志来看一下原理。

- 开启预编译功能

在代码中编写url时需要加上以下参数。而我们之前根本就没有开启预编译功能，只是解决了SQL注入漏洞。

```
useServerPrepStmts=true
```

- 配置MySQL执行日志（重启mysql服务后生效）

在mysql配置文件（my.ini）中添加如下配置

```
log-output=FILE
general-log=1
general_log_file="D:\mysql.log"
slow-query-log=1
slow_query_log_file="D:\mysql_slow.log"
long_query_time=2
```

- java测试代码如下：

```
/**
 * PreparedStatement原理
 * @throws Exception
 */
@Test
public void testPreparedStatement2() throws
Exception {
```

```
//2. 获取连接：如果连接的是本机mysql并且端口是默认的  
3306 可以简化书写  
// useServerPrepStmts=true 参数开启预编译功能  
String url = "jdbc:mysql://db1?  
useSSL=false&useServerPrepStmts=true";  
String username = "root";  
String password = "1234";  
Connection conn =  
DriverManager.getConnection(url, username,  
password);  
  
// 接收用户输入 用户名和密码  
String name = "zhangsan";  
String pwd = "' or '1' = '1';  
  
// 定义sql  
String sql = "select * from tb_user where  
username = ? and password = ?";  
  
// 获取stmt对象  
PreparedStatement pstmt =  
conn.prepareStatement(sql);  
  
Thread.sleep(10000);  
// 设置? 的值  
pstmt.setString(1, name);  
pstmt.setString(2, pwd);  
ResultSet rs = null;  
// 执行sql  
rs = pstmt.executeQuery();  
  
// 设置? 的值  
pstmt.setString(1, "aaa");  
pstmt.setString(2, "bbb");  
// 执行sql  
rs = pstmt.executeQuery();
```

```

// 判断登录是否成功
if(rs.next()){
    System.out.println("登录成功~");
}else{
    System.out.println("登录失败~");
}

//7. 释放资源
rs.close();
pstmt.close();
conn.close();
}

```

- 执行SQL语句，查看 D:\mysql.log 日志如下：

```

2021-04-25T14:39:06.494002Z      4 Query SET character_set_results = NULL
2021-04-25T14:39:06.494348Z      4 Query SET autocommit=1
2021-04-25T14:39:06.516950Z      4 Prepare select * from tb_user where username = ? and password = ?
2021-04-25T14:39:06.518524Z      4 Execute select * from tb_user where username = 'zhangsan' and password = '\' or
\'1\' = '\1'
2021-04-25T14:39:06.519042Z      4 Execute select * from tb_user where username = 'aaa' and password = 'bbb'
2021-04-25T14:39:06.519405Z      4 Close stmt
2021-04-25T14:39:06.524935Z      4 Quit

```

上图中第三行中的 `Prepare` 是对SQL语句进行预编译。第四行和第五行是执行了两次SQL语句，而第二次执行前并没有对SQL进行预编译。

==小结：==

- 在获取PreparedStatement对象时，将sql语句发送给mysql服务器进行检查，编译（这些步骤很耗时）
- 执行时就不用再进行这些步骤了，速度更快
- 如果sql模板一样，则只需要进行一次检查、编译

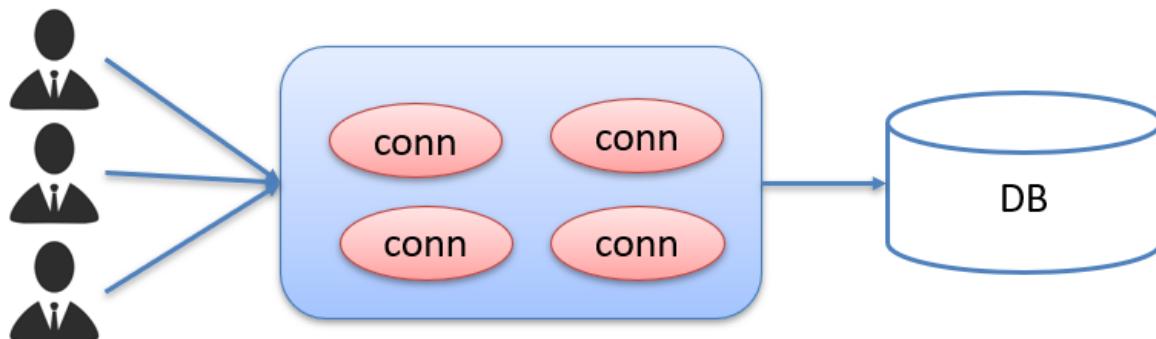
4. 数据库连接池

4.1 数据库连接池简介

- 数据库连接池是个容器，负责分配、管理数据库连接 (Connection)
- 它允许应用程序重复使用一个现有的数据库连接，而不是再重新建立一个；
- 释放空闲时间超过最大空闲时间的数据库连接来避免因为没有释放数据库连接而引起的数据库连接遗漏
- 好处
 - 资源重用
 - 提升系统响应速度
 - 避免数据库连接遗漏

之前我们代码中使用连接是没有使用都创建一个Connection对象，使用完毕就会将其销毁。这样重复创建销毁的过程是特别耗费计算机的性能的及消耗时间的。

而数据库使用了数据库连接池后，就能达到Connection对象的复用，如下图



连接池是在一开始就创建好了一些连接 (Connection) 对象存储起来。用户需要连接数据库时，不需要自己创建连接，而只需要从连接池中获取一个连接进行使用，使用完毕后再将连接对象归还给连接池；这样就可以起到资源重用，也节省了频繁创建连接销毁连接所花费的时间，从而提升了系统响应的速度。

4.2 数据库连接池实现

- 标准接口：==DataSource==

官方(SUN) 提供的数据库连接池标准接口，由第三方组织实现此接口。该接口提供了获取连接的功能：

```
Connection getConnection()
```

那么以后就不需要通过 `DriverManager` 对象获取 `Connection` 对象，而是通过连接池（DataSource）获取 `Connection` 对象。

- 常见的数据库连接池

- DBCP
- C3P0
- Druid

我们现在使用更多的是Druid，它的性能比其他两个会好一些。

- Druid（德鲁伊）

- Druid连接池是阿里巴巴开源的数据库连接池项目
- 功能强大，性能优秀，是Java语言最好的数据库连接池之一

4.3 Druid使用

- 导入jar包 druid-1.1.12.jar
- 定义配置文件
- 加载配置文件
- 获取数据库连接池对象
- 获取连接

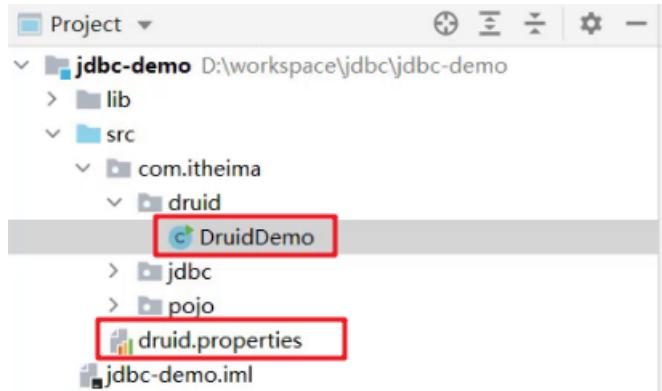
现在通过代码实现，首先需要先将druid的jar包放到项目下的lib下并添加为库文件

```

// 定义sql
String sql = "select * from tb_user where username
// 获取pstmt对象
PreparedStatement pstmt = conn.prepareStatement(sql
Thread.sleep( millis: 10000);
// 设置? 的值
ps
Name: druid-1.1.12
Re
Level: Project Library
Add to module: Global Library
Project Library
Module Library OK Cancel
// 设置? 的值
pstmt.setString( parameterIndex: 1 x: "aaa") .

```

项目结构如下：



编写配置文件如下：

```

driverClassName=com.mysql.jdbc.Driver
url=jdbc:mysql:///db1?
useSSL=false&useServerPrepStmts=true
username=root
password=1234
# 初始化连接数量
initialSize=5
# 最大连接数
maxActive=10
# 最大等待时间
maxWait=3000

```

使用druid的代码如下：

```
/**
```

```
* Druid数据库连接池演示
*/
public class DruidDemo {

    public static void main(String[] args) throws
Exception {
        //1. 导入jar包
        //2. 定义配置文件
        //3. 加载配置文件
        Properties prop = new Properties();
        prop.load(new FileInputStream("jdbc-
demo/src/druid.properties"));
        //4. 获取连接池对象
        DataSource dataSource =
DruidDataSourceFactory.createDataSource(prop);

        //5. 获取数据库连接 Connection
        Connection connection =
dataSource.getConnection();
        System.out.println(connection); //获取到了连接
后就可以继续做其他操作了

        //System.out.println(System.getProperty("user.dir"))
};

}

}
```

5, JDBC练习

5.1 需求

完成商品品牌数据的增删改查操作

- 查询：查询所有数据
- 添加：添加品牌

- 修改：根据id修改
- 删除：根据id删除

5.2 案例实现

5.2.1 环境准备

- 数据库表 tb_brand

```
-- 删除tb_brand表
drop table if exists tb_brand;
-- 创建tb_brand表
create table tb_brand (
    -- id 主键
    id int primary key auto_increment,
    -- 品牌名称
    brand_name varchar(20),
    -- 企业名称
    company_name varchar(20),
    -- 排序字段
    ordered int,
    -- 描述信息
    description varchar(100),
    -- 状态: 0: 禁用  1: 启用
    status int
);
-- 添加数据
insert into tb_brand (brand_name, company_name,
ordered, description, status)
values ('三只松鼠', '三只松鼠股份有限公司', 5, '好吃不上火', 0),
       ('华为', '华为技术有限公司', 100, '华为致力于把
数字世界带入每个人、每个家庭、每个组织，构建万物互联的智能世
界', 1),
       ('小米', '小米科技有限公司', 50, 'are you ok',
1);
```

- 在pojo包下实体类 Brand

```
/*
 * 品牌
 * alt + 鼠标左键: 整列编辑
 * 在实体类中, 基本数据类型建议使用其对应的包装类型
 */
public class Brand {
    // id 主键
    private Integer id;
    // 品牌名称
    private String brandName;
    // 企业名称
    private String companyName;
    // 排序字段
    private Integer ordered;
    // 描述信息
    private String description;
    // 状态: 0: 禁用 1: 启用
    private Integer status;

    public Integer getId() {
        return id;
    }

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

    public String getBrandName() {
        return brandName;
    }

    public void setBrandName(String brandName) {
        this.brandName = brandName;
    }
}
```

```
public string getCompanyName() {
    return companyName;
}

public void setCompanyName(string
companyName) {
    this.companyName = companyName;
}

public Integer getOrdered() {
    return ordered;
}

public void setOrdered(Integer ordered) {
    this.ordered = ordered;
}

public string getDescription() {
    return description;
}

public void setDescription(string
description) {
    this.description = description;
}

public Integer getStatus() {
    return status;
}

public void setStatus(Integer status) {
    this.status = status;
}

@Override
public string toString() {
    return "Brand{" +

```

```

        "id=" + id +
        ", brandName='" + brandName +
        "'\'' +
        ", companyName='" + companyName +
        "'\'' +
        ", ordered=" + ordered +
        ", description='" + description +
        "'\'' +
        ", status=" + status +
        '}';
    }
}

```

5.2.2 查询所有

```

/*
 * 查询所有
 * 1. SQL: select * from tb_brand;
 * 2. 参数: 不需要
 * 3. 结果: List<Brand>
 */

@Test
public void testselectAll() throws Exception {
    //1. 获取Connection
    //3. 加载配置文件
    Properties prop = new Properties();
    prop.load(new FileInputStream("jdbc-
demo/src/druid.properties"));
    //4. 获取连接池对象
    DataSource dataSource =
DruidDataSourceFactory.createDataSource(prop);

    //5. 获取数据库连接 Connection
    Connection conn = dataSource.getConnection();
    //2. 定义SQL
    String sql = "select * from tb_brand;"

```

```
//3. 获取stmt对象
PreparedStatement pstmt =
conn.prepareStatement(sql);

//4. 设置参数
//5. 执行SQL
ResultSet rs = pstmt.executeQuery();
//6. 处理结果 List<Brand> 封装Brand对象，装载List集合
Brand brand = null;
List<Brand> brands = new ArrayList<>();
while (rs.next()){
    //获取数据
    int id = rs.getInt("id");
    String brandName =
rs.getString("brand_name");
    String companyName =
rs.getString("company_name");
    int ordered = rs.getInt("ordered");
    String description =
rs.getString("description");
    int status = rs.getInt("status");
    //封装Brand对象
    brand = new Brand();
    brand.setId(id);
    brand.setBrandName(brandName);
    brand.setCompanyName(companyName);
    brand.setOrdered(ordered);
    brand.setDescription(description);
    brand.setStatus(status);

    //装载集合
    brands.add(brand);
}
System.out.println(brands);
//7. 释放资源
rs.close();
pstmt.close();
conn.close();
```

```
}
```

5.2.3 添加数据

```
/**  
 * 添加  
 * 1. SQL: insert into tb_brand(brand_name,  
 company_name, ordered, description, status)  
 values(?, ?, ?, ?, ?);  
 * 2. 参数: 需要, 除了id之外的所有参数信息  
 * 3. 结果: boolean  
 */  
  
@Test  
public void testAdd() throws Exception {  
    // 接收页面提交的参数  
    String brandName = "香飘飘";  
    String companyName = "香飘飘";  
    int ordered = 1;  
    String description = "绕地球一圈";  
    int status = 1;  
  
    //1. 获取Connection  
    //3. 加载配置文件  
    Properties prop = new Properties();  
    prop.load(new FileInputStream("jdbc-  
demo/src/druid.properties"));  
    //4. 获取连接池对象  
    DataSource dataSource =  
    DruidDataSourceFactory.createDataSource(prop);  
    //5. 获取数据库连接 Connection  
    Connection conn = dataSource.getConnection();  
    //2. 定义SQL  
    String sql = "insert into tb_brand(brand_name,  
 company_name, ordered, description, status)  
 values(?, ?, ?, ?, ?);";  
    //3. 获取pst  
}
```

```

PreparedStatement pstmt =
conn.prepareStatement(sql);
    //4. 设置参数
    pstmt.setString(1,brandName);
    pstmt.setString(2,companyName);
    pstmt.setInt(3,ordered);
    pstmt.setString(4,description);
    pstmt.setInt(5,status);

    //5. 执行SQL
    int count = pstmt.executeUpdate(); // 影响的行数
    //6. 处理结果
    System.out.println(count > 0);

    //7. 释放资源
    pstmt.close();
    conn.close();
}

```

5.2.4 修改数据

```

/**
 * 修改
 * 1. SQL:
 *
update tb_brand
    set brand_name  = ?,
        company_name= ?,
        ordered      = ?,
        description = ?,
        status       = ?
    where id = ?

* 2. 参数: 需要, 所有数据
* 3. 结果: boolean
*/

```

```
@Test
public void testupdate() throws Exception {
    // 接收页面提交的参数
    String brandName = "香飘飘";
    String companyName = "香飘飘";
    int ordered = 1000;
    String description = "绕地球三圈";
    int status = 1;
    int id = 4;

    //1. 获取Connection
    //3. 加载配置文件
    Properties prop = new Properties();
    prop.load(new FileInputStream("jdbc-
demo/src/druid.properties"));
    //4. 获取连接池对象
    DataSource dataSource =
DruidDataSourceFactory.createDataSource(prop);
    //5. 获取数据库连接 Connection
    Connection conn = dataSource.getConnection();
    //2. 定义SQL
    String sql = " update tb_brand\n" +
        "           set brand_name = ?,\n" +
        "           company_name= ?,\n" +
        "           ordered      = ?,\n" +
        "           description  = ?,\n" +
        "           status       = ?\n" +
        "           where id = ?";

    //3. 获取stmt对象
    PreparedStatement pstmt =
conn.prepareStatement(sql);

    //4. 设置参数
    pstmt.setString(1,brandName);
    pstmt.setString(2,companyName);
    pstmt.setInt(3,ordered);
```

```

        pstmt.setString(4,description);
        pstmt.setInt(5,status);
        pstmt.setInt(6,id);

        //5. 执行SQL
        int count = pstmt.executeUpdate(); // 影响的行数
        //6. 处理结果
        System.out.println(count > 0);

        //7. 释放资源
        pstmt.close();
        conn.close();
    }
}

```

5.2.5 删除数据

```

/**
 * 删除
 * 1. SQL:
 *         delete from tb_brand where id = ?
 * 2. 参数: 需要, id
 * 3. 结果: boolean
 */
@Test
public void testDeleteById() throws Exception {
    // 接收页面提交的参数
    int id = 4;
    //1. 获取Connection
    //3. 加载配置文件
    Properties prop = new Properties();
    prop.load(new FileInputStream("jdbc-
demo/src/druid.properties"));
    //4. 获取连接池对象
    DataSource dataSource =
DruidDataSourceFactory.createDataSource(prop);
    //5. 获取数据库连接 Connection
    Connection conn = dataSource.getConnection();
}

```

```
//2. 定义SQL
String sql = " delete from tb_brand where id =
?";

//3. 获取stmt对象
PreparedStatement pstmt =
conn.prepareStatement(sql);

//4. 设置参数
pstmt.setInt(1,id);
//5. 执行SQL
int count = pstmt.executeUpdate(); // 影响的行数
//6. 处理结果
System.out.println(count > 0);

//7. 释放资源
pstmt.close();
conn.close();

}
```

Maven&MyBatis

目标

- 能够使用Maven进行项目的管理
- 能够完成Mybatis代理方式查询数据
- 能够理解Mybatis核心配置文件的配置

1. Maven

Maven是专门用于管理和构建Java项目的工具，它的主要功能有：

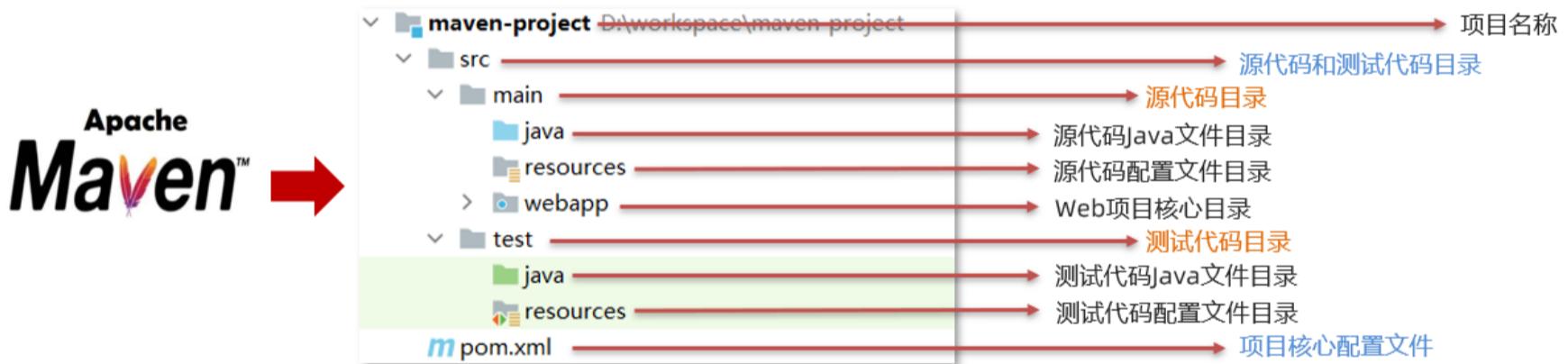
- 提供了一套标准化的项目结构
- 提供了一套标准化的构建流程（编译，测试，打包，发布……）
- 提供了一套依赖管理机制

标准化的项目结构：

项目结构我们都知道，每一个开发工具（IDE）都有自己不同的项目结构，它们互相之间不通用。我再eclipse中创建的目录，无法在idea中进行使用，这就造成了很大的不方便，如下图：前两个是以后开发经常使用的开发工具



而Maven提供了一套标准化的项目结构，所有的IDE使用Maven构建的项目完全一样，所以IDE创建的Maven项目可以通用。如下图右边就是Maven构建的项目结构。



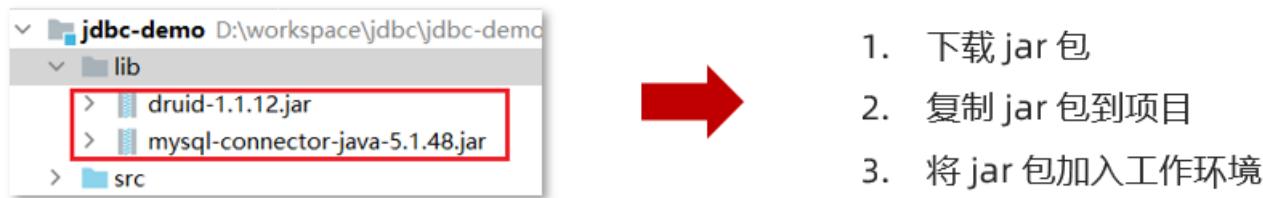
标准化的构建流程：



如上图所示我们开发了一套系统，代码需要进行编译、测试、打包、发布，这些操作如果需要反复进行就显得特别麻烦，而Maven提供了一套简单的命令来完成项目构建。

依赖管理：

依赖管理其实就是管理你项目所依赖的第三方资源（jar包、插件）。如之前我们项目中需要使用JDBC和Druid的话，就需要去网上下载对应的依赖包（当前之前是老师已经下载好提供给大家了），复制到项目中，还要将jar包加入工作环境这一系列的操作。如下图所示

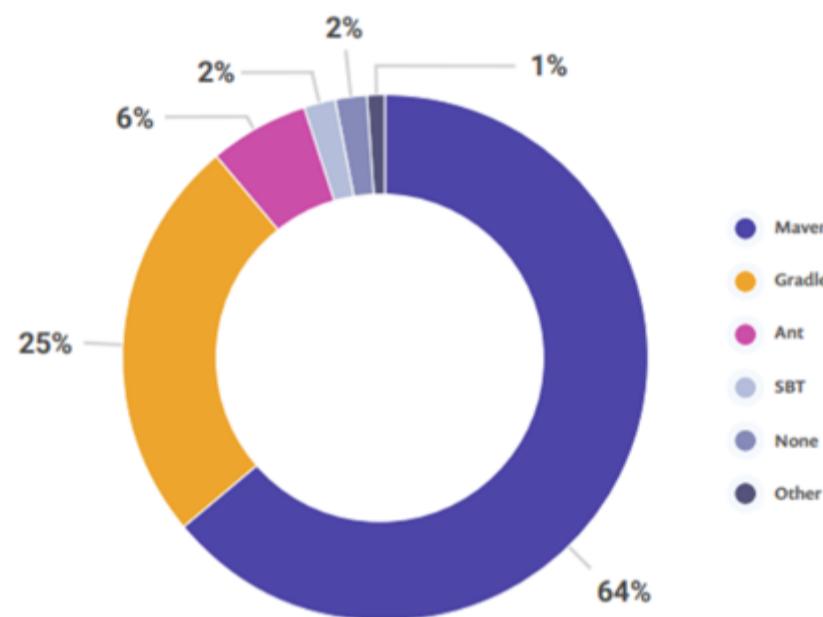


而Maven使用标准的 **坐标** 配置来管理各种依赖，只需要简单的配置就可以完成依赖管理。



如上图右边所示就是mysql驱动包的坐标，在项目中只需要写这段配置，其他都不需要我们担心，Maven都帮我们进行操作了。

市面上有很多构建工具，而Maven依旧还是主流构建工具，如下图是常用构建工具的使用占比



1.1 Maven简介

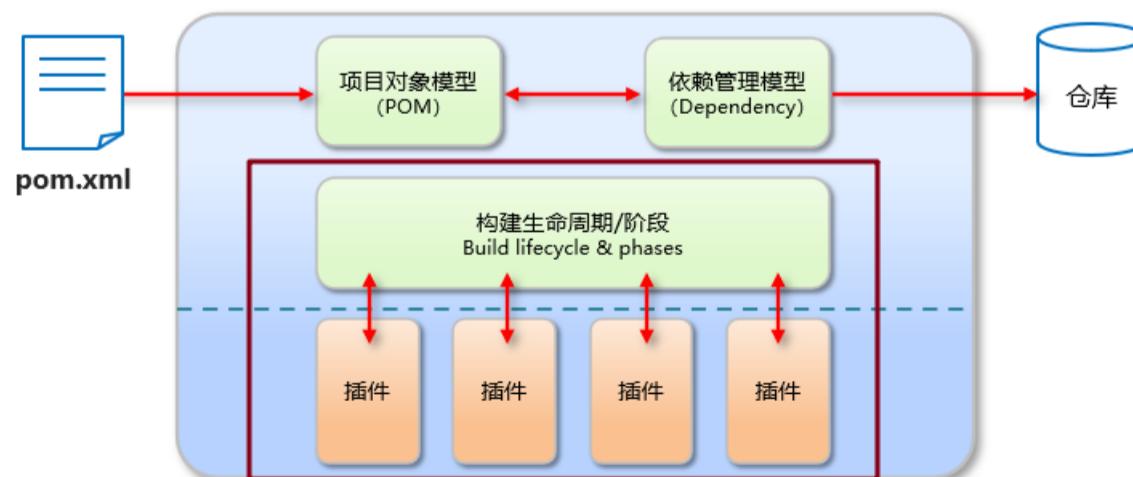
Apache Maven 是一个项目管理和构建**工具**，它基于项目对象模型(POM)的概念，通过一小段描述信息来管理项目的构建、报告和文档。

官网：<http://maven.apache.org/>

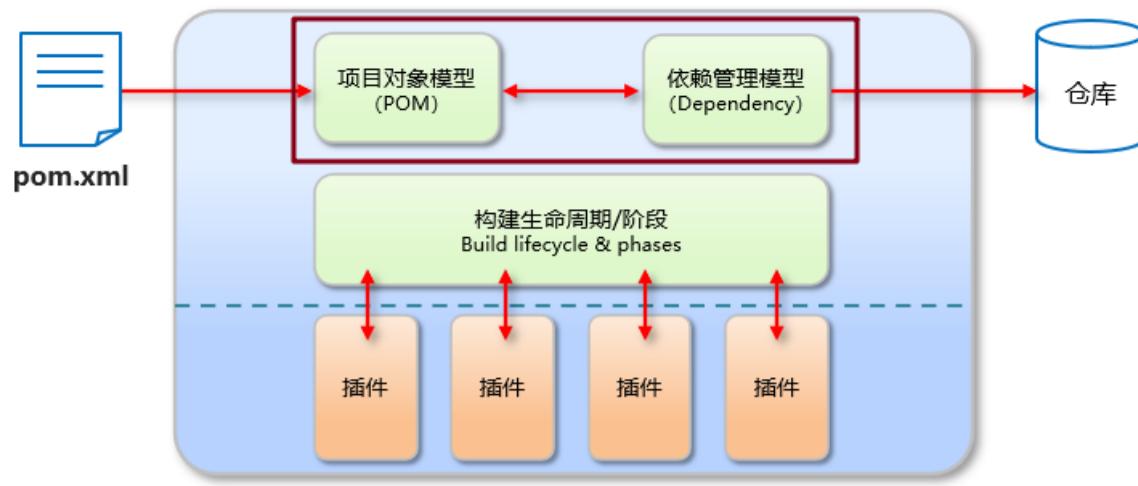
通过上面的描述大家只需要知道Maven是一个工具即可。Apache 是一个开源组织，将来我们会学习很多Apache提供的项目。

1.1.1 Maven模型

- 项目对象模型 (Project Object Model)
- 依赖管理模型(Dependency)
- 插件(Plugin)



如上图所示就是Maven的模型，而我们先看紫色框起来的部分，他就是用来完成 **标准化构建流程** 。如我们需要编译，Maven提供了一个编译插件供我们使用，我们需要打包，Maven就提供了一个打包插件提供我们使用等。



上图中紫色框起来的部分，项目对象模型就是将我们自己抽象成一个对象模型，有自己专属的坐标，如下图所示是一个 Maven项目：

A screenshot of the Eclipse IDE interface. On the left is the 'Project' view showing a project named 'jdbc-demo' with a sub-project 'maven-project'. The 'src' folder and 'pom.xml' file are visible. The main editor window shows the XML content of 'pom.xml'. A red box highlights the 'maven-project' section. In the code, the 'groupId', 'artifactId', and 'version' tags are highlighted with a red box and labeled '当前项目的坐标' (Current project's coordinate).

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.itheima</groupId>
    <artifactId>maven-project</artifactId>
    <version>1.0-SNAPSHOT</version>
```

依赖管理模型则是使用坐标来描述当前项目依赖哪儿些第三方jar包，如下图所示

A screenshot of the Eclipse IDE interface, similar to the previous one. The 'Project' view shows the same 'jdbc-demo' and 'maven-project' structure. The main editor window shows the 'pom.xml' file. A red box highlights the 'maven-project' section. In the code, the 'dependencies' section is highlighted with a red box and labeled '项目依赖mysql驱动包' (Project dependency on MySQL driver package).

```
<groupId>com.itheima</groupId>
<artifactId>maven-project</artifactId>
<version>1.0-SNAPSHOT</version>

<properties>
    <maven.compiler.source>8</maven.compiler.source>
    <maven.compiler.target>8</maven.compiler.target>
</properties>

<!-- 引入 mysql 驱动jar包-->
<dependencies>
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>5.1.32</version>
    </dependency>
</dependencies>
```

上述Maven模型图中还有一部分是仓库。如何理解仓库呢？

1.1.2 仓库

大家想想这样的场景，我们创建Maven项目，在项目中使用坐标来指定项目的依赖，那么依赖的jar包到底存储在什么地方呢？其实依赖jar包是存储在我们的本地仓库中。而项目运行时从本地仓库中拿需要的依赖jar包。

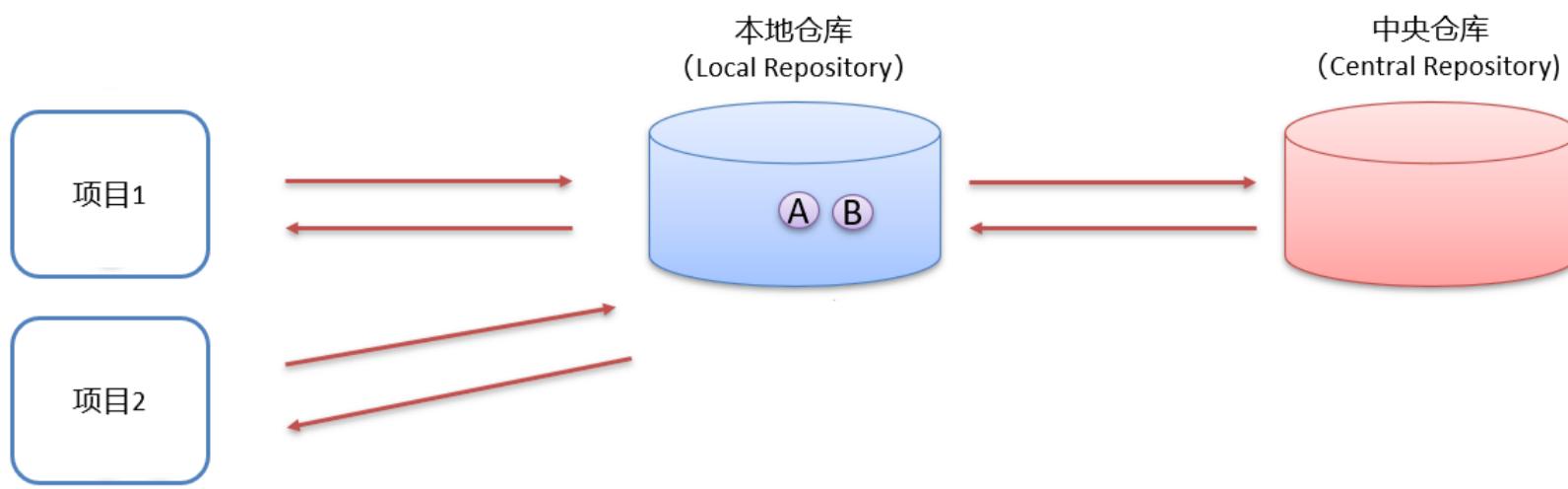
仓库分类：

- 本地仓库：自己计算机上的一个目录
- 中央仓库：由Maven团队维护的全球唯一的仓库
 - 地址：<https://repo1.maven.org/maven2/>
- 远程仓库(私服)：一般由公司团队搭建的私有仓库

今天我们只学习远程仓库的使用，并不会搭建。

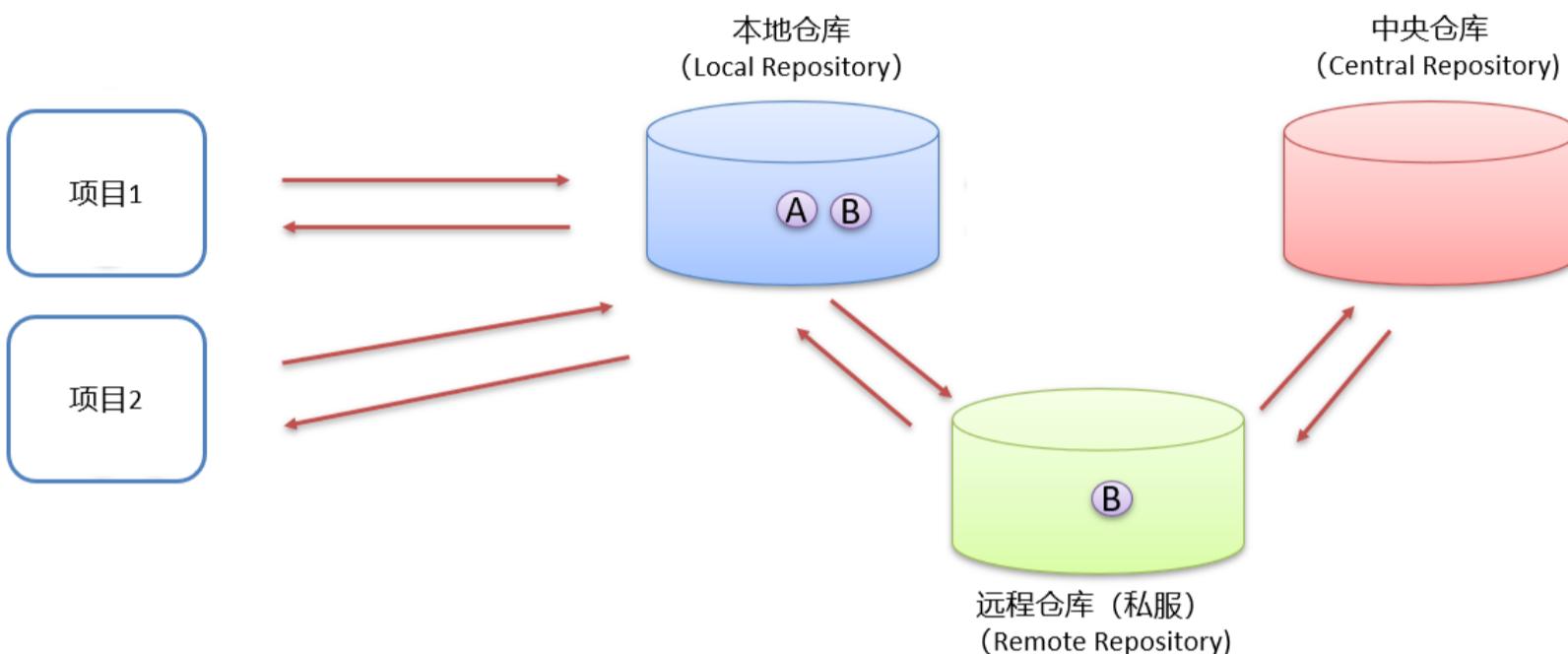
当项目中使用坐标引入对应依赖jar包后，首先会查找本地仓库中是否有对应的jar包：

- 如果有，则在项目直接引用；
- 如果没有，则去中央仓库中下载对应的jar包到本地仓库。



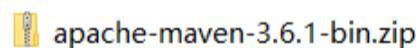
如果还可以搭建远程仓库，将来jar包的查找顺序则变为：

本地仓库 --> 远程仓库--> 中央仓库



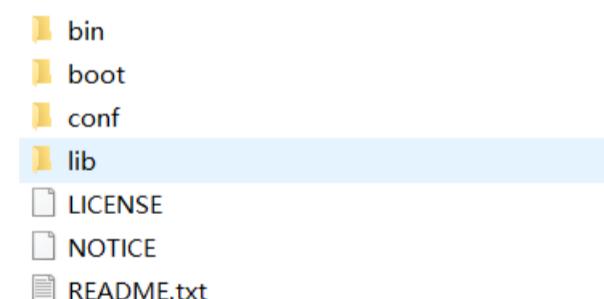
1.2 Maven安装配置

- 解压 apache-maven-3.6.1.rar 既安装完成



建议解压缩到没有中文、特殊字符的路径下。如课程中解压缩到 D:\software 下。

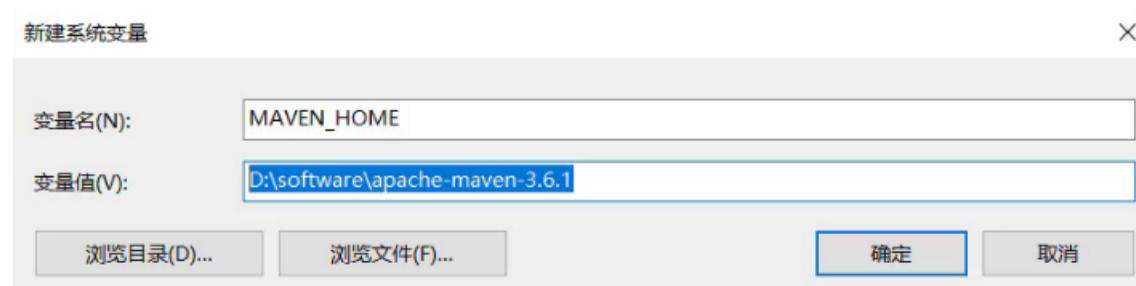
解压缩后的目录结构如下：



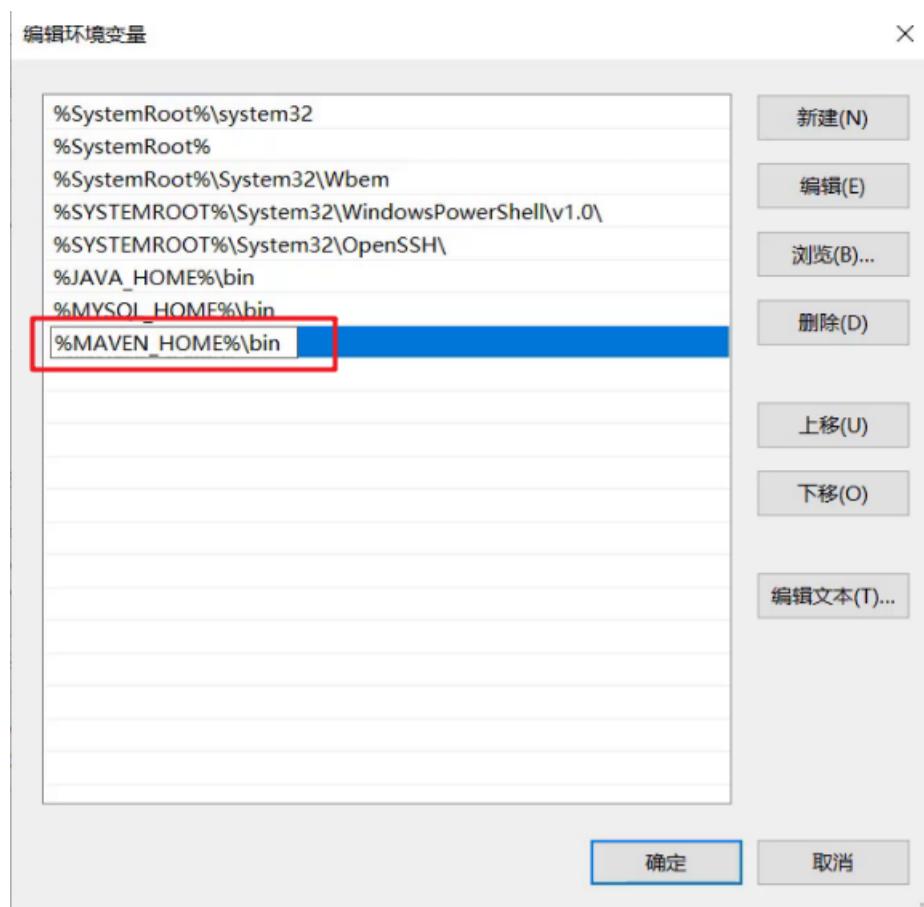
- bin 目录：存放的是可执行命令。mvn 命令重点关注。
- conf 目录：存放Maven的配置文件。settings.xml 配置文件后期需要修改。
- lib 目录：存放Maven依赖的jar包。Maven也是使用java开发的，所以它也依赖其他的jar包。
- 配置环境变量 MAVEN_HOME 为安装路径的bin目录

此电脑 右键 --> 高级系统设置 --> 高级 --> 环境变量

在系统变量处新建一个变量 MAVEN_HOME



在 Path 中进行配置



打开命令提示符进行验证，出现如图所示表示安装成功

```
C:\Users\super>mvn -version
Apache Maven 3.6.1 (d66c9c0b3152b2e69ee9bac180bb8fcc8e6af555; 2019-04-05T03:00:29+08:00)
Maven home: D:\software\apache-maven-3.6.1\bin\..
Java version: 1.8.0_172, vendor: Oracle Corporation, runtime: D:\software\jdk8_72\jre
Default locale: zh_CN, platform encoding: GBK
OS name: "windows 10", version: "10.0", arch: "amd64", family: "windows"
```

- 配置本地仓库

修改 conf/settings.xml 中的 为一个指定目录作为本地仓库，用来存储jar包。

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0 http://maven.apache.org/xsd/settings-1.0.0.xsd
  >
  <!-- localRepository
    | The path to the local repository maven will use to store artifacts.
    |
    | Default: ${user.home}/.m2/repository
  <localRepository>/path/to/local/repo</localRepository>
  <!-- interactiveMode
    |
```

- 配置阿里云私服

中央仓库在国外，所以下载jar包速度可能比较慢，而阿里公司提供了一个远程仓库，里面基本也都有开源项目的jar包。

修改 conf/settings.xml 中的 标签，为其添加如下子标签：

```
1 <mirror>
2   <id>alimaven</id>
3   <name>aliyun maven</name>
4   <url>http://maven.aliyun.com/nexus/content/groups/public/</url>
5   <mirrorOf>central</mirrorOf>
6 </mirror>
```

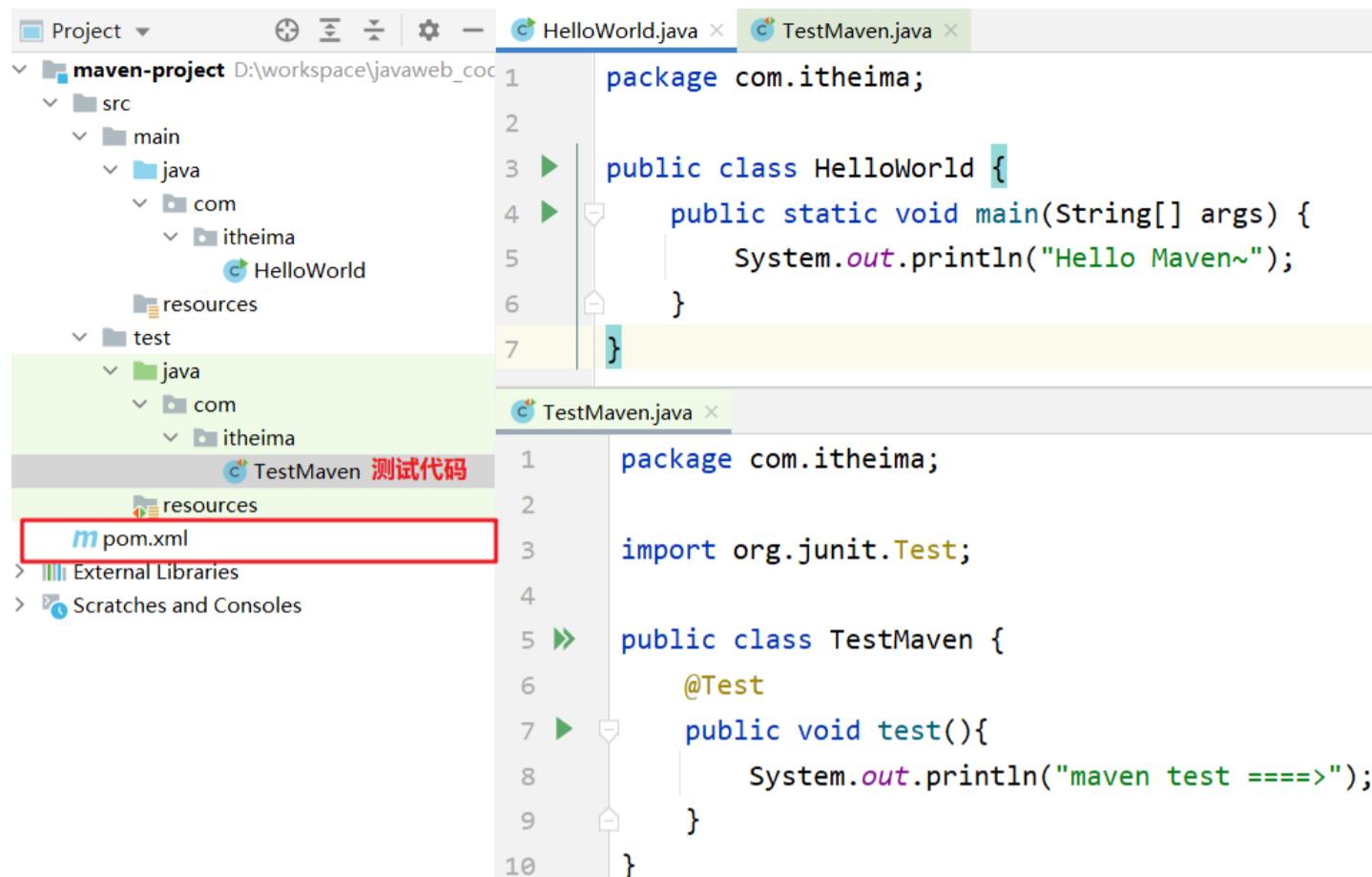
1.3 Maven基本使用

1.3.1 Maven 常用命令

- compile：编译
- clean：清理
- test：测试
- package：打包
- install：安装

命令演示：

在 `资料\代码\maven-project` 提供了一个使用Maven构建的项目，项目结构如下：



而我们使用上面命令需要在磁盘上进入到项目的 `pom.xml` 目录下，打开命令提示符

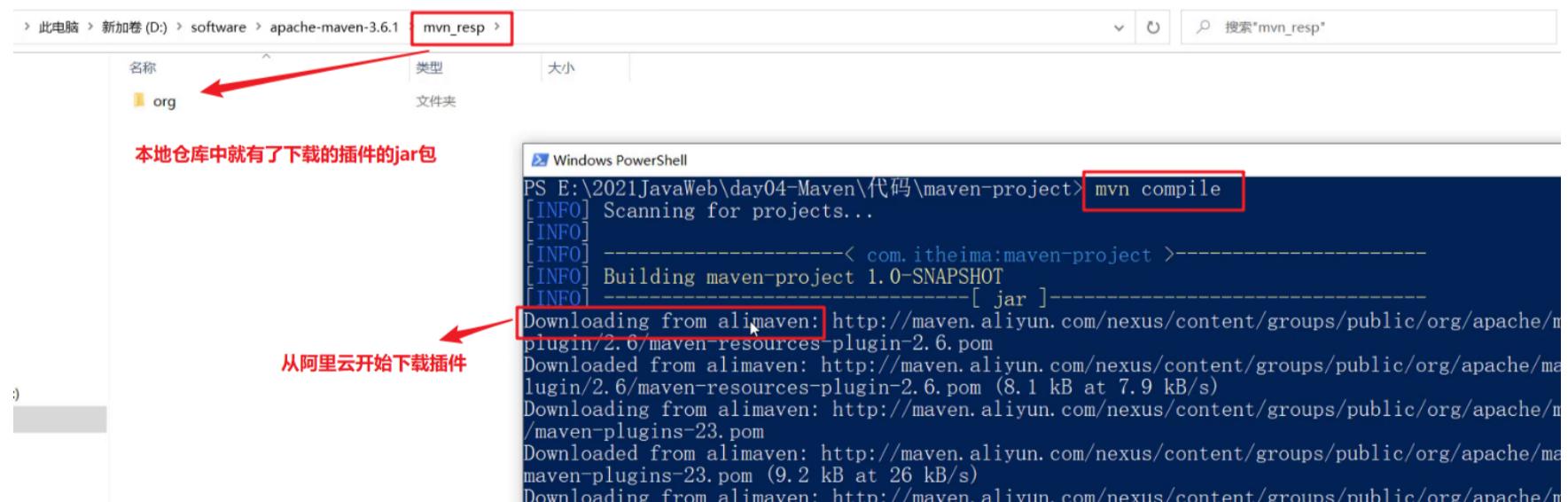


编译命令演示：

1 | `compile` : 编译

执行上述命令可以看到：

- 从阿里云下载编译需要的插件的jar包，在本地仓库也能看到下载好的插件
- 在项目下会生成一个 `target` 目录



同时在项目下会出现一个 `target` 目录，编译后的字节码文件就放在该目录下



清理命令演示：

```
1 | mvn clean
```

执行上述命令可以看到

- 从阿里云下载清理需要的插件jar包
- 删除项目下的 target 目录

```
PS E:\2021JavaWeb\day04-Maven\代码\maven-project> mvn clean
[INFO] Scanning for projects...
[INFO]
[INFO] -----< com.itheima:maven-project >-----
[INFO] Building maven-project 1.0-SNAPSHOT
[INFO] -----[ jar ]-----
Downloading from alimaven: http://maven.aliyun.com/nexus/content/groups/public/org/apache/maven/2.5/maven-clean-plugin-2.5.pom
Downloaded from alimaven: http://maven.aliyun.com/nexus/content/groups/public/org/apache/maven/2.5/maven-clean-plugin-2.5.pom (3.9 kB at 4.1 kB/s)
Downloading from alimaven: http://maven.aliyun.com/nexus/content/groups/public/org/apache/maven/maven-plugins-22.pom
Downloaded from alimaven: http://maven.aliyun.com/nexus/content/groups/public/org/apache/maven/maven-plugins-22.pom (13 kB at 40 kB/s)
Downloading from alimaven: http://maven.aliyun.com/nexus/content/groups/public/org/apache/maven/2.5/maven-clean-plugin-2.5.jar
Downloaded from alimaven: http://maven.aliyun.com/nexus/content/groups/public/org/apache/maven/2.5/maven-clean-plugin-2.5.jar (25 kB at 77 kB/s)
[INFO]
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ maven-project ---
Downloading from alimaven: http://maven.aliyun.com/nexus/content/groups/public/org/codehaus/us-utils-3.0.pom
Progress (1) · 4 1 kB
```

打包命令演示：

```
1 | mvn package
```

执行上述命令可以看到：

- 从阿里云下载打包需要的插件jar包
- 在项目的 target 目录下有一个jar包（将当前项目打成的jar包）

```
PS E:\2021JavaWeb\day04-Maven\代码\maven-project> mvn package
[INFO] Scanning for projects...
[INFO]
[INFO] -----< com.itheima:maven-project >-----
[INFO] Building maven-project 1.0-SNAPSHOT
[INFO] -----[ jar ]-----
Downloading from alimaven: http://maven.aliyun.com/nexus/content/groups/public/org/apache/maven/plugin/2.12.4/maven-surefire-plugin-2.12.4.pom
Downloaded from alimaven: http://maven.aliyun.com/nexus/content/groups/public/org/apache/maven/plugin/2.12.4/maven-surefire-plugin-2.12.4.pom (10 kB at 12 kB/s)
Downloading from alimaven: http://maven.aliyun.com/nexus/content/groups/public/org/apache/maven/
```

测试命令演示：

```
1 | mvn test
```

该命令会执行所有的测试代码。执行上述命令效果如下

```
-----  
T E S T S  
-----  
Running com.itheima.TestMaven  
maven test ===>  
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.066 sec  
  
Results :  
  
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0  
  
[INFO] -----  
[INFO] BUILD SUCCESS  
[INFO] -----  
[INFO] Total time: 1.955 s  
[INFO] Finished at: 2021-05-06T21:04:49+08:00  
[INFO] -----
```

安装命令演示：

```
1 | mvn install
```

该命令会将当前项目打成jar包，并安装到本地仓库。执行完上述命令后到本地仓库查看结果如下：

i > 新加卷 (D:) > software > apache-maven-3.6.1 > mvn_resp > com > itheima > maven-project > 1.0-SNAPSHOT		
名称	类型	大小
_remote.repositories	REPOSITORIES ...	1 KB
maven-metadata-local.xml	XML 文件	1 KB
maven-project-1.0-SNAPSHOT.jar	Executable Jar File	3 KB
maven-project-1.0-SNAPSHOT.pom	POM 文件	2 KB

1.3.2 Maven 生命周期

Maven 构建项目生命周期描述的是一次构建过程经历经历了多少个事件

Maven 对项目构建的生命周期划分为3套：

- clean：清理工作。
- default：核心工作，例如编译，测试，打包，安装等。
- site：产生报告，发布站点等。这套声明周期一般不会使用。

同一套生命周期内，执行后边的命令，前面的所有命令会自动执行。例如默认（default）生命周期如下：



当我们执行 `install`（安装）命令时，它会先执行 `compile` 命令，再执行 `test` 命令，再执行 `package` 命令，最后执行 `install` 命令。

当我们执行 `package`（打包）命令时，它会先执行 `compile` 命令，再执行 `test` 命令，最后执行 `package` 命令。

默认的生命周期也有对应的很多命令，其他的一般都不会使用，我们只关注常用的：

validate (校验)	校验项目是否正确并且所有必要的信息可以完成项目的构建过程。
initialize (初始化)	初始化构建状态，比如设置属性值。
generate-sources (生成源代码)	生成包含在编译阶段中的任何源代码。
process-sources (处理源代码)	处理源代码，比如说，过滤任意值。
generate-resources (生成资源文件)	生成将会包含在项目包中的资源文件。
process-resources (处理资源文件)	复制和处理资源到目标目录，为打包阶段做好准备。
compile (编译)	编译项目的源代码。
process-classes (处理类文件)	处理编译生成的文件，比如说对Java class文件做字节码改善优化。
generate-test-sources (生成测试源代码)	生成包含在编译阶段中的任何测试源代码。
process-test-sources (处理测试源代码)	处理测试源代码，比如说，过滤任意值。
generate-test-resources (生成测试资源文件)	为测试创建资源文件。
process-test-resources (处理测试资源文件)	复制和处理测试资源到目标目录。
test-compile (编译测试源码)	编译测试源代码到测试目标目录。
process-test-classes (处理测试类文件)	处理测试源码编译生成的文件。
test (测试)	使用合适的单元测试框架运行测试（ <u>Junit</u> 是其中之一）。
prepare-package (准备打包)	在实际打包之前，执行任何的必要的操作为打包做准备。
package (打包)	将编译后的代码打包成可分发格式的文件，比如JAR、WAR或者EAR文件。
pre-integration-test (集成测试前)	在执行集成测试前进行必要的动作。比如说，搭建需要的环境。
integration-test (集成测试)	处理和部署项目到可以运行集成测试环境中。
post-integration-test (集成测试后)	在执行集成测试完成后进行必要的动作。比如说，清理集成测试环境。
verify (验证)	运行任意的检查来验证项目包有效且达到质量标准。
install (安装)	安装项目包到本地仓库，这样项目包可以用作其他本地项目的依赖。
deploy (部署)	将最终的项目包复制到远程仓库中与其他开发者和项目共享。

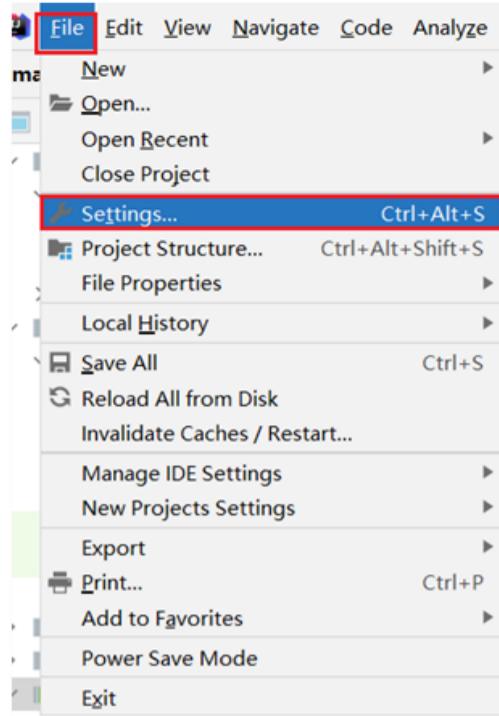
1.4 IDEA使用Maven

以后开发中我们肯定会在高级开发工具中使用Maven管理项目，而我们常用的高级开发工具是IDEA，所以接下来我们会讲解Maven在IDEA中的使用。

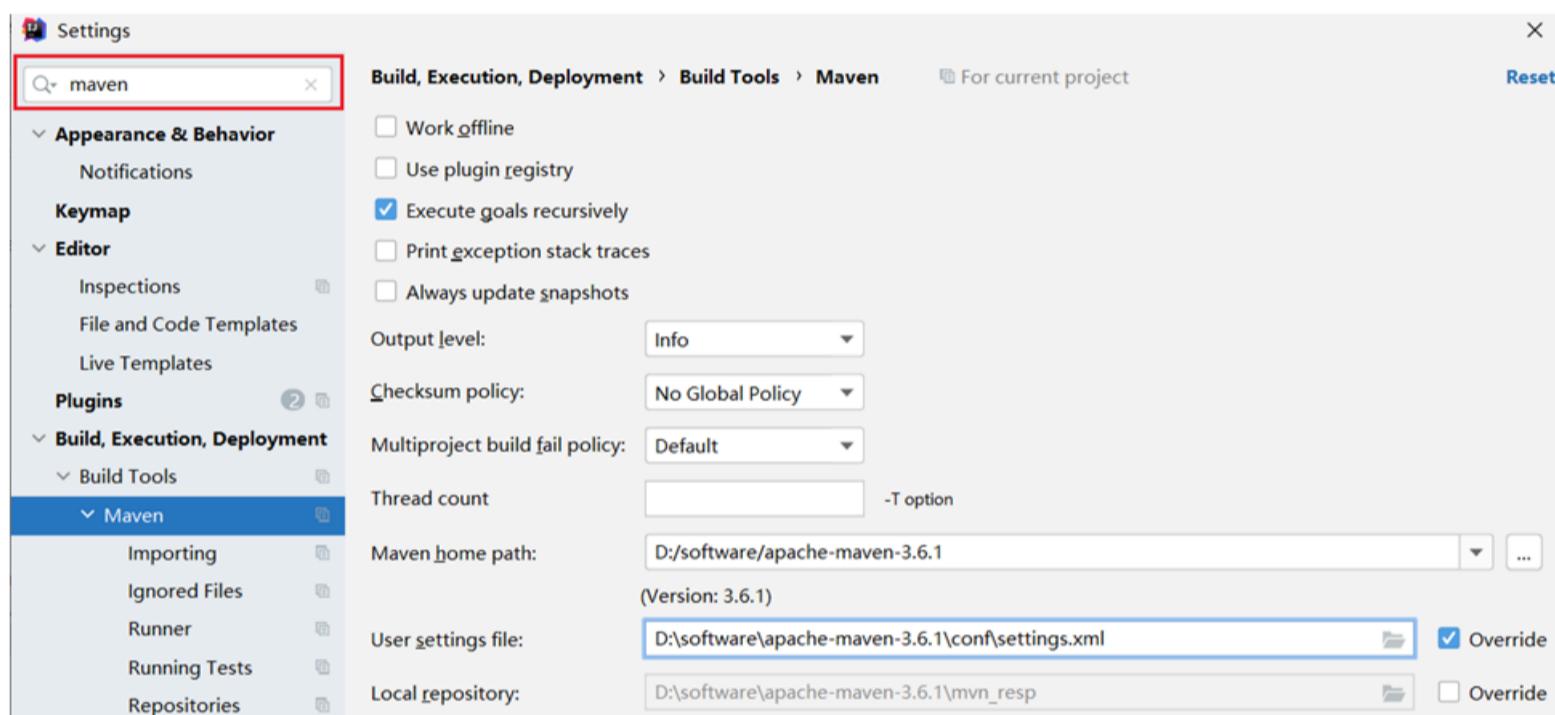
1.4.1 IDEA配置Maven环境

我们需要先在IDEA中配置Maven环境：

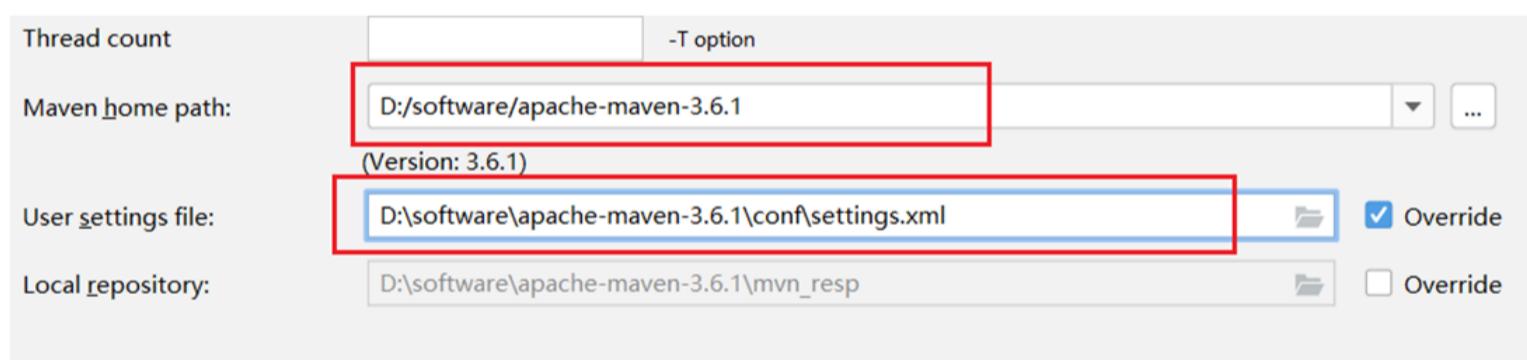
- 选择 IDEA 中 File --> Settings



- 搜索 maven



- 设置 IDEA 使用本地安装的 Maven，并修改配置文件路径



1.4.2 Maven 坐标详解

什么是坐标？

- Maven 中的坐标是**资源的唯一标识**
- 使用坐标来定义项目或引入项目中需要的依赖

Maven 坐标主要组成

- groupId: 定义当前Maven项目隶属组织名称（通常是域名反写，例如：com.itheima）
- artifactId: 定义当前Maven项目名称（通常是模块名称，例如 order-service、goods-service）
- version: 定义当前项目版本号

如下图就是使用坐标表示一个项目：

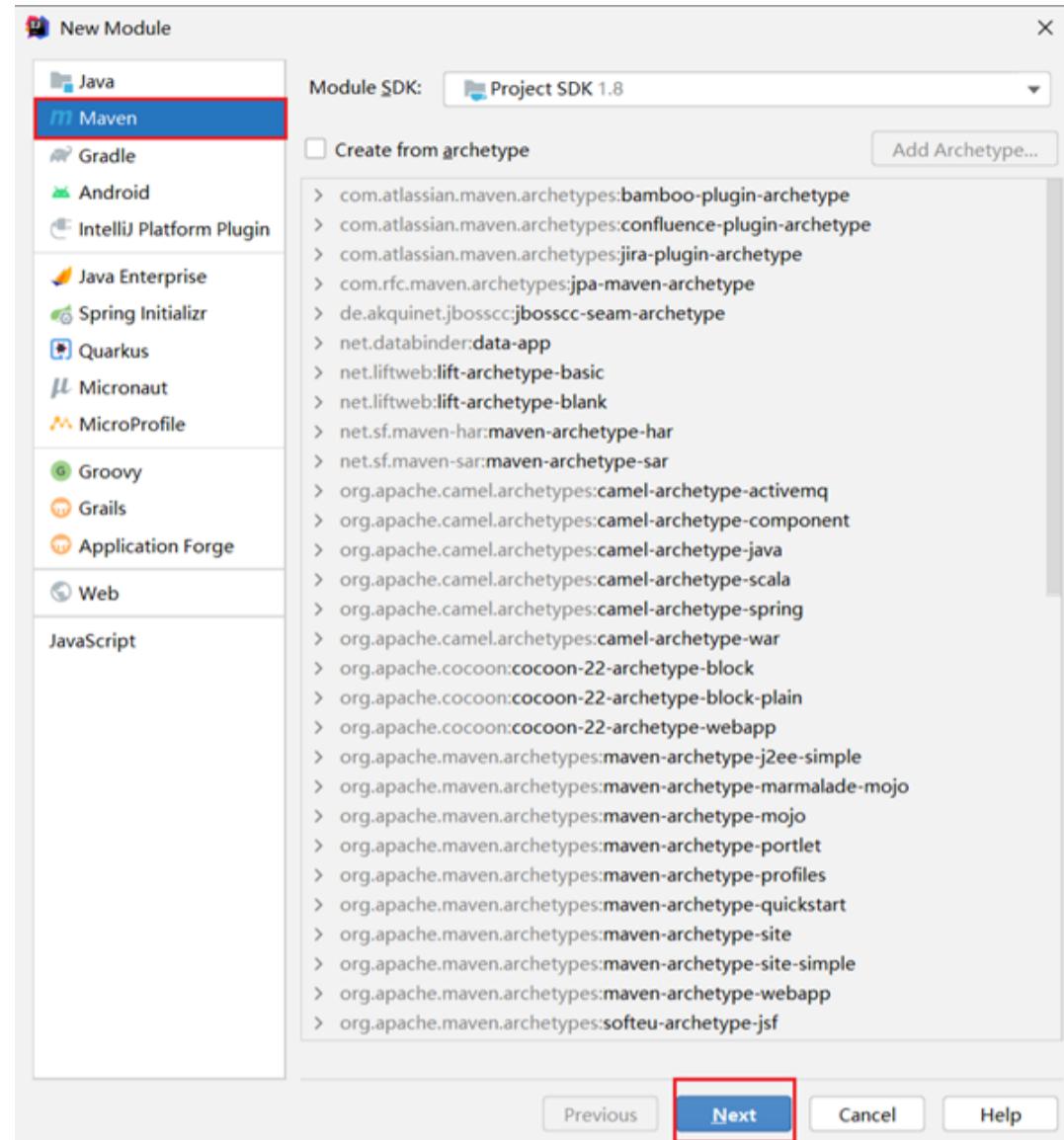
```
<groupId>com.itheima</groupId>
<artifactId>maven-demo</artifactId>
<version>1.0-SNAPSHOT</version>
```

注意：

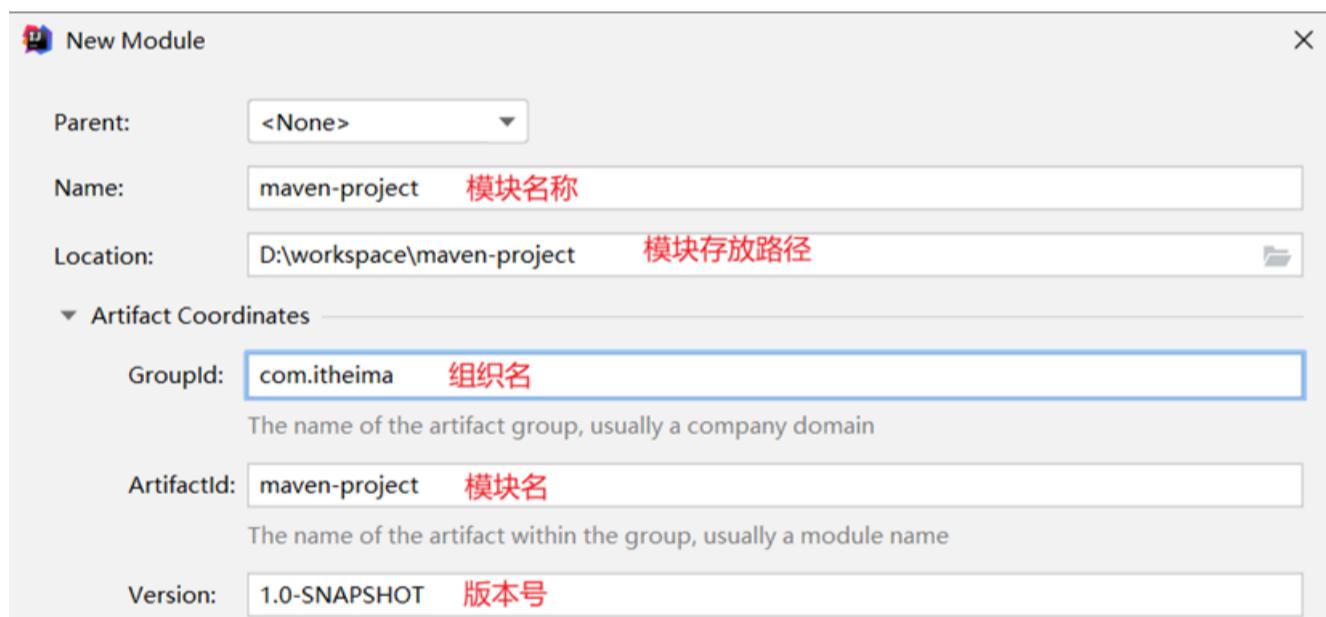
- 上面所说的资源可以是插件、依赖、当前项目。
- 我们的项目如果被其他的项目依赖时，也是需要坐标来引入的。

1.4.3 IDEA 创建 Maven项目

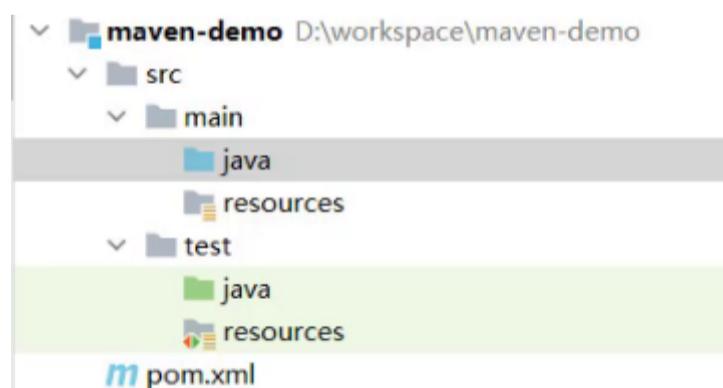
- 创建模块，选择Maven，点击Next



- 填写模块名称，坐标信息，点击finish，创建完成



创建好的项目目录结构如下：

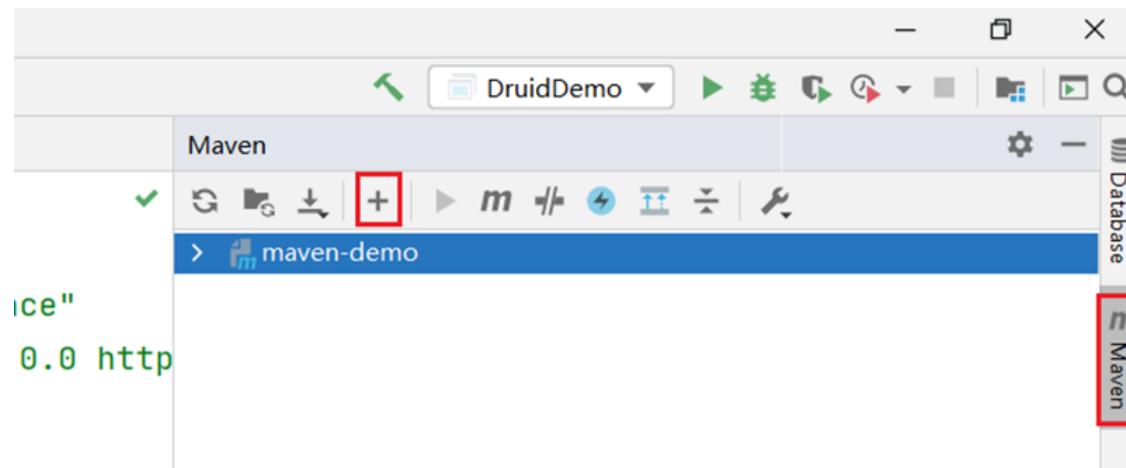


- 编写 HelloWorld，并运行

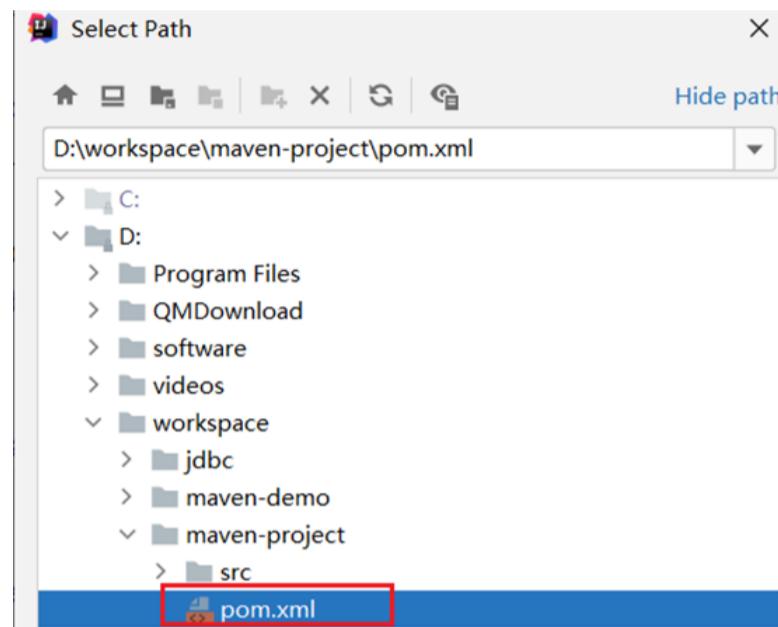
1.4.4 IDEA 导入 Maven项目

大家在学习时可能需要看老师的代码，当然也就需要将老师的代码导入到自己的IDEA中。我们可以通过以下步骤进行项目的导入：

- 选择右侧Maven面板，点击 + 号

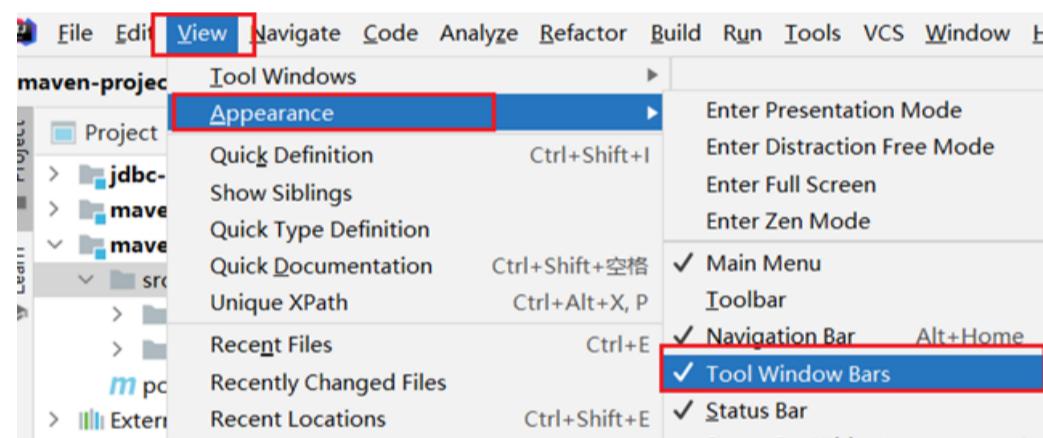


- 选中对应项目的pom.xml文件，双击即可

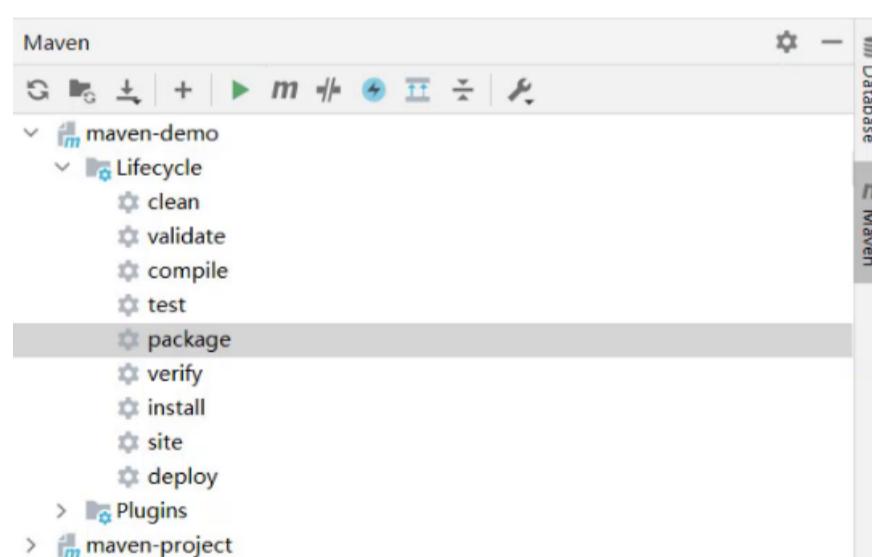


- 如果没有Maven面板，选择

View --> Appearance --> Tool Window Bars

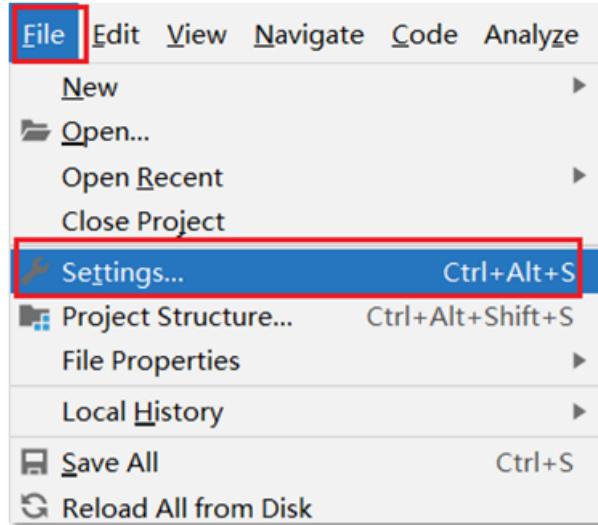


可以通过下图所示进行命令的操作：

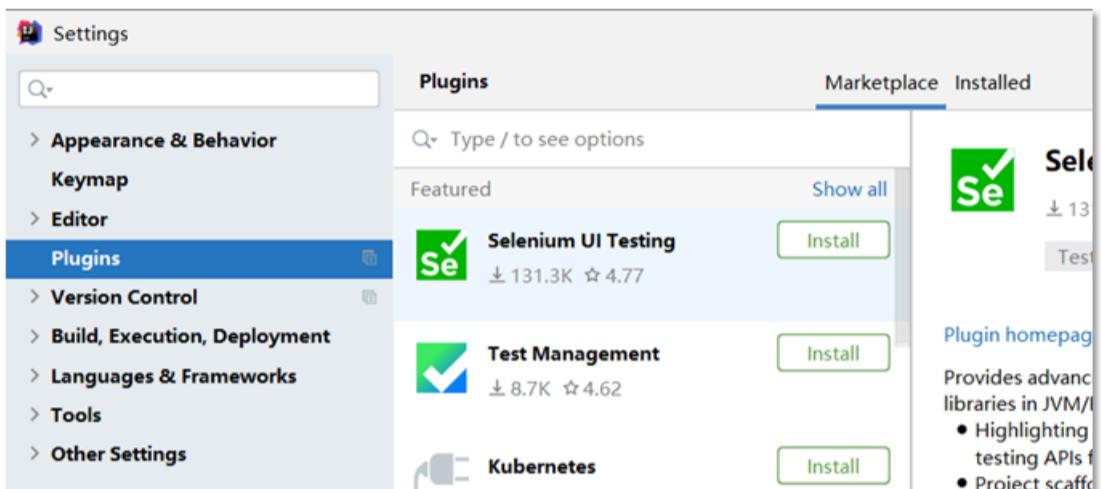


配置 Maven-Helper 插件

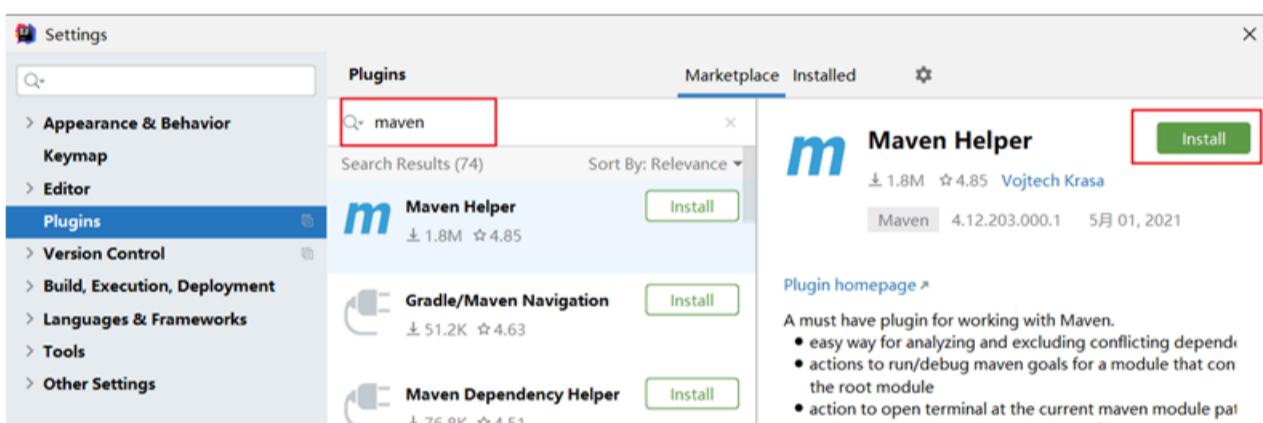
- 选择 IDEA 中 File --> Settings



- 选择 Plugins

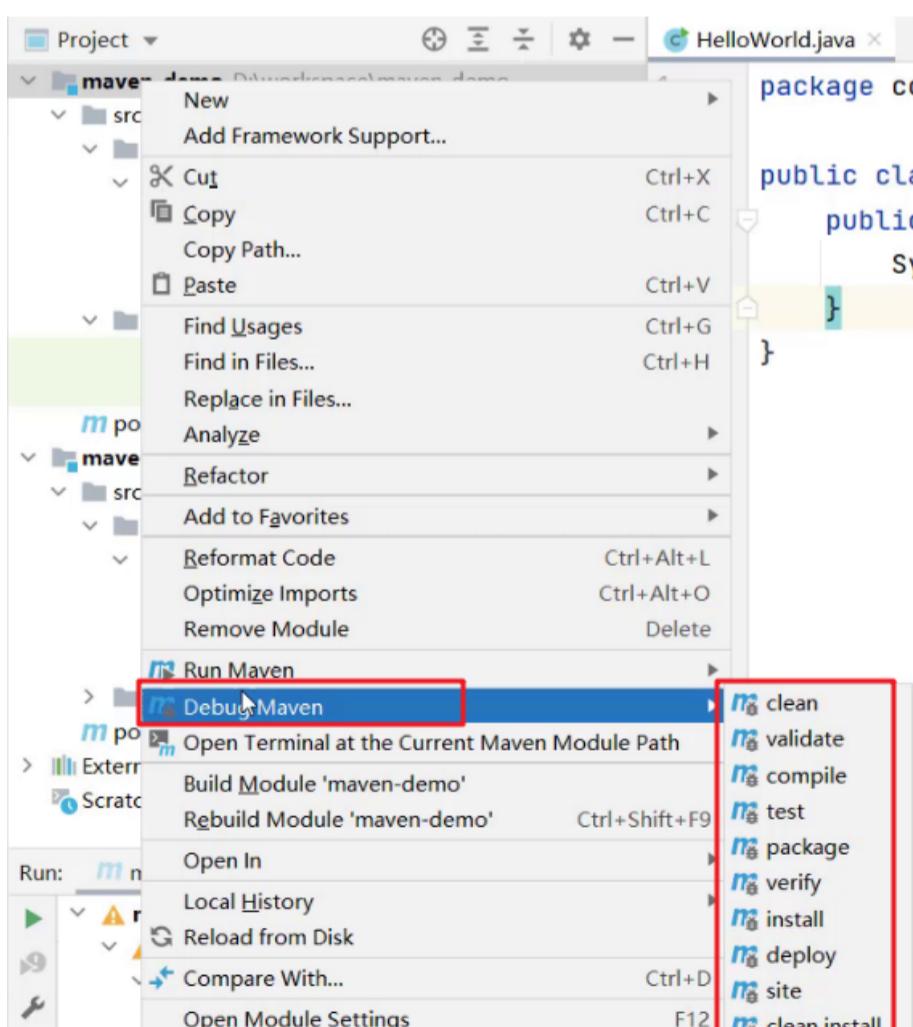


- 搜索 Maven，选择第一个 Maven Helper，点击Install安装，弹出面板中点击Accept



- 重启 IDEA

安装完该插件后可以通过 选中项目右键进行相关命令操作，如下图所示：



1.5 依赖管理

1.5.1 使用坐标引入jar包

使用坐标引入jar包的步骤：

- 在项目的 pom.xml 中编写 标签
- 在 标签中 使用 引入坐标
- 定义坐标的 groupId, artifactId, version

```
<dependencies>
    <!-- mysql 坐标 -->
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>5.1.32</version>
    </dependency>
</dependencies>
```

- 点击刷新按钮，使坐标生效



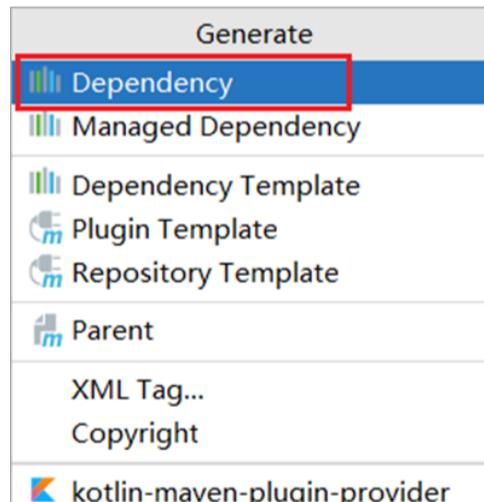
注意：

- 具体的坐标我们可以到如下网站进行搜索
- <https://mvnrepository.com/>

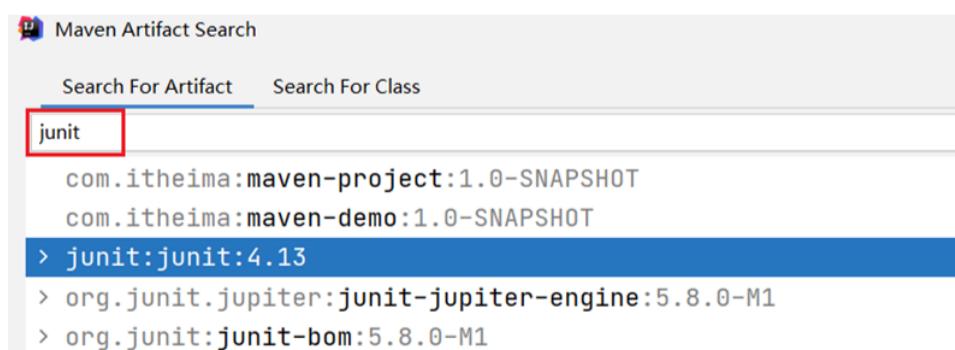
快捷方式导入jar包的坐标：

每次需要引入jar包，都去对应的网站进行搜索是比较麻烦的，接下来给大家介绍一种快捷引入坐标的方式

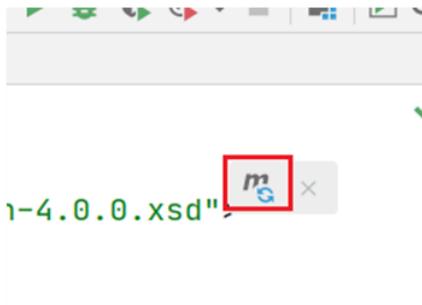
- 在 pom.xml 中按 alt + insert，选择 Dependency



- 在弹出的面板中搜索对应坐标，然后双击选中对应坐标



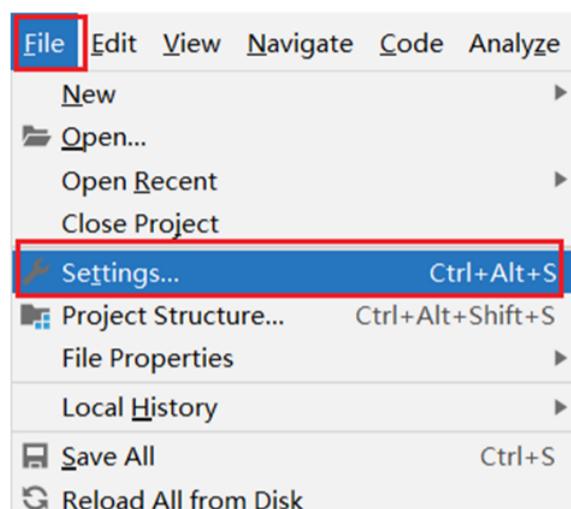
- 点击刷新按钮，使坐标生效



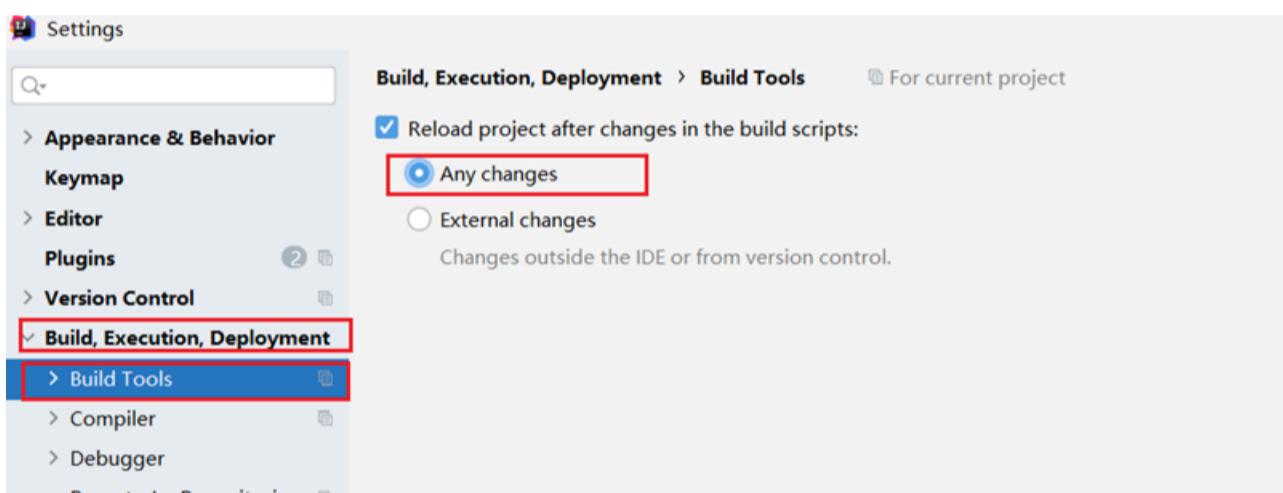
自动导入设置：

上面每次操作都需要点击刷新按钮，让引入的坐标生效。当然我们也可以通过设置让其自动完成

- 选择 IDEA 中 File --> Settings



- 在弹出的面板中找到 Build Tools



- 选择 Any changes，点击 ok 即可生效

1.5.2 依赖范围

通过设置坐标的依赖范围(scope)，可以设置 对应jar包的作用范围：编译环境、测试环境、运行环境。

如下图所示给 `junit` 依赖通过 `scope` 标签指定依赖的作用范围。那么这个依赖就只能作用在测试环境，其他环境下不能使用。

```

<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.13</version>
    <scope>test</scope>
</dependency>

```

那么 `scope` 都可以有哪些取值呢？

依赖范围	编译classpath	测试classpath	运行classpath	例子
compile	Y	Y	Y	logback
test	-	Y	-	Junit
provided	Y	Y	-	servlet-api
runtime	-	Y	Y	jdbc驱动
system	Y	Y	-	存储在本地的jar包

- compile：作用于编译环境、测试环境、运行环境。

- test：作用于测试环境。典型的就是Junit坐标，以后使用Junit时，都会将scope指定为该值
- provided：作用于编译环境、测试环境。我们后面会学习 `servlet-api`，在使用它时，必须将 `scope` 设置为该值，不然运行时就会报错
- runtime：作用于测试环境、运行环境。jdbc驱动一般将 `scope` 设置为该值，当然不设置也没有任何问题

注意：

- 如果引入坐标不指定 `scope` 标签时，默认就是 `compile` 值。以后大部分jar包都是使用默认值。

2, Mybatis

2.1 Mybatis概述

2.1.1 Mybatis概念

- MyBatis 是一款优秀的持久层框架，用于简化 JDBC 开发
- MyBatis 本是 Apache 的一个开源项目iBatis, 2010年这个项目由apache software foundation 迁移到了google code，并且改名为MyBatis 。2013年11月迁移到Github
- 官网：<https://mybatis.org/mybatis-3/zh/index.html>

持久层：

- 负责将数据到保存到数据库的那一层代码。

以后开发我们会将操作数据库的Java代码作为持久层。而Mybatis就是对jdbc代码进行了封装。

- JavaEE三层架构：表现层（页面展示）、业务层（逻辑处理）、持久层（数据存储）

三层架构在后期会给大家进行讲解，今天先简单的了解下即可。

框架：

- 框架就是一个半成品软件，是一套可重用的、通用的、软件基础代码模型
- 在框架的基础之上构建软件编写更加高效、规范、通用、可扩展

举例给大家简单的解释一下什么是半成品软件。大家小时候应该在公园见过给石膏娃娃涂鸦



如下图所示有一个石膏娃娃，这个就是一个半成品。你可以在这个半成品的基础上进行不同颜色的涂鸦



了解了什么是Mybatis后，接下来说说以前 `JDBC代码` 的缺点以及Mybatis又是如何解决的。

2.1.2 JDBC 缺点

下面是 JDBC 代码，我们通过该代码分析都存在什么缺点：

```
//1. 注册驱动  
Class.forName("com.mysql.jdbc.Driver");  
//2. 获取Connection连接  
String url = "jdbc:mysql://db1?useSSL=false"; ①  
String uname = "root";  
String pwd = "1234";  
Connection conn = DriverManager.getConnection(url, uname, pwd);  
//接收输入的查询条件  
String gender = "男";  
// 定义sql  
String sql = "select *from tb_user where gender = ?"; ②  
//获取PreparedStatement对象  
PreparedStatement pstmt = conn.prepareStatement(sql);  
// 设置? 的值  
pstmt.setString(1,gender); ③  
//执行sql  
ResultSet rs = pstmt.executeQuery();  
//遍历Result, 获取数据  
User user = null;  
ArrayList<User> users = new ArrayList<>();  
while (rs.next()) {  
    //获取数据  
    int id = rs.getInt("id");  
    String username = rs.getString("username");  
    String password = rs.getString("password"); ④  
    //创建对象, 设置属性值  
    user = new User();  
    user.setId(id);  
    user.setUsername(username);  
    user.setPassword(password);  
    user.setGender(gender);  
    //装入集合  
    users.add(user);  
}
```

- 硬编码

- 注册驱动、获取连接

上图标1的代码有很多字符串，而这些是连接数据库的四个基本信息，以后如果要将Mysql数据库换成其他的关系型数据库的话，这四个地方都需要修改，如果放在此处就意味着要修改我们的源代码。

- SQL语句

上图标2的代码。如果表结构发生变化，SQL语句就要进行更改。这也不方便后期的维护。

- 操作繁琐

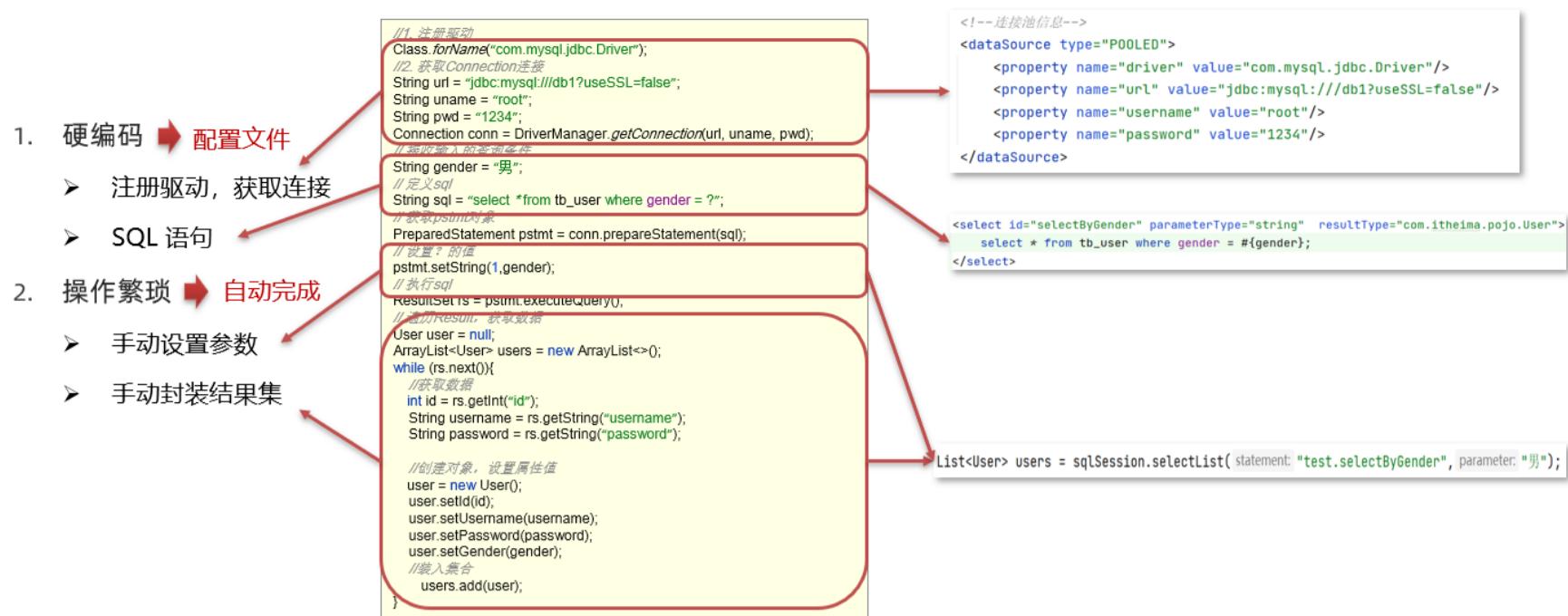
- 手动设置参数
 - 手动封装结果集

上图标4的代码是对查询到的数据进行封装，而这部分代码是没有什么技术含量，而且特别耗费时间的。

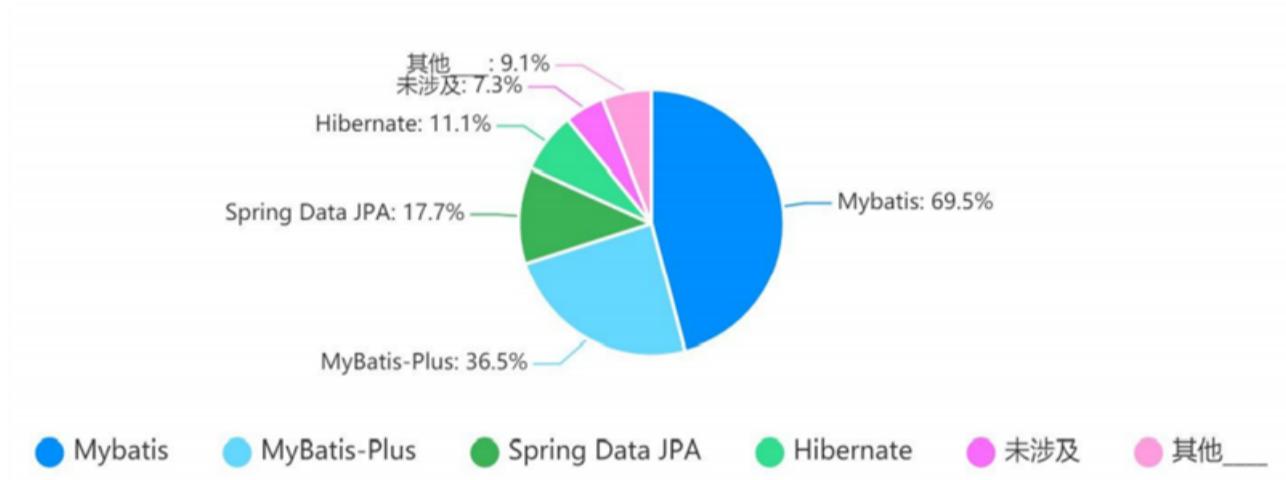
2.1.3 Mybatis 优化

- 硬编码可以配置到**配置文件**
- 操作繁琐的地方mybatis都**自动完成**

如图所示



下图是持久层框架的使用占比。



2.2 Mybatis快速入门

需求：查询user表中所有的数据

- 创建user表，添加数据

```

1  create database mybatis;
2  use mybatis;
3
4  drop table if exists tb_user;
5
6  create table tb_user(
7      id int primary key auto_increment,
8      username varchar(20),
9      password varchar(20),
10     gender char(1),
11     addr varchar(30)
12 );
13
14 INSERT INTO tb_user VALUES (1, 'zhangsan', '123', '男', '北京');
15 INSERT INTO tb_user VALUES (2, '李四', '234', '女', '天津');
16 INSERT INTO tb_user VALUES (3, '王五', '11', '男', '西安');

```

- 创建模块，导入坐标

在创建好的模块中的 pom.xml 配置文件中添加依赖的坐标

```

1 <dependencies>
2     <!--mybatis 依赖-->
3     <dependency>
4         <groupId>org.mybatis</groupId>
5         <artifactId>mybatis</artifactId>
6         <version>3.5.5</version>
7     </dependency>
8
9     <!--mysql 驱动-->
10    <dependency>

```

```

11      <groupId>mysql</groupId>
12      <artifactId>mysql-connector-java</artifactId>
13      <version>5.1.46</version>
14  </dependency>
15
16  <!--junit 单元测试-->
17  <dependency>
18      <groupId>junit</groupId>
19      <artifactId>junit</artifactId>
20      <version>4.13</version>
21      <scope>test</scope>
22  </dependency>
23
24  <!-- 添加slf4j日志api -->
25  <dependency>
26      <groupId>org.slf4j</groupId>
27      <artifactId>slf4j-api</artifactId>
28      <version>1.7.20</version>
29  </dependency>
30  <!-- 添加logback-classic依赖 -->
31  <dependency>
32      <groupId>ch.qos.logback</groupId>
33      <artifactId>logback-classic</artifactId>
34      <version>1.2.3</version>
35  </dependency>
36  <!-- 添加logback-core依赖 -->
37  <dependency>
38      <groupId>ch.qos.logback</groupId>
39      <artifactId>logback-core</artifactId>
40      <version>1.2.3</version>
41  </dependency>
42 </dependencies>

```

注意：需要在项目的 resources 目录下创建logback的配置文件

- 编写 MyBatis 核心配置文件 --> 替换连接信息 解决硬编码问题

在模块下的 resources 目录下创建mybatis的配置文件 `mybatis-config.xml`，内容如下：

```

1  <?xml version="1.0" encoding="UTF-8" ?>
2  <!DOCTYPE configuration
3      PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
4      "http://mybatis.org/dtd/mybatis-3-config.dtd">
5 <configuration>
6
7     <typeAliases>
8         <package name="com.itheima.pojo"/>
9     </typeAliases>
10
11    <!--
12        environments: 配置数据库连接环境信息。可以配置多个environment，通过default属性切换不同的
13        environment
14        -->
15        <environments default="development">
16            <environment id="development">
17                <transactionManager type="JDBC"/>
18                <dataSource type="POOLED">
19                    <!--数据库连接信息-->
20                    <property name="driver" value="com.mysql.jdbc.Driver"/>
21                    <property name="url" value="jdbc:mysql:///mybatis?useSSL=false"/>
22                    <property name="username" value="root"/>
23                    <property name="password" value="1234"/>
24                </dataSource>
25            </environment>
26            <environment id="test">

```

```

27         <transactionManager type="JDBC"/>
28     <dataSource type="POOLED">
29         <!--数据库连接信息-->
30         <property name="driver" value="com.mysql.jdbc.Driver"/>
31         <property name="url" value="jdbc:mysql:///mybatis?useSSL=false"/>
32         <property name="username" value="root"/>
33         <property name="password" value="1234"/>
34     </dataSource>
35   </environment>
36 </environments>
37 <mappers>
38   <!--加载sql映射文件-->
39   <mapper resource="UserMapper.xml"/>
40 </mappers>
41 </configuration>

```

- 编写 SQL 映射文件 -> 统一管理sql语句，解决硬编码问题

在模块的 `resources` 目录下创建映射配置文件 `UserMapper.xml`，内容如下：

```

1 <?xml version="1.0" encoding="UTF-8" ?>
2 <!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
 "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
3 <mapper namespace="test">
4   <select id="selectAll" resultType="com.itheima.pojo.User">
5     select * from tb_user;
6   </select>
7 </mapper>

```

- 编码

- 在 `com.itheima.pojo` 包下创建 User类

```

1 public class User {
2     private int id;
3     private String username;
4     private String password;
5     private String gender;
6     private String addr;
7
8     //省略了 setter 和 getter
9 }

```

- 在 `com.itheima` 包下编写 MybatisDemo 测试类

```

1 public class MyBatisDemo {
2
3     public static void main(String[] args) throws IOException {
4         //1. 加载mybatis的核心配置文件，获取 SqlSessionFactory
5         String resource = "mybatis-config.xml";
6         InputStream inputStream = Resources.getResourceAsStream(resource);
7         SqlSessionFactory sqlSessionFactory = new
8         SqlSessionFactoryBuilder().build(inputStream);
9
10        //2. 获取SqlSession对象，用它来执行sql
11        SqlSession sqlSession = sqlSessionFactory.openSession();
12        //3. 执行sql
13        List<User> users = sqlSession.selectList("test.selectAll"); //参数是一个字符串，该
14        //字符串必须是映射配置文件的namespace.id
15        System.out.println(users);
16        //4. 释放资源
17        sqlSession.close();
18    }
19 }

```

解决SQL映射文件的警告提示：

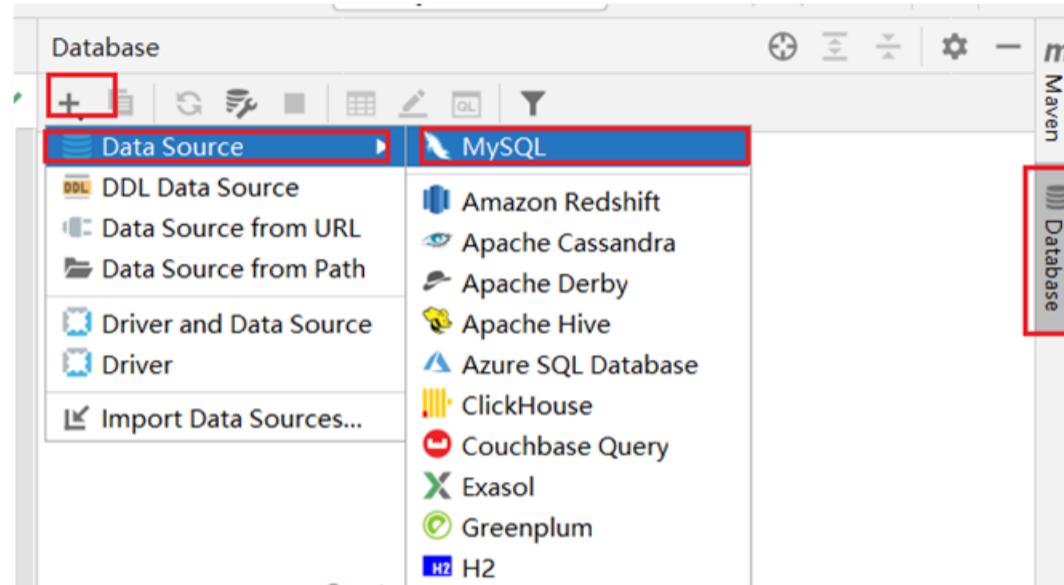
在入门案例映射配置文件中存在报红的情况。问题如下：

```
<mapper namespace="test">
    <select id="selectAll" resultType="com.itheima.pojo.User">
        select * from tb_user;
    </select>
</mapper>
```

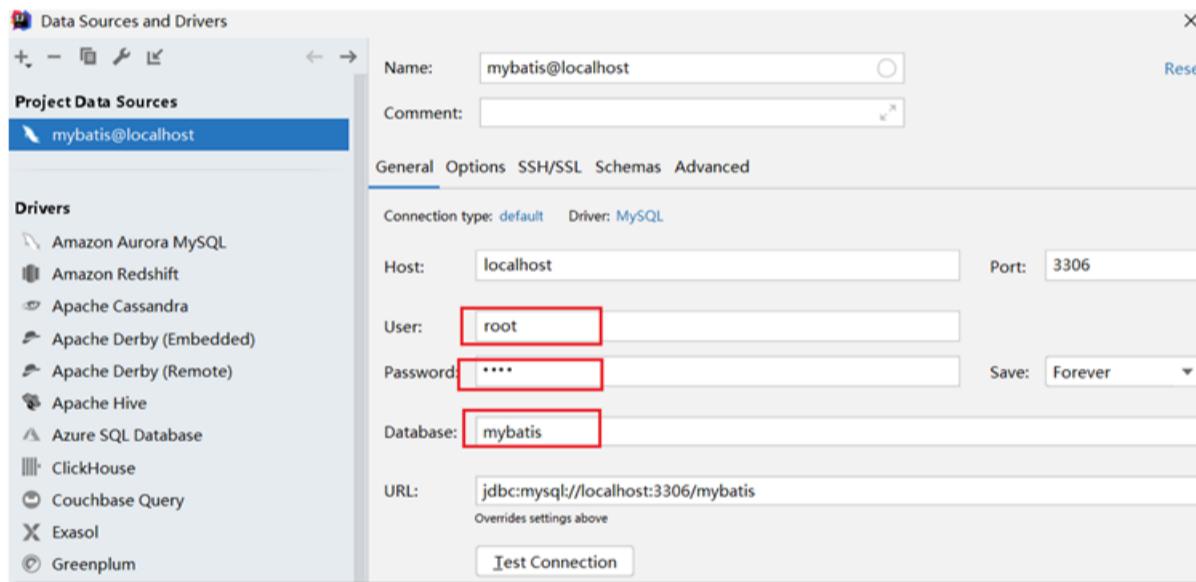
- 产生的原因：Idea和数据库没有建立连接，不识别表信息。但是大家一定要记住，它并不影响程序的执行。
- 解决方式：在Idea中配置MySQL数据库连接。

IDEA中配置MySQL数据库连接

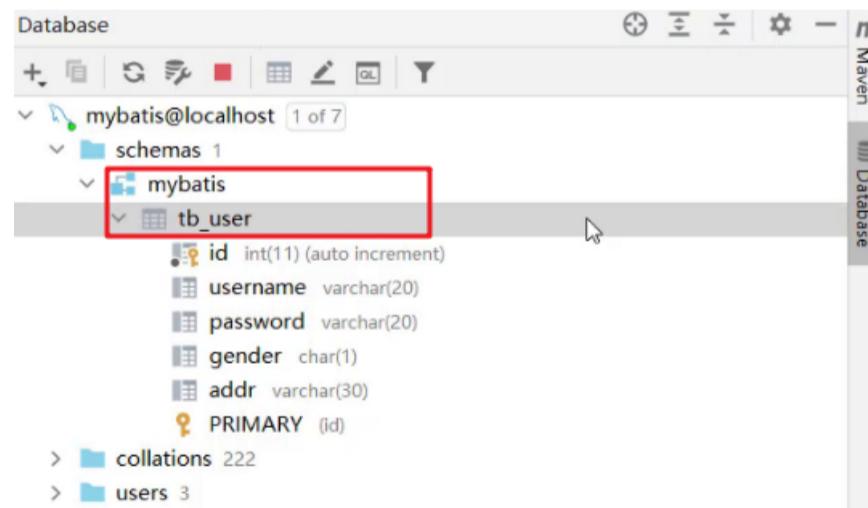
- 点击IDEA右边框的 **Database**，在展开的界面点击 **+** 选择 **Data Source**，再选择 **MySQL**



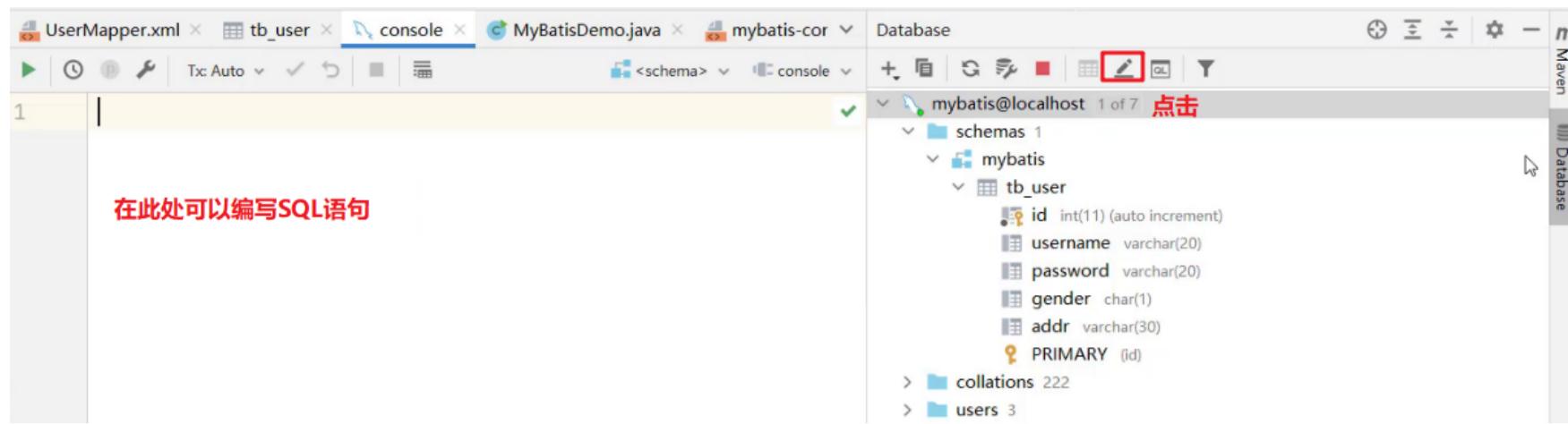
- 在弹出的界面进行基本信息的填写



- 点击完成后就能看到如下界面



而此界面就和 **navicat** 工具一样可以进行数据库的操作。也可以编写SQL语句



2.3 Mapper代理开发

2.3.1 Mapper代理开发概述

之前我们写的代码是基本使用方式，它也存在硬编码的问题，如下：

```
//3. 执行sql  
List<User> users = sqlSession.selectList(statement: "test.selectAll");  
System.out.println(users);
```

这里调用 `selectList()` 方法传递的参数是映射配置文件中的 `namespace.id` 值。这样写也不便于后期的维护。如果使用 Mapper 代理方式（如下图）则不存在硬编码问题。

```
//3. 获取接口代理对象  
UserMapper userMapper = sqlSession.getMapper(UserMapper.class);  
//4. 执行方法，其实就是执行sql语句  
List<User> users = userMapper.selectAll();
```

通过上面的描述可以看出 Mapper 代理方式的目的：

- 解决原生方式中的硬编码
- 简化后期执行SQL

Mybatis 官网也是推荐使用 Mapper 代理的方式。下图是截止官网的图片

为了这个简单的例子，我们似乎写了不少配置，但其实并不多。在一个 XML 映射文件中，可以定义无数个映射语句，这样一来，XML 头部和文档类型声明部分就显得微不足道了。文档的其它部分很直白，容易理解。它在命名空间 “org.mybatis.example.BlogMapper” 中定义了一个名为 “selectBlog” 的映射语句，这样你就可以用全限定名 “org.mybatis.example.BlogMapper.selectBlog” 来调用映射语句了，就像上面例子中那样：

```
Blog blog = (Blog) session.selectOne("org.mybatis.example.BlogMapper.selectBlog", 101);
```

你可能会注意到，这种方式和用全限定名调用 Java 对象的方法类似。这样，该命名就可以直接映射到在命名空间中同名的映射器类，并将已映射的 select 语句匹配到对应名称、参数和返回类型的方法。因此你就可以像上面那样，不费吹灰之力地在对应的映射器接口调用方法，就像下面这样：

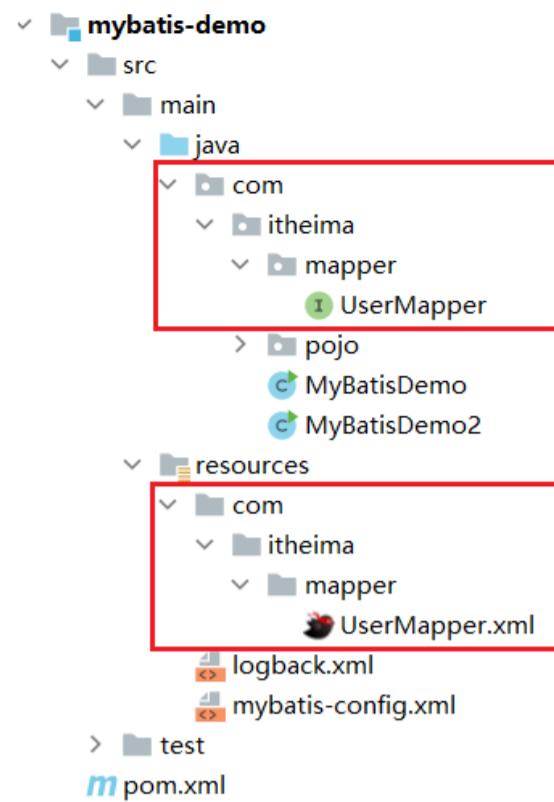
```
BlogMapper mapper = session.getMapper(BlogMapper.class);  
Blog blog = mapper.selectBlog(101);
```

第二种方法有很多优势，首先它不依赖于字符串字面值，会更安全一点；其次，如果你的 IDE 有代码补全功能，那么代码补全可以帮你快速选择到映射好的 SQL 语句。

2.3.2 使用Mapper代理要求

使用Mapper代理方式，必须满足以下要求：

- 定义与SQL映射文件同名的Mapper接口，并且将Mapper接口和SQL映射文件放置在同一目录下。如下图：



- 设置SQL映射文件的namespace属性为Mapper接口全限定名

```

<!--
  namespace: 名称空间。必须是对应接口的全限定名
-->
<mapper namespace="com.itheima.mapper.UserMapper">

```

- 在Mapper接口中定义方法，方法名就是SQL映射文件中sql语句的id，并保持参数类型和返回值类型一致

```

UserMapper.xml
<!--
  namespace: 名称空间。必须是对应接口的全限定名
-->
<mapper namespace="com.itheima.mapper.UserMapper">
  <select id="selectAll" resultType="com.itheima.pojo.User">
    select *
    from tb_user;
  </select>
</mapper>

UserMapper.java
package com.itheima.mapper;

import ...

public interface UserMapper {
  List<User> selectAll();
}

```

2.3.3 案例代码实现

- 在 `com.itheima.mapper` 包下创建 `UserMapper` 接口，代码如下：

```

1 public interface UserMapper {
2     List<User> selectAll();
3     User selectById(int id);
4 }

```

- 在 `resources` 下创建 `com/itheima/mapper` 目录，并在该目录下创建 `UserMapper.xml` 映射配置文件

```

1 <!--
2   namespace: 名称空间。必须是对应接口的全限定名
3 -->
4 <mapper namespace="com.itheima.mapper.UserMapper">
5   <select id="selectAll" resultType="com.itheima.pojo.User">
6     select *
7     from tb_user;
8   </select>
9 </mapper>

```

- 在 `com.itheima` 包下创建 `MybatisDemo2` 测试类，代码如下：

```

1  /**
2  * Mybatis 代理开发
3  */
4  public class MyBatisDemo2 {
5
6      public static void main(String[] args) throws IOException {
7
8          //1. 加载mybatis的核心配置文件, 获取 SqlSessionFactory
9          String resource = "mybatis-config.xml";
10         InputStream inputStream = Resources.getResourceAsStream(resource);
11         SqlSessionFactory sqlSessionFactory = new
12         SqlSessionFactoryBuilder().build(inputStream);
13
14         //2. 获取SqlSession对象, 用它来执行sql
15         SqlSession sqlSession = sqlSessionFactory.openSession();
16         //3. 执行sql
17         //3.1 获取UserMapper接口的代理对象
18         UserMapper userMapper = sqlSession.getMapper(UserMapper.class);
19         List<User> users = userMapper.selectAll();
20
21         System.out.println(users);
22         //4. 释放资源
23         sqlSession.close();
24     }
25 }
```

注意：

如果Mapper接口名称和SQL映射文件名称相同，并在同一目录下，则可以使用包扫描的方式简化SQL映射文件的加载。也就是将核心配置文件的加载映射配置文件的配置修改为

```

1 <mappers>
2     <!--加载sql映射文件-->
3     <!-- <mapper resource="com/itheima/mapper/UserMapper.xml"/>-->
4     <!--Mapper代理方式-->
5     <package name="com.itheima.mapper"/>
6 </mappers>
```

2.4 核心配置文件

核心配置文件中现有的配置之前已经给大家进行了解释，而核心配置文件中还可以配置很多内容。我们可以通过查询官网看可以配置的内容



配置

MyBatis 的配置文件包含了会深深影响 MyBatis 行为的设置和属性信息。配置文档的顶层结构如下：

- configuration (配置)
 - properties (属性)
 - settings (设置)
 - typeAliases (类型别名)
 - typeHandlers (类型处理器)
 - objectFactory (对象工厂)
 - plugins (插件)
 - environments (环境配置)
 - environment (环境变量)
 - transactionManager (事务管理器)
 - dataSource (数据源)
 - databaseIdProvider (数据库厂商标识)
 - mappers (映射器)

接下来我们先对里面的一些配置进行讲解。

2.4.1 多环境配置

在核心配置文件的 `environments` 标签中其实是可以配置多个 `environment`，使用 `id` 给每段环境起名，在 `environments` 中使用 `default='环境id'` 来指定使用哪儿段配置。我们一般就配置一个 `environment` 即可。

```
1 <environments default="development">
2   <environment id="development">
3     <transactionManager type="JDBC"/>
4     <dataSource type="POOLED">
5       <!--数据库连接信息-->
6       <property name="driver" value="com.mysql.jdbc.Driver"/>
7       <property name="url" value="jdbc:mysql://mybatis?useSSL=false"/>
8       <property name="username" value="root"/>
9       <property name="password" value="1234"/>
10      </dataSource>
11    </environment>
12
13    <environment id="test">
14      <transactionManager type="JDBC"/>
15      <dataSource type="POOLED">
16        <!--数据库连接信息-->
17        <property name="driver" value="com.mysql.jdbc.Driver"/>
18        <property name="url" value="jdbc:mysql://mybatis?useSSL=false"/>
19        <property name="username" value="root"/>
20        <property name="password" value="1234"/>
21      </dataSource>
22    </environment>
23 </environments>=
```

2.4.2 类型别名

在映射配置文件中的 `resultType` 属性需要配置数据封装的类型（类的全限定名）。而每次这样写是特别麻烦的，Mybatis 提供了 `类型别名 (typeAliases)` 可以简化这部分的书写。

首先需要在核心配置文件中配置类型别名，也就意味着给pojo包下所有的类起了别名（别名就是类名），不区分大小写。内容如下：

```
1 <typeAliases>
2   <!--name属性的值是实体类所在包-->
3   <package name="com.itheima.pojo"/>
4 </typeAliases>
```

通过上述的配置，我们就可以简化映射配置文件中 `resultType` 属性值的编写

```
1 <mapper namespace="com.itheima.mapper.UserMapper">
2   <select id="selectAll" resultType="user">
3     select * from tb_user;
4   </select>
5 </mapper>
```

Mybatis练习

目标

- 能够使用映射配置文件实现CRUD操作
- 能够使用注解实现CRUD操作

1. 配置文件实现CRUD

The screenshot shows a web-based product management system. On the left, there's a sidebar with links like '商品列表', '新增商品', '商品评价', '商品回收站', '商品属性', '商品分类', '新增分类', '商品品牌', '新增品牌', and '商品规格'. The main area has tabs for '商品管理' (selected), '订单管理', '物流管理', '促销管理', '文章管理', '权限管理', and 'VIP原型'. A navigation bar at the top includes '首页', '当前状态' dropdown, '企业名称' search input, '品牌名称' search input, a '搜索' button, and a '重置' button. Below these are buttons for '数据列表' and '删除'. The central part displays a table with data:

序号	品牌LOGO	品牌名称	企业名称	排序	当前状态	操作
010		三只松鼠	这里是企业名称	6	<input checked="" type="checkbox"/>	删除 编辑 查看详情
009		优衣库	这里是企业名称	2	<input checked="" type="checkbox"/>	删除 编辑 查看详情
008		小米	这里是企业名称	1	<input checked="" type="checkbox"/>	删除 编辑 查看详情
007		阿迪达斯	这里是企业名称	2	<input checked="" type="checkbox"/>	删除 编辑 查看详情
006		百草味	这里是企业名称	13	<input type="checkbox"/>	删除 编辑 查看详情

如上图所示产品原型，里面包含了品牌数据的查询、按条件查询、添加、删除、批量删除、修改等功能，而这些功能其实就是对数据库表中的数据进行CRUD操作。接下来我们就使用Mybatis完成品牌数据的增删改查操作。以下是我们要完成功能列表：

- 查询
 - 查询所有数据
 - 查询详情
 - 条件查询
- 添加
- 修改
 - 修改全部字段
 - 修改动态字段
- 删除
 - 删除一个
 - 批量删除

我们先将必要的环境准备一下。

1.1 环境准备

- 数据库表 (tb_brand) 及数据准备

```
1 -- 删除tb_brand表
2 drop table if exists tb_brand;
3 -- 创建tb_brand表
4 create table tb_brand
5 (
6     -- id 主键
7     id      int primary key auto_increment,
8     -- 品牌名称
9     brand_name  varchar(20),
10    -- 企业名称
11    company_name varchar(20),
12    -- 排序字段
13    ordered      int,
14    -- 描述信息
15    description  varchar(100),
16    -- 状态: 0: 禁用 1: 启用
```

```

17     status      int
18 );
19 -- 添加数据
20 insert into tb_brand (brand_name, company_name, ordered, description, status)
21 values ('三只松鼠', '三只松鼠股份有限公司', 5, '好吃不上火', 0),
22         ('华为', '华为技术有限公司', 100, '华为致力于把数字世界带入每个人、每个家庭、每个组织，构建万物互联
23         的智能世界', 1),
24         ('小米', '小米科技有限公司', 50, 'are you ok', 1);

```

- 实体类 Brand

在 `com.itheima.pojo` 包下创建 Brand 实体类。

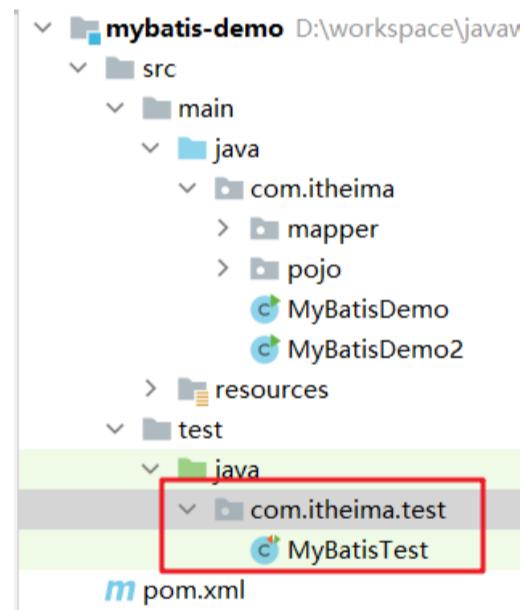
```

1 public class Brand {
2     // id 主键
3     private Integer id;
4     // 品牌名称
5     private String brandName;
6     // 企业名称
7     private String companyName;
8     // 排序字段
9     private Integer ordered;
10    // 描述信息
11    private String description;
12    // 状态: 0: 禁用 1: 启用
13    private Integer status;
14
15    //省略 setter and getter。自己写时要补全这部分代码
16 }

```

- 编写测试用例

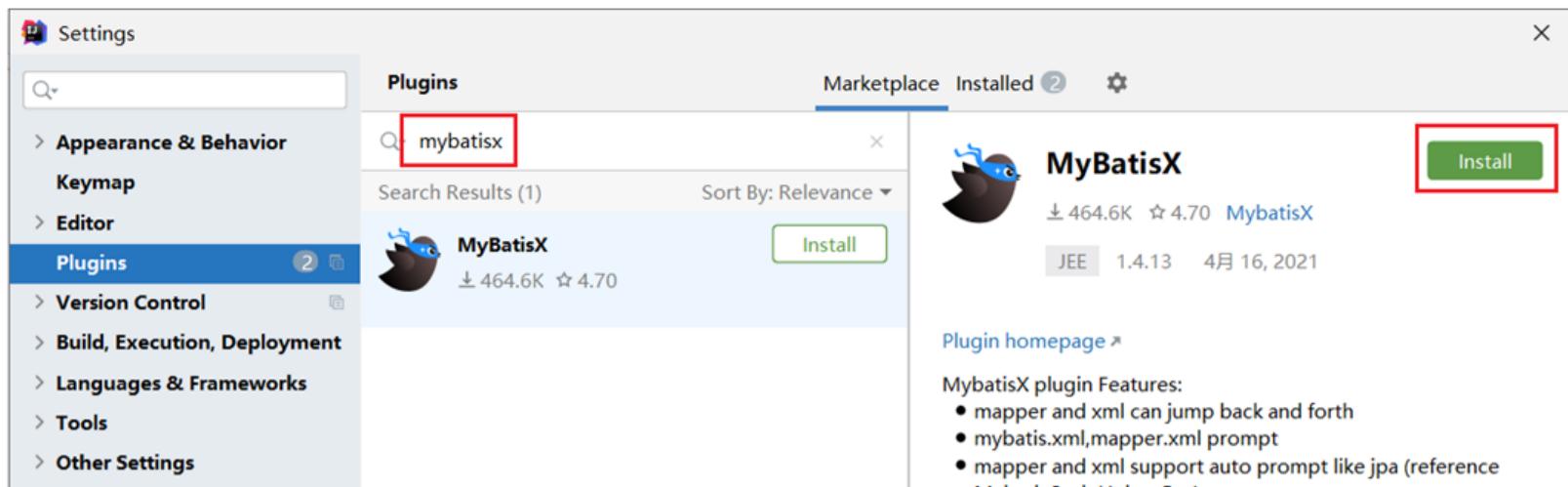
测试代码需要在 `test/java` 目录下创建包及测试用例。项目结构如下：



- 安装 MyBatisX 插件

- MybatisX 是一款基于 IDEA 的快速开发插件，为效率而生。
- 主要功能
 - XML映射配置文件 和 接口方法 间相互跳转
 - 根据接口方法生成 statement
- 安装方式

点击 `file`，选择 `settings`，就能看到如下图所示界面



注意：安装完毕后需要重启IDEA

- 插件效果

红色头绳的表示映射配置文件，蓝色头绳的表示mapper接口。在mapper接口点击红色头绳的小鸟图标会自动跳转到对应的映射配置文件，在映射配置文件中点击蓝色头绳的小鸟图标会自动跳转到对应的mapper接口。也可以在 mapper接口中定义方法，自动生成映射文件中的 statement，如图所示



1.2 查询所有数据

<input type="checkbox"/>	序号	品牌LOGO	品牌名称	企业名称	排序	当前状态
<input type="checkbox"/>	010		三只松鼠	这里是企业名称	6	<input checked="" type="checkbox"/>
<input type="checkbox"/>	009		优衣库	这里是企业名称	2	<input checked="" type="checkbox"/>
<input type="checkbox"/>	008		小米	这里是企业名称	1	<input checked="" type="checkbox"/>
<input type="checkbox"/>	007		阿迪达斯	这里是企业名称	2	<input checked="" type="checkbox"/>
<input type="checkbox"/>	006		百草味	这里是企业名称	13	<input type="checkbox"/>

如上图所示就页面上展示的数据，而这些数据需要从数据库进行查询。接下来我们就来讲查询所有数据功能，而实现该功能我们分以下步骤进行实现：

- 编写接口方法：Mapper接口

- 参数：无

查询所有数据功能是不需要根据任何条件进行查询的，所以此方法不需要参数。

```
List<Brand> selectAll();
```

- 结果: List

我们会将查询出来的每一条数据封装成一个 `Brand` 对象，而多条数据封装多个 `Brand` 对象，需要将这些对象封装到 `List` 集合中返回。

```
<select id="selectAll" resultType="brand">
    select * from tb_brand;
</select>
```

- 执行方法、测试

1.2.1 编写接口方法

在 `com.itheima.mapper` 包下创建名为 `BrandMapper` 的接口。并在该接口中定义 `List<Brand> selectAll()` 方法。

```
1 public interface BrandMapper {
2
3     /**
4      * 查询所有
5      */
6     List<Brand> selectAll();
7 }
```

1.2.2 编写SQL语句

在 `resources` 下创建 `com/itheima/mapper` 目录结构，并在该目录下创建名为 `BrandMapper.xml` 的映射配置文件

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <!DOCTYPE mapper
3     PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
4     "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
5
6 <mapper namespace="com.itheima.mapper.BrandMapper">
7     <select id="selectAll" resultType="brand">
8         select *
9         from tb_brand;
10    </select>
11 </mapper>
```

1.2.3 编写测试方法

在 `MybatisTest` 类中编写测试查询所有的方法

```
1 @Test
2 public void testSelectAll() throws IOException {
3     //1. 获取SqlSessionFactory
4     String resource = "mybatis-config.xml";
5     InputStream inputStream = Resources.getResourceAsStream(resource);
6     SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);
7
8     //2. 获取SqlSession对象
9     SqlSession sqlSession = sqlSessionFactory.openSession();
10
11    //3. 获取Mapper接口的代理对象
12    BrandMapper brandMapper = sqlSession.getMapper(BrandMapper.class);
13
14    //4. 执行方法
15    List<Brand> brands = brandMapper.selectAll();
16    System.out.println(brands);
17
18    //5. 释放资源
19    sqlSession.close();
20}
```

注意：现在我们感觉测试这部分代码写起来特别麻烦，我们可以先忍忍。以后我们只会写上面的第3步的代码，其他的都不需要我们来完成。

执行测试方法结果如下：

```

Tests passed: 1 of 1 test - 700 ms
MyBatisTest (700 ms) [DEBUG] 12:19:32.101 [main] o.a.i.i.ResolverUtil - Checking to see if class com.itheima.mapper.BrandMapper matches criteria [is ass
[DEBUG] 12:19:32.101 [main] o.a.i.i.ResolverUtil - Checking to see if class com.itheima.mapper.UserMapper matches criteria [is ass
[DEBUG] 12:19:32.193 [main] o.a.i.t.j.JdbcTransaction - Opening JDBC Connection
[DEBUG] 12:19:32.399 [main] o.a.i.d.p.PooledDataSource - Created connection 1629687658.
[DEBUG] 12:19:32.399 [main] o.a.i.t.j.JdbcTransaction - Setting autocommit to false on JDBC Connection [com.mysql.jdbc.JDBC4Connect
[DEBUG] 12:19:32.403 [main] c.i.m.B.selectAll - ==> Preparing: select * from tb_brand; 执行的sql语句
[DEBUG] 12:19:32.428 [main] c.i.m.B.selectAll - ==> Parameters:
[DEBUG] 12:19:32.446 [main] c.i.m.B.selectAll - <== Total: 3 查询到的结果
[Brand{id=1, brandName='null', companyName='null', ordered=5, description='好吃不上火', status=0}, Brand{id=2, brandName='null', co
[DEBUG] 12:19:32.446 [main] o.a.i.t.j.JdbcTransaction - Resetting autocommit to true on JDBC Connection [com.mysql.jdbc.JDBC4Connec
[DEBUG] 12:19:32.446 [main] o.a.i.t.j.JdbcTransaction - Closing JDBC Connection [com.mysql.jdbc.JDBC4Connection@61230f6a]
[DEBUG] 12:19:32.447 [main] o.a.i.d.p.PooledDataSource - Returned connection 1629687658 to pool.

```

从上面结果我们看到了问题，有些数据封装成功了，而有些数据并没有封装成功。为什么这样呢？

这个问题可以通过两种方式进行解决：

- 给字段起别名
- 使用resultMap定义字段和属性的映射关系

1.2.4 起别名解决上述问题

从上面结果可以看到 `brandName` 和 `companyName` 这两个属性的数据没有封装成功，查询 实体类 和 表中的字段 发现，在实体类中属性名是 `brandName` 和 `companyName`，而表中的字段名为 `brand_name` 和 `company_name`，如下图所示。那么我们只需要保持这两部分的名称一致这个问题就迎刃而解。

```

public class Brand {
    // id 主键
    private Integer id;
    // 品牌名称
    private String brandName;
    // 企业名称
    private String companyName;
    // 排序字段
    private Integer ordered;
}

```

栏位	索引	外键	触发器	选项	注释	SQL 预览
名						类型
id						int
brand_name						varchar
company_name						varchar
ordered						int
description						varchar
status						int

我们可以在写sql语句时给这两个字段起别名，将别名定义成和属性名一致即可。

```

1 <select id="selectAll" resultType="brand">
2     select
3         id, brand_name as brandName, company_name as companyName, ordered, description, status
4     from tb_brand;
5 </select>

```

而上面的SQL语句中的字段列表书写麻烦，如果表中还有更多的字段，同时其他的功能也需要查询这些字段时就显得我们的代码不够精炼。Mybatis提供了 `sql` 片段可以提高sql的复用性。

SQL片段：

- 将需要复用的SQL片段抽取到 `sql` 标签中

```

1 <sql id="brand_column">
2     id, brand_name as brandName, company_name as companyName, ordered, description, status
3 </sql>

```

`id`属性值是唯一标识，引用时也是通过该值进行引用。

- 在原sql语句中进行引用

使用 `include` 标签引用上述的 SQL 片段，而 `refid` 指定上述 SQL 片段的 `id` 值。

```
1 <select id="selectAll" resultType="brand">
2   select
3     <include refid="brand_column" />
4   from tb_brand;
5 </select>
```

1.2.5 使用resultMap解决上述问题

起别名 + sql片段的方式可以解决上述问题，但是它也存在问题。如果还有功能只需要查询部分字段，而不是查询所有字段，那么我们就需要再定义一个SQL片段，这就显得不是那么灵活。

那么我们也可以使用resultMap来定义字段和属性的映射关系的方式解决上述问题。

- 在映射配置文件中使用resultMap定义 字段 和 属性 的映射关系

```
1 <resultMap id="brandResultMap" type="brand">
2   <!--
3     id: 完成主键字段的映射
4     column: 表的列名
5     property: 实体类的属性名
6     result: 完成一般字段的映射
7     column: 表的列名
8     property: 实体类的属性名
9   -->
10  <result column="brand_name" property="brandName"/>
11  <result column="company_name" property="companyName"/>
12 </resultMap>
```

注意：在上面只需要定义 字段名 和 属性名 不一样的映射，而一样的则不需要专门定义出来。

- SQL语句正常编写

```
1 <select id="selectAll" resultMap="brandResultMap">
2   select *
3   from tb_brand;
4 </select>
```

1.2.6 小结

实体类属性名 和 数据库表列名 不一致，不能自动封装数据

- 起别名：**在SQL语句中，对不一样的列名起别名，别名和实体类属性名一样
 - 可以定义 片段，提升复用性
- resultMap：**定义 完成不一致的属性名和列名的映射

而我们最终选择使用 resultMap的方式。查询映射配置文件中查询所有的 statement 书写如下：

```
1 <resultMap id="brandResultMap" type="brand">
2   <!--
3     id: 完成主键字段的映射
4     column: 表的列名
5     property: 实体类的属性名
6     result: 完成一般字段的映射
7     column: 表的列名
8     property: 实体类的属性名
9   -->
10  <result column="brand_name" property="brandName"/>
11  <result column="company_name" property="companyName"/>
12 </resultMap>
13
14
15
16 <select id="selectAll" resultMap="brandResultMap">
17   select *
18   from tb_brand;
```

```
19 </select>
```

1.3 查询详情

品牌名称	企业名称	排序	当前状态	操作
三只松鼠	这里是企业名称	15	<input checked="" type="checkbox"/>	删除 编辑 查看详情
优衣库	这里是企业名称	18	<input checked="" type="checkbox"/>	删除 编辑 查看详情
小米	这里是企业名称	1	<input checked="" type="checkbox"/>	删除 编辑 查看详情

有些数据的属性比较多，在页面表格中无法全部实现，而只会显示部分，而其他属性数据的查询可以通过[查看详情](#)来进行查询，如上图所示。

查看详情功能实现步骤：

- 编写接口方法：Mapper接口

```
Brand selectById(int id);
```

- 参数：id

查看详情就是查询某一行数据，所以需要根据id进行查询。而id以后是由页面传递过来。

- 结果：Brand

根据id查询出来的数据只要一条，而将一条数据封装成一个Brand对象即可

- 编写SQL语句：SQL映射文件

```
<select id="selectById" parameterType="int" resultType="brand">
    select * from tb_brand where id = #{id};
</select>
```

- 执行方法、进行测试

1.3.1 编写接口方法

在 `BrandMapper` 接口中定义根据id查询数据的方法

```
1 /**
2  * 查看详情：根据ID查询
3  */
4 Brand selectById(int id);
```

1.3.2 编写SQL语句

在 `BrandMapper.xml` 映射配置文件中编写 `statement`，使用 `resultMap` 而不是使用 `resultType`

```
1 <select id="selectById" resultMap="brandResultMap">
2     select *
3     from tb_brand where id = #{id};
4 </select>
```

注意：上述SQL中的 `#{id}` 先这样写，一会我们再详细讲解

1.3.3 编写测试方法

在 `test/java` 下的 `com.itheima.mapper` 包下的 `MybatisTest` 类中 定义测试方法

```
1 @Test
2 public void testSelectById() throws IOException {
3     //接收参数，该id以后需要传递过来
4     int id = 1;
```

```

5
6 //1. 获取SqlSessionFactory
7 String resource = "mybatis-config.xml";
8 InputStream inputStream = Resources.getResourceAsStream(resource);
9 SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);
10
11 //2. 获取SqlSession对象
12 SqlSession sqlSession = sqlSessionFactory.openSession();
13
14 //3. 获取Mapper接口的代理对象
15 BrandMapper brandMapper = sqlSession.getMapper(BrandMapper.class);
16
17 //4. 执行方法
18 Brand brand = brandMapper.selectById(id);
19 System.out.println(brand);
20
21 //5. 释放资源
22 sqlSession.close();
23 }

```

执行测试方法结果如下：

The screenshot shows the JUnit test results for 'MyBatisTest'. The test 'testSelect' passed in 98ms. The log output shows the following sequence of events:

- Reader entry: #4
- Find JAR URL: file:/D:/workspace/mybatis-demo/target/classes/com/itheima/mapper/UserMapper.xml
- Not a JAR: file:/D:/workspace/mybatis-demo/target/classes/com/itheima/mapper/UserMapper.xml
- Reader entry: <?xml version="1.0" encoding="UTF-8" ?>
- Checking to see if class com.itheima.mapper.BrandMapper matches criteria [is assigned]
- Checking to see if class com.itheima.mapper.UserMapper matches criteria [is assigned]
- Opening JDBC Connection
- Created connection 1259014228.
- Setting autocommit to false on JDBC Connection [com.mysql.jdbc.JDBC4Connection]
- Preparing: select * from tb_brand where id = ?; **SQL语句**
- Parameters: 1(Integer) **传递的参数值**
- Total: 1
- nd{id=1, brandName='三只松鼠', companyName='三只松鼠股份有限公司', ordered=5, description='好吃不上火', status=0} **查询到的结果**
- Resetting autocommit to true on JDBC Connection [com.mysql.jdbc.JDBC4Connection]
- Closing JDBC Connection [com.mysql.jdbc.JDBC4Connection@4b0b0854]
- Returned connection 1259014228 to pool.

1.3.4 参数占位符

查询到的结果很好理解就是id为1的这行数据。而这里我们需要看控制台显示的SQL语句，能看到使用?进行占位。说明我们在映射配置文件中的写的 `#{}{}` 最终会被?进行占位。接下来我们就聊聊映射配置文件中的参数占位符。

mybatis提供了两种参数占位符：

- `#{}{}`：执行SQL时，会将 `#{}{}` 占位符替换为?，将来自动设置参数值。从上述例子可以看出使用`#{}{}` 底层使用的是 `PreparedStatement`
- `${}{}`：拼接SQL。底层使用的是 `Statement`，会存在SQL注入问题。如下图将 映射配置文件中的 `#{}{}` 替换成 `${}{}` 来看效果

```

1 <select id="selectById" resultMap="brandResultMap">
2   select *
3     from tb_brand where id = ${id};
4 </select>

```

重新运行查看结果如下：

The screenshot shows the JUnit test results for 'MyBatisTest'. The test 'testSelect' passed in 54ms. The log output shows the following sequence of events:

- Find JAR URL: file:/D:/workspace/mybatis-demo/target/classes/com/itheima/mapper/UserMapper.xml
- Not a JAR: file:/D:/workspace/mybatis-demo/target/classes/com/itheima/mapper/UserMapper.xml
- Reader entry: <?xml version="1.0" encoding="UTF-8" ?>
- Checking to see if class com.itheima.mapper.BrandMapper matches criteria [is assigned]
- Checking to see if class com.itheima.mapper.UserMapper matches criteria [is assigned]
- Opening JDBC Connection
- Created connection 1637290981.
- Setting autocommit to false on JDBC Connection [com.mysql.jdbc.JDBC4Connection]
- Preparing: select * from tb_brand where id = 1; **直接将数据拼接到SQL语句中**
- Parameters:
- Total: 1
- Brand{id=1, brandName='三只松鼠', companyName='三只松鼠股份有限公司', ordered=5, description='好吃不上火', status=0}
- Resetting autocommit to true on JDBC Connection [com.mysql.jdbc.JDBC4Connection]
- Closing JDBC Connection [com.mysql.jdbc.JDBC4Connection@619713e5]
- Returned connection 1637290981 to pool.

注意：从上面两个例子可以看出，以后开发我们使用 #{} 参数占位符。

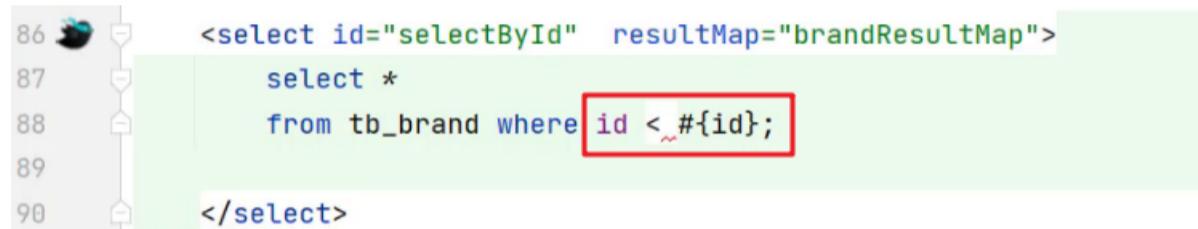
1.3.5 parameterType使用

对于有参数的mapper接口方法，我们在映射配置文件中应该配置 `parameterType` 来指定参数类型。只不过该属性都可以省略。如下图：

```
1 <select id="selectById" parameterType="int" resultMap="brandResultMap">
2   select *
3   from tb_brand where id = ${id};
4 </select>
```

1.3.6 SQL语句中特殊字段处理

以后肯定会在SQL语句中写一下特殊字符，比如某一个字段大于某个值，如下图

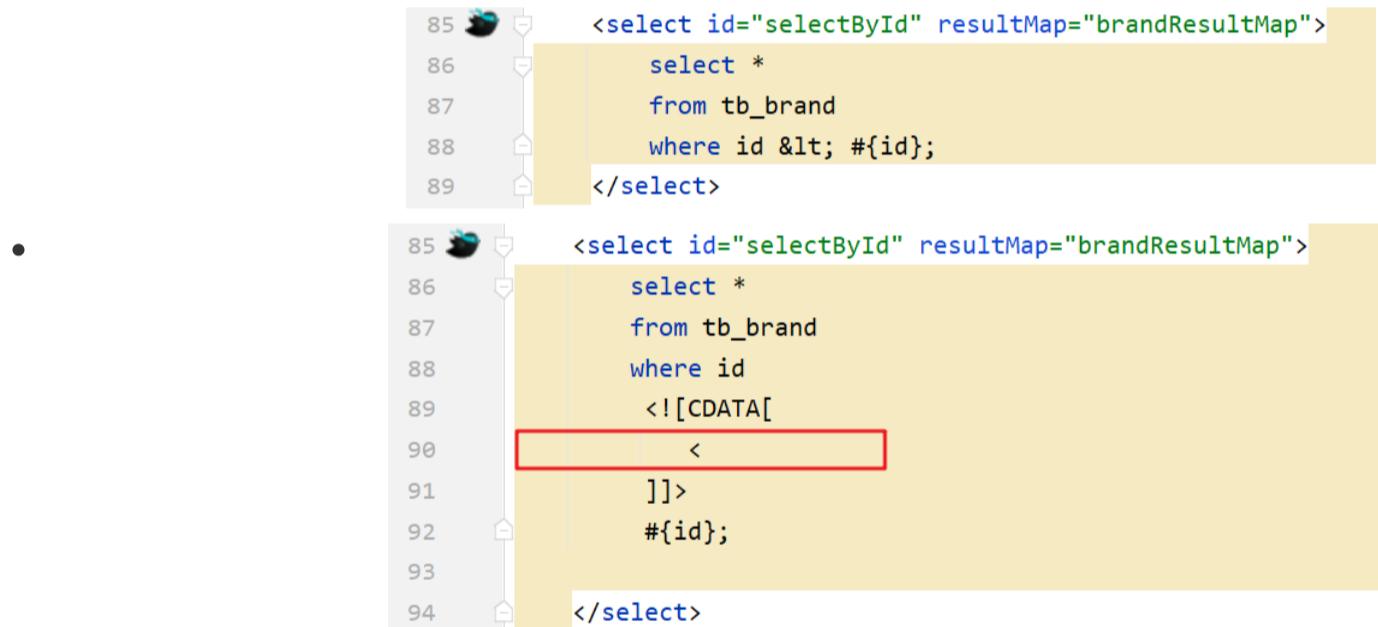


```
86 <select id="selectById" resultMap="brandResultMap">
87   select *
88   from tb_brand where id < #{id};
89 </select>
```

可以看出报错了，因为映射配置文件是xml类型的问题，而 > < 等这些字符在xml中有特殊含义，所以此时我们需要将这些符号进行转义，可以使用以下两种方式进行转义

- 转义字符

下图的 `<` 就是 `<` 的转义字符。



```
85 <select id="selectById" resultMap="brandResultMap">
86   select *
87   from tb_brand
88   where id &lt; #{id};
89 </select>
```



```
85 <select id="selectById" resultMap="brandResultMap">
86   select *
87   from tb_brand
88   where id
89     <![CDATA[
90       <
91       ]]>
92     #{id};
93 </select>
```

1.4 多条件查询



我们经常会遇到如上图所示的多条件查询，将多条件查询的结果展示在下方的数据列表中。而我们做这个功能需要分析最终的SQL语句应该是什么样，思考两个问题

- 条件表达式
- 如何连接

条件字段 `企业名称` 和 `品牌名称` 需要进行模糊查询，所以条件应该是：

简单的分析后，我们来看功能实现的步骤：

- 编写接口方法
 - 参数：所有查询条件
 - 结果：List
- 在映射配置文件中编写SQL语句
- 编写测试方法并执行

1.4.1 编写接口方法

在 `BrandMapper` 接口中定义多条件查询的方法。

而该功能有三个参数，我们就需要考虑定义接口时，参数应该如何定义。Mybatis针对多参数有多种实现

- 使用 `@Param("参数名称")` 标记每一个参数，在映射配置文件中就需要使用 `#{}{参数名称}` 进行占位

```
1 | List<Brand> selectByCondition(@Param("status") int status, @Param("companyName") String
  companyName, @Param("brandName") String brandName);
```

- 将多个参数封装成一个实体对象，将该实体对象作为接口的方法参数。该方式要求在映射配置文件的SQL中使用 `#{}{内容}` 时，里面的内容必须和实体类属性名保持一致。

```
1 | List<Brand> selectByCondition(Brand brand);
```

- 将多个参数封装到map集合中，将map集合作为接口的方法参数。该方式要求在映射配置文件的SQL中使用 `#{}{内容}` 时，里面的内容必须和map集合中键的名称一致。

```
1 | List<Brand> selectByCondition(Map map);
```

1.4.2 编写SQL语句

在 `BrandMapper.xml` 映射配置文件中编写 `statement`，使用 `resultMap` 而不是使用 `resultType`

```
1 | <select id="selectByCondition" resultMap="brandResultMap">
2 |   select *
3 |   from tb_brand
4 |   where status = #{status}
5 |   and company_name like #{companyName}
6 |   and brand_name like #{brandName}
7 | </select>
```

1.4.3 编写测试方法

在 `test/java` 下的 `com.itheima.mapper` 包下的 `MybatisTest` 类中 定义测试方法

```
1 | @Test
2 | public void testSelectByCondition() throws IOException {
3 |   //接收参数
4 |   int status = 1;
5 |   String companyName = "华为";
6 |   String brandName = "华为";
7 |
8 |   // 处理参数
9 |   companyName = "%" + companyName + "%";
10 |  brandName = "%" + brandName + "%";
11 |
12 |  //1. 获取sqlSessionFactory
```

```

13     String resource = "mybatis-config.xml";
14     InputStream inputStream = Resources.getResourceAsStream(resource);
15     SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);
16     //2. 获取SqlSession对象
17     SqlSession sqlSession = sqlSessionFactory.openSession();
18     //3. 获取Mapper接口的代理对象
19     BrandMapper brandMapper = sqlSession.getMapper(BrandMapper.class);
20
21     //4. 执行方法
22     //方式一：接口方法参数使用 @Param 方式调用的方法
23     //List<Brand> brands = brandMapper.selectByCondition(status, companyName, brandName);
24     //方式二：接口方法参数是 实体类对象 方式调用的方法
25     //封装对象
26     /* Brand brand = new Brand();
27         brand.setStatus(status);
28         brand.setCompanyName(companyName);
29         brand.setBrandName(brandName); */
30
31     //List<Brand> brands = brandMapper.selectByCondition(brand);
32
33     //方式三：接口方法参数是 map集合对象 方式调用的方法
34     Map map = new HashMap();
35     map.put("status", status);
36     map.put("companyName", companyName);
37     map.put("brandName", brandName);
38     List<Brand> brands = brandMapper.selectByCondition(map);
39     System.out.println(brands);
40
41     //5. 释放资源
42     sqlSession.close();
43 }

```

1.4.4 动态SQL

上述功能实现存在很大的问题。用户在输入条件时，肯定不会所有的条件都填写，这个时候我们的SQL语句就不能那样写的
例如用户只输入 当前状态 时， SQL语句就是

```
1 | select * from tb_brand where status = #{status}
```

而用户如果只输入企业名称时， SQL语句就是

```
1 | select * from tb_brand where company_name like #{companName}
```

而用户如果输入了 `当前状态` 和 `企业名称` 时， SQL语句又不一样

```
1 | select * from tb_brand where status = #{status} and company_name like #{companName}
```

针对上述的需要， Mybatis对动态SQL有很强大的支撑：

- if
- choose (when, otherwise)
- trim (where, set)
- foreach

我们先学习 if 标签和 where 标签：

- if 标签：条件判断
 - test 属性：逻辑表达式

```

1 | <select id="selectByCondition" resultMap="brandResultMap">
2 |   select *
3 |   from tb_brand
4 |   where
5 |     <if test="status != null">
```

```

6         and status = #{status}
7     </if>
8     <if test="companyName != null and companyName != ''">
9         and company_name like #{companyName}
10    </if>
11    <if test="brandName != null and brandName != ''">
12        and brand_name like #{brandName}
13    </if>
14 </select>

```

如上的这种SQL语句就会根据传递的参数值进行动态的拼接。如果此时status和companyName有值那么就会值拼接这两个条件。

执行结果如下：

The screenshot shows the MyBatis Test results window. It displays the following log output:

```

Tests passed: 1 of 1 test - 1s 243 ms
MyBatisTest 1s 243 ms
  ✓ testSele 1s 243 ms
    .j.JdbcTransaction - Opening JDBC Connection
    .p.PooledDataSource - Created connection 1780034814.
    .j.JdbcTransaction - Setting autocommit to false on JDBC Connection [com.mysql.jdbc.JDBC4Connection@6a192cfe]
    .selectByCondition - ==> Preparing: select * from tb_brand where status = ? and company_name like ?
    .selectByCondition - ==> Parameters: 1(Integer), %华为%(String)
    .selectByCondition - <== Total: 1
    nyName='华为技术有限公司', ordered=100, description='华为致力于把数字世界带入每个人、每个家庭、每个组织，构建万物互联的智能世界', status=1]
    .j.JdbcTransaction - Resetting autocommit to true on JDBC Connection [com.mysql.jdbc.JDBC4Connection@6a192cfe]
    .j.JdbcTransaction - Closing JDBC Connection [com.mysql.jdbc.JDBC4Connection@6a192cfe]

```

The line ".selectByCondition - ==> Preparing: select * from tb_brand where status = ? and company_name like ?" is highlighted with a red box.

但是它也存在问题，如果此时给的参数值是

```

1 Map map = new HashMap();
2 // map.put("status", status);
3 map.put("companyName", companyName);
4 map.put("brandName", brandName);

```

拼接的SQL语句就变成了

```
1 select * from tb_brand where and company_name like ? and brand_name like ?
```

而上面的语句中 where 关键后直接跟 and 关键字，这就是一条错误的SQL语句。这个就可以使用 where 标签解决

- where 标签

- 作用：

- 替换where关键字
- 会动态的去掉第一个条件前的 and
- 如果所有的参数没有值则不加where关键字

```

1 <select id="selectByCondition" resultMap="brandResultMap">
2     select *
3     from tb_brand
4     <where>
5         <if test="status != null">
6             and status = #{status}
7         </if>
8         <if test="companyName != null and companyName != ''">
9             and company_name like #{companyName}
10        </if>
11        <if test="brandName != null and brandName != ''">
12            and brand_name like #{brandName}
13        </if>
14     </where>
15 </select>

```

注意：需要给每个条件前都加上 and 关键字。

1.5 单个条件（动态SQL）

如上图所示，在查询时只能选择 品牌名称、当前状态、企业名称 这三个条件中的一个，但是用户到底选择哪一个，我们并不能确定。这种就属于单个条件的动态SQL语句。

这种需求需要使用到 `choose (when, otherwise)` 标签 实现，而 `choose` 标签类似于Java 中的switch语句。

通过一个案例来使用这些标签

1.5.1 编写接口方法

在 `BrandMapper` 接口中定义单条件查询的方法。

```

1 /**
2  * 单条件动态查询
3  * @param brand
4  * @return
5 */
6 List<Brand> selectByConditionSingle(Brand brand);

```

1.5.2 编写SQL语句

在 `BrandMapper.xml` 映射配置文件中编写 `statement`，使用 `resultMap` 而不是使用 `resultType`

```

1 <select id="selectByConditionSingle" resultMap="brandResultMap">
2     select *
3     from tb_brand
4     <where>
5         <choose><!--相当于switch-->
6             <when test="status != null"><!--相当于case-->
7                 status = #{status}
8             </when>
9             <when test="companyName != null and companyName != '' "><!--相当于case-->
10                companyName like #{companyName}
11            </when>
12            <when test="brandName != null and brandName != '' "><!--相当于case-->
13                brand_name like #{brandName}
14            </when>
15        </choose>
16    </where>
17 </select>

```

1.5.3 编写测试方法

在 `test/java` 下的 `com.itheima.mapper` 包下的 `MybatisTest`类中 定义测试方法

```

1 @Test
2 public void testSelectByConditionSingle() throws IOException {
3     //接收参数
4     int status = 1;
5     String companyName = "华为";
6     String brandName = "华为";
7
8     // 处理参数
9     companyName = "%" + companyName + "%";
10    brandName = "%" + brandName + "%";

```

```

11
12 //封装对象
13 Brand brand = new Brand();
14 //brand.setStatus(status);
15 brand.setCompanyName(companyName);
16 //brand.setBrandName(brandName);
17
18 //1. 获取SqlSessionFactory
19 String resource = "mybatis-config.xml";
20 InputStream inputStream = Resources.getResourceAsStream(resource);
21 SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);
22 //2. 获取SqlSession对象
23 SqlSession sqlSession = sqlSessionFactory.openSession();
24 //3. 获取Mapper接口的代理对象
25 BrandMapper brandMapper = sqlSession.getMapper(BrandMapper.class);
26 //4. 执行方法
27 List<Brand> brands = brandMapper.selectByConditionSingle(brand);
28 System.out.println(brands);
29
30 //5. 释放资源
31 sqlSession.close();
32 }

```

执行测试方法结果如下：

```

    Tests passed: 1 of 1 test - 1 s 254 ms
    MyBatisTest: 1 s 254 ms
      ✓ testSelect 1 s 254 ms
        [DEBUG] 21:14:54.908 [main] o.a.i.i.DefaultVFS - Reader entry: 4444
        [DEBUG] 21:14:54.908 [main] o.a.i.i.DefaultVFS - Find JAR URL: file:/D:/workspace/mybatis-demo/target/classes/com/itheima/mapper/UserMapper.xml
        [DEBUG] 21:14:54.909 [main] o.a.i.i.DefaultVFS - Not a JAR: file:/D:/workspace/mybatis-demo/target/classes/com/itheima/mapper/UserMapper.xml
        [DEBUG] 21:14:54.909 [main] o.a.i.i.DefaultVFS - Reader entry: <?xml version="1.0" encoding="UTF-8" ?>
        [DEBUG] 21:14:54.910 [main] o.a.i.i.ResolverUtil - Checking to see if class com.itheima.mapper.BrandMapper matches criteria [is abstract=false, is interface=true]
        [DEBUG] 21:14:54.910 [main] o.a.i.i.ResolverUtil - Checking to see if class com.itheima.mapper.UserMapper matches criteria [is abstract=false, is interface=true]
        [DEBUG] 21:14:55.096 [main] o.a.i.t.j.JdbcTransaction - Opening JDBC Connection
        [DEBUG] 21:14:55.477 [main] o.a.i.d.p.PooledDataSource - Created connection 959869407.
        [DEBUG] 21:14:55.478 [main] o.a.i.t.j.JdbcTransaction - Setting autocommit to false on JDBC Connection [com.mysql.jdbc.JDBC4Connection@393671df]
        [DEBUG] 21:14:55.483 [main] c.i.m.B.selectByConditionSingle - ==> Preparing: select * from tb_brand WHERE company_name like ?
        [DEBUG] 21:14:55.540 [main] c.i.m.B.selectByConditionSingle - ==> Parameters: %华为%(String)
        [DEBUG] 21:14:55.579 [main] c.i.m.B.selectByConditionSingle - <== Total: 1
        [Brand{id=2, brandName='华为', companyName='华为技术有限公司', ordered=100, description='华为致力于把数字世界带入每个人、每个家庭、每个组织, 构建万物互联的智能世界。'}]
        [DEBUG] 21:14:55.580 [main] o.a.i.t.j.JdbcTransaction - Resetting autocommit to true on JDBC Connection [com.mysql.jdbc.JDBC4Connection@393671df]
        [DEBUG] 21:14:55.581 [main] o.a.i.t.j.JdbcTransaction - Closing JDBC Connection [com.mysql.jdbc.JDBC4Connection@393671df]
        [DEBUG] 21:14:55.581 [main] o.a.i.d.p.PooledDataSource - Returned connection 959869407 to pool.

```

1.6 添加数据

新增商品品牌

品牌LOGO	<input type="button" value="+"/> 上传logo 支持JPG、PNG、GIF格式的图片，大小请小于200K，尺寸200*120px
* 品牌名称	<input type="text" value="请输入品牌名称"/>
* 企业名称	<input type="text" value="请输入企业名称"/>
排序	<input type="text" value="请输入大于0的正整数"/> 排序为空时，默认按新增时间倒序排在最前面
备注信息	<input type="text" value="0/200"/>
当前状态	<input checked="" type="checkbox"/>
<input type="button" value="提交"/> <input type="button" value="取消"/>	

如上图是我们平时在添加数据时展示的页面，而我们在该页面输入想要的数据后添加 提交 按钮，就会将这些数据添加到数据库中。接下来我们就来实现添加数据的操作。

- 编写接口方法

```
void add(Brand brand);
```

参数：除了id之外的所有的数据。id对应的是表中主键值，而主键我们是**自动增长**生成的。

- 编写SQL语句

```
<insert id="add">
    insert into tb_brand (brand_name, company_name, ordered, description, status)
    values (#{brandName}, #{companyName}, #{ordered}, #{description}, #{status});
</insert>
```

- 编写测试方法并执行

明确了该功能实现的步骤后，接下来我们进行具体的操作。

1.6.1 编写接口方法

在 `BrandMapper` 接口中定义添加方法。

```
1 /**
2  * 添加
3  */
4 void add(Brand brand);
```

1.6.2 编写SQL语句

在 `BrandMapper.xml` 映射配置文件中编写添加数据的 `statement`

```
1 <insert id="add">
2     insert into tb_brand (brand_name, company_name, ordered, description, status)
3     values (#{brandName}, #{companyName}, #{ordered}, #{description}, #{status});
4 </insert>
```

1.6.3 编写测试方法

在 `test/java` 下的 `com.itheima.mapper` 包下的 `MybatisTest`类中 定义测试方法

```
1 @Test
2 public void testAdd() throws IOException {
3     //接收参数
4     int status = 1;
5     String companyName = "波导手机";
6     String brandName = "波导";
7     String description = "手机中的战斗机";
8     int ordered = 100;
9
10    //封装对象
11    Brand brand = new Brand();
12    brand.setStatus(status);
13    brand.setCompanyName(companyName);
14    brand.setBrandName(brandName);
15    brand.setDescription(description);
16    brand.setOrdered(ordered);
17
18    //1. 获取sqlSessionFactory
19    String resource = "mybatis-config.xml";
20    InputStream inputStream = Resources.getResourceAsStream(resource);
21    SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);
22    //2. 获取sqlSession对象
23    SqlSession sqlSession = sqlSessionFactory.openSession();
24    //SqlSession sqlSession = sqlSessionFactory.openSession(true); //设置自动提交事务，这种情况不需要手动提交事务了
25    //3. 获取Mapper接口的代理对象
26    BrandMapper brandMapper = sqlSession.getMapper(BrandMapper.class);
27    //4. 执行方法
28    brandMapper.add(brand);
29    //提交事务
30    sqlSession.commit();
```

```

31     //5. 释放资源
32     sqlSession.close();
33 }

```

执行结果如下：

```

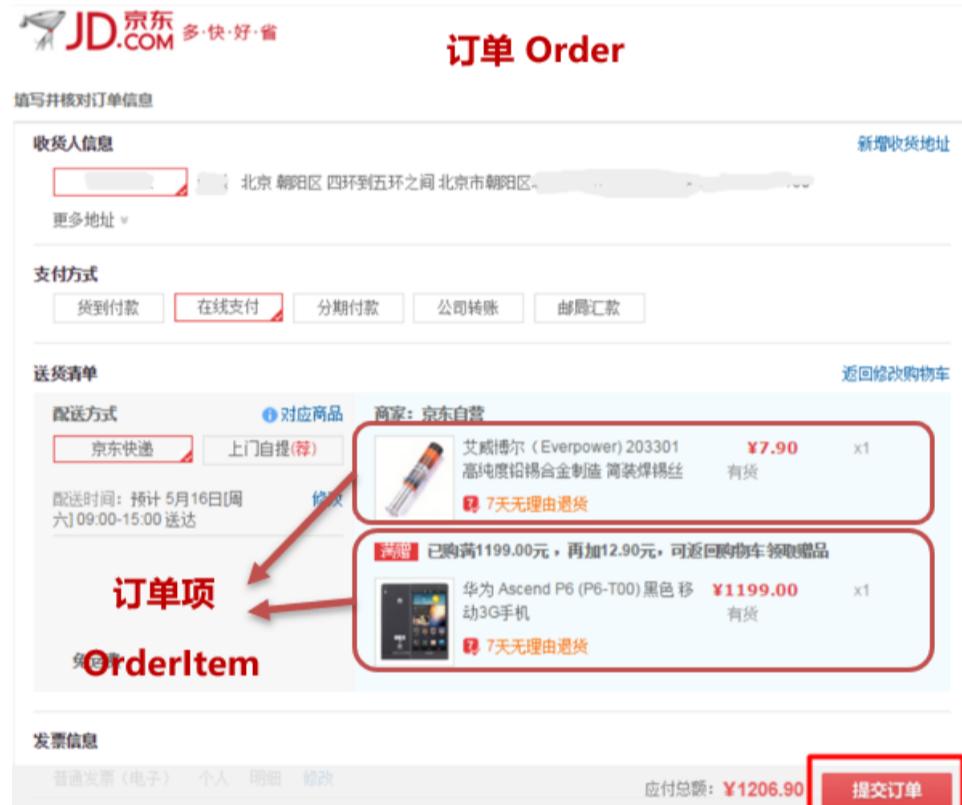
Tests passed: 1 of 1 test - 1s 128ms
[DEBUG] 21:55:58.154 [main] o.a.i.i.DefaultVFS - Find JAR URL: file:/D:/workspace/mybatis-demo/target/classes/com/itheima/mapper/UserMapper.xml
[DEBUG] 21:55:58.154 [main] o.a.i.i.DefaultVFS - Not a JAR: file:/D:/workspace/mybatis-demo/target/classes/com/itheima/mapper/UserMapper.xml
[DEBUG] 21:55:58.155 [main] o.a.i.i.DefaultVFS - Reader entry: <!-->
[DEBUG] 21:55:58.155 [main] o.a.i.i.DefaultVFS - Find JAR URL: file:/D:/workspace/mybatis-demo/target/classes/com/itheima/mapper/UserMapper.xml
[DEBUG] 21:55:58.155 [main] o.a.i.i.DefaultVFS - Not a JAR: file:/D:/workspace/mybatis-demo/target/classes/com/itheima/mapper/UserMapper.xml
[DEBUG] 21:55:58.156 [main] o.a.i.i.DefaultVFS - Reader entry: <?xml version="1.0" encoding="UTF-8" ?>
[DEBUG] 21:55:58.157 [main] o.a.i.i.ResolverUtil - Checking to see if class com.itheima.mapper.BrandMapper matches criteria [is abstract=false, is interface=true]
[DEBUG] 21:55:58.157 [main] o.a.i.i.ResolverUtil - Checking to see if class com.itheima.mapper.UserMapper matches criteria [is abstract=false, is interface=true]
[DEBUG] 21:55:58.318 [main] o.a.i.t.j.JdbcTransaction - Opening JDBC Connection
[DEBUG] 21:55:58.641 [main] o.a.i.d.p.PooledDataSource - Created connection 178049969.
[DEBUG] 21:55:58.641 [main] o.a.i.t.j.JdbcTransaction - Setting autocommit to false on JDBC Connection [com.mysql.jdbc.JDBC4Connection@a9cd3b1]
[DEBUG] 21:55:58.645 [main] c.i.m.B.add - ==> Preparing: insert into tb_brand (brand_name, company_name, ordered, description, stock)
[DEBUG] 21:55:58.687 [main] c.i.m.B.add - ==> Parameters: 波导(String), 波导手机(String), 100(Integer), 手机中的战斗机(String), 1(Integer)
[DEBUG] 21:55:58.689 [main] c.i.m.B.add - <-- Updates: 1
[DEBUG] 21:55:58.689 [main] o.a.i.t.j.JdbcTransaction - Committing JDBC Connection [com.mysql.jdbc.JDBC4Connection@a9cd3b1]
[DEBUG] 21:55:58.700 [main] o.a.i.t.j.JdbcTransaction - Resetting autocommit to true on JDBC Connection [com.mysql.jdbc.JDBC4Connection@a9cd3b1]
[DEBUG] 21:55:58.701 [main] o.a.i.t.j.JdbcTransaction - Closing JDBC Connection [com.mysql.jdbc.JDBC4Connection@a9cd3b1]
[DEBUG] 21:55:58.701 [main] o.a.i.d.p.PooledDataSource - Returned connection 178049969 to pool.

```

1.6.4 添加-主键返回

在数据添加成功后，有时候需要获取插入数据库数据的主键（主键是自增长）。

比如：添加订单和订单项，如下图就是京东上的订单



订单数据存储在订单表中，订单项存储在订单项表中。

- 添加订单数据

```

<insert id="addOrder" useGeneratedKeys="true" keyProperty="id">
    insert into tb_order (payment, payment_type, status)
    values (#{}{payment},#{paymentType},#{status});
</insert>

```

- 添加订单项数据，订单项中需要设置所属订单的id

```

<insert id="addOrder" useGeneratedKeys="true" keyProperty="id">
    insert into tb_order (payment, payment_type, status)
    values (#{}{payment},#{paymentType},#{status});
</insert>

```

明白了什么时候 `主键返回`。接下来我们简单模拟一下，在添加完数据后打印id属性值，能打印出来说明已经获取到了。

我们将上面添加品牌数据的案例中映射配置文件里 `statement` 进行修改，如下

```
1 <insert id="add" useGeneratedKeys="true" keyProperty="id">
2     insert into tb_brand (brand_name, company_name, ordered, description, status)
3         values (#{brandName}, #{companyName}, #{ordered}, #{description}, #{status});
4 </insert>
```

在 insert 标签上添加如下属性：

- useGeneratedKeys：足够获取自动增长的主键值。true 表示获取
- keyProperty：指定将获取到的主键值封装到哪个属性里

1.7 修改

如图所示是修改页面，用户在该页面书写需要修改的数据，点击 提交 按钮，就会将数据库中对应的数据进行修改。注意一点，如果哪几个输入框没有输入内容，我们是将表中数据对应字段值替换为空白还是保留字段之前的值？答案肯定是保留之前的数据。

接下来我们就具体来实现

1.7.1 编写接口方法

在 `BrandMapper` 接口中定义修改方法。

```
1 /**
2  * 修改
3  */
4 void update(Brand brand);
```

上述方法参数 `Brand` 就是封装了需要修改的数据，而 `id` 肯定是有数据的，这也是和添加方法的区别。

1.7.2 编写SQL语句

在 `BrandMapper.xml` 映射配置文件中编写修改数据的 `statement`。

```
1 <update id="update">
2     update tb_brand
3     <set>
4         <if test="brandName != null and brandName != ''">
5             brand_name = #{brandName},
6         </if>
7         <if test="companyName != null and companyName != ''">
8             company_name = #{companyName},
9         </if>
```

```

10      <if test="ordered != null">
11          ordered = #{ordered},
12      </if>
13      <if test="description != null and description != ''">
14          description = #{description},
15      </if>
16      <if test="status != null">
17          status = #{status}
18      </if>
19  </set>
20  where id = #{id};
21 </update>

```

`set` 标签可以用于动态包含需要更新的列，忽略其它不更新的列。

1.7.3 编写测试方法

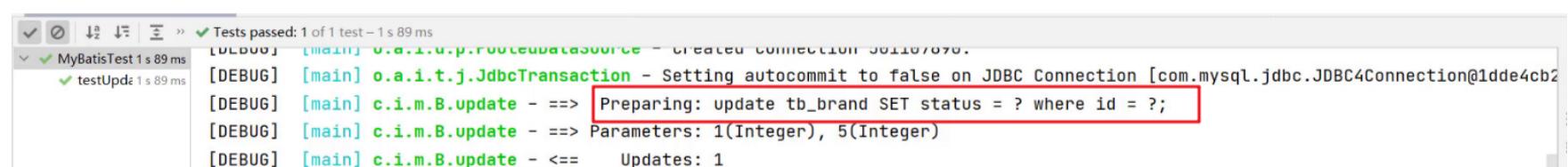
在 `test/java` 下的 `com.itheima.mapper` 包下的 `MybatisTest` 类中 定义测试方法

```

1 @Test
2 public void testUpdate() throws IOException {
3     //接收参数
4     int status = 0;
5     String companyName = "波导手机";
6     String brandName = "波导";
7     String description = "波导手机,手机中的战斗机";
8     int ordered = 200;
9     int id = 6;
10
11    //封装对象
12    Brand brand = new Brand();
13    brand.setStatus(status);
14    //        brand.setCompanyName(companyName);
15    //        brand.setBrandName(brandName);
16    //        brand.setDescription(description);
17    //        brand.setOrdered(ordered);
18    brand.setId(id);
19
20    //1. 获取sqlSessionFactory
21    String resource = "mybatis-config.xml";
22    InputStream inputStream = Resources.getResourceAsStream(resource);
23    SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);
24    //2. 获取SqlSession对象
25    SqlSession sqlSession = sqlSessionFactory.openSession();
26    //SqlSession sqlSession = sqlSessionFactory.openSession(true);
27    //3. 获取Mapper接口的代理对象
28    BrandMapper brandMapper = sqlSession.getMapper(BrandMapper.class);
29    //4. 执行方法
30    int count = brandMapper.update(brand);
31    System.out.println(count);
32    //提交事务
33    sqlSession.commit();
34    //5. 释放资源
35    sqlSession.close();
36 }

```

执行测试方法结果如下：



从结果中SQL语句可以看出，只修改了 `status` 字段值，因为我们给的数据中只给 `Brand` 实体对象的 `status` 属性设置值了。这就是 `set` 标签的作用。

1.8 删除一行数据

<input type="checkbox"/>	序号	品牌LOGO	品牌名称	企业名称	排序	当前状态	操作
<input type="checkbox"/>	010		三只松鼠	这里是企业名称	15	<input checked="" type="checkbox"/>	删除 编辑 查看详情
<input type="checkbox"/>	009		优衣库	这里是企业名称	18	<input checked="" type="checkbox"/>	删除 编辑 查看详情
<input type="checkbox"/>	008		小米	这里是企业名称	5	<input checked="" type="checkbox"/>	删除 编辑 查看详情

如上图所示，每行数据后面都有一个 `删除` 按钮，当用户点击了该按钮，就会将改行数据删除掉。那我们就需要思考，这种删除是根据什么进行删除呢？是通过主键id删除，因为id是表中数据的唯一标识。

接下来就来实现该功能。

1.8.1 编写接口方法

在 `BrandMapper` 接口中定义根据id删除方法。

```
1 /**
2  * 根据id删除
3  */
4 void deleteById(int id);
```

1.8.2 编写SQL语句

在 `BrandMapper.xml` 映射配置文件中编写删除一行数据的 `statement`

```
1 <delete id="deleteById">
2     delete from tb_brand where id = #{id};
3 </delete>
```

1.8.3 编写测试方法

在 `test/java` 下的 `com.itheima.mapper` 包下的 `MybatisTest`类中 定义测试方法

```
1 @Test
2 public void testDeleteById() throws IOException {
3     //接收参数
4     int id = 6;
5
6     //1. 获取sqlSessionFactory
7     String resource = "mybatis-config.xml";
8     InputStream inputStream = Resources.getResourceAsStream(resource);
9     SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);
10    //2. 获取sqlSession对象
11    SqlSession sqlSession = sqlSessionFactory.openSession();
12    //SqlSession sqlSession = sqlSessionFactory.openSession(true);
13    //3. 获取Mapper接口的代理对象
14    BrandMapper brandMapper = sqlSession.getMapper(BrandMapper.class);
15    //4. 执行方法
16    brandMapper.deleteById(id);
17    //提交事务
18    sqlSession.commit();
19    //5. 释放资源
20    sqlSession.close();
21 }
```

运行过程只要没报错，直接到数据库查询数据是否还存在。

1.9 批量删除

<input type="checkbox"/>	序号	品牌LOGO	品牌名称	企业名称
<input checked="" type="checkbox"/>	010		三只松鼠	这里是企业名称
<input checked="" type="checkbox"/>	009		优衣库	这里是企业名称
<input checked="" type="checkbox"/>	008		小米	这里是企业名称
<input type="checkbox"/>	007		阿迪达斯	这里是企业名称

如上图所示，用户可以选择多条数据，然后点击上面的 `删除` 按钮，就会删除数据库中对应的多行数据。

1.9.1 编写接口方法

在 `BrandMapper` 接口中定义删除多行数据的方法。

```

1 /**
2  * 批量删除
3  */
4 void deleteByIds(int[] ids);

```

参数是一个数组，数组中存储的是多条数据的id

1.9.2 编写SQL语句

在 `BrandMapper.xml` 映射配置文件中编写删除多条数据的 `statement`。

编写SQL时需要遍历数组来拼接SQL语句。Mybatis 提供了 `foreach` 标签供我们使用

foreach 标签

用来迭代任何可迭代的对象（如数组，集合）。

- collection 属性：
 - mybatis会将数组参数，封装为一个Map集合。
 - 默认：array = 数组
 - 使用@Param注解改变map集合的默认key的名称
- item 属性：本次迭代获取到的元素。
- separator 属性：集合项迭代之间的分隔符。`foreach` 标签不会错误地添加多余的分隔符。也就是最后一次迭代不会加分隔符。
- open 属性：该属性值是在拼接SQL语句之前拼接的语句，只会拼接一次
- close 属性：该属性值是在拼接SQL语句拼接后拼接的语句，只会拼接一次

```

1 <delete id="deleteByIds">
2   delete from tb_brand where id
3   in
4   <foreach collection="array" item="id" separator="," open="(" close=")">
5     #{id}
6   </foreach>
7   ;
8 </delete>

```

假如数组中的id数据是{1,2,3}，那么拼接后的sql语句就是：

```
1 delete from tb_brand where id in (1,2,3);
```

1.9.3 编写测试方法

在 `test/java` 下的 `com.itheima.mapper` 包下的 `MybatisTest` 类中 定义测试方法

```
1 @Test
2 public void testDeleteByIds() throws IOException {
3     //接收参数
4     int[] ids = {5,7,8};
5
6     //1. 获取SqlSessionFactory
7     String resource = "mybatis-config.xml";
8     InputStream inputStream = Resources.getResourceAsStream(resource);
9     SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);
10    //2. 获取SqlSession对象
11    SqlSession sqlSession = sqlSessionFactory.openSession();
12    //SqlSession sqlSession = sqlSessionFactory.openSession(true);
13    //3. 获取Mapper接口的代理对象
14    BrandMapper brandMapper = sqlSession.getMapper(BrandMapper.class);
15    //4. 执行方法
16    brandMapper.deleteByIds(ids);
17    //提交事务
18    sqlSession.commit();
19    //5. 释放资源
20    sqlSession.close();
21 }
```

1.10 Mybatis参数传递

Mybatis 接口方法中可以接收各种各样的参数，如下：

- 多个参数
- 单个参数：单个参数又可以是如下类型
 - POJO 类型
 - Map 集合类型
 - Collection 集合类型
 - List 集合类型
 - Array 类型
 - 其他类型

1.10.1 多个参数

如下面的代码，就是接收两个参数，而接收多个参数需要使用 `@Param` 注解，那么为什么要加该注解呢？这个问题要弄明白就必须来研究Mybatis 底层对于这些参数是如何处理的。

```
1 User select(@Param("username") String username,@Param("password") String password);

1 <select id="select" resultType="user">
2     select *
3     from tb_user
4     where
5         username=#{username}
6         and password=#{password}
7 </select>
```

我们在接口方法中定义多个参数，Mybatis 会将这些参数封装成 Map 集合对象，值就是参数值，而键在没有使用 `@Param` 注解时有以下命名规则：

- 以 `arg` 开头：第一个参数就叫 `arg0`，第二个参数就叫 `arg1`，以此类推。如：

```
map.put("arg0", 参数值1);
map.put("arg1", 参数值2);
```

- 以 `param` 开头：第一个参数就叫 `param1`，第二个参数就叫 `param2`，依次类推。如：

```
map.put("param1", 参数值1);
map.put("param2", 参数值2);
```

代码验证：

- 在 `UserMapper` 接口中定义如下方法

```
1 User select(String username, String password);
```

- 在 `UserMapper.xml` 映射配置文件中定义SQL

```
1 <select id="select" resultType="user">
2     select *
3     from tb_user
4     where
5         username=#{arg0}
6         and password=#{arg1}
7 </select>
```

或者

```
1 <select id="select" resultType="user">
2     select *
3     from tb_user
4     where
5         username=#{param1}
6         and password=#{param2}
7 </select>
```

- 运行代码结果如下

```
✓ Tests passed: 1 of 1 test - 860 ms
[DEBUG] [main] o.a.i.i.ResolverUtil - Checking to see if class com.itheima.mapper.UserMapper matches criteria [is assignable to Object]
[DEBUG] [main] o.a.i.t.j.JdbcTransaction - Opening JDBC Connection
[DEBUG] [main] o.a.i.d.p.PooledDataSource - Created connection 1546693040.
[DEBUG] [main] o.a.i.t.j.JdbcTransaction - Setting autocommit to false on JDBC Connection [com.mysql.jdbc.JDBC4Connection@5c30a9b0]
[DEBUG] [main] c.i.m.U.select - ==> Preparing: select * from tb_user where username = ? and password = ?
[DEBUG] [main] c.i.m.U.select - ==> Parameters: zhangsan(String), 123(String)
[DEBUG] [main] c.i.m.U.select - <== Total: 1
User{id=1, username='zhangsan', password='123', gender='男', addr='北京'}
[DEBUG] [main] o.a.i.t.j.JdbcTransaction - Resetting autocommit to true on JDBC Connection [com.mysql.jdbc.JDBC4Connection@5c30a9b0]
[DEBUG] [main] o.a.i.t.j.JdbcTransaction - Closing JDBC Connection [com.mysql.jdbc.JDBC4Connection@5c30a9b0]
[DEBUG] [main] o.a.i.d.p.PooledDataSource - Returned connection 1546693040 to pool.
```

在映射配合文件的SQL语句中使用用 `arg` 开头的和 `param` 书写，代码的可读性会变的特别差，此时可以使用 `@Param` 注解。

在接口方法参数上使用 `@Param` 注解，Mybatis 会将 `arg` 开头的键名替换为对应注解的属性值。

代码验证：

- 在 `UserMapper` 接口中定义如下方法，在 `username` 参数前加上 `@Param` 注解

```
1 User select(@Param("username") String username, String password);
```

Mybatis 在封装 Map 集合时，键名就会变成如下：

```
map.put("username", 参数值1);
map.put("arg1", 参数值2);
map.put("param1", 参数值1);
map.put("param2", 参数值2);
```

- 在 `UserMapper.xml` 映射配置文件中定义SQL

```

1 <select id="select" resultType="user">
2   select *
3   from tb_user
4   where
5     username=#{username}
6     and password=#{param2}
7 </select>

```

- 运行程序结果没有报错。而如果将 `#{} 中的 username` 还是写成 `arg0`

```

1 <select id="select" resultType="user">
2   select *
3   from tb_user
4   where
5     username=#{arg0}
6     and password=#{param2}
7 </select>

```

- 运行程序则可以看到错误



结论：以后接口参数是多个时，在每个参数上都使用 `@Param` 注解。这样代码的可读性更高。

1.10.2 单个参数

- POJO 类型

直接使用。要求 `属性名` 和 `参数占位符名称` 一致

- Map 集合类型

直接使用。要求 `map集合的键名` 和 `参数占位符名称` 一致

- Collection 集合类型

Mybatis 会将集合封装到 map 集合中，如下：

```

map.put("arg0", collection集合);
map.put("collection", collection集合);

```

可以使用 `@Param` 注解替换map集合中默认的 arg 键名。

- List 集合类型

Mybatis 会将集合封装到 map 集合中，如下：

```

map.put("arg0", list集合);
map.put("collection", list集合);
map.put("list", list集合);

```

可以使用 `@Param` 注解替换map集合中默认的 arg 键名。

- Array 类型

Mybatis 会将集合封装到 map 集合中，如下：

```

map.put("arg0", 数组);
map.put("array", 数组);

```

可以使用 `@Param` 注解替换map集合中默认的 arg 键名。

- 其他类型

比如int类型，参数占位符名称叫什么都可以。尽量做到见名知意

2. 注解实现CRUD

使用注解开发会比配置文件开发更加方便。如下就是使用注解进行开发

```
1 @Select(value = "select * from tb_user where id = #{id}")
2 public User select(int id);
```

注意：

- 注解是用来替换映射配置文件方式配置的，所以使用了注解，就不需要再映射配置文件中书写对应的 statement

Mybatis 针对 CURD 操作都提供了对应的注解，已经做到见名知意。如下：

- 查询：@Select
- 添加：@Insert
- 修改：@Update
- 删除：@Delete

接下来我们做一个案例来使用 Mybatis 的注解开发

代码实现：

- 将之前案例中 `UserMapper.xml` 中的根据id查询数据的 `statement` 注释掉

```
<mapper namespace="com.itheima.mapper.UserMapper">

    <!--statement-->
    <select id="selectAll" resultType="User">
        select *
        from tb_user;
    </select>
    <!-- select id="selectById" resultType="User">
        select *
        from tb_user where id = #{id};
    </select>-->

    <select id="select" resultType="User">
        select *
        from tb_user
        where
            username = #{arg0}
            and password = #{param2}
    </select>
```

- 在 `UserMapper` 接口的 `selectById` 方法上添加注解

```
public interface UserMapper {
    List<User> selectAll();
    @Select("select * from tb_user where id = #{id}")
    User selectById(int id);
```

- 运行测试程序也能正常查询到数据

我们课程上只演示这一个查询的注解开发，其他的同学们下来可以自己实现，都是比较简单。

注意：在官方文档中 `入门` 中有这样的一段话：

使用注解来映射简单语句会使代码显得更加简洁，但对于稍微复杂一点的语句，Java 注解不仅力不从心，还会让你本就复杂的 SQL 语句更加混乱不堪。因此，如果你需要做一些很复杂的操作，最好用 XML 来映射语句。

选择何种方式来配置映射，以及认为是否应该要统一映射语句定义的形式，完全取决于你和你的团队。换句话说，永远不要拘泥于一种方式，你可以很轻松的在基于注解和 XML 的语句映射方式间自由移植和切换。

所以，**注解完成简单功能，配置文件完成复杂功能。**

而我们之前写的动态 SQL 就是复杂的功能，如果用注解使用的话，就需要使用到 Mybatis 提供的SQL构建器来完成，而对应的代码如下：

```
String sql = new SQL() {{
    SELECT("P.ID, P.USERNAME, P.PASSWORD, P.FIRST_NAME, P.LAST_NAME");
    FROM("PERSON P");
    if (id != null) {
        WHERE("P.ID like #{id}");
    }
    if (firstName != null) {
        WHERE("P.FIRST_NAME like #{firstName}");
    }
    if (lastName != null) {
        WHERE("P.LAST_NAME like #{lastName}");
    }
    ORDER_BY("P.LAST_NAME");
}}.toString();
```

上述代码将java代码和SQL语句融到了一块，使得代码的可读性大幅度降低。

HTML&CSS

今日目标：

- 能够掌握课程中讲解的标签的使用
- 了解css的使用

1, HTML

1.1 介绍

HTML 是一门语言，所有的网页都是用HTML 这门语言编写出来的，也就是HTML是用来写网页的，像京东，12306等网站有很多网页。



这些都是网页展示出来的效果。而HTML也有专业的解释

HTML(HyperText Markup Language): 超文本标记语言:

- 超文本：超越了文本的限制，比普通文本更强大。除了文字信息，还可以定义图片、音频、视频等内容
如上图看到的页面，我们除了能看到一些文字，同时也有大量的图片展示；有些网页也有视频，音频等。这种展示效果超越了文本展示的限制。
- 标记语言：由标签构成的语言

之前学习的XML就是标记语言，由一个一个的标签组成，HTML 也是由标签组成。我们在浏览器页面右键可以查看页面的源代码，如下



可以看到如下内容，就是由一个一个的标签组成的

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge, chrome=1">
    <title>中国铁路12306</title>
    <script>
        window.startTime = new Date().getTime(); //window.onload外开始时间:, window.startTime
    </script>
    <link rel="shortcut icon" href="/images/favicon.ico" type="image/x-icon" />
    <!-- <link href="/css/index.css" rel="stylesheet">
    <link href="/css/global.css" rel="stylesheet">
    <link href="/css/public.css" rel="stylesheet"> -->
    <link href="/fonts/iconfont.css" rel="stylesheet">
    <!-- 日期城市控件 -->
    <!-- <link href="/css/common/calendarNew.css" rel="stylesheet">
    <link href="/css/common/table.css" rel="stylesheet">
    <link href="/css/common/station.css" rel="stylesheet">
    <link rel="stylesheet" href="/css/main.css" -->
    <link rel="stylesheet" href="/css/index_v30001.css">
</head>
<style>
    .typeahead li {
        width: 265px;
        padding-left: 10px;
    }
    a:hover {
        color: #000;
    }
</style>
<body>
    <!-- 页面加载进度 -->
    <div id="page-loading" class="page-loading"></div>
    <!-- 头部 -->
    <div class="header">
        <div class="header">
            <div class="wrapper">
                <!-- 头部内容 -->
                <div class="header-con">
                    <h1 class="logo">
                        <a name="g_href" data-type="1" data-href="index.html" data-redirect="1" href="javascript:;">中国铁路12306</a>
                    <!-- 头部内容 -->
                </div>
            </div>
        </div>
    </div>
    <!-- 内容 -->
    <div class="content">
        <!-- 内容 -->
    </div>
    <!-- 底部 -->
    <div class="footer">
        <div class="footer">
            <div class="wrap">
                <!-- 底部内容 -->
            </div>
        </div>
    </div>
</body>
```

这些标签不像XML那样可以自定义，**HTML中的标签都是预定义好的，运行在浏览器上并由浏览器解析**，然后展示出对应的效果。例如我们想在浏览器上展示出图片就需要使用预定义的 `img` 标签；想展示可以点击的链接的效果就可以使用预定义的 `a` 标签等。

HTML 预定义了很多标签，由于我们是Java工程师、是做后端开发，所以不会每个都学习，页面开发是有专门的前端工程来开发。那为什么我们还要学习呢？在公司中或多或少大家也会涉及到前端开发。

简单的给大家聊一下开发流程：

以后我们是通过Java程序从数据库中查询出来数据，然后交给页面进行展示，这样用户就能通过在浏览器通过页面看到数据。

W3C标准：

W3C是万维网联盟，这个组成是用来定义标准的。他们规定了一个网页是由三部分组成，分别是：

- 结构：对应的是 HTML 语言
- 表现：对应的是 CSS 语言
- 行为：对应的是 JavaScript 语言

HTML定义页面的整体结构；CSS是用来美化页面，让页面看起来更加美观；JavaScript可以使网页动起来，比如轮播图也就是多张图片自动的进行切换等效果。

为了更好的给大家表述这三种语言的作用。我们通过具体的页面给大家说明。

如下只是使用HTML语言编写的页面的结构：

- [Welcome to Edustar center !](#)
- [9856055260](#)
- [Sign](#)
- [Register](#)

EDU[☆]STAR

Type your keyword 提交

- [Home](#)
- [About](#)
- [Contact](#)

Education and Training Center

Edustar is world top education consultancy ipsum dolor sit amet, consectetur adipiscing elit. Cras interdum ante vel aliquet euismod.

[learn more](#)

Education and Training Center

Edustar is world top education consultancy ipsum dolor sit amet, consectetur adipiscing elit. Cras interdum ante vel aliquet euismod.

[learn more](#)

Modern Class

Find the price of your ideal course. Lorem ipsum dolor sit amet

[learn more](#)

Global Students

可以看到页面是比较丑的，但是每一部分其实都已经包含了。接下来咱们加上CSS进行美化看到的效果如下：

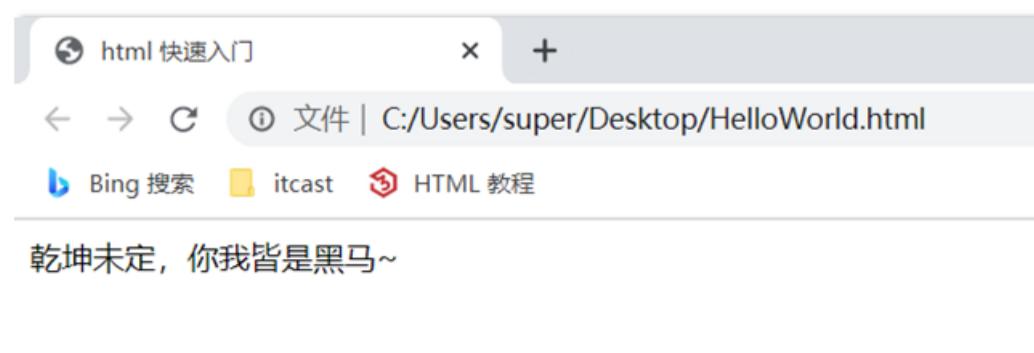
瞬间感觉好看多了，这就是CSS的作用，用来美化页面的。接下来再加上JavaScript试试

在上图中可以看到多了轮播图，在浏览器上它是会自动切换图片的，并且切换的动态效果是很不错的。

看到了前端编写的这三个技术效果后，我们今天学习的是HTML，学习HTML其实就是学习预定义的这些标签。

1.2 快速入门

需求：编写如下图效果的页面



要实现这个页面，我们需要从以下三步进行实现

- 新建文本文件，后缀名改为 .html

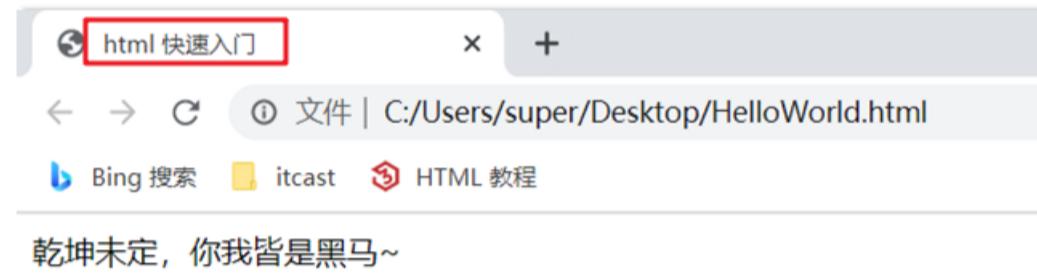
页面文件的后缀名是 .html，所以需要该后缀名

- 编写 HTML 结构标签

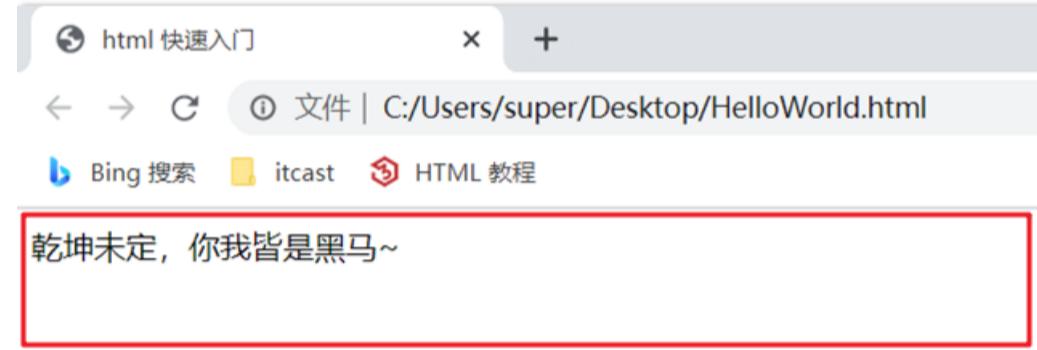
HTML 是由一个一个的标签组成的，但是它也用于表示结构的标签

```
1 <html>
2   <head>
3     <title> </title>
4   </head>
5   <body>
6
7     </body>
8 </html>
```

html 标签是根标签，下面有 head 标签和 body 标签这两个子标签。而 head 标签的 title 子标签是用来定义页面标题名称的，它定义的内容会展示在浏览器的标题位置，如下图红框标记



body 标签的内容会被展示在内容区中，如下图红框标记



- 在中定义文字

代码如下：

```
1 <html>
2   <head>
3     <title>html 快速入门</title>
4   </head>
5   <body>
6     乾坤未定，你我皆是黑马~
7   </body>
8 </html>
```

同学们在访问其他网站页面时会看到字体颜色是五颜六色的，我们可以该字体颜色吗？当然可以了

font 标签就可以使用，该标签有一个 color 属性可以设置字体颜色，如：就是将文字设置成了红颜色。那么我们只需要将需要变成红色的文字放在标签体部分就可以了，如下：

```
1 <html>
2   <head>
3     <title>html 快速入门</title>
4   </head>
5   <body>
6     <font color='red'>乾坤未定，你我皆是黑马~</font>
7   </body>
8 </html>
```

总结：

- HTML 文件以.htm或.html为扩展名
- HTML 结构标签

标签	描述
<HTML>	定义 HTML 文档
<head>	定义关于文档的信息
<title>	定义文档的标题
<body>	定义文档的主体

- HTML 标签不区分大小写

如上案例中的 `font` 写成 `Font` 也是一样可以展示出对应的效果的。

- HTML 标签属性值 单双引皆可

如上案例中的color属性值使用双引号也是可以的。

- HTML 语法松散

比如 `font` 标签不加结束标签也是可以展示出效果的。但是建议同学们在写的时候还是不要这样做，严格按照要求去写。

1.3 基础标签

基础标签就是一些和文字相关的标签，如下：

标签	描述
<h1> ~ <h6>	定义标题，h1最大，h6最小
	定义文本的字体、字体尺寸、字体颜色
	定义粗体文本
<i>	定义斜体文本
<u>	定义文本下划线
<center>	定义文本居中
<p>	定义段落
 	定义折行
<hr>	定义水平线

接下来我们挨个进行讲解

1.3.1 标题标签

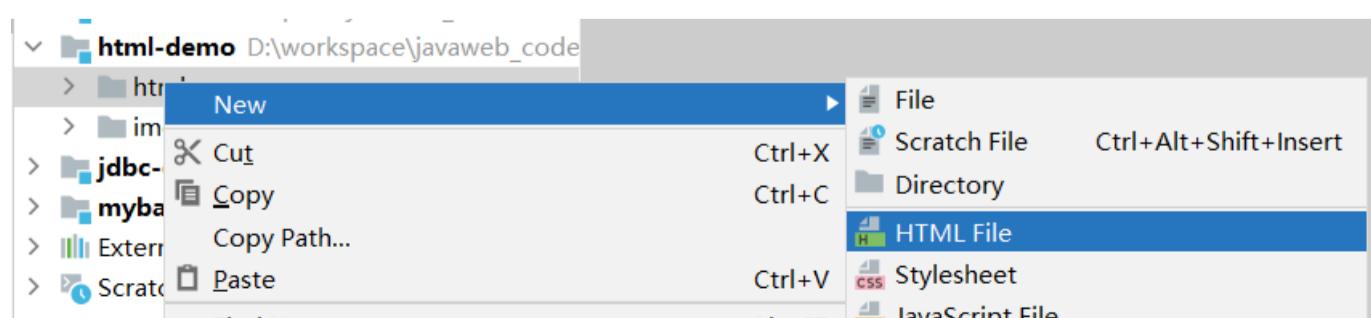
- 创建模块

在 Idea 中创建模块，而我们现在不需要写java代码，所以 `src` 目录就可以删除掉。在模块下创建一个html文件夹，该我们今天的所有的页面文件都放在该文件夹下。模块目录如下



- 创建页面文件

选中 `html` 文件夹右键创建页面文件 (01-基础标签.html)



创建好后 idea 会自动加上结构标签，如下

```

<!-- html5 标识-->
<!DOCTYPE html>
<html lang="en">
  <head>
    <!-- 页面的字符集-->
    <meta charset="UTF-8">
    <title>Title</title>
  </head>
  <body>
    </body>
  </html>

```

我们只需要在 `body` 标签中书写标签。

- 书写标题标签

标题标签中 h1最大， h6最小。

```

1 <h1>我是标题 h1</h1>
2 <h2>我是标题 h2</h2>
3 <h3>我是标题 h3</h3>
4 <h4>我是标题 h4</h4>
5 <h5>我是标题 h5</h5>
6 <h6>我是标题 h6</h6>

```

- 通过浏览器查看效果

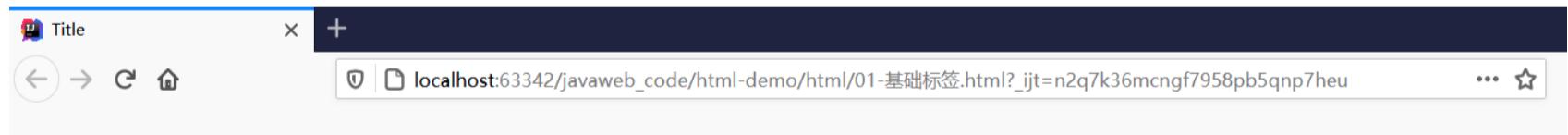
idea 提供了快捷的打开方式，如下图

```

01-基础标签.html ×
1 <!-- html5 标识-->
2 <!DOCTYPE html>
3 <html lang="en">
4   <head>
5     <!-- 页面的字符集-->
6     <meta charset="UTF-8">
7     <title>Title</title>
8   </head>
9   <body>
10
11   <h1>我是标题 h1</h1>
12   <h2>我是标题 h2</h2>
13   <h3>我是标题 h3</h3>
14   <h4>我是标题 h4</h4>
15   <h5>我是标题 h5</h5>
16   <h6>我是标题 h6</h6>

```

浏览器展示效果如下：



我是标题 h1

我是标题 h2

我是标题 h3

我是标题 h4

我是标题 h5

我是标题 h6

1.3.2 hr标签

`hr` 标签在浏览器中呈现出横线的效果。

在页面文件中书写 `hr` 标签

```

1 <hr>

```

效果如下：

1.3.3 字体标签

font：字体标签

- face 属性：用来设置字体。如 "楷体"、"宋体"等
- color 属性：设置文字颜色。颜色有三种表示方式
 - 英文单词：red,pink,blue...
 - 这种方式表示的颜色特别有限，所以一般不用。
 - rgb(值1,值2,值3)：值的取值范围：0~255
◦ 此种方式也就是三原色（红绿蓝）设置方式。例如：rgb(255,0,0)。
◦ 这种书写起来比较麻烦，一般不用。
 - #值1值2值3：值的范围：00~FF
◦ 这种方式是rgb方式的简化写法，以后基本都用此方式。
◦ 值1表示红色的范围，值2表示绿色的范围，值3表示蓝色范围。例如：#ff0000
- size 属性：设置文字大小

代码演示：

```
1 | <font face="楷体" size="5" color="#ff0000">传智教育</font>
```

效果如下：

传智教育

注意：

font 标签已经不建议使用了，以后如果要改变文字字体，大小，颜色可以使用 CSS 进行设置。

1.3.4 换行标签

在页面文件中书写如下内容

- ```
1 | 刚察草原绿草如茵，沙柳河水流淌入湖。藏族牧民索南才让家中，茶几上摆着馓子、麻花和水果，炉子上刚煮开的奶茶香气四溢.....
2 |
3 | 6月8日下午，习近平总书记来到青海省海北藏族自治州刚察县沙柳河镇果洛藏贡麻村，走进牧民索南才让家中，看望慰问藏族群众。
```

在浏览器展示的效果如下：

刚察草原绿草如茵，沙柳河水流淌入湖。藏族牧民索南才让家中，茶几上摆着馓子、麻花和水果，炉子上刚煮开的奶茶香气四溢..... 6月8日下午，习近平总书记来到青海省海北藏族自治州刚察县沙柳河镇果洛藏贡麻村，走进牧民索南才让家中，看望慰问藏族群众。

我们可以看到并没有换行。如果要实现换行效果，需要使用 换行标签（br标签）。

修改页面文件内容如下：

- ```
1 | 刚察草原绿草如茵，沙柳河水流淌入湖。藏族牧民索南才让家中，茶几上摆着馓子、麻花和水果，炉子上刚煮开的奶茶香气四溢.....  
2 | <br>  
3 | 6月8日下午，习近平总书记来到青海省海北藏族自治州刚察县沙柳河镇果洛藏贡麻村，走进牧民索南才让家中，看望慰问藏族群众。
```

浏览器打开效果如下：

刚察草原绿草如茵，沙柳河水流淌入湖。藏族牧民索南才让家中，茶几上摆着馓子、麻花和水果，炉子上刚煮开的奶茶香气四溢..... 6月8日下午，习近平总书记来到青海省海北藏族自治州刚察县沙柳河镇果洛藏贡麻村，走进牧民索南才让家中，看望慰问藏族群众。

现在就有换行效果了。

1.3.5 段落标签

上面文字展示的效果还是不太好，我们想让每一段上下都加空行。此时就需要使用段落标签（p标签）

在页面文件中书写如下内容：

```
1 <p>
2 刚察草原绿草如茵，沙柳河水流淌入湖。藏族牧民索南才让家中，茶几上摆着馓子、麻花和水果，炉子上刚煮开的奶茶香气四溢……
3 </p>
4 <p>
5 6月8日下午，习近平总书记来到青海省海北藏族自治州刚察县沙柳河镇果洛藏贡麻村，走进牧民索南才让家中，看望慰问藏族群众。
6 </p>
```

在浏览器展示的效果如下：

刚察草原绿草如茵，沙柳河水流淌入湖。藏族牧民索南才让家中，茶几上摆着馓子、麻花和水果，炉子上刚煮开的奶茶香气四溢……

6月8日下午，习近平总书记来到青海省海北藏族自治州刚察县沙柳河镇果洛藏贡麻村，走进牧民索南才让家中，看望慰问藏族群众。

这种效果就会比之前的效果好一些，呈现出段落的效果。

1.3.6 加粗、斜体、下划线标签

- b：加粗标签
- i：斜体标签
- u：下划线标签，在文字的下方有一条横线

代码如下：

```
1 <b>沙柳河水流淌</b><br>
2 <i>沙柳河水流淌</i><br>
3 <u>沙柳河水流淌</u><br>
```

在浏览器展示的效果如下：

沙柳河水流淌
沙柳河水流淌
沙柳河水流淌

1.3.7 居中标签

center：文本居中

代码如下：

```
1 <hr>
2 <center>
3     <b>沙柳河水流淌</b>
4 </center>
```

在浏览器效果如下：

沙柳河水流淌

1.3.8 案例

实现如下图所示页面效果：

公司简介

传智教育(股票代码 003032)，隶属江苏传智播客教育科技股份有限公司，注册资本4亿元，是第一个实现A股IPO的教育企业，公司致力于培养高精尖数字化人才，主要培养人工智能、python+大数据开发、智能制造、软件、互联网、区块链等数字化专业人才及数据分析、网络营销、新媒体等数字化应用人才。公司由一批拥有10年以上开发管理经验，且来自互联网或研究机构的IT精英组成，负责研究、开发教学模式和课程内容。公司具有完善的课程研发体系，一直走在整个行业发展的前端，在行业内竖立起了良好的品质口碑。

民族振兴靠人才，中华民族正处于伟大复兴之路上，要赢得国际竞争，需要拥有大量的科技人才，我们将肩负起民族使命，在三尺讲台诲人不倦 著书立说，为科技行业培养出大量的优秀人才，促进民族伟大复兴！我们的使命是：**为中华民族伟大复兴而讲课，为千万学生少走弯路而著书**。

探索教育之路，长途漫漫。传智教育希望通过自己的努力，寻找出一条更符合人类自然成长规律的教育之路，建立起一个新的教育生态环境，让中国的家长和孩子们在现有的教育体系之外，再多一些选择的机会。因此“**探索教育本源，开辟教育新生态**”便成为了所有传智人为之奋斗的终极愿景，也是所有传智人共同努力的目标。为此，15年来，传智人不曾有一丝懈怠，相信在传智人的不懈努力下，大道不远，终在脚下。

此案例同学们自己实现，用我们学过的基础标签。

注意：在上图页面中版权所有里有特殊字符，需要使用转义字符。有如下转义字符：

HTML 原代码	显示结果	描述
<	<	小于号或显示标记
>	>	大于号或显示标记
&	&	可用于显示其它特殊字符
"	"	引号
®	®	已注册
©	©	版权
™	™	商标
 		不断行的空白

1.4 图片、音频、视频标签

标签	描述
	定义图片
<audio>	定义音频
<video>	定义视频

- img: 定义图片
 - src: 规定显示图像的 URL (统一资源定位符)
 - height: 定义图像的高度
 - width: 定义图像的宽度
- audio: 定义音频。支持的音频格式: MP3、WAV、OGG
 - src: 规定音频的 URL
 - controls: 显示播放控件
- video: 定义视频。支持的音频格式: MP4, WebM、OGG
 - src: 规定视频的 URL
 - controls: 显示播放控件

尺寸单位:

height属性和width属性有两种设置方式：

- 像素：单位是px
- 百分比。占父标签的百分比。例如宽度设置为 50%，意思就是占它的父标签宽度的一半（50%）

资源路径:

图片，音频，视频标签都有src属性，而src是用来指定对应的图片，音频，视频文件的路径。此处的图片，音频，视频就称为资源。资源路径有如下两种设置方式：

- 绝对路径：完整路径

这里的绝对路径是网络中的绝对路径。格式为：协议://ip地址:端口号/资源名称。

如：

```
1 
```

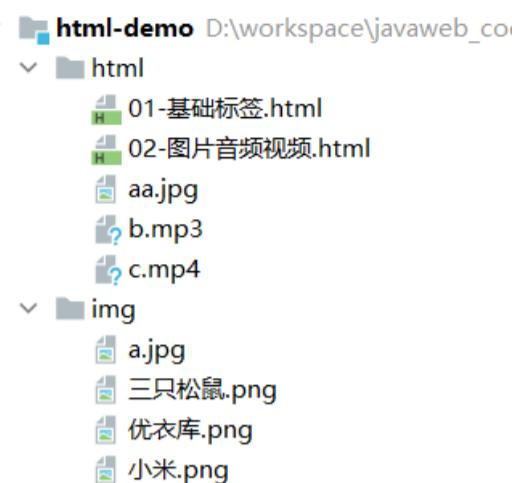
这里src属性的值就是网络中的绝对路径。

- 相对路径：相对位置关系

找页面和其他资源的相对路径。

./ 表示当前路径
../ 表示上一级路径
../../ 表示上两级路径

如模块目录结构如下：



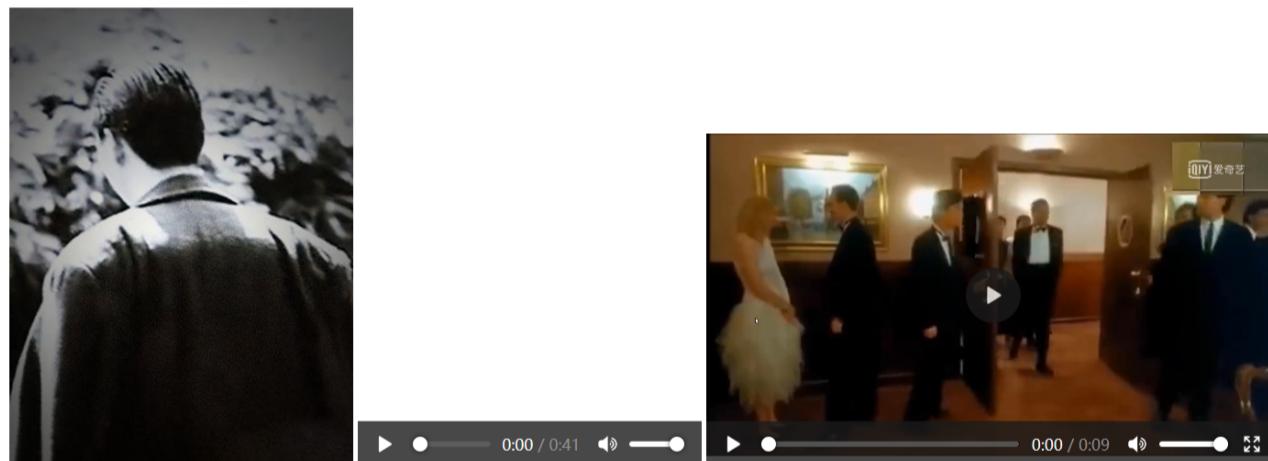
在 `01-基础标签.html` 里的标签中找不同的图片，路径写法不同

```
1 <!--在该页面找a.jpg，就需要先回到上一级目录，该级目录有img目录，进入该目录就可以找到 a.jpg图片-->
2 
3 <!--该页面和aa.jpg 是在同一级下，所以可以直接写 图片的名称，也可以写成 ./aa.jpg-->
4 
```

使用这些标签的代码如下：

```
1 
2 <audio src="b.mp3" controls></audio>
3 <video src="c.mp4" controls width="500" height="300"></video>
```

在浏览器展示的效果如下：



1.5 超链接标签

在网页中可以看到很多超链接标签，如下

相关搜索

- [唯美图片大全](#)
- [二次元图片大全](#)
- [图片素材网](#)
- [全屏动漫壁纸](#)
- [超大图片](#)
- [找图网站](#)
- [图片 可爱卡通](#)
- [风景图片大全 大自然](#)

上图红框中的都是超链接，当我们点击这些超链接时会跳转到其他的页面或者资源。而超链接使用的是 `a` 标签。

标签	描述
<code><a></code>	定义超链接，用于链接到另一个资源

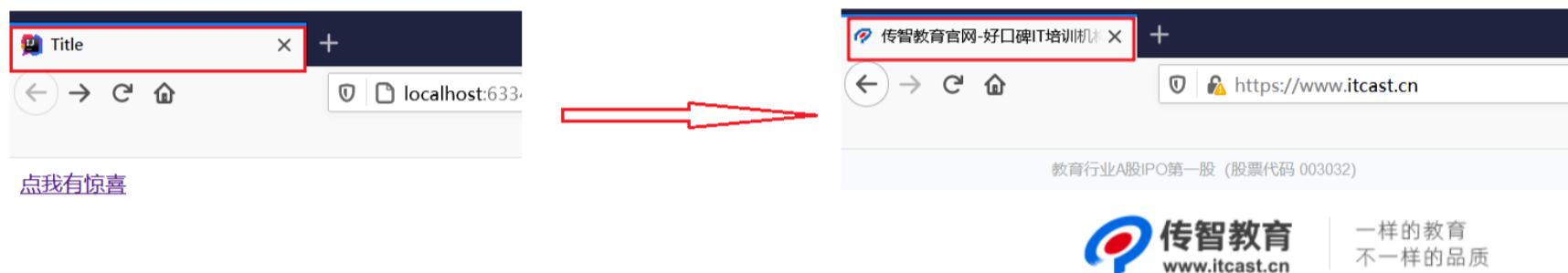
a 标签属性:

- href: 指定访问资源的URL
- target: 指定打开资源的方式
 - _self: 默认值, 在当前页面打开
 - _blank: 在空白页面打开

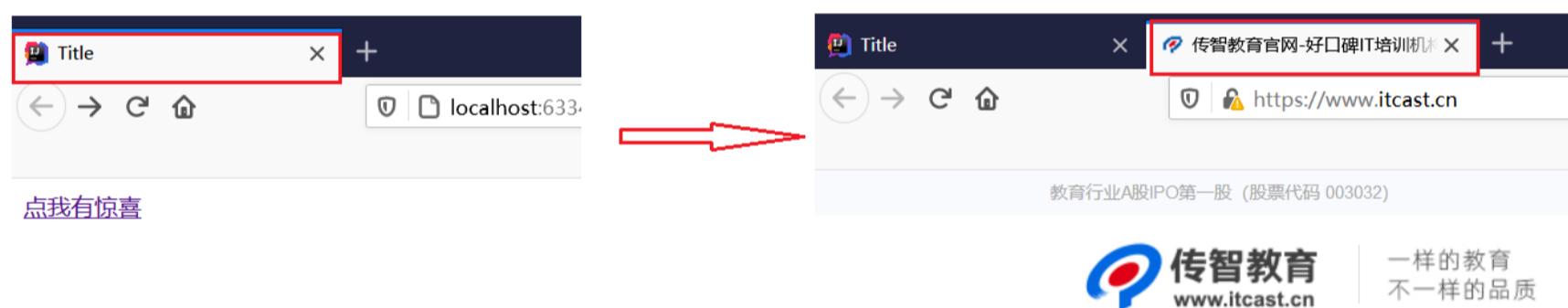
代码演示:

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>Title</title>
6 </head>
7 <body>
8   <a href="https://www.itcast.cn" target="_self">点我有惊喜</a>
9 </body>
10 </html>
```

效果图示:



当我们将 `target` 属性值设置为 `_blank`, 效果图示:



1.6 列表标签

HTML 中列表分为

- 有序列表

如下图, 页面效果中是有标号对每一项进行标记的。

<p>1. 咖啡 效果 2. 茶 3. 牛奶</p>	<p> 咖啡 茶 牛奶 </p>
---	--

- 无序列表

如下图, 页面效果中没有标号对每一项进行标记, 而是使用 点 进行标记。

<p>• 咖啡 效果 • 茶 • 牛奶</p>	<p> 咖啡 茶 牛奶 </p>
--------------------------------------	--

标签说明:

标签	描述
	定义有序列表
	定义无序列表
	定义列表项

有序列表中的 `type` 属性用来指定标记的标号的类型 (数字、字母、罗马数字等)

无序列表中的 `type` 属性用来指定标记的形状

代码演示：

```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>Title</title>
6 </head>
7 <body>
8   <ol type="A">
9     <li>咖啡</li>
10    <li>茶</li>
11    <li>牛奶</li>
12  </ol>
13
14  <ul type="circle">
15    <li>咖啡</li>
16    <li>茶</li>
17    <li>牛奶</li>
18  </ul>
19 </body>
20 </html>

```

1.7 表格标签

序号	品牌logo	品牌名称	企业名称
010		三只松鼠	三只松鼠
009		优衣库	优衣库
008		小米	小米科技有限公司

如上图就是一个表格，表格可以使用如下标签定义

- table：定义表格
 - border: 规定表格边框的宽度
 - width : 规定表格的宽度
 - cellspacing: 规定单元格之间的空白
- tr：定义行
 - align: 定义表格行的内容对齐方式
- td：定义单元格
 - rowspan:规定单元格可横跨的行数
 - colspan:规定单元格可横跨的列数
- th: 定义表头单元格

代码演示：

```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>

```

```

4     <meta charset="UTF-8">
5     <title>Title</title>
6 </head>
7 <body>
8
9 <table border="1" cellspacing="0" width="500">
10    <tr>
11        <th>序号</th>
12        <th>品牌logo</th>
13        <th>品牌名称</th>
14        <th>企业名称</th>
15    </tr>
16    <tr align="center">
17        <td>010</td>
18        <td></td>
19        <td>三只松鼠</td>
20        <td>三只松鼠</td>
21    </tr>
22
23    <tr align="center">
24        <td>009</td>
25        <td></td>
26        <td>优衣库</td>
27        <td>优衣库</td>
28    </tr>
29
30    <tr align="center">
31        <td>008</td>
32        <td></td>
33        <td>小米</td>
34        <td>小米科技有限公司</td>
35    </tr>
36 </table>
37 </body>
38 </html>

```

1.8 布局标签

标签	描述
<div>	定义 HTML 文档中的一个区域部分，经常与 CSS 一起使用，用来布局网页
	用于组合行内元素。

这两个标签，一般都是和css结合到一块使用来实现页面的布局。

`div` 标签在浏览器上会有换行的效果，而 `span` 标签在浏览器上没有换行效果。

代码演示：

```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <title>Title</title>
6 </head>
7 <body>
8     <div>我是div</div>
9     <div>我是div</div>
10    <span>我是span</span>
11    <span>我是span</span>
12 </body>
13 </html>

```

浏览器效果如下：

我是div
我是div
我是span 我是span

1.9 表单标签

表单标签效果大家其实都不陌生，像登陆页面、注册页面等都是表单。

The image shows two side-by-side screenshots of web forms. On the left is a 'Baidu' login form with fields for '手机/邮箱/用户名' (Phone number/Email/Username) and '密码' (Password), both with placeholder text. A blue '登录' (Login) button is at the bottom. On the right is a '欢迎注册' (Welcome to Register) form with fields for '用户名' (Username), '手机号' (Mobile number), '密 码' (Password), and '验证码' (Verification code). Each field has a placeholder. Below the fields are '立即注册' (Register Now) and '注册' (Register) buttons. A checkbox for accepting terms and conditions is at the bottom.

像这样的表单就是用来采集用户输入的数据，然后将数据发送到服务端，服务端会对数据库进行操作，比如注册就是将数据保存到数据库中，而登陆就是根据用户名和密码进行数据库的查询操作。

表单是很重要的标签，需要大家重点来学习。

1.9.1 表单标签概述

表单：在网页中主要负责数据采集功能，使用
标签定义表单
表单项(元素)：不同类型的 input 元素、下拉列表、文本域等

标签	描述
<form>	定义表单
<input>	定义表单项，通过type属性控制输入形式
<label>	为表单项定义标注
<select>	定义下拉列表
<option>	定义下拉列表的列表项
<textarea>	定义文本域

`form` 是表单标签，它在页面上没有任何展示的效果。需要借助于表单项标签来展示不同的效果。如下图就是不同的表单项标签展示出来的效果。

1.9.2 form标签属性

- action:** 规定当提交表单时向何处发送表单数据，该属性值就是URL

以后会将数据提交到服务端，该属性需要书写服务端的URL。而今天我们可以书写`#`，表示提交到当前页面来看效果。

- method :** 规定用于发送表单数据的方式

method取值有如下两种：

- get: 默认值。如果不设置method属性则默认就是该值
 - 请求参数会拼接在URL后边
 - url的长度有限制 4KB
- post:
 - 浏览器会将数据放到http请求消息体中
 - 请求参数无限制的

1.9.3 代码演示

由于表单标签在页面上没有任何展示的效果，所以在演示的过程是会先使用`input`这个表单项标签展示输入框效果。

代码如下：

```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>Title</title>
6 </head>
7 <body>
8   <form>
9     <input type="text">
10    <input type="submit">
11  </form>
12 </body>
13 </html>

```

浏览器展示效果如下：



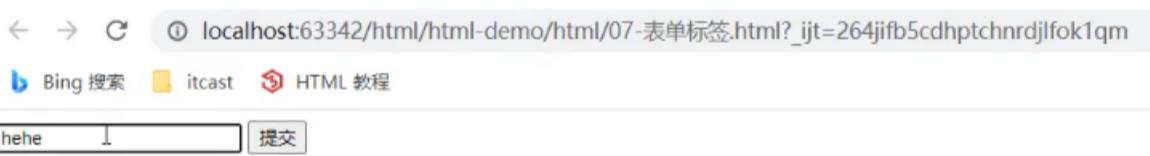
从效果可以看到页面有一个输入框，用户可以在数据框中输入自己想输入的内容，点击提交按钮以后会将数据发送到服务端，当然现在肯定不能实现。现在我们可以将 `form` 标签的 `action` 属性值设置为 `#`，将其将数据提交到当前页面。还需要注意一点，要想提交数据，`input` 输入框必须设置 `name` 属性。代码如下：

```

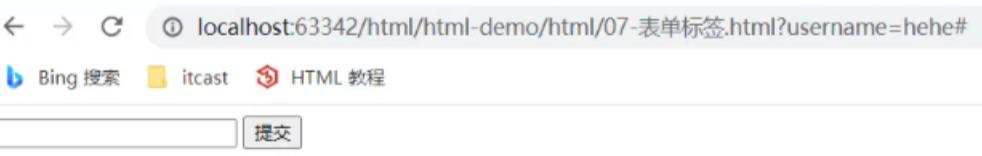
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>Title</title>
6 </head>
7 <body>
8   <form action="#">
9     <input type="text" name="username">
10    <input type="submit">
11  </form>
12 </body>
13 </html>

```

浏览器展示效果如下：



在输入框输入 `hehe`，然后点击 `提交` 按钮，就能看到如下效果

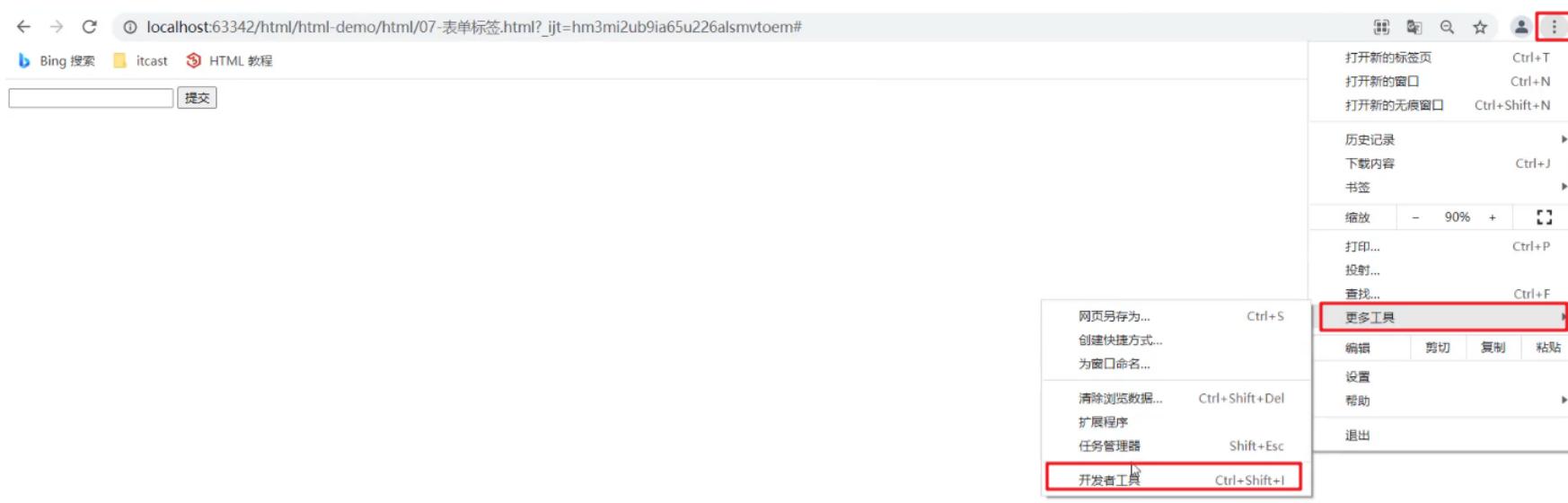


我们可以看到在浏览器的地址栏的URL后拼接了我们提交的数据。`username` 就是输入框 `name` 属性值，而 `hehe` 就是我们输入框输入的内容。

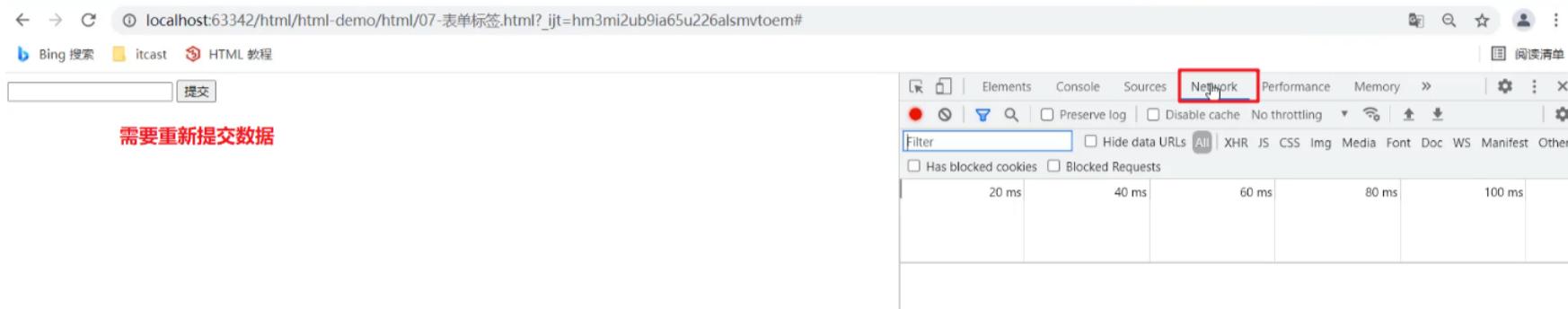
接下来我们来聊 `method` 属性，默认是 `method = 'get'`，所以该取值就会将数据拼接到URL的后面。那我们将 `method` 属性值设置为 `post`，浏览器的效果如下：



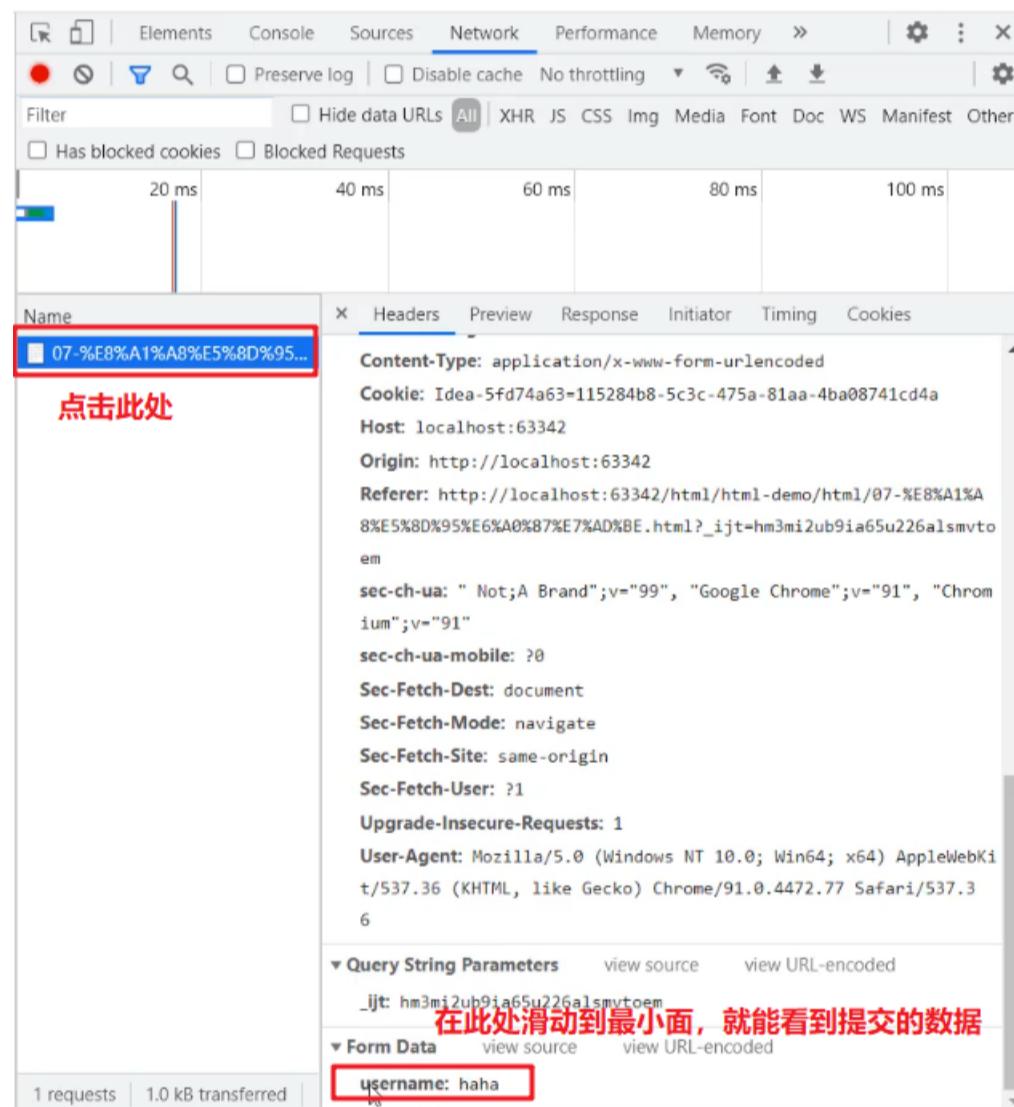
从上图可以看出数据并没有拼接到 URL 后，那怎么看提交的数据呢？我们可以使用浏览器的开发者工具来查看



按照如上步骤操作能看到如下页面



重新提交数据后，可以看到提交的数据，如下图



1.10 表单项标签

表单项标签有很多，不同的表单项标签有不同的展示效果。表单项标签可以分为以下三个：

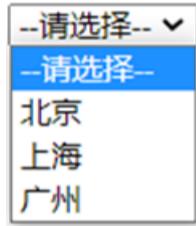
- <input>：表单项，通过type属性控制输入形式

<input> 标签有个 type 属性。 type 属性的取值不同，展示的效果也不一样

type 取值	描述
text	默认值。定义单行的输入字段 <input type="text" value="superbaby"/>
password	定义密码字段 <input type="password" value="....."/>
radio	定义单选按钮 <input checked="" type="radio"/> 男 <input type="radio"/> 女
checkbox	定义复选框 <input checked="" type="checkbox"/> 旅游 <input checked="" type="checkbox"/> 电影 <input type="checkbox"/> 游戏
file	定义文件上传按钮 <input type="file"/> 未选择任何文件
hidden	定义隐藏的输入字段
submit	定义提交按钮，提交按钮会把表单数据发送到服务器 <input type="submit" value="提交"/>
reset	定义重置按钮，重置按钮会清除表单中的所有数据 <input type="reset" value="重置"/>
button	定义可点击按钮 <input type="button" value="按钮"/>

- <select>：定义下拉列表，<option> 定义列表项

如下图就是下拉列表的效果：



- <textarea>：文本域

如下图就是文本域效果。它可以输入多行文本，而 `input` 数据框只能输入一行文本。



注意：

- 以上标签项的内容要想提交，必须得定义 `name` 属性。
- 每一个标签都有 `id` 属性，`id` 属性值是唯一的标识。
- 单选框、复选框、下拉列表需要使用 `value` 属性指定提交的值。

代码演示：

```

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>Title</title>
6  </head>
7  <body>
8      <form action="#" method="post">
9          <input type="hidden" name="id" value="123">
10
11         <label for="username">用户名: </label>
12         <input type="text" name="username" id="username"><br>
13
14         <label for="password">密码: </label>
15         <input type="password" name="password" id="password"><br>
16
17         性别:
18         <input type="radio" name="gender" value="1" id="male"> <label for="male">男</label>
19         <input type="radio" name="gender" value="2" id="female"> <label for="female">女
</label>
20         <br>
21
22         爱好:
23         <input type="checkbox" name="hobby" value="1"> 旅游
24         <input type="checkbox" name="hobby" value="2"> 电影
25         <input type="checkbox" name="hobby" value="3"> 游戏

```

```

26     <br>
27
28     头像:
29     <input type="file"><br>
30
31     城市:
32     <select name="city">
33         <option>北京</option>
34         <option value="shanghai">上海</option>
35         <option>广州</option>
36     </select>
37     <br>
38
39     个人描述:
40     <textarea cols="20" rows="5" name="desc"></textarea>
41     <br>
42     <br>
43     <input type="submit" value="免费注册">
44     <input type="reset" value="重置">
45     <input type="button" value="一个按钮">
46   </form>
47 </body>
48 </html>

```

在浏览器的效果如下:

用户名:

密码:

性别: 男 女

爱好: 旅游 电影 游戏

头像: 浏览... 未选择文件。

城市: 北京 ▾

个人描述:

2, CSS

2.1 概述

CSS 是一门语言，用于控制网页表现。我们之前介绍过W3C标准。W3C标准规定了网页是由以下组成：

- 结构：HTML
- 表现：CSS
- 行为：JavaScript

CSS也有一个专业的名字：**Cascading Style Sheet (层叠样式表)**。

如下面的代码，`style` 标签中定义的就是css代码。该代码描述了将 `div` 标签的内容的字体颜色设置为 红色。

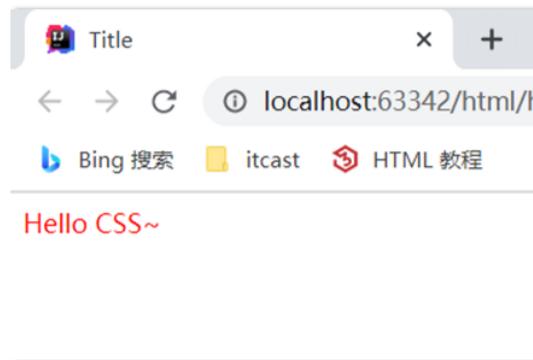
```

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>Title</title>
6      <style>
7          div{
8              color: red;
9          }
10     </style>
11 </head>
12 <body>

```

```
13     <div>Hello CSS~</div>
14 </body>
15 </html>
```

在浏览器中的效果如下：



2.2 css 导入方式

css 导入方式其实就是 css 代码和 html 代码的结合方式。CSS 导入 HTML 有三种方式：

- 内联样式：在标签内部使用 style 属性，属性值是 css 属性键值对

```
1 | <div style="color: red">Hello CSS~</div>
```

这种方式只能作用在这一个标签上，如果其他的标签也想使用同样的样式，那就需要在其他标签上写上相同的样式。复用性太差。

- 内部样式：定义

JavaScript

今日目标

- 掌握 JavaScript 的基础语法
- 掌握 JavaScript 的常用对象 (Array、String)
- 能根据需求灵活运用定时器及通过 js 代码进行页面跳转
- 能通过DOM 对象对标签进行常规操作
- 掌握常用的事件
- 能独立完成表单校验案例

1, JavaScript简介

JavaScript 是一门跨平台、面向对象的脚本语言，而Java语言也是跨平台的、面向对象的语言，只不过Java是编译语言，是需要编译成字节码文件才能运行的；JavaScript是脚本语言，不需要编译，由浏览器直接解析并执行。

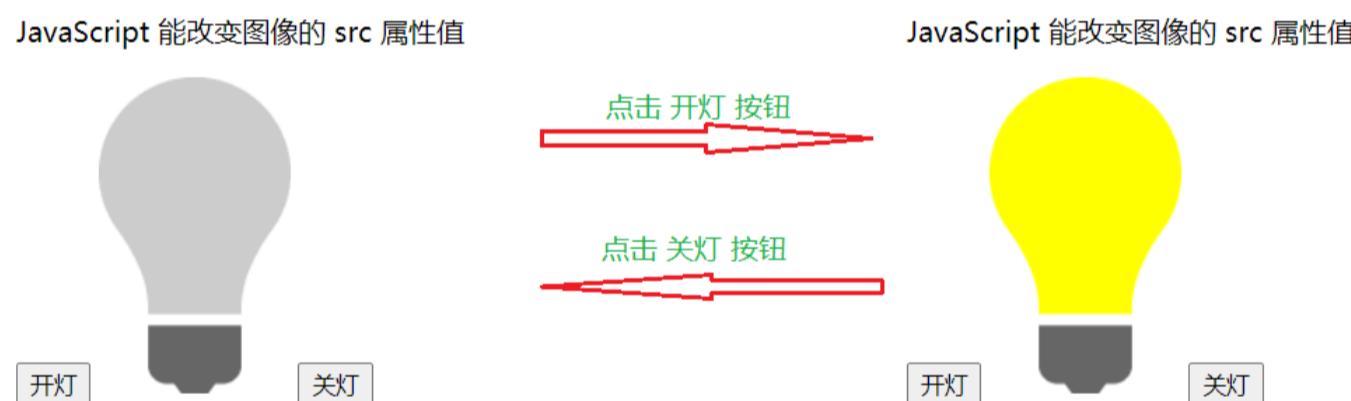
JavaScript 是用来控制网页行为的，它能使网页可交互；那么它可以做什么呢？如改变页面内容、修改指定元素的属性值、对表单进行校验等，下面是这些功能的效果展示：

- 改变页面内容



当我点击上面左图的 `点击我` 按钮，按钮上面的文本就改为上面右图内容，这就是js 改变页面内容的功能。

- 修改指定元素的属性值



当我们点击上图的 `开灯` 按钮，效果就是上面右图效果；当我点击 `关灯` 按钮，效果就是上面左图效果。其他这个功能中有两张灯泡的图片（使用img标签进行展示），通过修改 img 标签的 src 属性值改变展示的图片来实现。

- 对表单进行校验



在上面左图的输入框输入用户名，如果输入的用户名是不满足规则的就展示右图(上)的效果；如果输入的用户名是满足规则的就展示右图(下)的效果。

JavaScript 和 Java 是完全不同的语言，不论是概念还是设计，只是名字比较像而已。但是**基础语法类似**，所以我们有java的学习经验，再学习JavaScript 语言就相对比较容易些。

JavaScript (简称: JS) 在 1995 年由 Brendan Eich 发明，并于 1997 年成为一部 ECMA 标准。ECMA 规定了一套标准 就叫 **ECMAScript**，所有的客户端校验语言必须遵守这个标准，当然 JavaScript 也遵守了这个标准。ECMAScript 6 (简称ES6) 是最新的 JavaScript 版本 (发布于 2015 年)，我们的课程就是基于最新的 **ES6** 进行讲解。

2, JavaScript引入方式

JavaScript 引入方式就是 HTML 和 JavaScript 的结合方式。JavaScript引入方式有两种：

- 内部脚本：将 JS 代码定义在 HTML 页面中
- 外部脚本：将 JS 代码定义在外部 JS 文件中，然后引入到 HTML 页面中

2.1 内部脚本

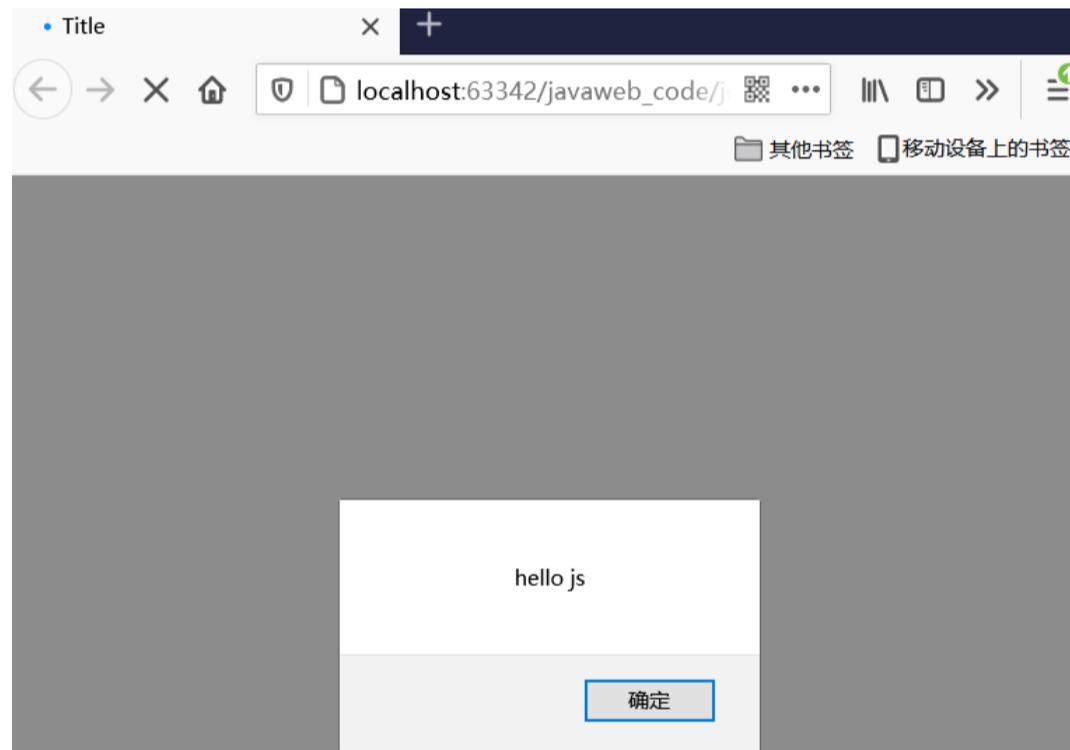
在 HTML 中，JavaScript 代码必须位于 `<script>` 与 `</script>` 标签之间

代码如下：

`alert(数据)` 是 JavaScript 的一个方法，作用是将参数数据以浏览器弹框的形式输出出来。

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>Title</title>
6 </head>
7 <body>
8
9 <script>
10   alert("hello js1");
11 </script>
12 </body>
13 </html>
```

效果如下：



从结果可以看到 js 代码已经执行了。

提示：

- 在 HTML 文档中可以在任意地方，放置任意数量的标签。如下图

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>Title</title>
6   <script>
7     alert("hello js1");
8   </script>
9 </head>
10 <body>
11
12 <script>
13   alert("hello js1");
14 </script>
15
16 </body>
```

```
17 </html>
18 <script>
19     alert("hello js1");
20 </script>
```

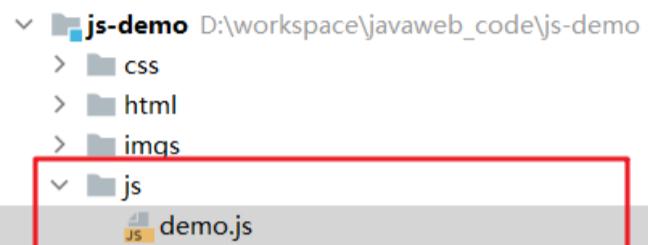
- 一般把脚本置于元素的底部，可改善显示速度

因为浏览器在加载页面的时候会从上往下进行加载并解析。我们应该让用户看到页面内容，然后再展示动态的效果。

2.2 外部脚本

第一步：定义外部 js 文件。如定义名为 demo.js 的文件

项目结构如下：



demo.js 文件内容如下：

```
1 | alert("hello js");
```

第二步：在页面中引入外部的js文件

在页面使用 `script` 标签中使用 `src` 属性指定 js 文件的 URL 路径。

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <title>Title</title>
6 </head>
7 <body>
8
9 <script src="../js/demo.js"></script>
10 </body>
11 </html>
```

注意：

- 外部脚本不能包含 `<script>` 标签
在js文件中直接写 js 代码即可，不要在js文件中写 `script` 标签
- `<script>` 标签不能自闭合
在页面中引入外部js文件时，不能写成 `<script src="../js/demo.js" />`。

3, JavaScript基础语法

3.1 书写语法

- 区分大小写：与 Java 一样，变量名、函数名以及其他一切东西都是区分大小写的
- 每行结尾的分号可有可无
如果一行上写多个语句时，必须加分号用来区分多个语句。
- 注释
 - 单行注释：`// 注释内容`
 - 多行注释：`/* 注释内容 */`

注意：JavaScript 没有文档注释

- 大括号表示代码块

下面语句大家肯定能看懂，和 java 一样 大括号表示代码块。

```
1 if (count == 3) {  
2     alert(count);  
3 }
```

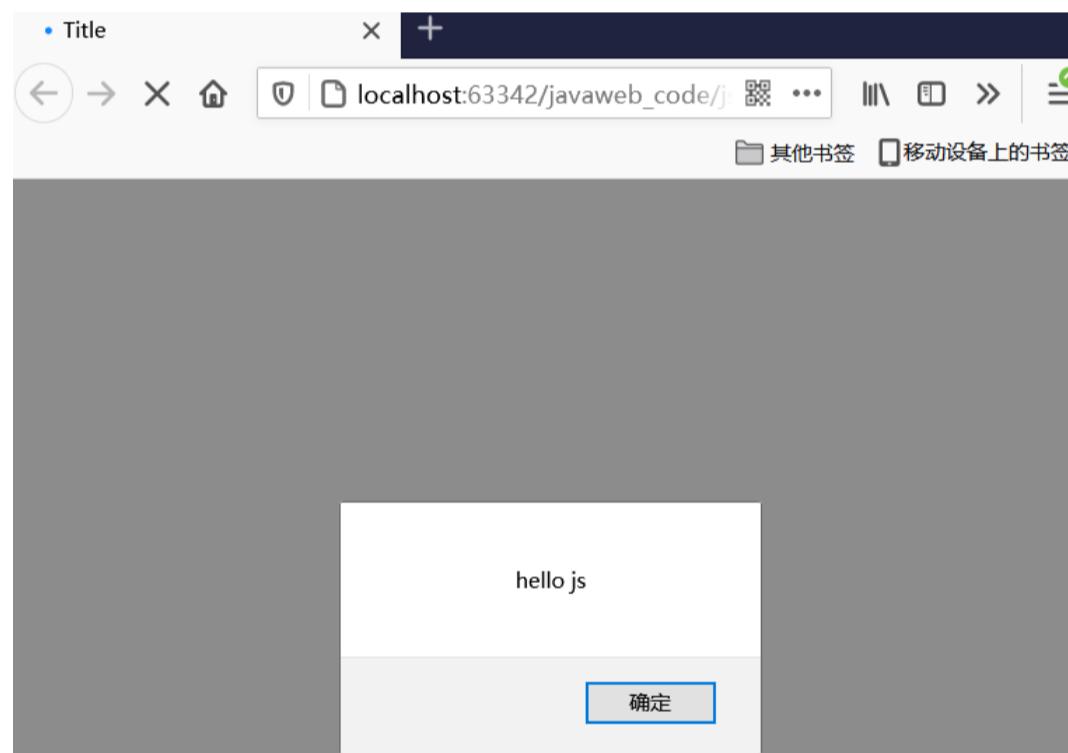
3.2 输出语句

js 可以通过以下方式进行内容的输出，只不过不同的语句输出到的位置不同

- 使用 `window.alert()` 写入警告框

```
1 <!DOCTYPE html>  
2 <html lang="en">  
3 <head>  
4     <meta charset="UTF-8">  
5     <title>Title</title>  
6 </head>  
7 <body>  
8  
9 <script>  
10    window.alert("hello js");//写入警告框  
11 </script>  
12 </body>  
13 </html>
```

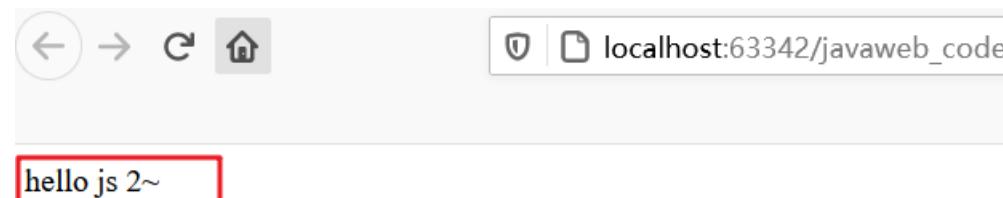
上面代码通过浏览器打开，我们可以看到如下图弹框效果



- 使用 `document.write()` 写入 HTML 输出

```
1 <!DOCTYPE html>  
2 <html lang="en">  
3 <head>  
4     <meta charset="UTF-8">  
5     <title>Title</title>  
6 </head>  
7 <body>  
8  
9 <script>  
10    document.write("hello js 2~");//写入html页面  
11 </script>  
12 </body>  
13 </html>
```

上面代码通过浏览器打开，我们可以在页面上看到 `document.write(内容)` 输出的内容



- 使用 `console.log()` 写入浏览器控制台

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>Title</title>
6 </head>
7 <body>
8
9 <script>
10   console.log("hello js 3");//写入浏览器的控制台
11 </script>
12 </body>
13 </html>
```

上面代码通过浏览器打开，我们可以在不能页面上看到 `console.log(内容)` 输出的内容，它是输出在控制台了，而怎么在控制台查看输出的内容呢？在浏览器界面按 `F12` 就可以看到下图的控制台



3.3 变量

JavaScript 中用 `var` 关键字（variable 的缩写）来声明变量。格式 `var 变量名 = 数据值;`。而在 JavaScript 是一门弱类型语言，变量**可以存放不同类型的值**；如下在定义变量时赋值为数字数据，还可以将变量的值改为字符串类型的数

```
1 var test = 20;
2 test = "张三";
```

js 中的变量名命名也有如下规则，和java语言基本都相同

- 组成字符可以是任何字母、数字、下划线（_）或美元符号（\$）
- 数字不能开头
- 建议使用驼峰命名

JavaScript 中 `var` 关键字有点特殊，有以下地方和其他语言不一样

- 作用域：全局变量

```
1 {
2   var age = 20;
3 }
4 alert(age); // 在代码块中定义的age 变量，在代码块外边还可以使用
```

- 变量可以重复定义

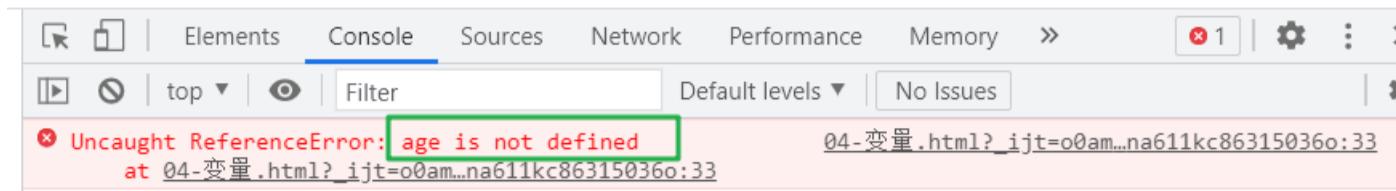
```
1 {
2   var age = 20;
3   var age = 30;//JavaScript 会用 30 将之前 age 变量的 20 替换掉
4 }
5 alert(age); //打印的结果是 30
```

针对如上的问题，**ECMAScript 6 新增了 `let` 关键字来定义变量**。它的用法类似于 `var`，但是所声明的变量，只在 `let` 关键字所在的代码块内有效，且不允许重复声明。

例如：

```
1 {
2     let age = 20;
3 }
4 alert(age);
```

运行上面代码，浏览器并没有弹框输出结果，说明这段代码是有问题的。通过 F12 打开开发者模式可以看到如下错误信息



而如果在代码块中定义两个同名的变量，IDEA 开发工具就直接报错了

```
{  
    let age = 30;  
    let age = 20;  
}  
alert(age);
```

ECMAScript 6 新增了 const 关键字，用来声明一个只读的常量。一旦声明，常量的值就不能改变。 通过下面的代码看一下常用的特点就可以了

```
const PI = 3.14;  
PI = 3;
```

我们可以看到给 PI 这个常量重新赋值时报错了。

3.4 数据类型

JavaScript 中提供了两类数据类型：原始类型 和 引用类型。

使用 typeof 运算符可以获取数据类型

```
alert(typeof age); // 以弹框的形式将 age 变量的数据类型输出
```

原始数据类型：

- **number**: 数字（整数、小数、NaN(Not a Number)）

```
1 var age = 20;  
2 var price = 99.8;  
3  
4 alert(typeof age); // 结果是 : number  
5 alert(typeof price); // 结果是 : number
```

注意： NaN 是一个特殊的 number 类型的值，后面用到再说

- **string**: 字符、字符串，单双引皆可

```
1 var ch = 'a';  
2 var name = '张三';  
3 var addr = "北京";  
4  
5 alert(typeof ch); // 结果是 string  
6 alert(typeof name); // 结果是 string  
7 alert(typeof addr); // 结果是 string
```

注意： 在 js 中 双引号和单引号都表示字符串类型的数据

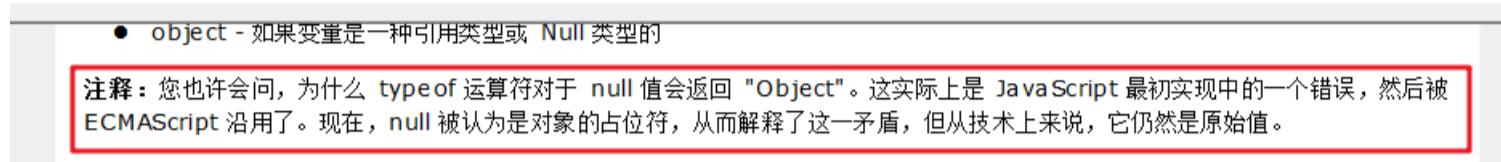
- **boolean**: 布尔。true, false

```
1 var flag = true;
2 var flag2 = false;
3
4 alert(typeof flag); //结果是 boolean
5 alert(typeof flag2); //结果是 boolean
```

- **null**: 对象为空

```
1 var obj = null;
2
3 alert(typeof obj); //结果是 object
```

为什么打印上面的 obj 变量的数据类型，结果是object；这个官方给出了解释，下面是从官方文档截的图



- **undefined**: 当声明的变量未初始化时，该变量的默认值是 undefined

```
1 var a ;
2 alert(typeof a); //结果是 undefined
```

3.5 运算符

JavaScript 提供了如下的运算符。大部分和 Java 语言都是一样的，不同的是 JS 关系运算符中的 `==` 和 `===`，一会儿我们只演示这两个的区别，其他运算符将不做演示

- 一元运算符: `++`, `--`
- 算术运算符: `+`, `-`, `*`, `/`, `%`
- 赋值运算符: `=`, `+=`, `-=`...
- 关系运算符: `>`, `<`, `>=`, `<=`, `!=`, `==`, `====`...
- 逻辑运算符: `&&`, `||`, `!`
- 三元运算符: 条件表达式 `? true_value : false_value`

3.5.1 == 和 === 区别

概述:

- `==`:
 1. 判断类型是否一样，如果不一样，则进行类型转换
 2. 再去比较其值
- `===`: js 中的全等于
 1. 判断类型是否一样，如果不一样，直接返回false
 2. 再去比较其值

代码:

```
1 var age1 = 20;
2 var age2 = "20";
3
4 alert(age1 == age2); // true
5 alert(age1 === age2); // false
```

3.5.2 类型转换

上述讲解 `==` 运算符时，发现会进行类型转换，所以接下来我们来详细的讲解一下 JavaScript 中的类型转换。

- 其他类型转为 number
 - string 转换为 number 类型：按照字符串的字面值，转为数字。如果字面值不是数字，则转为NaN
- 将 string 转换为 number 有两种方式：
- 使用 `+` 正号运算符：

```
1 var str = +"20";
2 alert(str + 1) //21
```

- 使用 `parseInt()` 函数(方法):

```
1 var str = "20";
2 alert(parseInt(str) + 1);
```

建议使用 `parseInt()` 函数进行转换。

- boolean 转换为 number 类型: true 转为1, false转为0

```
1 var flag = +false;
2 alert(flag); // 0
```

- 其他类型转为boolean

- number 类型转换为 boolean 类型: 0和NaN转为false, 其他的数字转为true
- string 类型转换为 boolean 类型: 空字符串转为false, 其他的字符串转为true
- null类型转换为 boolean 类型是 false
- undefined 转换为 boolean 类型是 false

代码如下:

```
1 // var flag = 3;
2 // var flag = "";
3 var flag = undefined;
4
5 if(flag){
6     alert("转为true");
7 }else {
8     alert("转为false");
9 }
```

使用场景:

在 Java 中使用字符串前, 一般都会先判断字符串不是null, 并且不是空字符才会做其他的一些操作, JavaScript也有类型的操作, 代码如下:

```
1 var str = "abc";
2
3 //健壮性判断
4 if(str != null && str.length > 0){
5     alert("转为true");
6 }else {
7     alert("转为false");
8 }
```

但是由于JavaScript 会自动进行类型转换, 所以上述的判断可以进行简化, 代码如下:

```
1 var str = "abc";
2
3 //健壮性判断
4 if(str){
5     alert("转为true");
6 }else {
7     alert("转为false");
8 }
```

3.6 流程控制语句

JavaScript 中提供了和 Java 一样的流程控制语句，如下

- if
- switch
- for
- while
- dowhile

3.6.1 if 语句

```
1 var count = 3;
2 if (count == 3) {
3     alert(count);
4 }
```

3.6.2 switch 语句

```
1 var num = 3;
2 switch (num) {
3     case 1:
4         alert("星期一");
5         break;
6     case 2:
7         alert("星期二");
8         break;
9     case 3:
10        alert("星期三");
11        break;
12    case 4:
13        alert("星期四");
14        break;
15    case 5:
16        alert("星期五");
17        break;
18    case 6:
19        alert("星期六");
20        break;
21    case 7:
22        alert("星期日");
23        break;
24    default:
25        alert("输入的星期有误");
26        break;
27 }
```

3.6.3 for 循环语句

```
1 var sum = 0;
2 for (let i = 1; i <= 100; i++) { //建议for循环小括号中定义的变量使用let
3     sum += i;
4 }
5 alert(sum);
```

3.6.4 while 循环语句

```
1 var sum = 0;
2 var i = 1;
3 while (i <= 100) {
4     sum += i;
5     i++;
6 }
7 alert(sum);
```

3.6.5 dowhile 循环语句

```
1 var sum = 0;
2 var i = 1;
3 do {
4     sum += i;
5     i++;
6 }
7 while (i <= 100);
8 alert(sum);
```

3.7 函数

函数（就是Java中的方法）是被设计为执行特定任务的代码块；JavaScript 函数通过 `function` 关键词进行定义。

3.7.1 定义格式

函数定义格式有两种：

- 方式1

```
1 function 函数名(参数1,参数2...){
2     要执行的代码
3 }
```

- 方式2

```
1 var 函数名 = function (参数列表){
2     要执行的代码
3 }
```

注意：

- 形式参数不需要类型。因为JavaScript是弱类型语言

```
1 function add(a, b){
2     return a + b;
3 }
```

上述函数的参数 `a` 和 `b` 不需要定义数据类型，因为在每个参数前加上 `var` 也没有任何意义。

- 返回值也不需要定义类型，可以在函数内部直接使用`return`返回即可

3.7.2 函数调用

函数调用函数：

```
1 函数名称(实际参数列表);
```

eg:

```
1 let result = add(10,20);
```

注意：

- JS中，函数调用可以传递任意个数参数
- 例如 `let result = add(1,2,3);`

它是将数据 1 传递给了变量 `a`，将数据 2 传递给了变量 `b`，而数据 3 没有变量接收。

4, JavaScript常用对象

JavaScript 提供了很多对象供使用者来使用。这些对象总共分类三类

- 基本对象
-

JavaScript 对象参考手册

本参考手册描述每个对象的属性和方法，并提供实例。

- [Array](#)
 - [Boolean](#)
 - [Date](#)
 - [Math](#)
 - [Number](#)
 - [String](#)
 - [RegExp](#)
 - [Global](#)
-

- BOM 对象
-

Browser 对象参考手册

本参考手册描述每个对象的属性和方法，并提供实例。

- [Window](#)
 - [Navigator](#)
 - [Screen](#)
 - [History](#)
 - [Location](#)
-

- DOM对象

DOM 中的对象就比较多了，下图只是截取部分

HTML DOM 对象参考手册

本参考手册描述每个对象的属性和方法，并提供实例。

- [Document](#)
- [Anchor](#)
- [Area](#)

这小节我们先学习基本对象，而我们先学习 `Array` 数组对象和 `String` 字符串对象。

4.1 Array对象

JavaScript Array对象用于定义数组

4.1.1 定义格式

数组的定义格式有两种：

- 方式1

```
1 | var 变量名 = new Array(元素列表);
```

例如：

```
1 | var arr = new Array(1,2,3); //1,2,3 是存储在数组中的数据（元素）
```

- 方式2

```
1 | var 变量名 = [元素列表];
```

例如：

```
1 | var arr = [1,2,3]; //1,2,3 是存储在数组中的数据（元素）
```

注意：Java中的数组静态初始化使用的是{}定义，而JavaScript 中使用的是[] 定义

4.1.2 元素访问

访问数组中的元素和 Java 语言的一样，格式如下：

```
1 | arr[索引] = 值;
```

代码演示：

```
1 // 方式一
2 var arr = new Array(1,2,3);
3 // alert(arr);
4
5 // 方式二
6 var arr2 = [1,2,3];
7 //alert(arr2);
8
9 // 访问
10 arr2[0] = 10;
11 alert(arr2)
```

4.1.3 特点

JavaScript 中的数组相当于 Java 中集合。数组的长度是可以变化的，而 JavaScript 是弱类型，所以可以存储任意类型的数
据。

例如如下代码：

```
1 // 变长
2 var arr3 = [1,2,3];
3 arr3[10] = 10;
4 alert(arr3[10]); // 10
5 alert(arr3[9]); // undefined
```

上面代码在定义数组中给了三个元素，又给索引是 10 的位置添加了数据 10，那么 索引3 到 索引9 位置的元素是什么呢？我
们之前就介绍了，在 JavaScript 中没有赋值的话，默认就是 `undefined`。

如果给 `arr3` 数组添加字符串的数据，也是可以添加成功的

```
1 arr3[5] = "hello";
2 alert(arr3[5]); // hello
```

4.1.4 属性

Array 对象提供了很多属性，如下图是官方文档截取的

Array 对象属性

属性	描述
<code>constructor</code>	返回对创建此对象的数组函数的引用。
<code>length</code>	设置或返回数组中元素的数目。
<code>prototype</code>	使您有能力向对象添加属性和方法。

而我们只讲解 `length` 属性，该数组可以动态的获取数组的长度。而有这个属性，我们就可以遍历数组了

```
1 var arr = [1,2,3];
2 for (let i = 0; i < arr.length; i++) {
3     alert(arr[i]);
4 }
```

4.1.5 方法

Array 对象同样也提供了很多方法，如下图是官方文档截取的

Array 对象方法

方法	描述
<code>concat()</code>	连接两个或更多的数组，并返回结果。
<code>join()</code>	把数组的所有元素放入一个字符串。元素通过指定的分隔符进行分隔。
<code>pop()</code>	删除并返回数组的最后一个元素
<code>push()</code>	向数组的末尾添加一个或更多元素，并返回新的长度。
<code>reverse()</code>	颠倒数组中元素的顺序。
<code>shift()</code>	删除并返回数组的第一个元素
<code>slice()</code>	从某个已有的数组返回选定的元素
<code>sort()</code>	对数组的元素进行排序
<code>splice()</code>	删除元素，并向数组添加新元素。
<code>toSource()</code>	返回该对象的源代码。
<code>toString()</code>	把数组转换为字符串，并返回结果。
<code>toLocaleString()</code>	把数组转换为本地数组，并返回结果。
<code>unshift()</code>	向数组的开头添加一个或更多元素，并返回新的长度。
<code>valueOf()</code>	返回数组对象的原始值

而我们在课堂中只演示 `push` 函数和 `splice` 函数。

- `push` 函数：给数组添加元素，也就是在数组的末尾添加元素

参数表示要添加的元素

```
1 // push:添加方法
2 var arr5 = [1,2,3];
3 arr5.push(10);
4 alert(arr5); //数组的元素是 {1,2,3,10}
```

- `splice` 函数：删除元素

参数1：索引。表示从哪个索引位置删除

参数2：个数。表示删除几个元素

```
1 // splice:删除元素
2 var arr5 = [1,2,3];
3 arr5.splice(0,1); //从 0 索引位置开始删除，删除一个元素
4 alert(arr5); // {2,3}
```

4.2 String对象

String对象的创建方式有两种

- 方式1：

```
1 var 变量名 = new String(s);
```

- 方式2：

```
1 var 变量名 = "数组";
```

属性：

String对象提供了很多属性，下面给大家列举了一个属性 `length`，该属性是用于动态的获取字符串的长度

<code>length</code>	字符串的长度
---------------------	--------

函数：

String对象提供了很多函数（方法），下面给大家列举了两个方法。

charAt()

返回在指定位置的字符。

indexOf()

检索字符串。

String对象还有一个函数 `trim()`，该方法在文档中没有体现，但是所有的浏览器都支持；它是用来去掉字符串两端的空格。

代码演示：

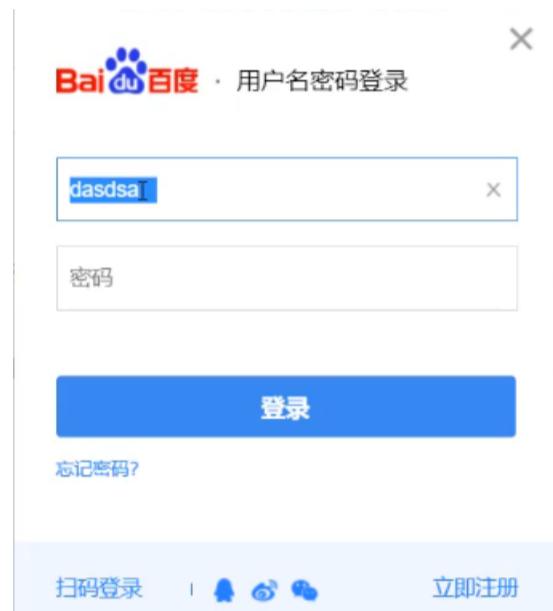
```
1 var str4 = ' abc ';
2 alert(1 + str4 + 1);
```

上面代码会输出内容 `1 abc 1`，很明显可以看到 abc 字符串左右两边是有空格的。接下来使用 `trim()` 函数

```
1 var str4 = ' abc ';
2 alert(1 + str4.trim() + 1);
```

输出的内容是 `1abc1`。这就是 `trim()` 函数的作用。

`trim()` 函数在以后开发中还是比较常用的，例如下图所示是登陆界面



用户在输入用户名和密码时，可能会习惯的输入一些空格，这样在我们后端程序中判断用户名和密码是否正确，结果肯定是失败。所以我们一般都会对用户输入的字符串数据进行去除前后空格的操作。

4.3 自定义对象

在 JavaScript 中自定义对象特别简单，下面就是自定义对象的格式：

```
1 var 对象名称 = {
2   属性名称1: 属性值1,
3   属性名称2: 属性值2,
4   ...
5   函数名称: function (形参列表){},
6   ...
7};
```

调用属性的格式：

```
1 对象名.属性名
```

调用函数的格式：

```
1 对象名.函数名()
```

接下来通过代码演示一下，让大家体验一下 JavaScript 中自定义对象

```

1 var person = {
2     name : "zhangsan",
3     age : 23,
4     eat: function () {
5         alert("干饭~");
6     }
7 };
8
9
10 alert(person.name); //zhangsan
11 alert(person.age); //23
12
13 person.eat(); //干饭~

```

5, BOM

BOM: Browser Object Model 浏览器对象模型。也就是 JavaScript 将浏览器的各个组成部分封装为对象。

我们要操作浏览器的各个组成部分就可以通过操作 BOM 中的对象来实现。比如：我现在想将浏览器地址栏的地址改为 `https://www.itheima.com` 就可以通过使用 BOM 中定义的 `Location` 对象的 `href` 属性，代码：`location.href = "https://itheima.com";`

BOM 中包含了如下对象：

- Window: 浏览器窗口对象
- Navigator: 浏览器对象
- Screen: 屏幕对象
- History: 历史记录对象
- Location: 地址栏对象

下图是 BOM 中的各个对象和浏览器的各个组成部分的对应关系



BOM 中的 `Navigator` 对象和 `screen` 对象基本不会使用，所以我们的课堂只对 `window`、`History`、`Location` 对象进行讲解。

5.1 Window对象

`window` 对象是 JavaScript 对浏览器的窗口进行封装的对象。

5.1.1 获取window对象

该对象不需要创建直接使用 `window`，其中 `window.` 可以省略。比如我们之前使用的 `alert()` 函数，其实质就是 `window` 对象的函数，在调用是可以写成如下两种

- 显式使用 `window` 对象调用

```
1 window.alert("abc");
```

- 隐式调用

```
1 | alert("abc")
```

5.1.2 window对象属性

window 对象提供了用于获取其他 BOM 组成对象的属性

history	对 History 对象的只读引用。请参见 History 对象 。	4	1	9
location	用于窗口或框架的 Location 对象。请参见 Location 对象 。	4	1	9
Navigator	对 Navigator 对象的只读引用。请参见 Navigator 对象 。	4	1	9
Screen	对 Screen 对象的只读引用。请参见 Screen 对象 。	4	1	9

也就是说，我们想使用 Location 对象的话，就可以使用 window 对象获取；写成 window.location，而 window. 可以省略，简化写成 location 来获取 Location 对象。

5.1.3 window对象函数

window 对象提供了很多函数供我们使用，而很多都不常用；下面给大家列举了一些比较常用的函数

alert()	显示带有一段消息和一个确认按钮的警告框。	4	1	9
confirm()	显示带有一段消息以及确认按钮和取消按钮的对话框。	4	1	9
setInterval()	按照指定的周期（以毫秒计）来调用函数或计算表达式。	4	1	9
setTimeout()	在指定的毫秒数后调用函数或计算表达式。	4	1	9

`setTimeout(function,毫秒值)` : 在一定的时间间隔后执行一个function，只执行一次
`setInterval(function,毫秒值)` : 在一定的时间间隔后执行一个function，循环执行

confirm代码演示：

```
1 // confirm(), 点击确定按钮, 返回true, 点击取消按钮, 返回false
2 var flag = confirm("确认删除? ");
3
4 alert(flag);
```

下图是 confirm() 函数的效果。当我们点击 确定 按钮，flag 变量值记录的就是 true；当我们点击 取消 按钮，flag 变量值记录的就是 false。



而以后我们在页面删除数据时候如下图每一条数据后都有 删除 按钮，有可能是用户的一些误操作，所以对于删除操作需要用户进行再次确认，此时就需要用到 confirm() 函数。

课程标题	课程类型	课程类别	学员总数	绑定课程数量	单次积分	创建时间	操作
每日诵读	在线看图说话	A类	12500	5	5	2020-2-24	课程详情 编辑 删除
万物皆诗公益在线	在线听故事	B类	6000	8	8	2020-2-23	课程详情 编辑 删除
cs	在线诵读	A类	1000	6	6	2020-2-22	课程详情 编辑 删除
国文专栏课程	在线文博课	X类	2000	4	4	2020-2-21	课程详情 编辑 删除

定时器代码演示：

```
1 setTimeout(function () {
2   alert("hehe");
3 },3000);
```

当我们打开浏览器，3秒后才会弹框输出 hehe，并且只会弹出一次。

```
1 setInterval(function () {
2     alert("hehe");
3 },2000);
```

当我们打开浏览器，每隔2秒都会弹框输出 hehe。

5.1.4 案例

需求：每隔1秒，灯泡切换一次状态



需求说明：

有如下页面效果，实现定时进行开灯、关灯功能



初始页面环境

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <title>JavaScript演示</title>
6 </head>
7 <body>
8
9 <input type="button" onclick="on()" value="开灯">
10 
11 <input type="button" onclick="off()" value="关灯">
12
13 <script>
14     function on(){
15         document.getElementById('myImage').src='../imgs/on.gif';
16     }
17
18     function off(){
19         document.getElementById('myImage').src='../imgs/off.gif'
20     }
21
22 </script>
23 </body>
24 </html>
```

代码实现：

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <title>JavaScript演示</title>
6 </head>
```

```

7 <body>
8
9 <input type="button" onclick="on()" value="开灯">
10 
11 <input type="button" onclick="off()" value="关灯">
12
13 <script>
14     function on(){
15         document.getElementById('myImage').src='../imgs/on.gif';
16     }
17
18     function off(){
19         document.getElementById('myImage').src='../imgs/off.gif'
20     }
21
22     //定义一个变量，用来记录灯的状态，偶数是开灯状态，奇数是关灯状态
23     var x = 0;
24     //使用循环定时器
25     setInterval(function (){
26         if(x % 2 == 0){//表示是偶数，开灯状态，调用 on() 函数
27             on();
28         }else { //表示是奇数，关灯状态，调用 off() 函数
29             off();
30         }
31         x++; //改变变量的值
32     },1000);
33
34 </script>
35 </body>
36 </html>

```

5.2 History对象

History 对象是 JavaScript 对历史记录进行封装的对象。

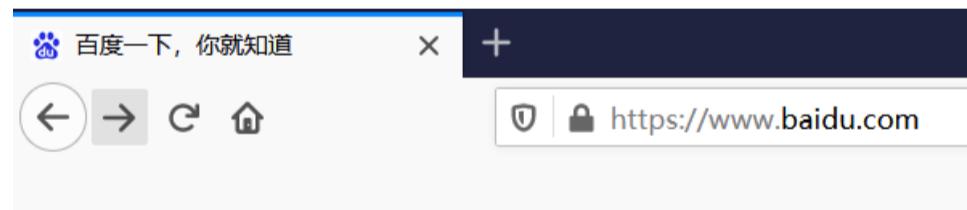
- History 对象的获取

使用 window.history 获取，其中window. 可以省略

- History 对象的函数

<u>back()</u>	加载 history 列表中的前一个 URL。
<u>forward()</u>	加载 history 列表中的下一个 URL。

这两个函数我们平时在访问其他的一些网站时经常使用对应的效果，如下图



当我们点击向左的箭头，就跳转到前一个访问的页面，这就是 `back()` 函数的作用；当我们点击向右的箭头，就跳转到下一个访问的页面，这就是 `forward()` 函数的作用。

5.3 Location对象



Location 对象是 JavaScript 对地址栏封装的对象。可以通过操作该对象，跳转到任意页面。

5.3.1 获取Location对象

使用 `window.location` 获取，其中window. 可以省略

```

1 window.location.方法();
2 location.方法();

```

5.3.2 Location对象属性

Location对象提供了很对属性。以后常用的只有一个属性 `href`

href	设置或返回完整的 URL。	4	1	9
----------------------	---------------	---	---	---

代码演示：

```
1 alert("要跳转了");
2 location.href = "https://www.baidu.com";
```

在浏览器首先会弹框显示 `要跳转了`，当我们点击了 `确定` 就会跳转到 百度 的首页。

5.3.3 案例

需求：3秒跳转到百度首页

分析：

1. 3秒跳转，由此可以确定需要使用到定时器，而只跳转一次，所以使用 `setTimeOut()`
2. 要进行页面跳转，所以需要用到 `location` 对象的 `href` 属性实现

代码实现：

```
1 document.write("3秒跳转到首页...");  
2 setTimeout(function (){  
3     location.href = "https://www.baidu.com"  
4 },3000);
```

6, DOM

6.1 概述

DOM: Document Object Model 文档对象模型。也就是 JavaScript 将 HTML 文档的各个组成部分封装为对象。

DOM 其实我们并不陌生，之前在学习 XML 就接触过，只不过 XML 文档中的标签需要我们写代码解析，而 HTML 文档是浏览器解析。封装的对象分为

- Document: 整个文档对象
- Element: 元素对象
- Attribute: 属性对象
- Text: 文本对象
- Comment: 注释对象

如下图，左边是 HTML 文档内容，右边是 DOM 树



作用：

JavaScript 通过 DOM，就能够对 HTML 进行操作了

- 改变 HTML 元素的内容
- 改变 HTML 元素的样式 (CSS)
- 对 HTML DOM 事件作出反应
- 添加和删除 HTML 元素

DOM相关概念：

DOM 是 W3C (万维网联盟) 定义了访问 HTML 和 XML 文档的标准。该标准被分为 3 个不同的部分：

1. 核心 DOM：针对任何结构化文档的标准模型。XML 和 HTML 通用的标准

- Document：整个文档对象
- Element：元素对象
- Attribute：属性对象
- Text：文本对象
- Comment：注释对象

2. XML DOM：针对 XML 文档的标准模型

3. HTML DOM：针对 HTML 文档的标准模型

该标准是在核心 DOM 基础上，对 HTML 中的每个标签都封装成了不同的对象

- 例如：`` 标签在浏览器加载到内存中时会被封装成 `Image` 对象，同时该对象也是 `Element` 对象。
- 例如：`<input type='button'>` 标签在浏览器加载到内存中时会被封装成 `Button` 对象，同时该对象也是 `Element` 对象。

6.2 获取 Element 对象

HTML 中的 `Element` 对象可以通过 `Document` 对象获取，而 `Document` 对象是通过 `window` 对象获取。

`Document` 对象中提供了以下获取 `Element` 元素对象的函数

- `getElementById()`：根据id属性值获取，返回单个Element对象
- `getElementsByName()`：根据标签名称获取，返回Element对象数组
- `getElementsByClassName()`：根据name属性值获取，返回Element对象数组
- `getElementsByClassName()`：根据class属性值获取，返回Element对象数组

代码演示：

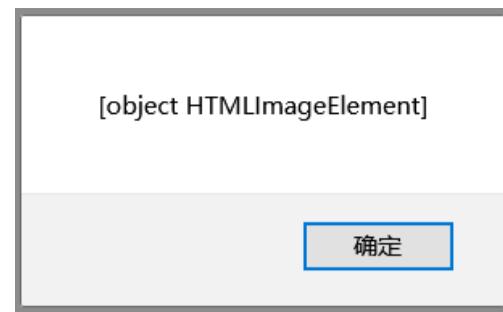
下面有提前准备好的页面：

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>Title</title>
6 </head>
7 <body>
8    <br>
9
10  <div class="cls">传智教育</div> <br>
11  <div class="cls">黑马程序员</div> <br>
12
13  <input type="checkbox" name="hobby"> 电影
14  <input type="checkbox" name="hobby"> 旅游
15  <input type="checkbox" name="hobby"> 游戏
16  <br>
17  <script>
18    //在此处书写js代码
19  </script>
20 </body>
21 </html>
```

1. 根据 `id` 属性值获取上面的 `img` 元素对象，返回单个对象

```
1 var img = document.getElementById("light");
2 alert(img);
```

结果如下：



从弹框输出的内容，也可以看出是一个图片元素对象。

2. 根据标签名称获取所有的 `div` 元素对象

```
1 var divs = document.getElementsByTagName("div");// 返回一个数组，数组中存储的是 div 元素对象
2 // alert(divs.length); //输出 数组的长度
3 //遍历数组
4 for (let i = 0; i < divs.length; i++) {
5     alert(divs[i]);
6 }
```

3. 获取所有的满足 `name = 'hobby'` 条件的元素对象

```
1 //3. getElementsByTagName: 根据name属性值获取，返回Element对象数组
2 var hobbys = document.getElementsByTagName("hobby");
3 for (let i = 0; i < hobbys.length; i++) {
4     alert(hobbys[i]);
5 }
```

4. 获取所有的满足 `class='cls'` 条件的元素对象

```
1 //4. getElementsByClassName: 根据class属性值获取，返回Element对象数组
2 var cls = document.getElementsByClassName("cls");
3 for (let i = 0; i < cls.length; i++) {
4     alert(cls[i]);
5 }
```

6.3 HTML Element对象使用

HTML 中的 `Element` 元素对象有很多，不可能全部记住，以后是根据具体的需求查阅文档使用。

下面我们通过具体的案例给大家演示文档的查询和对象的使用；下面提前给大家准备好的页面

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <title>Title</title>
6 </head>
7 <body>
8      <br>
9
10    <div class="cls">传智教育</div> <br>
11    <div class="cls">黑马程序员</div> <br>
12
13    <input type="checkbox" name="hobby"> 电影
14    <input type="checkbox" name="hobby"> 旅游
15    <input type="checkbox" name="hobby"> 游戏
16    <br>
17    <script>
18        //在此处写js低吗
19    </script>
20 </body>
21 </html>
```

需求：

1. 点亮灯泡

此案例由于需要改变 `img` 标签的图片，所以我们查询文档，下图是查看文档的流程：



代码实现：

```
1 //1, 根据 id='light' 获取 img 元素对象
2 var img = document.getElementById("light");
3 //2, 修改 img 对象的 src 属性来改变图片
4 img.src = "../imgs/on.gif";
```

2. 将所有的 `div` 标签的内容替换为 呵呵

```
1 //1, 获取所有的 div 元素对象
2 var divs = document.getElementsByTagName("div");
3 /*
4     style: 设置元素css样式
5     innerHTML: 设置元素内容
6 */
7 //2, 遍历数组, 获取到每一个 div 元素对象, 并修改元素内容
8 for (let i = 0; i < divs.length; i++) {
9     //divs[i].style.color = 'red';
10    divs[i].innerHTML = "呵呵";
11 }
```

3. 使所有的复选框呈现被选中的状态

此案例我们需要看 复选框 元素对象有什么属性或者函数是来操作 复选框的选中状态。下图是文档的查看

HTML DOM 对象参考手册

本参考手册描述每个对象的属性和方法，并提供实例。

- [Document](#)
- [Anchor](#)
- [Area](#)
- [Base](#)
- [Body](#)
- [Button](#)
- [Canvas](#)
- [Event](#)
- [Form](#)
- [Frame](#)
- [Frameset](#)
- [IFrame](#)
- [Image](#)
- [Input Button](#)
- [Input Checkbox](#) ①
- [Input File](#)
- [Input Hidden](#)
- [Input Password](#)

Checkbox 对象的属性

属性	描述
accessKey	设置或返回访问 checkbox 的快捷键。
alt ②	设置或返回不支持 checkbox 时显示的替代文本。
checked	设置或返回 checked 属性是否被选中。
defaultChecked	返回 checked 属性的默认值。
disabled	设置或返回 checkbox 是否应被禁用。
form	返回对包含 checkbox 的表单的引用。
id	设置或返回 checkbox 的 id。

代码实现：

```
1 //1, 获取所有的 复选框 元素对象
2 var hobbys = document.getElementsByName("hobby");
3 //2, 遍历数组, 通过将 复选框 元素对象的 checked 属性值设置为 true 来改变复选框的选中状态
4 for (let i = 0; i < hobbys.length; i++) {
5     hobbys[i].checked = true;
6 }
```

7. 事件监听

要想知道什么是事件监听，首先先聊聊什么是事件？

HTML 事件是发生在 HTML 元素上的“事情”。比如：页面上的 按钮被点击、鼠标移动到元素之上、按下键盘按键 等都是事件。

事件监听是 JavaScript 可以在事件被侦测到时 执行一段逻辑代码。例如下图当我们点击 开灯 按钮，就需要通过 js 代码实现替换图片



再比如下图输入框，当我们输入了用户名 `光标离开` 输入框，就需要通过 js 代码对输入的内容进行校验，没通过校验就在输入框后提示 `用户名格式有误！`

用户名： **用户名格式有误！**

7.1 事件绑定

JavaScript 提供了两种事件绑定方式：

- 方式一：通过 HTML 标签中的事件属性进行绑定

如下面代码，有一个按钮元素，我们是在该标签上定义 `事件属性`，在事件属性中绑定函数。`onclick` 就是 `单击事件` 的事件属性。`onclick='on ()'` 表示该点击事件绑定了一个名为 `on()` 的函数

```
1 <input type="button" onclick='on()>
```

下面是点击事件绑定的 `on()` 函数

```
1 function on(){
2     alert("我被点了");
3 }
```

- 方式二：通过 DOM 元素属性绑定

如下面代码是按钮标签，在该标签上我们并没有使用 `事件属性`，绑定事件的操作需要在 js 代码中实现

```
1 <input type="button" id="btn">
```

下面 js 代码是获取了 `id='btn'` 的元素对象，然后将 `onclick` 作为该对象的属性，并且绑定匿名函数。该函数是在事件触发后自动执行

```
1 document.getElementById("btn").onclick = function (){
2     alert("我被点了");
3 }
```

代码演示：

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <title>Title</title>
6 </head>
7 <body>
8     <!-- 方式1：在下面input标签上添加 onclick 属性，并绑定 on() 函数--&gt;
9     &lt;input type="button" value="点我" onclick="on()&gt; &lt;br&gt;
10    &lt;input type="button" value="再点我" id="btn"&gt;
11
12    &lt;script&gt;
13        function on(){
14            alert("我被点了");
15        }
16        // 方式2：获取 id="btn" 元素对象，通过调用 onclick 属性 绑定点击事件
17        document.getElementById("btn").onclick = function (){
18            alert("我被点了");
19        }
</pre>
```

```
20     </script>
21 </body>
22 </html>
```

7.2 常见事件

上面案例中使用到了 `onclick` 事件属性，那都有哪些事件属性供我们使用呢？下面就给大家列举一些比较常用的事件属性

事件属性名	说明
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
onmouseover	鼠标被移到某元素之上
onmouseout	鼠标从某元素移开

- `onfocus` 获得焦点事件。

如下图，当点击了输入框后，输入框就获得了焦点。而下图示例是当获取焦点后会更改输入框的背景颜色。

请输入您的姓名：

当输入字段获得焦点时，会触发一个更改背景颜色的函数。

- `onblur` 失去焦点事件。

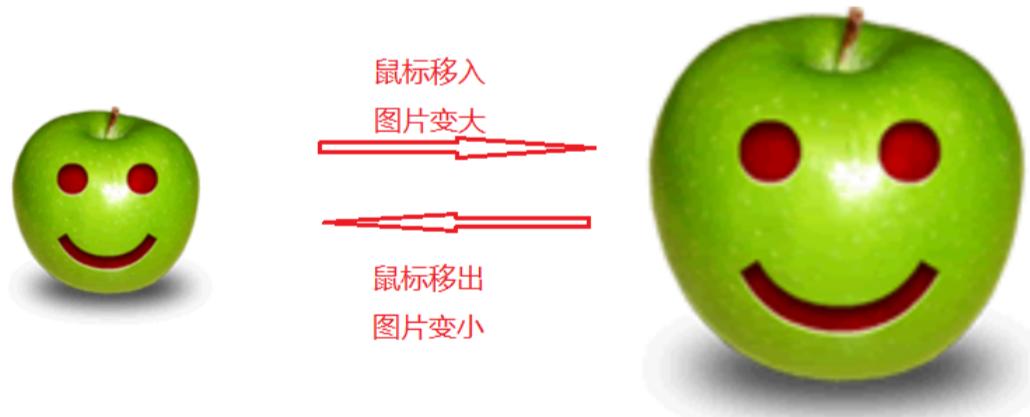
如下图，当点击了输入框后，输入框就获得了焦点；再点击页面其他位置，那输入框就失去了焦点了。下图示例是将输入的文本转换为大写。

请输入您的姓名：

当您离开输入字段时，会触发一个将输入文本转换为大写的函数。

- `onmouseout` 鼠标移出事件。
- `onmouseover` 鼠标移入事件。

如下图，当鼠标移到到 苹果 图片上时，苹果图片变大；当鼠标移出 苹果图片时，苹果图片变小。



- `onsubmit` 表单提交事件

如下是带有表单的页面

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>Title</title>
6  </head>
7  <body>
8      <form id="register" action="#" >
9          <input type="text" name="username" />
10         <input type="submit" value="提交">
11     </form>
```

```
12 <script>
13
14 </script>
15 </body>
16 </html>
```

如上代码的表单，当我们点击 提交 按钮后，表单就会提交，此处默认使用的是 GET 提交方式，会将提交的数据拼接到 URL 后。现需要通过 js 代码实现阻止表单提交的功能，js 代码实现如下：

1. 获取 form 表单元素对象。
2. 给 form 表单元素对象绑定 onsubmit 事件，并绑定匿名函数。
3. 该匿名函数如果返回的是true，提交表单；如果返回的是false，阻止表单提交。

```
1 document.getElementById("register").onsubmit = function () {
2     //onsubmit 返回true，则表单会被提交，返回false，则表单不提交
3     return true;
4 }
```

8. 表单验证案例

8.1 需求



有如下注册页面，对表单进行校验，如果输入的用户名、密码、手机号符合规则，则允许提交；如果不符，则不允许提交。

完成以下需求：

1. 当输入框失去焦点时，验证输入内容是否符合要求
2. 当点击注册按钮时，判断所有输入框的内容是否都符合要求，如果不符则阻止表单提交

8.2 环境准备

下面是初始页面

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <title>欢迎注册</title>
6     <link href="../css/register.css" rel="stylesheet">
7 </head>
8 <body>
9     <div class="form-div">
10        <div class="reg-content">
11            <h1>欢迎注册</h1>
12            <span>已有帐号? </span> <a href="#">登录</a>
13        </div>
14        <form id="reg-form" action="#" method="get">
15            <table>
16                <tr>
```

```

17         <td>用户名</td>
18         <td class="inputs">
19             <input name="username" type="text" id="username">
20             <br>
21             <span id="username_err" class="err_msg" style="display: none">用户名不太
受欢迎</span>
22         </td>
23     </tr>
24
25     <tr>
26         <td>密码</td>
27         <td class="inputs">
28             <input name="password" type="password" id="password">
29             <br>
30             <span id="password_err" class="err_msg" style="display: none">密码格式有
误</span>
31         </td>
32     </tr>
33
34     <tr>
35         <td>手机号</td>
36         <td class="inputs"><input name="tel" type="text" id="tel">
37             <br>
38             <span id="tel_err" class="err_msg" style="display: none">手机号格式有误
</span>
39         </td>
40     </tr>
41 </table>
42 <div class="buttons">
43     <input value="注 册" type="submit" id="reg_btn">
44 </div>
45 <br class="clear">
46 </form>
47
48 </div>
49
50
51 <script>
52
53 </script>
54 </body>
55 </html>

```

8.3 验证输入框

此小节完成如下功能：

- 校验用户名。当用户名输入框失去焦点时，判断输入的内容是否符合 `长度是 6-12 位 规则`，不符合使 `id='username_err'` 的 `span` 标签显示出来，给出用户提示。
- 校验密码。当密码输入框失去焦点时，判断输入的内容是否符合 `长度是 6-12 位 规则`，不符合使 `id='password_err'` 的 `span` 标签显示出来，给出用户提示。
- 校验手机号。当手机号输入框失去焦点时，判断输入的内容是否符合 `长度是 11 位 规则`，不符合使 `id='tel_err'` 的 `span` 标签显示出来，给出用户提示。

代码如下：

```

1 //1. 验证用户名是否符合规则
2 //1.1 获取用户名的输入框
3 var usernameInput = document.getElementById("username");
4
5 //1.2 绑定onblur事件 失去焦点
6 usernameInput.onblur = function () {
7     //1.3 获取用户输入的用户名
8     var username = usernameInput.value.trim();
9

```

```

10 //1.4 判断用户名是否符合规则: 长度 6~12
11 if (username.length >= 6 && username.length <= 12) {
12     //符合规则
13     document.getElementById("username_err").style.display = 'none';
14 } else {
15     //不合符规则
16     document.getElementById("username_err").style.display = '';
17 }
18 }

19 //1. 验证密码是否符合规则
20 //1.1 获取密码的输入框
21 var passwordInput = document.getElementById("password");
22
23 //1.2 绑定onblur事件 失去焦点
24 passwordInput.onblur = function() {
25     //1.3 获取用户输入的密码
26     var password = passwordInput.value.trim();
27
28     //1.4 判断密码是否符合规则: 长度 6~12
29     if (password.length >= 6 && password.length <= 12) {
30         //符合规则
31         document.getElementById("password_err").style.display = 'none';
32     } else {
33         //不合符规则
34         document.getElementById("password_err").style.display = '';
35     }
36 }
37 }

38 //1. 验证手机号是否符合规则
39 //1.1 获取手机号的输入框
40 var telInput = document.getElementById("tel");
41
42 //1.2 绑定onblur事件 失去焦点
43 telInput.onblur = function() {
44     //1.3 获取用户输入的手机号
45     var tel = telInput.value.trim();
46
47     //1.4 判断手机号是否符合规则: 长度 11
48     if (tel.length == 11) {
49         //符合规则
50         document.getElementById("tel_err").style.display = 'none';
51     } else {
52         //不合符规则
53         document.getElementById("tel_err").style.display = '';
54     }
55 }
56 }

```

8.3 验证表单

当用户点击 `注册` 按钮时，需要同时对输入的 `用户名`、`密码`、`手机号`，如果都符合规则，则提交表单；如果有一个不符合规则，则不允许提交表单。实现该功能需要获取表单元素对象，并绑定 `onsubmit` 事件

```

1 //1. 获取表单对象
2 var regForm = document.getElementById("reg-form");
3
4 //2. 绑定onsubmit 事件
5 regForm.onsubmit = function () {
6
7 }

```

`onsubmit` 事件绑定的函数需要对输入的 `用户名`、`密码`、`手机号` 进行校验，这些校验我们之前都已经实现过了，这里我们还需要再校验一次吗？不需要，只需要对之前校验的代码进行改造，把每个校验的代码专门抽象到有名字的函数中，方便调用；并且每个函数都要返回结果来决定是提交表单还是阻止表单提交，代码如下：

```
1 //1. 验证用户名是否符合规则
2 //1.1 获取用户名的输入框
3 var usernameInput = document.getElementById("username");
4
5 //1.2 绑定onblur事件 失去焦点
6 usernameInput.onblur = checkUsername;
7
8 function checkUsername() {
9     //1.3 获取用户输入的用户名
10    var username = usernameInput.value.trim();
11
12    //1.4 判断用户名是否符合规则: 长度 6~12
13    var flag = username.length >= 6 && username.length <= 12;
14    if (flag) {
15        //符合规则
16        document.getElementById("username_err").style.display = 'none';
17    } else {
18        //不合符规则
19        document.getElementById("username_err").style.display = '';
20    }
21    return flag;
22}
23
24 //1. 验证密码是否符合规则
25 //1.1 获取密码的输入框
26 var passwordInput = document.getElementById("password");
27
28 //1.2 绑定onblur事件 失去焦点
29 passwordInput.onblur = checkPassword;
30
31 function checkPassword() {
32     //1.3 获取用户输入的密码
33     var password = passwordInput.value.trim();
34
35     //1.4 判断密码是否符合规则: 长度 6~12
36     var flag = password.length >= 6 && password.length <= 12;
37     if (flag) {
38         //符合规则
39         document.getElementById("password_err").style.display = 'none';
40     } else {
41         //不合符规则
42         document.getElementById("password_err").style.display = '';
43     }
44     return flag;
45}
46
47 //1. 验证手机号是否符合规则
48 //1.1 获取手机号的输入框
49 var telInput = document.getElementById("tel");
50
51 //1.2 绑定onblur事件 失去焦点
52 telInput.onblur = checkTel;
53
54 function checkTel() {
55     //1.3 获取用户输入的手机号
56     var tel = telInput.value.trim();
57
58     //1.4 判断手机号是否符合规则: 长度 11
59     var flag = tel.length == 11;
60     if (flag) {
61         //符合规则
62         document.getElementById("tel_err").style.display = 'none';
63     } else {
64         //不合符规则
65         document.getElementById("tel_err").style.display = '';
66     }
67 }
```

```
66      }
67      return flag;
68 }
```

而 `onsubmit` 绑定的函数需要调用 `checkUsername()` 函数、`checkPassword()` 函数、`checkTel()` 函数。

```
1 //1. 获取表单对象
2 var regForm = document.getElementById("reg-form");
3
4 //2. 绑定onsubmit 事件
5 regForm.onsubmit = function () {
6     //挨个判断每一个表单项是否都符合要求，如果有任何一个不符，则返回false
7
8     var flag = checkUsername() && checkPassword() && checkTel();
9
10    return flag;
11 }
```

9, RegExp对象

RegExp 是正则对象。正则对象是判断指定字符串是否符合规则。

如下图是百度贴吧中的帖子



我们可以通过爬虫技术去爬取该页面源代码，然后获取页面中所有的邮箱，后期我们可以给这些邮箱地址发送推广的邮件。那么问题来了，如何才能知道页面内容中哪些事邮箱地址呢？这里就可以使用正则表达式来匹配邮箱。

在 js 中对正则表达式封装的对象就是正则对象。

9.1 正则对象使用

9.1.1 创建对象

正则对象有两种创建方式：

- 直接量方式：注意不要加引号

```
1 | var reg = /正则表达式/;
```

- 创建 RegExp 对象

```
1 | var reg = new RegExp("正则表达式");
```

9.1.2 函数

`test(str)`：判断指定字符串是否符合规则，返回 true 或 false

9.2 正则表达式

从上面创建正则对象的格式中可以看出不管哪种方式都需要正则表达式，那么什么是正则表达式呢？

正则表达式定义了字符串组成的规则。也就是判断指定的字符串是否符合指定的规则，如果符合返回true，如果不符返回false。

正则表达式是和语言无关的。很多语言都支持正则表达式，Java语言也支持，只不过正则表达式在不同的语言中的使用方式不同，js中需要使用正则对象来使用正则表达式。

正则表达式常用的规则如下：

- ^: 表示开始
- \$: 表示结束
- []: 代表某个范围内的单个字符，比如：[0-9] 单个数字字符
- .: 代表任意单个字符，除了换行和行结束符
- \w: 代表单词字符：字母、数字、下划线()，相当于 [A-Za-z0-9]
- \d: 代表数字字符：相当于 [0-9]

量词：

- +: 至少一个
- *: 零个或多个
- ?: 零个或一个
- {x}: x个
- {m,}: 至少m个
- {m,n}: 至少m个，最多n个

代码演示：

```
1 // 规则：单词字符，6~12
2 //1, 创建正则对象，对正则表达式进行封装
3 var reg = /^[^\w{6,12}}$/;
4
5 var str = "abcccc";
6 //2, 判断 str 字符串是否符合 reg 封装的正则表达式的规则
7 var flag = reg.test(str);
8 alert(flag);
```

9.3 改进表单校验案例

表单校验案例中的规则是我们进行一系列的判断来实现的，现在学习了正则对象后，就可以使用正则对象来改进这个案例。

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>欢迎注册</title>
6   <link href="../css/register.css" rel="stylesheet">
7 </head>
8 <body>
9
10 <div class="form-div">
11   <div class="reg-content">
12     <h1>欢迎注册</h1>
13     <span>已有帐号? </span> <a href="#">登录</a>
14   </div>
15   <form id="reg-form" action="#" method="get">
16
17     <table>
18
19       <tr>
20         <td>用户名</td>
21         <td class="inputs">
22           <input name="username" type="text" id="username">
23           <br>
```

```
24             <span id="username_err" class="err_msg" style="display: none">用户名不太受欢迎
25             迎</span>
26         </td>
27     </tr>
28
29     <tr>
30         <td>密码</td>
31         <td class="inputs">
32             <input name="password" type="password" id="password">
33             <br>
34             <span id="password_err" class="err_msg" style="display: none">密码格式有误
35         </td>
36     </tr>
37
38
39     <tr>
40         <td>手机号</td>
41         <td class="inputs"><input name="tel" type="text" id="tel">
42             <br>
43             <span id="tel_err" class="err_msg" style="display: none">手机号格式有误
44         </td>
45     </tr>
46
47 </table>
48
49 <div class="buttons">
50     <input value="注 册" type="submit" id="reg_btn">
51 </div>
52 <br class="clear">
53 </form>
54
55 </div>
56
57
58 <script>
59
60 //1. 验证用户名是否符合规则
61 //1.1 获取用户名的输入框
62 var usernameInput = document.getElementById("username");
63
64 //1.2 绑定onblur事件 失去焦点
65 usernameInput.onblur = checkusername;
66
67 function checkusername() {
68     //1.3 获取用户输入的用户名
69     var username = usernameInput.value.trim();
70
71     //1.4 判断用户名是否符合规则: 长度 6~12, 单词字符组成
72     var reg = /\w{6,12}$/;
73     var flag = reg.test(username);
74
75     //var flag = username.length >= 6 && username.length <= 12;
76     if (flag) {
77         //符合规则
78         document.getElementById("username_err").style.display = 'none';
79     } else {
80         //不合符规则
81         document.getElementById("username_err").style.display = '';
82     }
83     return flag;
84 }
85 }
```

```
86 //1. 验证密码是否符合规则
87 //1.1 获取密码的输入框
88 var passwordInput = document.getElementById("password");
89
90 //1.2 绑定onblur事件 失去焦点
91 passwordInput.onblur = checkPassword;
92
93 function checkPassword() {
94     //1.3 获取用户输入的密码
95     var password = passwordInput.value.trim();
96
97     //1.4 判断密码是否符合规则: 长度 6~12
98     var reg = /\w{6,12}$/;
99     var flag = reg.test(password);
100
101     //var flag = password.length >= 6 && password.length <= 12;
102     if (flag) {
103         //符合规则
104         document.getElementById("password_err").style.display = 'none';
105     } else {
106         //不合符规则
107         document.getElementById("password_err").style.display = '';
108     }
109     return flag;
110 }
111
112 //1. 验证手机号是否符合规则
113 //1.1 获取手机号的输入框
114 var telInput = document.getElementById("tel");
115
116 //1.2 绑定onblur事件 失去焦点
117 telInput.onblur = checkTel;
118
119 function checkTel() {
120     //1.3 获取用户输入的手机号
121     var tel = telInput.value.trim();
122
123     //1.4 判断手机号是否符合规则: 长度 11, 数字组成, 第一位是1
124     //var flag = tel.length == 11;
125     var reg = /^[1]\d{10}$/;
126     var flag = reg.test(tel);
127     if (flag) {
128         //符合规则
129         document.getElementById("tel_err").style.display = 'none';
130     } else {
131         //不合符规则
132         document.getElementById("tel_err").style.display = '';
133     }
134     return flag;
135 }
136
137 //1. 获取表单对象
138 var regForm = document.getElementById("reg-form");
139
140 //2. 绑定onsubmit 事件
141 regForm.onsubmit = function () {
142     //挨个判断每一个表单项是否都符合要求, 如果有一个不合符, 则返回false
143
144     var flag = checkUsername() && checkPassword() && checkTel();
145
146     return flag;
147 }
148 </script>
149 </body>
150 </html>
```


HTTP&Tomcat&Servlet

今日目标：

- 了解JavaWeb开发的技术栈
- 理解HTTP协议和HTTP请求与响应数据的格式
- 掌握Tomcat的使用
- 掌握在IDEA中使用Tomcat插件
- 理解Servlet的执行流程和生命周期
- 掌握Servlet的使用和相关配置

1. Web概述

1.1 Web和JavaWeb的概念

Web是全球广域网，也称为万维网(www)，能够通过浏览器访问的网站。

在我们日常的生活中，经常会使用浏览器去访问 百度、 京东、 传智官网 等这些网站，这些网站统称为 Web网站。如下就是通过浏览器访问传智官网的界面：



我们知道了什么是Web，那么JavaWeb又是什么呢？顾名思义JavaWeb就是用Java技术来解决相关web互联网领域的技术栈。

等学习完JavaWeb之后，同学们就可以使用Java语言开发我们上述所说的网站。而国内很多大型网站公司也是首选Java语言来解决web互联网相关的问题。那都有哪些公司的系统是使用Java语言的呢？



使用Java语言开发互联网系统是有很多技术栈需要大家了解，具体都有哪些呢？

1.2 JavaWeb技术栈

了解JavaWeb技术栈之前，有一个很重要的概念要介绍。

1.2.1 B/S架构

什么是B/S架构？

B/S 架构：Browser/Server，浏览器/服务器 架构模式，它的特点是，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器端。浏览器只需要请求服务器，获取Web资源，服务器把Web资源发送给浏览器即可。大家可以通过下面这张图来回想下我们平常的上网过程：



- 打开浏览器访问百度首页，输入要搜索的内容，点击回车或百度一下，就可以获取和搜索相关的内容
- 思考下搜索的内容并不在我们自己的点上，那么这些内容从何而来？答案很明显是从百度服务器返回给我们的
- 日常百度的小细节，逢年过节百度的logo会更换不同的图片，服务端发生变化，客户端不需做任务事情就能获取最新内容
- 所以说B/S架构的好处：易于维护升级：服务器端升级后，客户端无需任何部署就可以使用到新的版本。

了解了什么是B/S架构后，作为后台开发工程师的我们将来主要关注的是服务端的开发和维护工作。在服务端将来会放很多资源，都有哪些资源呢？

1.2.2 静态资源

- 静态资源主要包含HTML、CSS、JavaScript、图片等，主要负责页面的展示。
- 我们之前已经学过前端网页制作三剑客 (HTML+CSS+JavaScript)，使用这些技术我们就可以制作出效果比较丰富的网页，将来展现给用户。但是由于做出来的这些内容都是静态的，这就会导致所有的人看到的内容将是一模一样。
- 在日常上网的过程中，我们除了看到这些好看的页面以外，还会碰到很多动态内容，比如我们常见的百度登录效果：



百度一下

张三 登录以后在网页的右上角看到的是 张三，而 李四 登录以后看到的则是 李四。所以不同的用户访问相同的资源看到的内容大多数是不一样的，要想实现这样的效果，光靠静态资源是无法实现的。

1.2.3 动态资源

- 动态资源主要包含Servlet、JSP等，主要用来负责逻辑处理。
- 动态资源处理完逻辑后会把得到的结果交给静态资源来进行展示，动态资源和静态资源要结合一起使用。
- 动态资源虽然可以处理逻辑，但是当用户来登录百度的时候，就需要输入用户名和密码，这个时候我们就又需要解决的一个问题是，用户在注册的时候填入的用户名和密码、以及我们经常会访问到一些数据列表的内容展示(如下图所示)，这些数据都存储在哪里？我们需要的时候又是从哪里来取呢？

The screenshot shows a web-based administration interface for managing brands. The top navigation bar includes links for Product Management, Order Management, Logistics Management, Promotional Management, Article Management, Permission Management, and VIP Model. A search bar at the top right allows filtering by status, company name, and brand name. Below the search bar is a table titled 'Data List' containing six rows of brand information. Each row includes a checkbox, a sequence number (序号), a logo thumbnail (品牌LOGO), the brand name (品牌名称), the company name (企业名称), a sort order (排序), a current status switch (当前状态), and an operations column (操作) with three buttons: Delete, Edit, and View Details.

序号	品牌LOGO	品牌名称	企业名称	排序	当前状态	操作
010		三只松鼠	这里是企业名称	8	<input checked="" type="checkbox"/>	删除 编辑 查看详情
009		优农库	这里是企业名称	11	<input checked="" type="checkbox"/>	删除 编辑 查看详情
008		小米	这里是企业名称	9	<input checked="" type="checkbox"/>	删除 编辑 查看详情
007		闻迪斯	这里是企业名称	12	<input checked="" type="checkbox"/>	删除 编辑 查看详情
006		百草味	这里是企业名称	19	<input type="checkbox"/>	删除 编辑 查看详情

1.2.4 数据库

- 数据库主要负责存储数据。
- 整个Web的访问过程就如下图所示：



- (1) 浏览器发送一个请求到服务端，去请求所需要的相关资源；
- (2) 资源分为动态资源和静态资源，动态资源可以是使用Java代码按照Servlet和JSP的规范编写的内容；
- (3) 在Java代码可以进行业务处理也可以从数据库中读取数据；
- (4) 拿到数据后，把数据交给HTML页面进行展示，再结合CSS和JavaScript使展示效果更好；
- (5) 服务端将静态资源响应给浏览器；
- (6) 浏览器将这些资源进行解析；
- (7) 解析后将效果展示在浏览器，用户就可以看到最终的结果。

在整个Web的访问过程中，会设计到很多技术，这些技术有已经学习过的，也有还未涉及到的内容，都有哪些还没有涉及到呢？

1.2.5 HTTP协议

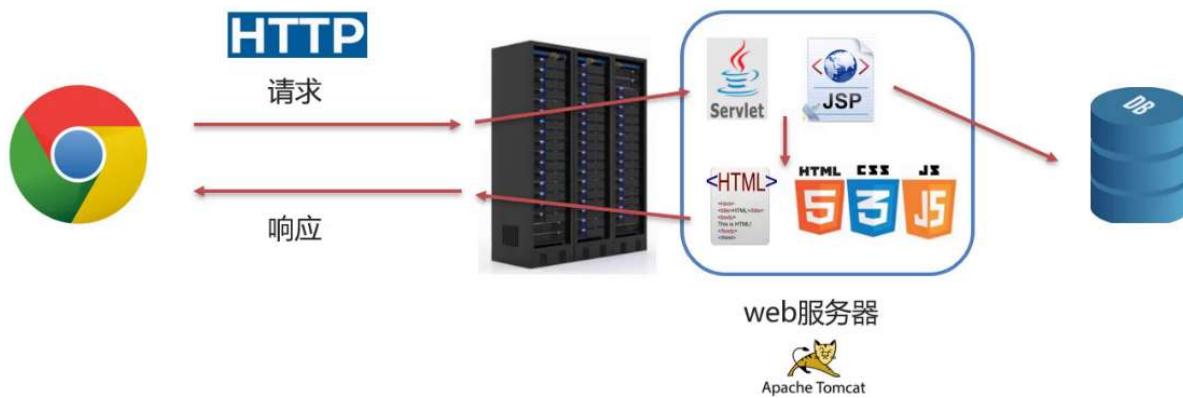
- HTTP协议：主要定义通信规则
- 浏览器发送请求给服务器，服务器响应数据给浏览器，这整个过程都需要遵守一定的规则，之前大家学习过TCP、UDP，这些都属于规则，这里我们需要使用的是HTTP协议，这也是一种规则。

1.2.6 Web服务器

- Web服务器：负责解析HTTP协议，解析请求数据，并发送响应数据
- 浏览器按照HTTP协议发送请求和数据，后台就需要一个Web服务器软件来根据HTTP协议解析请求和数据，然后把处理结果再按照HTTP协议发送给浏览器
- Web服务器软件有很多，我们课程中将学习的是目前最为常用的Tomcat服务器

到目前为止，关于JavaWeb中用到的技术栈我们就介绍完了，这里面就只有HTTP协议、Servlet、JSP以及Tomcat这些知识是没有学习过的，所以整个Web核心主要就是来学习这些技术。

1.3 Web核心课程安排



整个Web核心，我们总共有六天的学习内容，分别是：

- 第一天：HTTP、Tomcat、Servlet
- 第二天：Request(请求)、Response(响应)
- 第三天：JSP、会话技术(Cookie、Session)
- 第四天：Filter(过滤器)、Listener(监听器)
- 第五天：Ajax、Vue、ElementUI
- 第六天：综合案例

(1) Request是从客户端向服务端发出的请求对象，

(2) Response是从服务端响应给客户端的结果对象，

(3) JSP是动态网页技术，

(4) 会话技术是用来存储客户端和服务端交互所产生的数据，

(5) 过滤器是用来拦截客户端的请求，

(6) 监听器是用来监听特定事件，

(7) Ajax、Vue、ElementUI都是属于前端技术

这些技术都该如何来使用，我们后面会一个个进行详细的讲解。接下来我们来学习下HTTP、Tomcat和Servlet。

2, HTTP

2.1 简介

HTTP概念

HyperText Transfer Protocol，超文本传输协议，规定了浏览器和服务器之间数据传输的规则。

- 数据传输的规则指的是请求数据和响应数据需要按照指定的格式进行传输。
- 如果想知道具体的格式，可以打开浏览器，点击 F12 打开开发者工具，点击 Network 来查看某一次请求的请求数据和响应数据具体的格式内容，如下图所示：

注意：在浏览器中如果看不到上述内容，需要清除浏览器的浏览数据。chrome浏览器可以使用 **ctrl+shift+Del** 进行清除。

所以学习HTTP主要就是学习请求和响应数据的具体格式内容。

HTTP协议特点

HTTP协议有它自己的一些特点，分别是：

- 基于TCP协议：面向连接，安全

TCP是一种面向连接的(建立连接之前是需要经过三次握手)、可靠的、基于字节流的传输层通信协议，在数据传输方面更安全。

- 基于请求-响应模型的：一次请求对应一次响应

请求和响应是一一对应关系

- HTTP协议是无状态协议：对于事物处理没有记忆能力。每次请求-响应都是独立的

无状态指的是客户端发送HTTP请求给服务端之后，服务端根据请求响应数据，响应完后，不会记录任何信息。这种特性有优点也有缺点，

- 缺点：多次请求间不能共享数据
- 优点：速度快

请求之间无法共享数据会引发的问题，如：

- 京东购物，加入购物车 和 去购物车结算 是两次请求，
- HTTP协议的无状态特性，加入购物车请求响应结束后，并未记录加入购物车是何商品
- 发起去购物车结算的请求后，因为无法获取哪些商品加入了购物车，会导致此次请求无法正确展示数据

具体使用的时候，我们发现京东是可以正常展示数据的，原因是Java早已考虑到这个问题，并提出了使用会话技术(Cookie、Session)来解决这个问题。具体如何来做，我们后面会详细讲到。刚才提到HTTP协议是规定了请求和响应数据的格式，那具体的格式是什么呢？

2.2 请求数据格式

2.2.1 格式介绍

请求数据总共分为三部分内容，分别是**请求行、请求头、请求体**

```
GET / HTTP/1.1
Host: www.itcast.cn
Connection: keep-alive
Cache-Control: max-age=0 Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 Chrome/91.0.4472.106
...
```

- 请求行: HTTP请求中的第一行数据, 请求行包含三块内容, 分别是 GET[请求方式] /[请求URL路径] HTTP/1.1[HTTP协议及版本]

请求方式有七种,最常用的是GET和POST

- 请求头: 第二行开始, 格式为key: value形式

请求头中会包含若干个属性, 常见的HTTP请求头有:

- 1 Host: 表示请求的主机名
- 2 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Mozilla/5.0 (Windows NT ...) like Gecko;
- 3 Accept: 表示浏览器能接收的资源类型, 如text/*, image/*或者 */*表示所有;
- 4 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;
- 5 Accept-Encoding: 表示浏览器可以支持的压缩类型, 例如gzip, deflate等。

这些数据有什么用处?

举例说明:服务端可以根据请求头中的内容来获取客户端的相关信息, 有了这些信息服务端就可以处理不同的业务需求, 比如:

- 不同浏览器解析HTML和CSS标签的结果会有不一致, 所以就会导致相同的代码在不同的浏览器会出现不同的效果
 - 服务端根据客户端请求头中的数据获取到客户端的浏览器类型, 就可以根据不同的浏览器设置不同的代码来达到一致的效果
 - 这就是我们常说的浏览器兼容问题
- 请求体: POST请求的最后一部分, 存储请求参数

```
POST / HTTP/1.1
Host: www.itcast.cn
Connection: keep-alive
Cache-Control: max-age=0 Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 Chrome/91.0.4472.106
```

```
username=superbaby&password=123456
```

如上图红线框的内容就是请求体的内容, 请求体和请求头之间是有一个空行隔开。此时浏览器发送的是POST请求, 为什么不能使用GET呢?这时就需要回顾GET和POST两个请求之间的区别了:

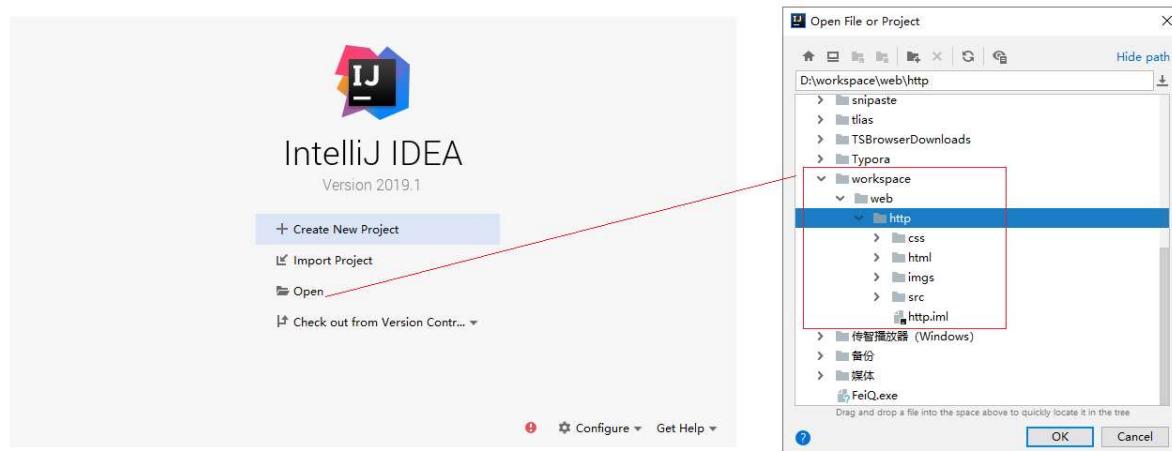
- GET请求请求参数在请求行中, 没有请求体, POST请求请求参数在请求体中
- GET请求请求参数大小有限制, POST没有

2.2.2 实例演示

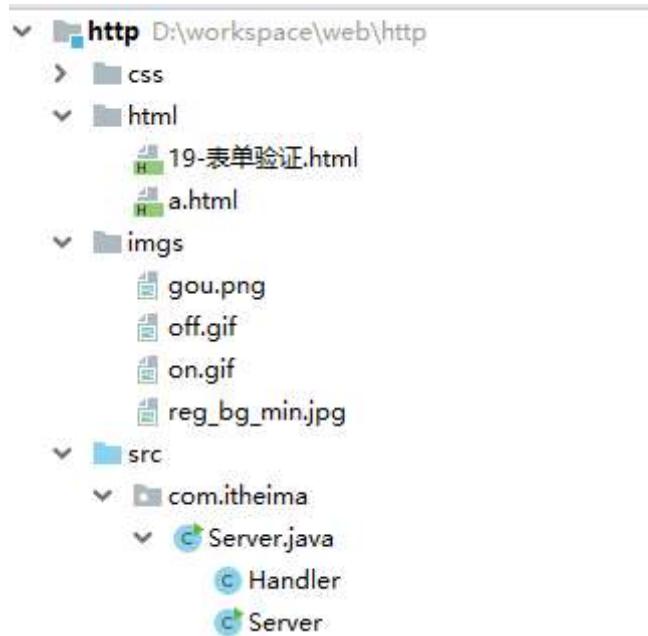
把 `代码\http` 拷贝到IDEA的工作目录中，比如 `D:\workspace\web` 目录，



使用IDEA打开



打开后，可以点击项目中的 `html\19-表单验证.html`，使用浏览器打开，通过修改页面中form表单的 `method` 属性来测试GET请求和POST请求的参数携带方式。



小结：

1. 请求数据中包含三部分内容，分别是请求行、请求头和请求体
2. POST请求数据在请求体中，GET请求数据在请求行上

2.3 响应数据格式

2.3.1 格式介绍

响应数据总共分为三部分内容，分别是**响应行**、**响应头**、**响应体**

```
HTTP/1.1 200 OK
Server: Tengine
Content-Type: text/html
Transfer-Encoding: chunked...

<html>
<head>
    <title></title>
</head>
<body></body>
</html>
```

- **响应行**: 响应数据的第一行, 响应行包含三块内容, 分别是 HTTP/1.1[HTTP协议及版本] 200[响应状态码] ok[状态码的描述]
- **响应头**: 第二行开始, 格式为key: value形式

响应头中会包含若干个属性, 常见的HTTP响应头有:

- 1 Content-Type: 表示该响应内容的类型, 例如text/html, image/jpeg;
- 2 Content-Length: 表示该响应内容的长度(字节数);
- 3 Content-Encoding: 表示该响应压缩算法, 例如gzip;
- 4 Cache-Control: 指示客户端应如何缓存, 例如max-age=300表示可以最多缓存300秒

- **响应体**: 最后一部分。存放响应数据

上图中...这部分内容就是响应体, 它和响应头之间有一个空行隔开。

2.3.2 响应状态码

参考: 资料/1.HTTP/《响应状态码.md》

关于响应状态码, 我们先主要认识三个状态码, 其余的等后期用到了再去掌握:

- 200 ok 客户端请求成功
- 404 Not Found 请求资源不存在
- 500 Internal Server Error 服务端发生不可预期的错误

2.3.3 自定义服务器

在前面我们导入到IDEA中的http项目中, 有一个Server.java类, 这里面就是自定义的一个服务器代码, 主要使用到的是 `ServerSocket` 和 `Socket`

```
1 package com.itheima;
2
3 import sun.misc.IOUtils;
4
5 import java.io.*;
```

```
6 import java.net.ServerSocket;
7 import java.net.Socket;
8 import java.nio.charset.StandardCharsets;
9 import java.nio.file.Files;
10 /*
11     自定义服务器
12 */
13 public class Server {
14     public static void main(String[] args) throws IOException {
15         ServerSocket ss = new ServerSocket(8080); // 监听指定端口
16         System.out.println("server is running...");
17         while (true){
18             Socket sock = ss.accept();
19             System.out.println("connected from " +
20             sock.getRemoteSocketAddress());
21             Thread t = new Handler(sock);
22             t.start();
23         }
24     }
25
26     class Handler extends Thread {
27         Socket sock;
28
29         public Handler(Socket sock) {
30             this.sock = sock;
31         }
32
33         public void run() {
34             try (InputStream input = this.sock.getInputStream()) {
35                 try (OutputStream output = this.sock.getOutputStream()) {
36                     handle(input, output);
37                 }
38             } catch (Exception e) {
39                 try {
40                     this.sock.close();
41                 } catch (IOException ioe) {
42                 }
43                 System.out.println("client disconnected.");
44             }
45         }
46
47         private void handle(InputStream input, OutputStream output) throws
48             IOException {
49             BufferedReader reader = new BufferedReader(new
50             InputStreamReader(input, StandardCharsets.UTF_8));
51             BufferedWriter writer = new BufferedWriter(new
52             OutputStreamWriter(output, StandardCharsets.UTF_8));
53             // 读取HTTP请求:
54             boolean requestok = false;
55             String first = reader.readLine();
56             if (first.startsWith("GET / HTTP/1.")) {
57                 requestok = true;
58             }
59             for (;;) {
60                 String header = reader.readLine();
61                 if (header.isEmpty()) { // 读取到空行时, HTTP Header读取完毕
62                     break;
63                 }
64             }
65             if (requestok) {
66                 writer.write("HTTP/1.1 200 OK\r\n");
67                 writer.write("Content-Type: text/html\r\n");
68                 writer.write("\r\n");
69                 writer.write("<h1>Hello, World!</h1>");
70             }
71             writer.flush();
72         }
73     }
74 }
```

```

60         }
61         System.out.println(header);
62     }
63     System.out.println(requestOk ? "Response OK" : "Response Error");
64     if (!requestOk) {
65         // 发送错误响应:
66         writer.write("HTTP/1.0 404 Not Found\r\n");
67         writer.write("Content-Length: 0\r\n");
68         writer.write("\r\n");
69         writer.flush();
70     } else {
71         // 发送成功响应:
72
73         //读取html文件, 转换为字符串
74         BufferedReader br = new BufferedReader(new
FileReader("http/html/a.html"));
75         StringBuilder data = new StringBuilder();
76         String line = null;
77         while ((line = br.readLine()) != null){
78             data.append(line);
79         }
80         br.close();
81         int length =
82             data.toString().getBytes(StandardCharsets.UTF_8).length;
83
84         writer.write("HTTP/1.1 200 OK\r\n");
85         writer.write("Connection: keep-alive\r\n");
86         writer.write("Content-Type: text/html\r\n");
87         writer.write("Content-Length: " + length + "\r\n");
88         writer.write("\r\n"); // 空行标识Header和Body的分隔
89         writer.write(data.toString());
90         writer.flush();
91     }
92 }

```

上面代码，大家不需要自己写，主要通过上述代码，只需要大家了解到服务器可以使用java完成编写，是可以接受页面发送的请求和响应数据给前端浏览器的，真正用到的Web服务器，我们不会自己写，都是使用目前比较流行的web服务器，比如Tomcat

小结

1. 响应数据中包含三部分内容，分别是响应行、响应头和响应体
2. 掌握200, 404, 500这三个响应状态码所代表含义，分布是成功、所访问资源不存在和服务的错误

3, Tomcat

3.1 简介

3.1.1 什么是Web服务器

Web服务器是一个应用程序（软件），对HTTP协议的操作进行封装，使得程序员不必直接对协议进行操作，让Web开发更加便捷。主要功能是“提供网上信息浏览服务”。



Web服务器是安装在服务器端的一款软件，将来我们把自己写的Web项目部署到Web Tomcat服务器软件中，当Web服务器软件启动后，部署在Web服务器软件中的页面就可以直接通过浏览器来访问了。

Web服务器软件使用步骤

- 准备静态资源
- 下载安装Web服务器软件
- 将静态资源部署到Web服务器上
- 启动Web服务器使用浏览器访问对应的资源

上述内容在演示的时候，使用的是Apache下的Tomcat软件，至于Tomcat软件如何使用，后面会详细的讲到。而对于Web服务器来说，实现的方案有很多，Tomcat只是其中的一种，而除了Tomcat以外，还有很多优秀的Web服务器，比如：



Tomcat就是一款软件，我们主要是以学习如何去使用为主。具体我们会从以下这些方向去学习：

1. 简介：初步认识下Tomcat
2. 基本使用：安装、卸载、启动、关闭、配置和项目部署，这些都是对Tomcat的基本操作
3. IDEA中如何创建Maven Web项目
4. IDEA中如何使用Tomcat，后面这两个都是我们以后开发经常会用到的方式

首先我们来认识下Tomcat。

Tomcat

Tomcat的相关概念：

- Tomcat是Apache软件基金会一个核心项目，是一个开源免费的轻量级Web服务器，支持Servlet/JSP少量JavaEE规范。
- 概念中提到了JavaEE规范，那什么又是JavaEE规范呢？

JavaEE: Java Enterprise Edition, Java企业版。指Java企业级开发的技术规范总和。包含13项技术规范：JDBC、JNDI、EJB、RMI、JSP、Servlet、XML、JMS、Java IDL、JTS、JTA、JavaMail、JAF。

- 因为Tomcat支持Servlet/JSP规范，所以Tomcat也被称为Web容器、Servlet容器。Servlet需要依赖Tomcat才能运行。
- Tomcat的官网：<https://tomcat.apache.org/> 从官网上可以下载对应的版本进行使用。

Tomcat的LOGO



小结

通过这一节的学习，我们需要掌握以下内容：

1. Web服务器的作用
 - 封装HTTP协议操作，简化开发
 - 可以将Web项目部署到服务器中，对外提供网上浏览服务
2. Tomcat是一个轻量级的Web服务器，支持Servlet/JSP少量JavaEE规范，也称为Web容器，Servlet容器。

3.2 基本使用

Tomcat总共分两部分学习，先来学习Tomcat的基本使用，包括Tomcat的下载、安装、卸载、启动和关闭。

3.2.1 下载

直接从官网下载

Binary Distributions 软件

- Core:
 - [zip \(pgp, sha512\)](#)
 - [tar.gz \(pgp, sha512\)](#)
 - [32-bit Windows zip \(pgp, sha512\)](#)
 - [64-bit Windows zip \(pgp, sha512\)](#)
 - [32-bit/64-bit Windows Service Installer \(pgp, sha512\)](#)
- Full documentation:
 - [tar.gz \(pgp, sha512\)](#)
- Deployer:
 - [zip \(pgp, sha512\)](#)
 - [tar.gz \(pgp, sha512\)](#)
- Embedded:
 - [tar.gz \(pgp, sha512\)](#)
 - [zip \(pgp, sha512\)](#)

Source Code Distributions 源码

- [tar.gz \(pgp, sha512\)](#)
- [zip \(pgp, sha512\)](#)

大家可以自行下载，也可以直接使用资料中已经下载好的资源，

Tomcat的软件程序 资料/2. Tomcat/apache-tomcat-8.5.68-windows-x64.zip

3.2.2 安装

Tomcat是绿色版,直接解压即可

- 在D盘的software目录下, 将 `apache-tomcat-8.5.68-windows-x64.zip` 进行解压缩, 会得到一个 `apache-tomcat-8.5.68` 的目录, Tomcat就已经安装成功。

注意, Tomcat在解压缩的时候, 解压所在的目录可以任意, 但最好解压到一个不包含中文和空格的目录, 因为后期在部署项目的时候, 如果路径有中文或者空格可能会导致程序部署失败。

- 打开 `apache-tomcat-8.5.68` 目录就能看到如下目录结构, 每个目录中包含的内容需要认识下,

	<code>bin</code>	可执行文件存放目录
	<code>conf</code>	配置文件存放目录
	<code>lib</code>	tomcat依赖的jar包
	<code>logs</code>	日志文件
	<code>temp</code>	临时文件
	<code>webapps</code>	应用发布目录
	<code>work</code>	工作目录

`bin`:目录下有两类文件, 一种是以 `.bat` 结尾的, 是Windows系统的可执行文件, 一种是以 `.sh` 结尾的, 是Linux系统的可执行文件。

`webapps`:就是以后项目部署的目录

到此, Tomcat的安装就已经完成。

3.2.3 卸载

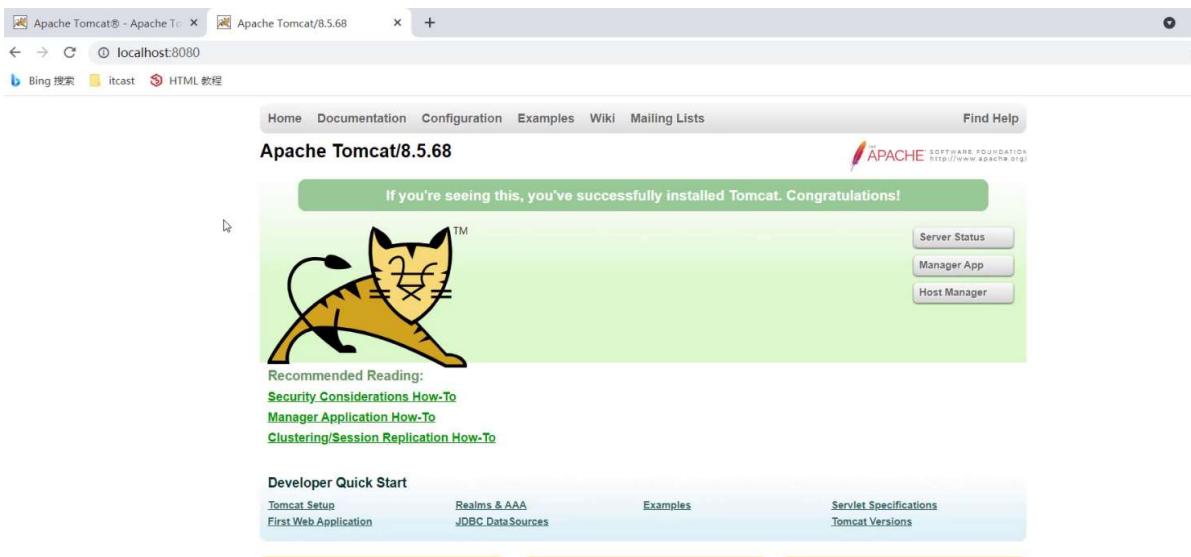
卸载比较简单, 可以直接删除目录即可

3.2.4 启动

双击: `bin\startup.bat`



启动后, 通过浏览器访问 `http://localhost:8080` 能看到Apache Tomcat的内容就说明Tomcat已经启动成功。



注意: 启动的过程中，控制台有中文乱码，需要修改conf/logging.properties

```
java.util.logging.ConsoleHandler.encoding = UTF-8 GBK
```

3.2.5 关闭

关闭有三种方式

- 直接x掉运行窗口: 强制关闭[不建议]
- bin\shutdown.bat: 正常关闭
- ctrl+c: 正常关闭

3.2.6 配置

修改端口

- Tomcat默认的端口号是8080，要想修改Tomcat启动的端口号，需要修改 conf/server.xml

```
<Connector port="8080" protocol="HTTP/1.1"
           connectionTimeout="20000"
           redirectPort="8443" />
```

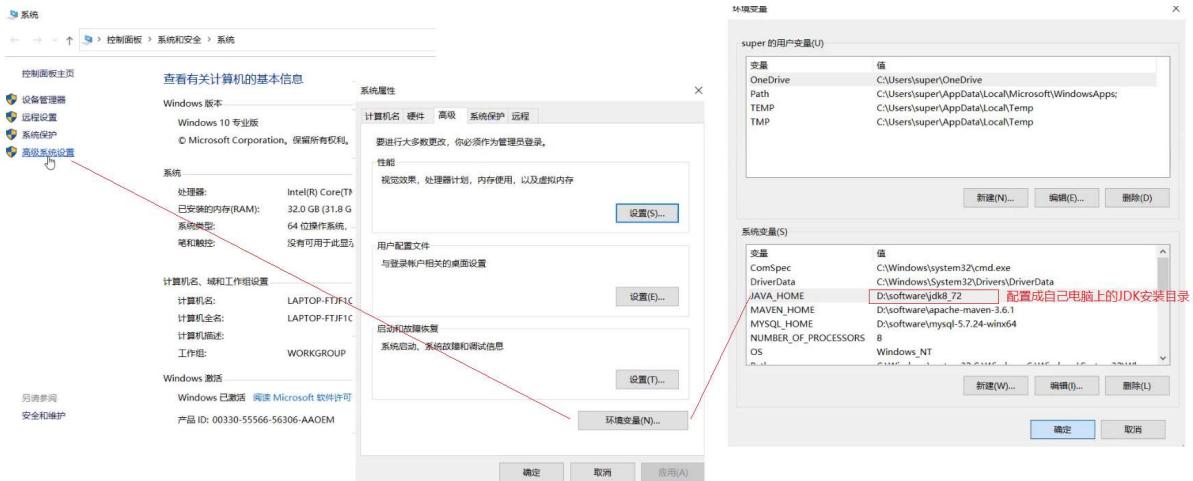
注: HTTP协议默认端口号为80，如果将Tomcat端口号改为80，则将来访问Tomcat时，将不用输入端口号。

启动时可能出现的错误

- Tomcat的端口号取值范围是0-65535之间任意未被占用的端口，如果设置的端口号被占用，启动的时候就会包如下的错误

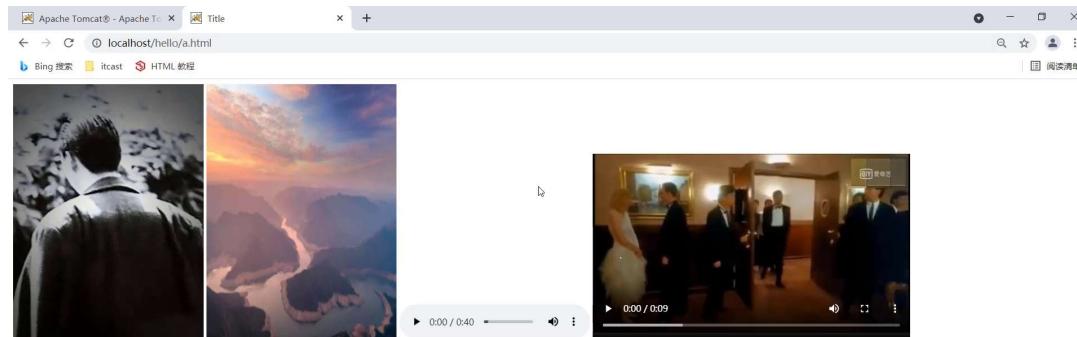
```
at org.apache.catalina.startup.Bootstrap.load(Bootstrap.java:84)
at org.apache.catalina.startup.Bootstrap.main(Bootstrap.java:46)
Caused by: java.net.BindException: Address already in use: bind
at sun.nio.ch.Net.bind0(Native Method)
at sun.nio.ch.Net.bind(Net.java:433)
at sun.nio.ch.Net.bind(Net.java:427)
```

- Tomcat启动的时候，启动窗口一闪而过: 需要检查JAVA_HOME环境变量是否正确配置



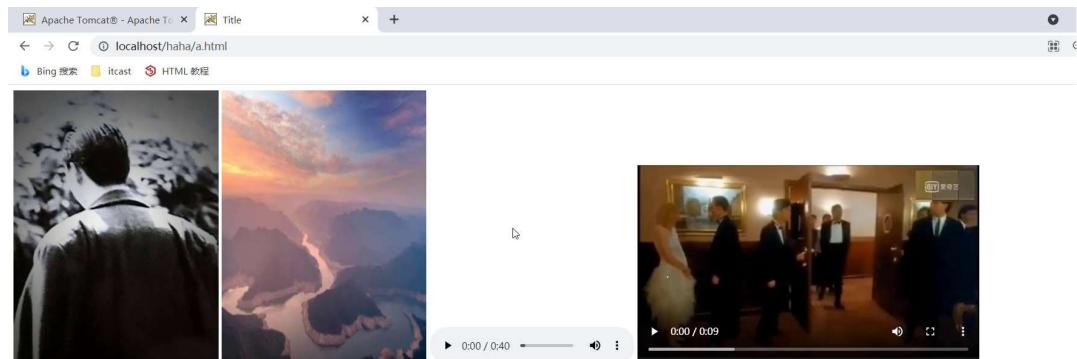
3.2.7 部署

- Tomcat部署项目：将项目放置到webapps目录下，即部署完成。
 - 将 资料/2. Tomcat/hello 目录拷贝到Tomcat的webapps目录下
 - 通过浏览器访问 `http://localhost/hello/a.html`，能看到下面的内容就说明项目已经部署成功。



但是呢随着项目的增大，项目中的资源也会越来越多，项目在拷贝的过程中也会越来越费时间，该如何解决呢？

- 一般JavaWeb项目会被打包称war包，然后将war包放到Webapps目录下，Tomcat会自动解压缩war文件
 - 将 资料/2. Tomcat/haha.war 目录拷贝到Tomcat的webapps目录下
 - Tomcat检测到war包后会自动完成解压缩，在webapps目录下就会多一个haha目录
 - 通过浏览器访问 `http://localhost/haha/a.html`，能看到下面的内容就说明项目已经部署成功。



至此，Tomcat的部署就已经完成了，至于如何获得项目对应的war包，后期我们会借助于IDEA工具来生成。

3.3 Maven创建Web项目

介绍完Tomcat的基本使用后，我们来学习在IDEA中如何创建Maven Web项目，学习这种方式的原因是以后Tomcat中运行的绝大多数都是Web项目，而使用Maven工具能更加简单快捷的把Web项目给创建出来，所以Maven的Web项目具体如何来构建呢？

在真正创建Maven Web项目之前，我们先要知道Web项目长什么样子，具体的结构是什么？

3.3.1 Web项目结构

Web项目的结构分为：开发中的项目和开发完可以部署的Web项目，这两种项目的结构是不一样的，我们一个个来介绍下：

- Maven Web项目结构：开发中的项目



- 开发完成部署的Web项目



- 开发项目通过执行Maven打包命令 **package**，可以获取到部署的Web项目目录
- 编译后的Java字节码文件和resources的资源文件，会被放到WEB-INF下的classes目录下
- pom.xml中依赖坐标对应的jar包，会被放入WEB-INF下的lib目录下

3.3.2 创建Maven Web项目

介绍完Maven Web的项目结构后，接下来使用Maven来创建Web项目，创建方式有两种：使用骨架和不使用骨架

使用骨架

具体的步骤包含：

1. 创建Maven项目

2.选择使用Web项目骨架

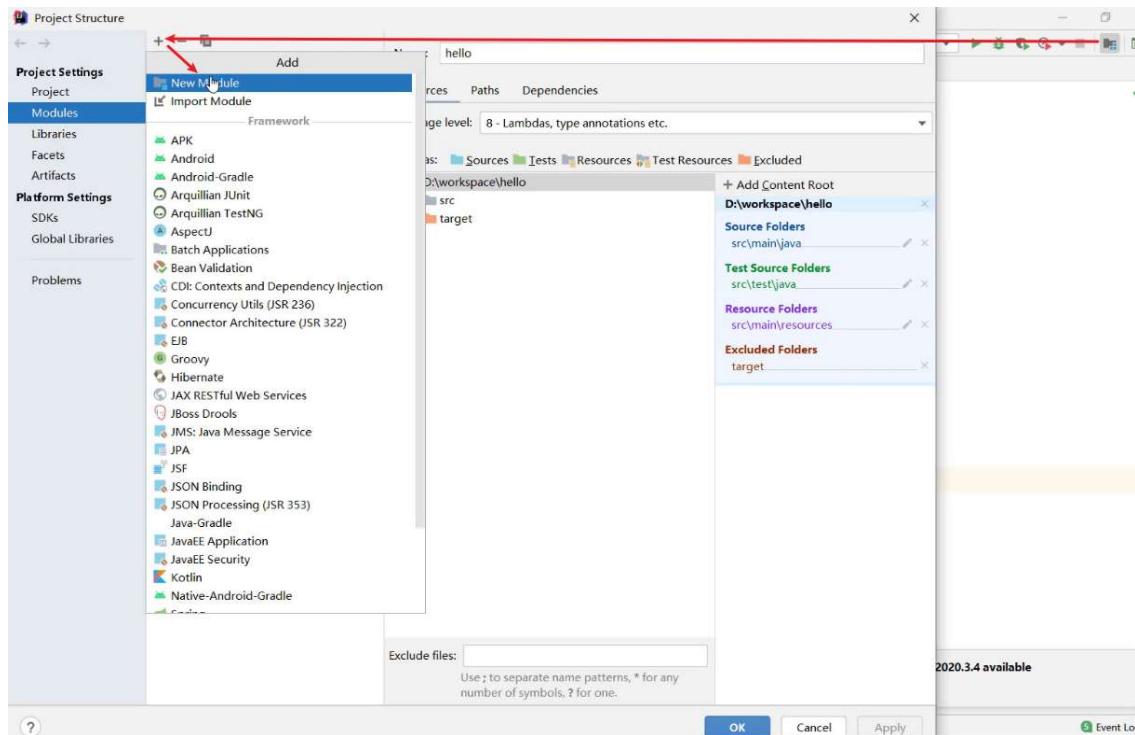
3.输入Maven项目坐标创建项目

4.确认Maven相关的配置信息后，完成项目创建

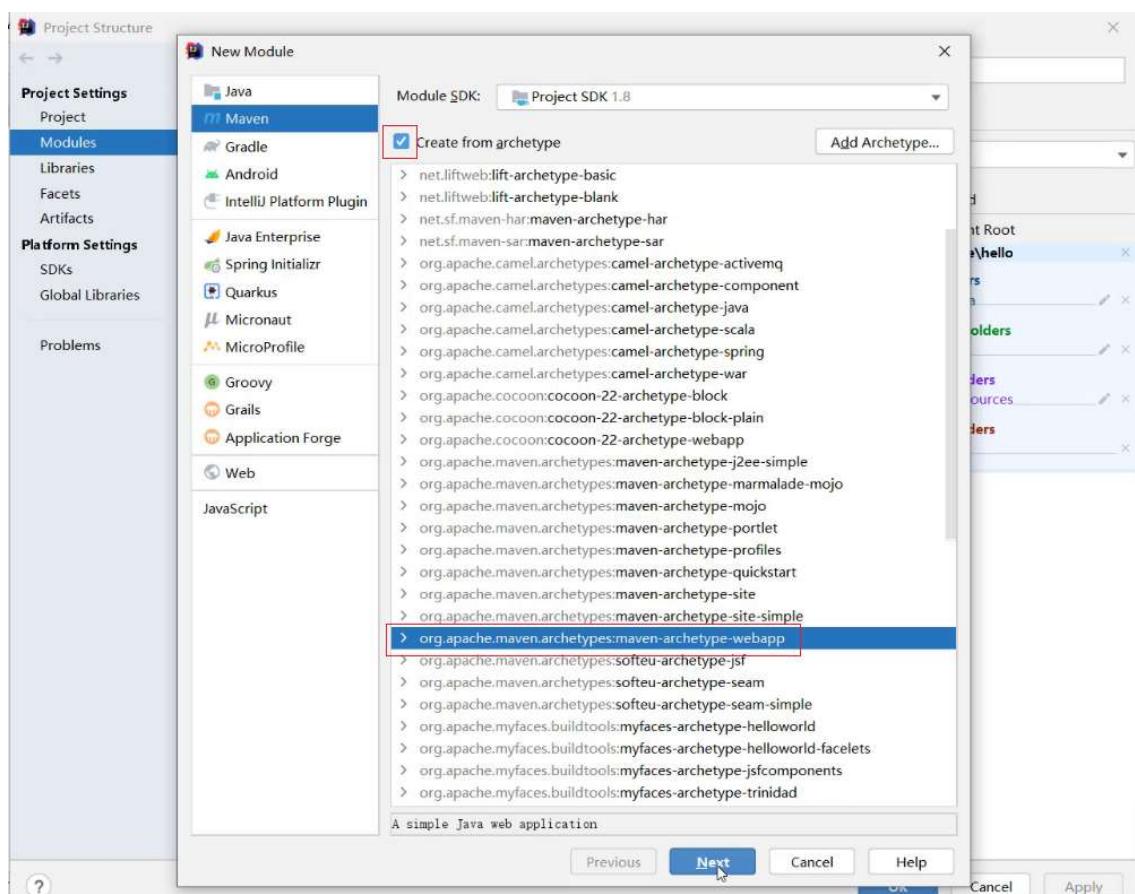
5.删除pom.xml中多余内容

6.补齐Maven Web项目缺失的目录结构

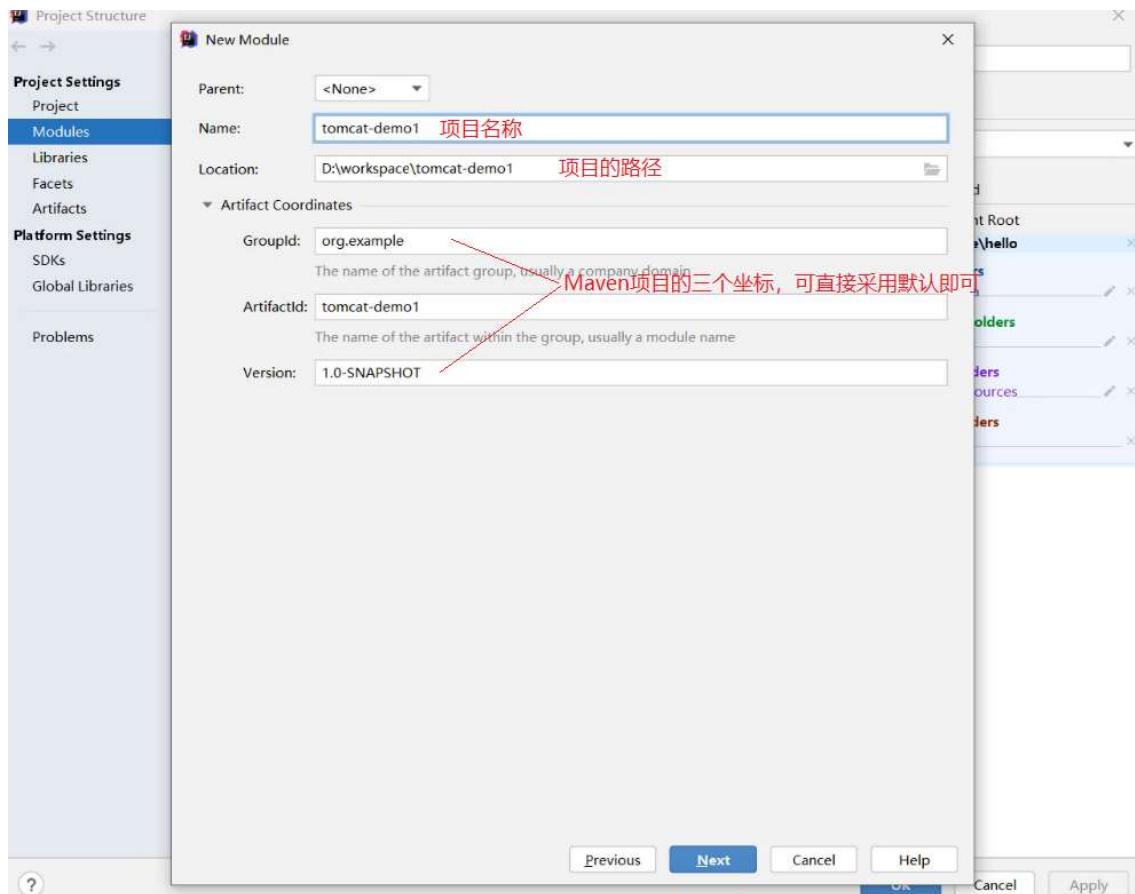
1. 创建Maven项目



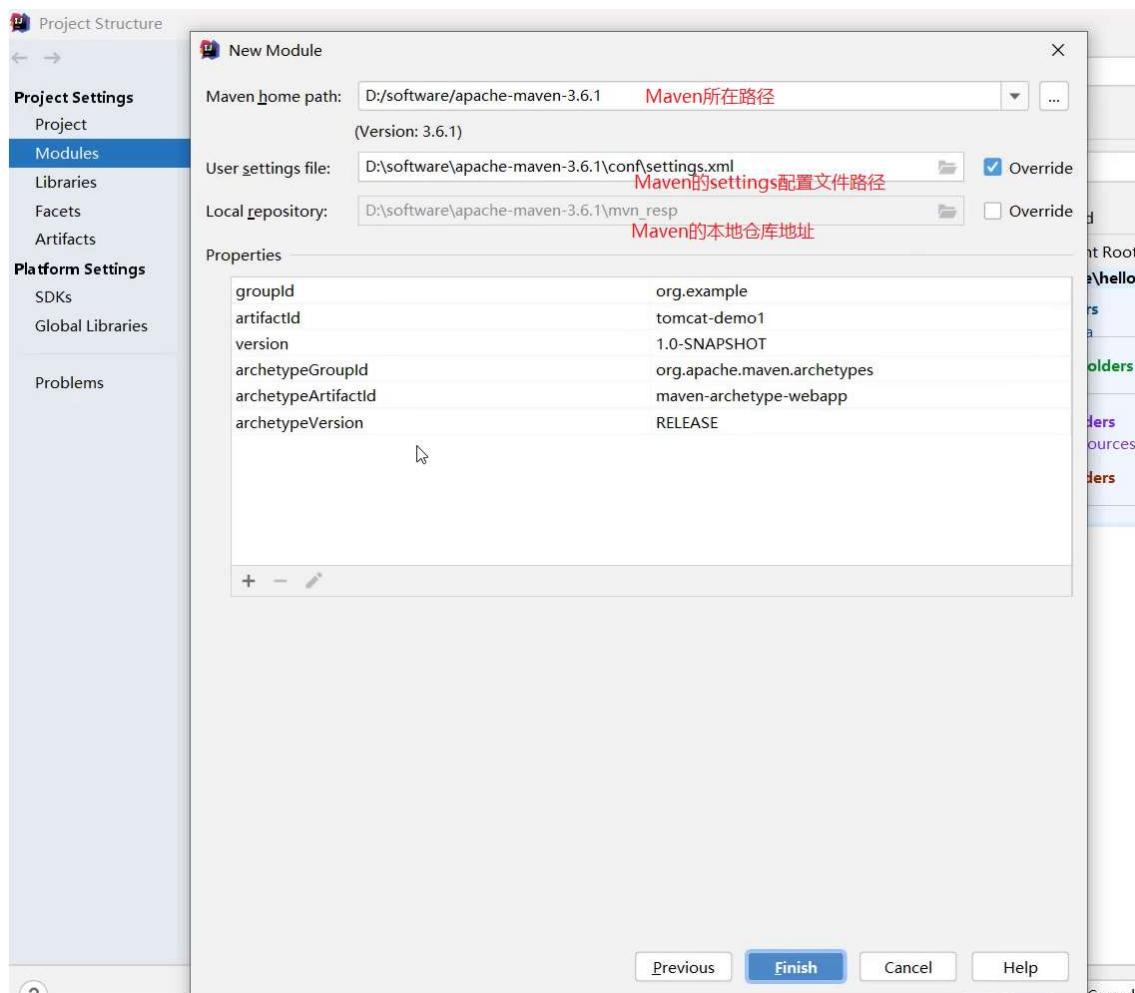
2. 选择使用Web项目骨架



3. 输入Maven项目坐标创建项目



4. 确认Maven相关的配置信息后，完成项目创建



5. 删除pom.xml中多余内容，只留下面的这些内容，注意打包方式jar和war的区别

```

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

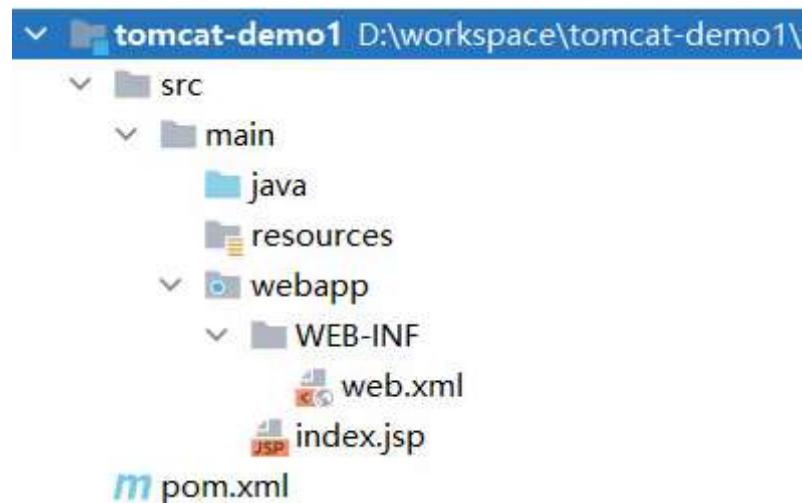
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>org.example</groupId>
  <artifactId>tomcat-demo1</artifactId>
  <version>1.0-SNAPSHOT</version>
  <!--
    <packaging>: 打包方式
    * jar: 默认值,
    * war: web项目
  -->
  <packaging>war</packaging>

</project>

```

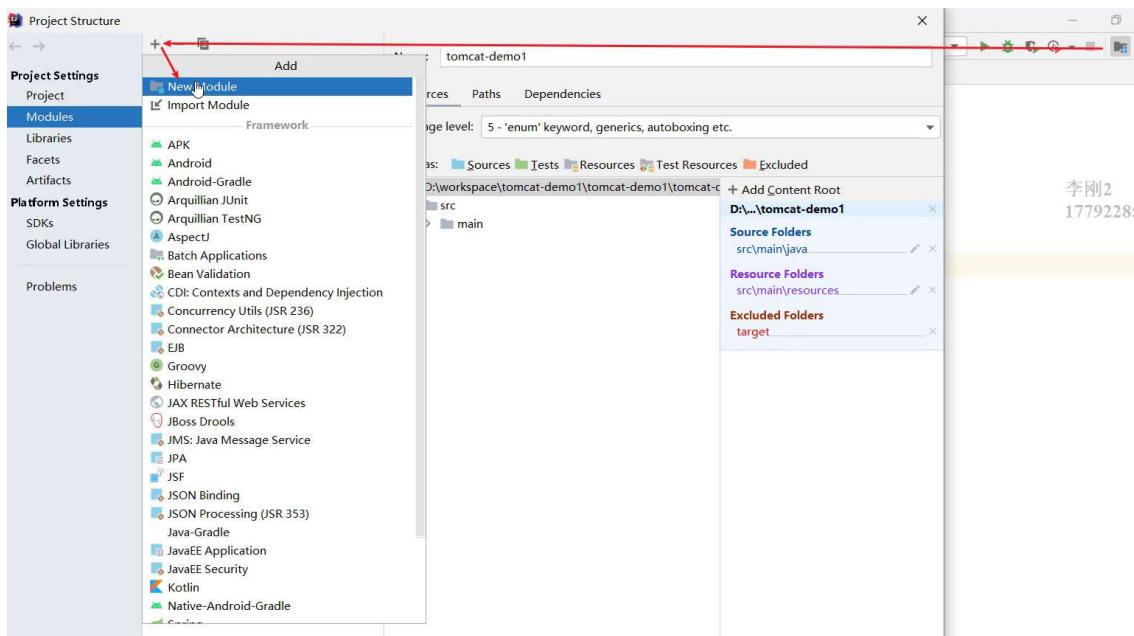
6. 补齐Maven Web项目缺失的目录结构，默认没有java和resources目录，需要手动完成创建补齐，最终的目录结果如下



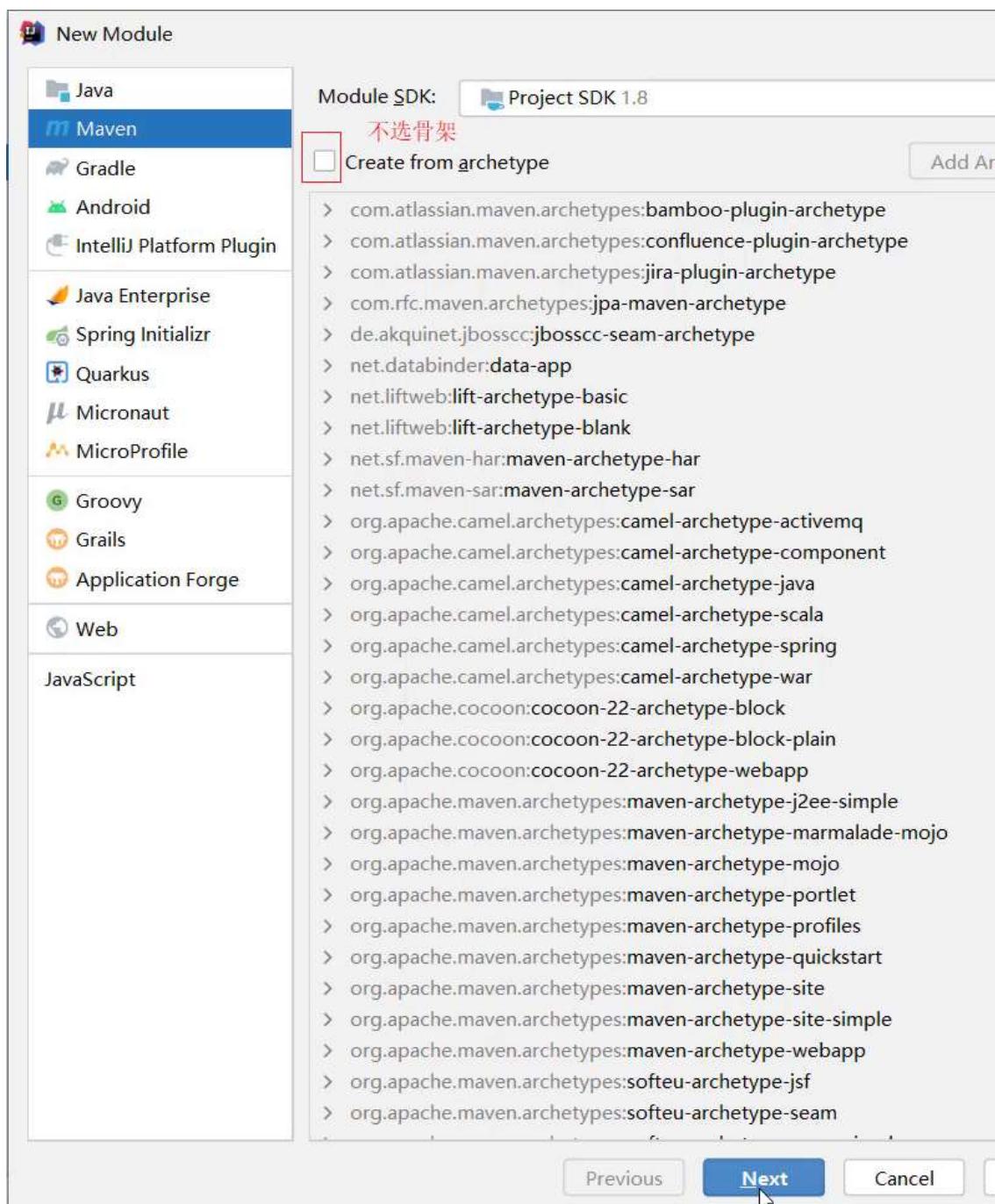
不使用骨架

具体的步骤包含：

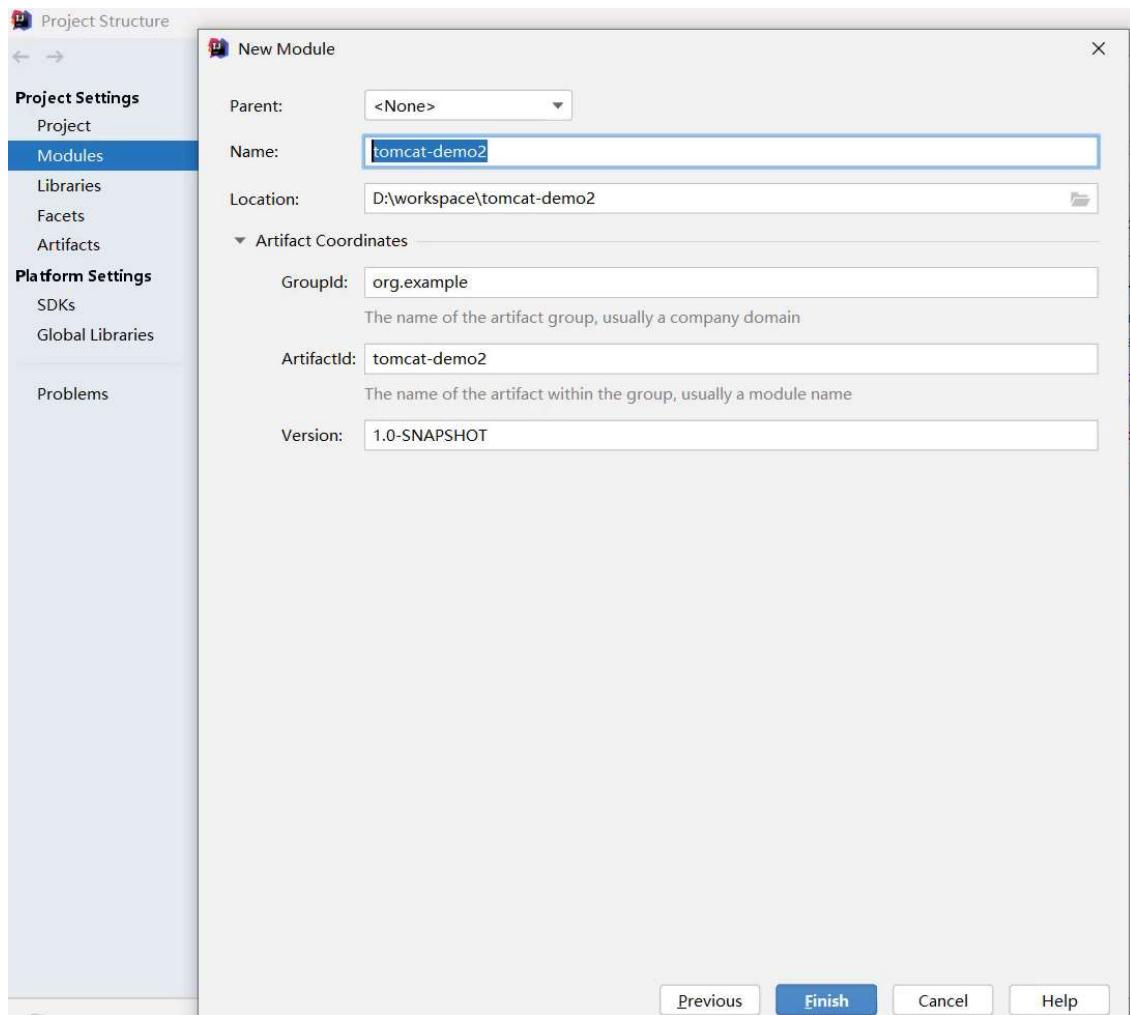
1. 创建Maven项目
 2. 选择不使用Web项目骨架
 3. 输入Maven项目坐标创建项目
 4. 在pom.xml设置打包方式为war
 5. 补齐Maven Web项目缺失webapp的目录结构
 6. 补齐Maven Web项目缺失WEB-INF/web.xml的目录结构
1. 创建Maven项目



2. 选择不使用Web项目骨架



3. 输入Maven项目坐标创建项目



4. 在pom.xml设置打包方式为war，默认是不写代表打包方式为jar

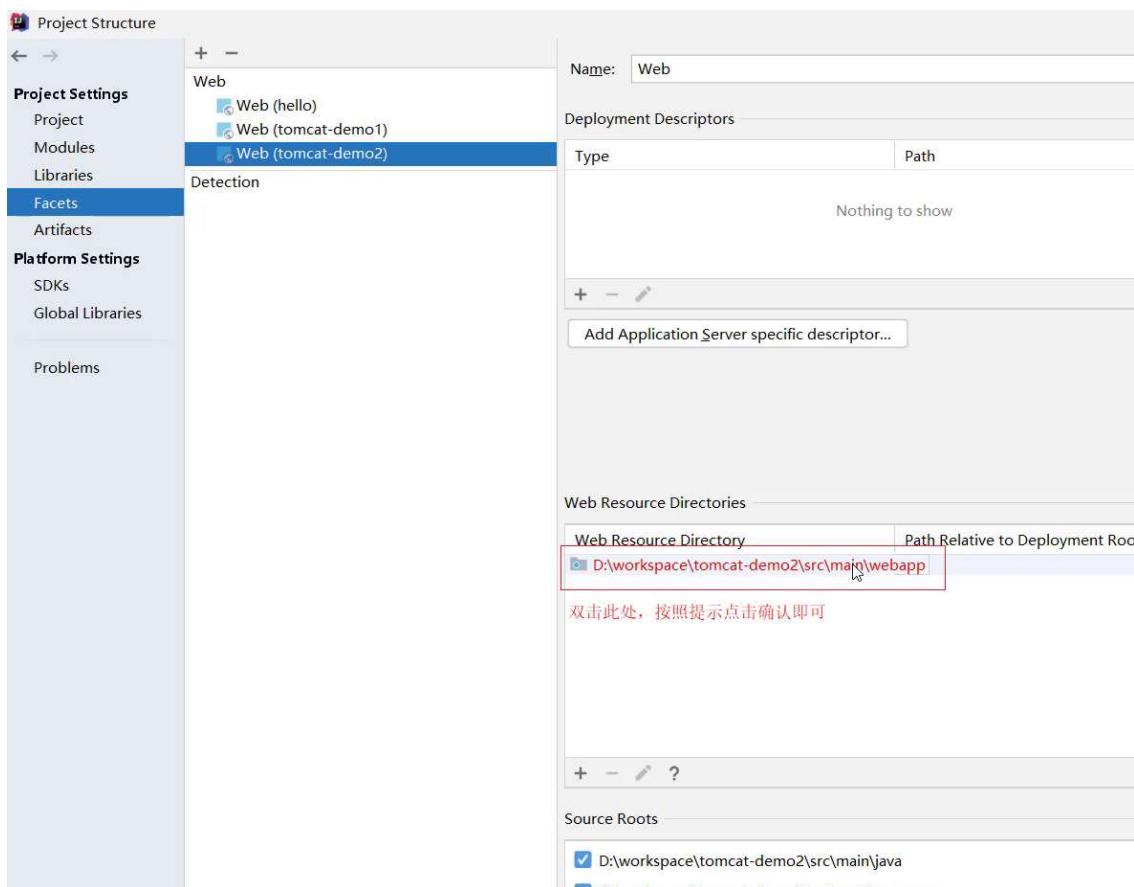
```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.a
<modelVersion>4.0.0</modelVersion>

<groupId>org.example</groupId>
<artifactId>tomcat-demo2</artifactId>
<version>1.0-SNAPSHOT</version>
[
    <packaging>war</packaging>
]

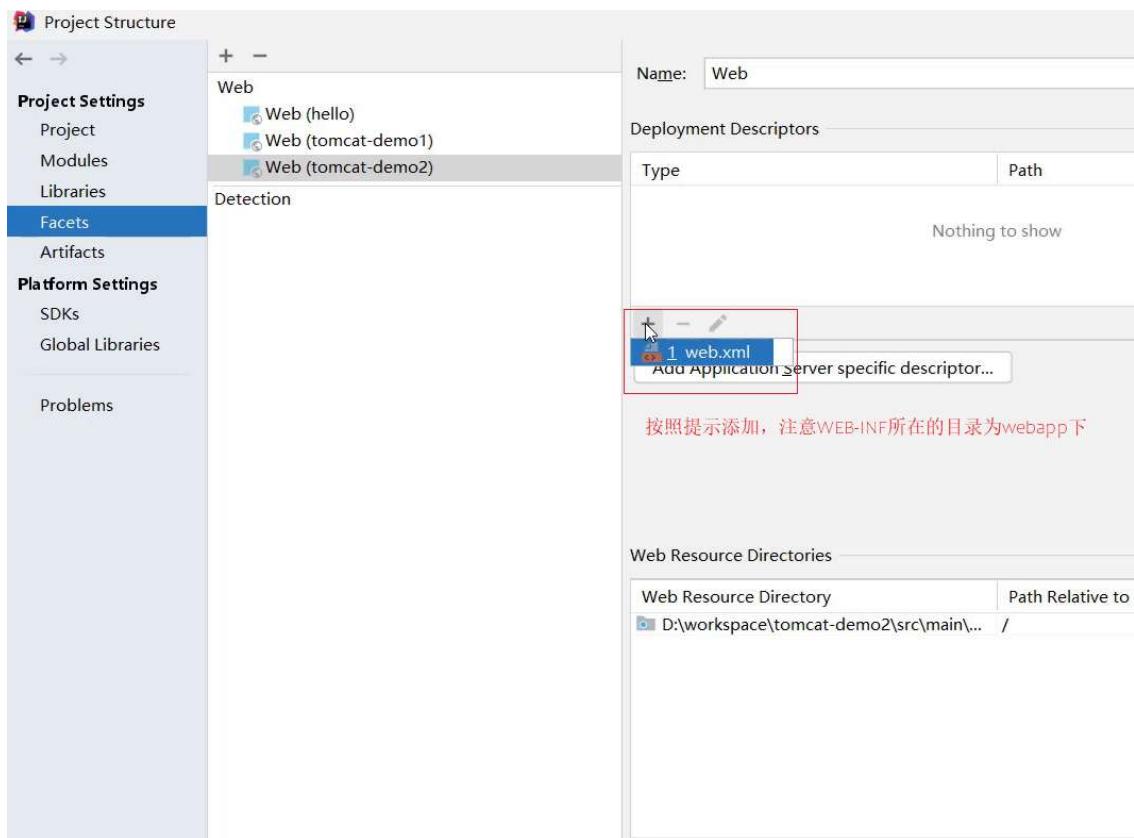
<properties>
    <maven.compiler.source>8</maven.compiler.source>
    <maven.compiler.target>8</maven.compiler.target>
</properties>

</project>
```

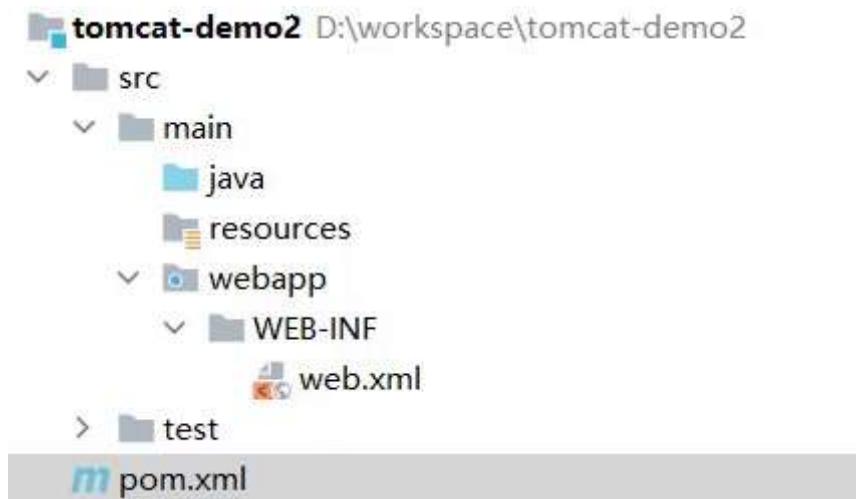
5. 补齐Maven Web项目缺失webapp的目录结构



6. 补齐Maven Web项目缺失WEB-INF/web.xml的目录结构



7. 补充完后，最终的项目结构如下：



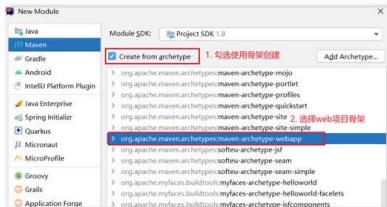
上述两种方式，创建的web项目，都不是很全，需要手动补充内容，至于最终采用哪种方式来创建 Maven Web项目，都是可以的，根据各自的喜好来选择使用即可。

小结

1. 掌握Maven Web项目的目录结构

2. 掌握使用骨架的方式创建Maven Web项目

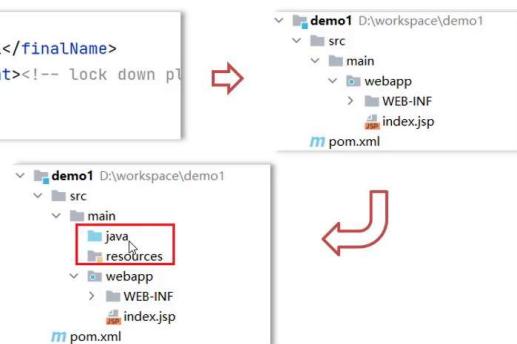
1. 选择web项目骨架，创建项目



2. 删除pom.xml中多余的坐标

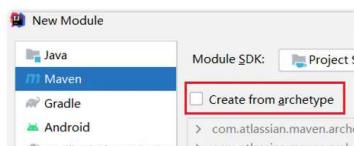
```
<build>
<finalName>demo1</finalName>
<pluginManagement><!-- lock down pl
<plugins>
<plugin>
```

3. 补齐缺失的目录结构



3. 掌握不使用骨架的方式创建Maven Web项目

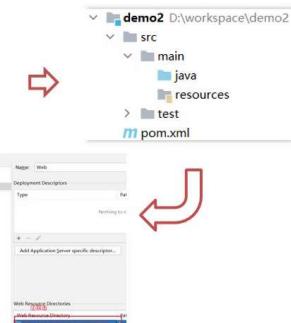
1. 选择web项目骨架，创建项目



2. pom.xml中添加打包方式为war

```
<groupId>com.itheima</groupId>
<artifactId>demo2</artifactId>
<version>1.0-SNAPSHOT</version>
<packaging>war</packaging>
```

3. 补齐缺失的目录结构：webapp



3.4 IDEA使用Tomcat

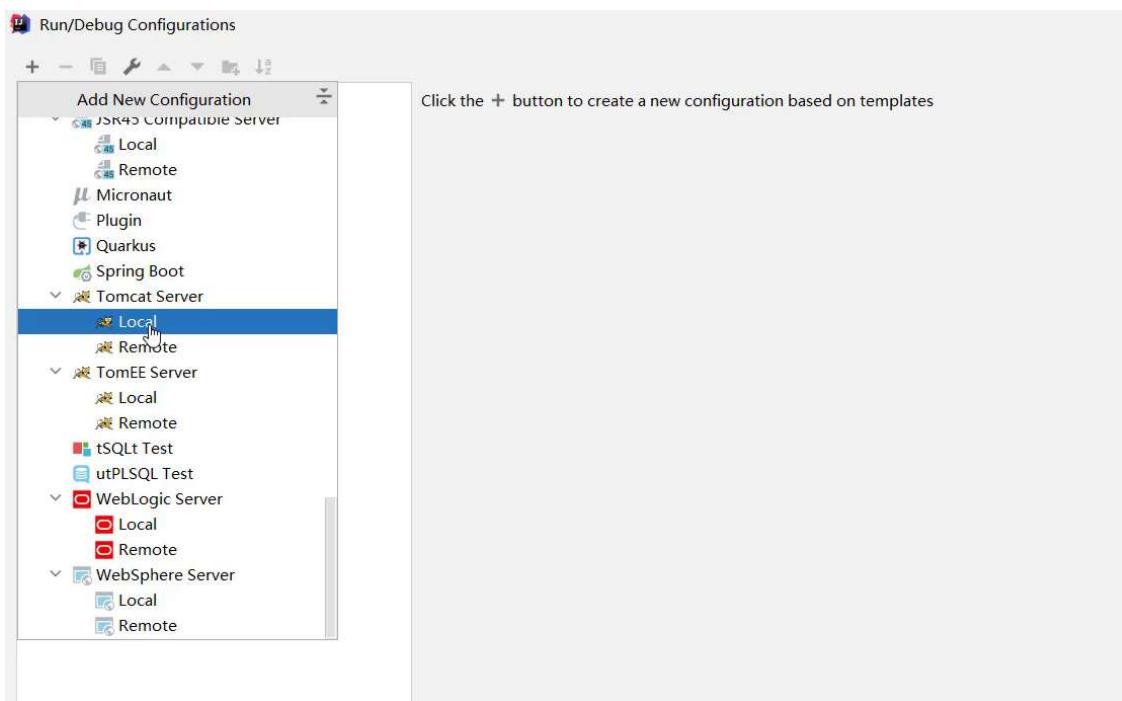
- Maven Web项目创建成功后，通过Maven的package命令可以将项目打包成war包，将war文件拷贝到Tomcat的webapps目录下，启动Tomcat就可以将项目部署成功，然后通过浏览器进行访问即可。
- 然而我们在开发的过程中，项目中的内容会经常发生变化，如果按照上面这种方式来部署测试，是非常不方便的
- 如何在IDEA中能快速使用Tomcat呢？

在IDEA中集成使用Tomcat有两种方式，分别是集成本地Tomcat和Tomcat Maven插件

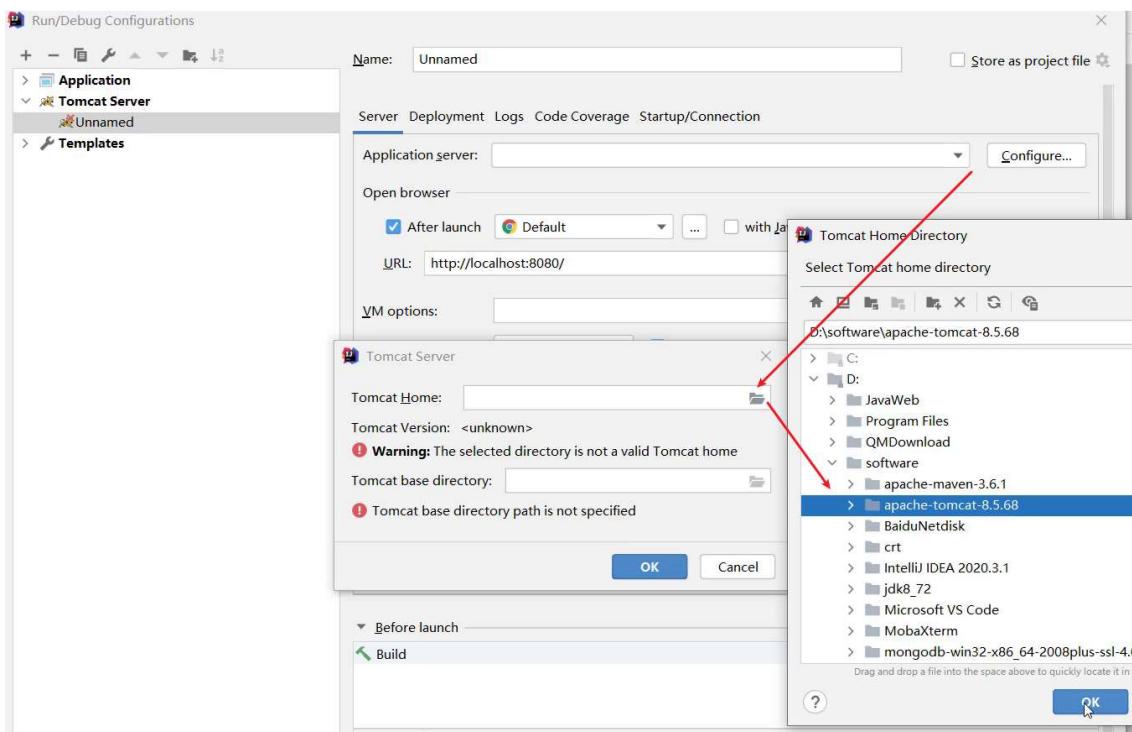
3.4.1 集成本地Tomcat

目标: 将刚才本地安装好的Tomcat8集成到IDEA中, 完成项目部署, 具体的实现步骤

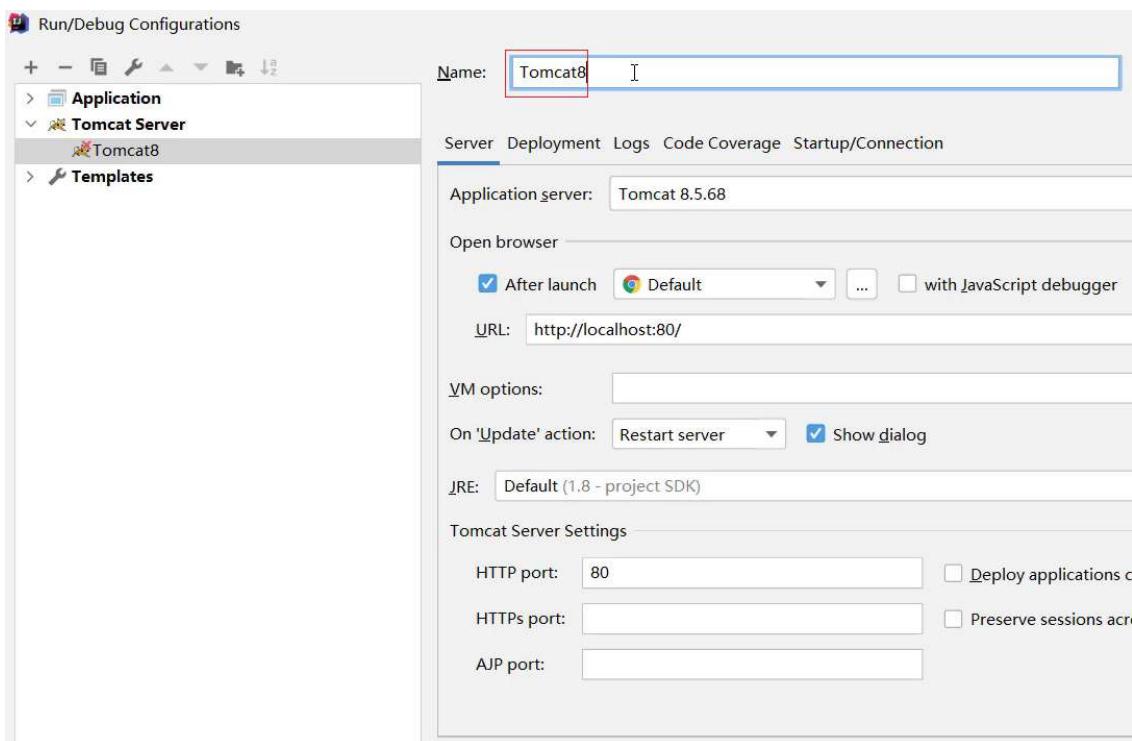
1. 打开添加本地Tomcat的面板



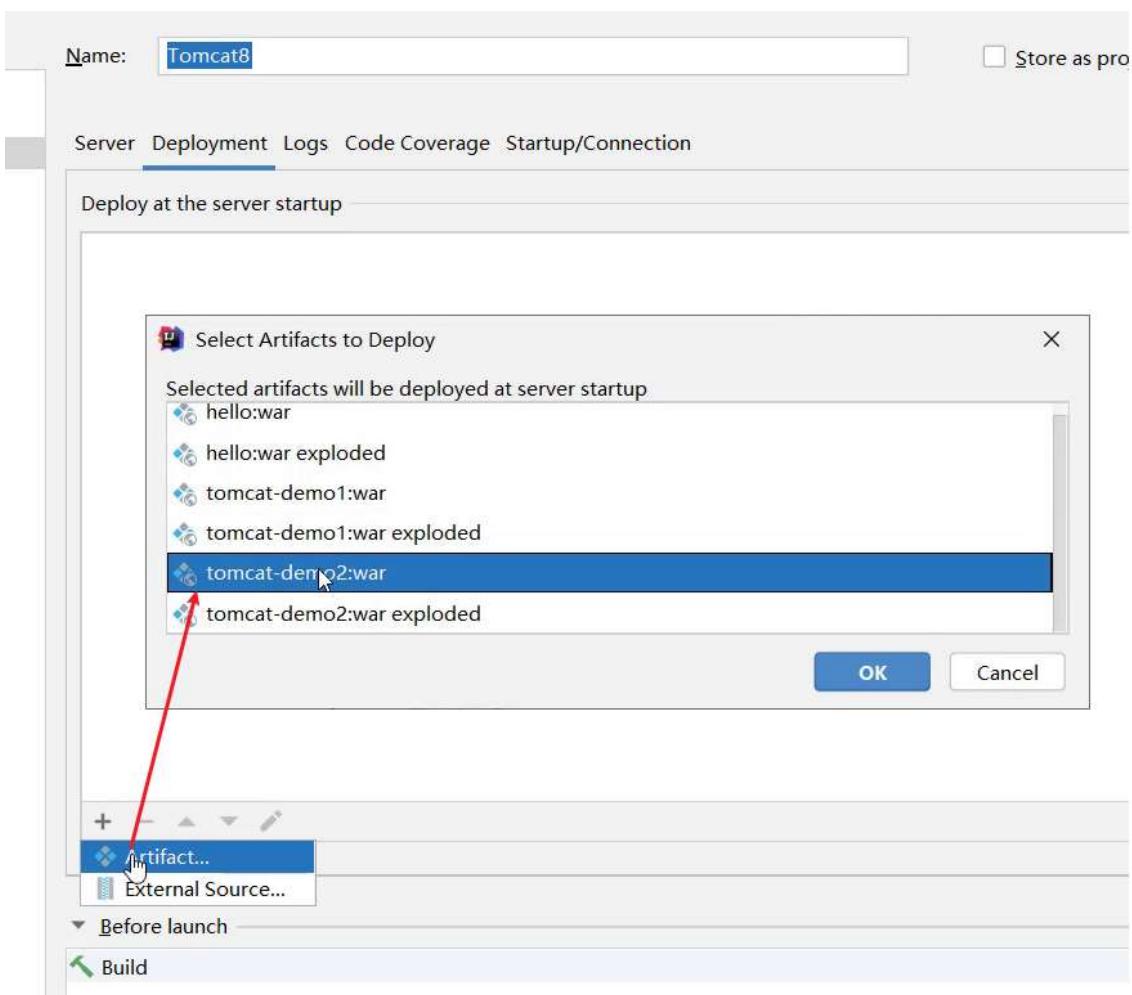
2. 指定本地Tomcat的具体路径



3. 修改Tomcat的名称, 此步骤可以不改, 只是让名字看起来更有意义, HTTP port中的端口也可以进行修改, 比如把8080改成80



4. 将开发项目部署项目到Tomcat中

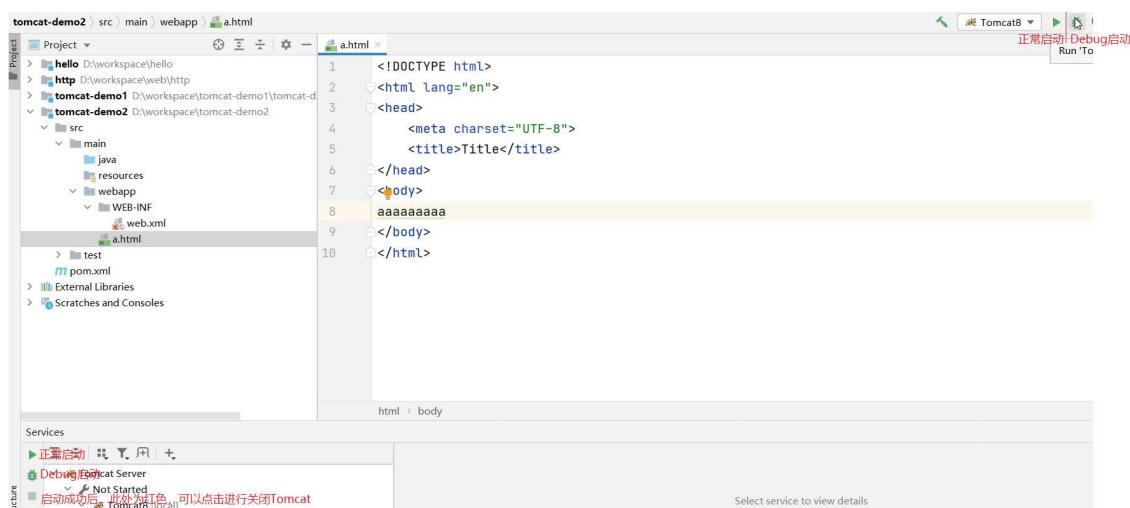


扩展内容： xxx.war 和 xxx.war exploded 这两种部署项目模式的区别？

- war 模式是将 WEB 工程打成 war 包，把 war 包发布到 Tomcat 服务器上
- war exploded 模式是将 WEB 工程以当前文件夹的位置关系发布到 Tomcat 服务器上
- war 模式部署成功后， Tomcat 的 webapps 目录下会有部署的项目内容
- war exploded 模式部署成功后， Tomcat 的 webapps 目录下没有，而使用的是项目的 target 目录下的内容进行部署

- 建议大家都选war模式进行部署，更符合项目部署的实际情况

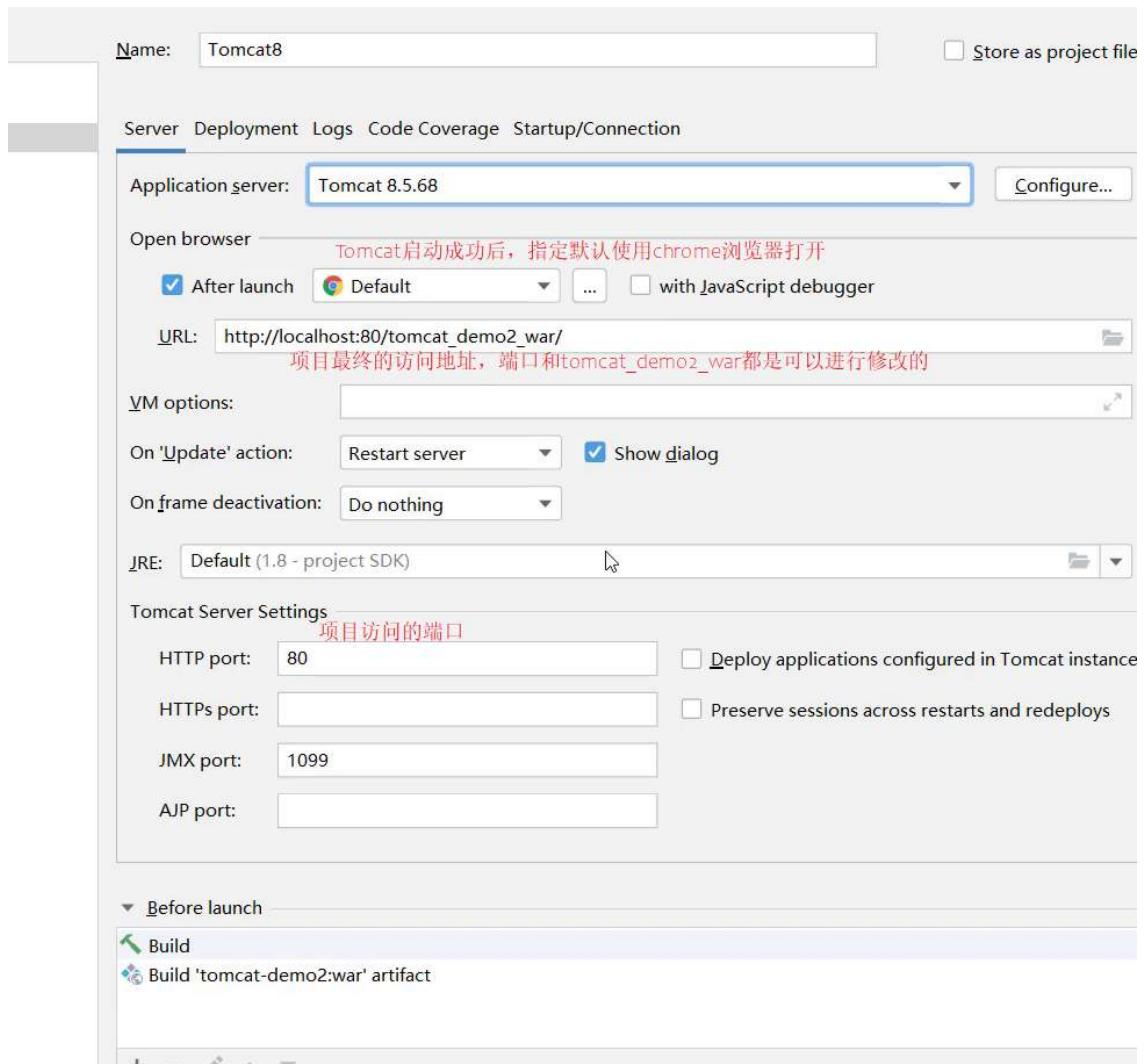
5. 部署成功后，就可以启动项目，为了能更好的看到启动的效果，可以在webapp目录下添加a.html页面



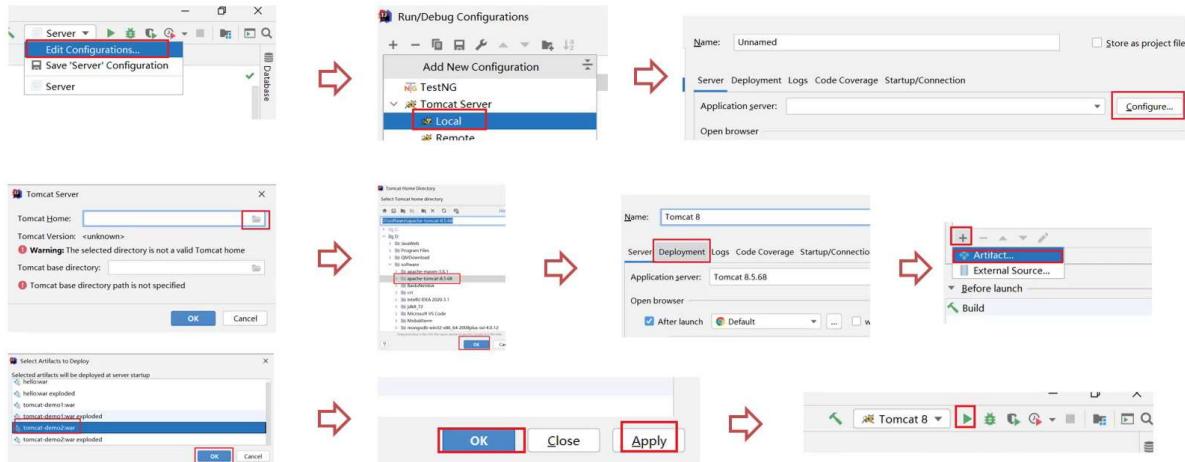
6. 启动成功后，可以通过浏览器进行访问测试



7. 最终的注意事项



至此，IDEA中集成本地Tomcat进行项目部署的内容我们就介绍完了，整体步骤如下，大家需要按照流程进行部署操作练习。



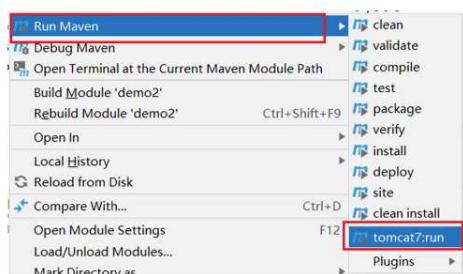
3.4.2 Tomcat Maven插件

在IDEA中使用本地Tomcat进行项目部署，相对来说步骤比较繁琐，所以我们需要一种更简便的方式来替换它，那就是直接使用Maven中的Tomcat插件来部署项目，具体的实现步骤，只需要两步，分别是：

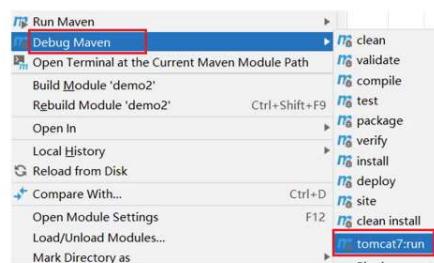
1. 在pom.xml中添加Tomcat插件

```
1 <build>
2   <plugins>
3     <!--Tomcat插件 -->
4     <plugin>
5       <groupId>org.apache.tomcat.maven</groupId>
6       <artifactId>tomcat7-maven-plugin</artifactId>
7       <version>2.2</version>
8     </plugin>
9   </plugins>
10  </build>
```

2. 使用Maven Helper插件快速启动项目，选中项目，右键-->Run Maven --> tomcat7:run

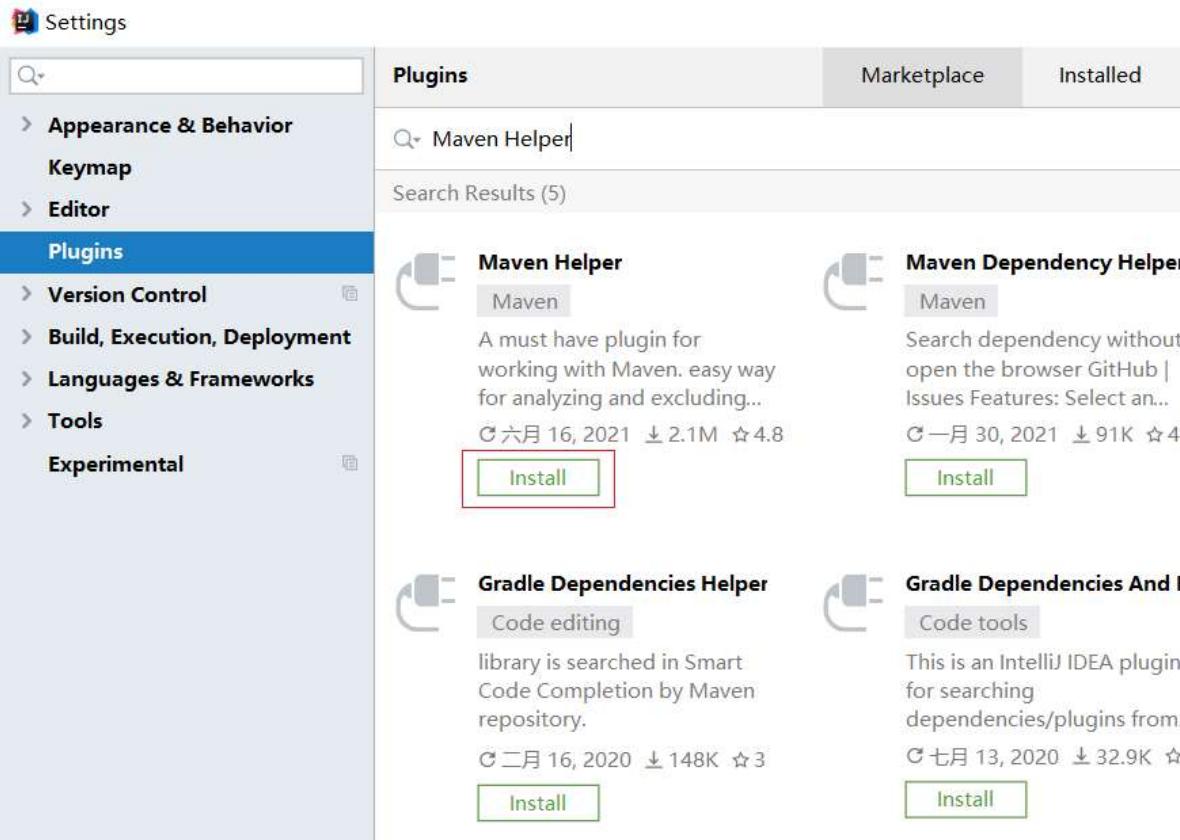


如果需要断点调试，选择 Debug Maven



注意：

- 如果选中项目并右键点击后，看不到Run Maven和Debug Maven，这个时候就需要在IDEA中下载 Maven Helper插件，具体的操作方式为：File --> Settings --> Plugins --> Maven Helper ---> Install,安装完后按照提示重启IDEA，就可以看到了。



- Maven Tomcat插件目前只有Tomcat7版本，没有更高的版本可以使用
- 使用Maven Tomcat插件，要想修改Tomcat的端口和访问路径，可以直接修改pom.xml

```
1 <build>
2   <plugins>
3     <!--Tomcat插件 -->
4     <plugin>
5       <groupId>org.apache.tomcat.maven</groupId>
6       <artifactId>tomcat7-maven-plugin</artifactId>
7       <version>2.2</version>
8       <configuration>
9         <port>80</port><!--访问端口号 -->
10        <!--项目访问路径
11          未配置访问路径: http://localhost:80/tomcat-demo2/a.html
12          配置/后访问路径: http://localhost:80/a.html
13          如果配置成 /hello,访问路径会变成什么?
14          答案: http://localhost:80/hello/a.html
15        -->
16        <path>/</path>
17      </configuration>
18    </plugin>
19  </plugins>
20</build>
```

小结

通过这一节的学习，大家要掌握在IDEA中使用Tomcat的两种方式，集成本地Tomcat和使用Maven的Tomcat插件。后者更简单，推荐大家使用，但是如果对于Tomcat的版本有比较高的要求，要在Tomcat7以上，这个时候就只能用前者了。

4, Servlet

4.1 简介



- Servlet是JavaWeb最为核心的内容，它是Java提供的一门动态web资源开发技术。
- 使用Servlet就可以实现，根据不同的登录用户在页面上动态显示不同内容。
- Servlet是JavaEE规范之一，其实就是一个接口，将来我们需要定义Servlet类实现Servlet接口，并由web服务器运行Servlet

```
public interface Servlet
```

Defines methods that all servlets must implement.

A servlet is a small Java program that runs within a Web server

介绍完Servlet是什么以后，接下来我们就按照 快速入门 -> 执行流程 -> 生命周期 -> 体系结构 -> urlPattern配置 -> XML配置 的学习步骤，一步步完成对Servlet的知识学习，首选我们来通过一个入门案例来快速把Servlet用起来。

4.2 快速入门

需求分析: 编写一个Servlet类，并使用IDEA中Tomcat插件进行部署，最终通过浏览器访问所编写的Servlet程序。

具体的实现步骤为：

1. 创建Web项目 web-demo，导入Servlet依赖坐标

```
1 <dependency>
2   <groupId>javax.servlet</groupId>
3   <artifactId>javax.servlet-api</artifactId>
4   <version>3.1.0</version>
5   <!--
6     此处为什么需要添加该标签？
7     provided指的是在编译和测试过程中有效，最后生成的war包时不会加入
8     因为Tomcat的lib目录中已经有servlet-api这个jar包，如果在生成war包的时候生效就会
9     和Tomcat中的jar包冲突，导致报错
10    -->
11    <scope>provided</scope>
12 </dependency>
```

2. 创建：定义一个类，实现Servlet接口，并重写接口中所有方法，并在service方法中输入一句话

```
1 package com.itheima.web;
2
```

```
3 import javax.servlet.*;
4 import java.io.IOException;
5
6 public class ServletDemo1 implements Servlet {
7
8     public void service(ServletRequest servletRequest, ServletResponse
servletResponse) throws ServletException, IOException {
9         System.out.println("servlet hello world~");
10    }
11    public void init(ServletConfig servletConfig) throws ServletException {
12    }
13    }
14
15    public ServletConfig getServletConfig() {
16        return null;
17    }
18
19    public String getServletInfo() {
20        return null;
21    }
22
23    public void destroy() {
24    }
25
26 }
```

3. 配置:在类上使用@WebServlet注解, 配置该Servlet的访问路径

```
1 | @WebServlet("/demo1")
```

4. 访问:启动Tomcat,浏览器中输入URL地址访问该Servlet

```
1 | http://localhost:8080/web-demo/demo1
```

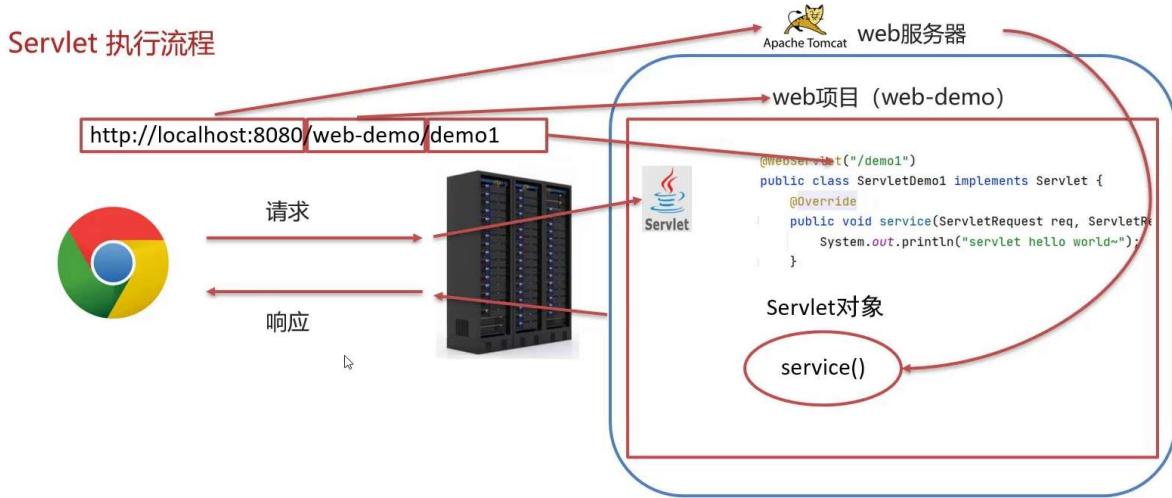
5. 器访问后, 在控制台会打印 `servlet hello world~` 说明servlet程序已经成功运行。

至此, Servlet的入门案例就已经完成, 大家可以按照上面的步骤进行练习了。

4.3 执行流程

Servlet程序已经能正常运行, 但是我们需要思考个问题: 我们并没有创建ServletDemo1类的对象, 也没有调用对象中的service方法, 为什么在控制台就打印了 `servlet hello world~` 这句话呢?

要想回答上述问题, 我们就需要对Servlet的执行流程进行一个学习。



- 浏览器发出 `http://localhost:8080/web-demo/demo1` 请求，从请求中可以解析出三部分内容，分别是 `localhost:8080`、`web-demo`、`demo1`
 - 根据 `localhost:8080` 可以找到要访问的Tomcat Web服务器
 - 根据 `web-demo` 可以找到部署在Tomcat服务器上的web-demo项目
 - 根据 `demo1` 可以找到要访问的是项目中的哪个Servlet类，根据@WebServlet后面的值进行匹配
- 找到ServletDemo1这个类后，Tomcat Web服务器就会为ServletDemo1这个类创建一个对象，然后调用对象中的service方法
 - ServletDemo1实现了Servlet接口，所以类中必然会重写service方法供Tomcat Web服务器进行调用
 - service方法中有ServletRequest和ServletResponse两个参数，ServletRequest封装的是请求数据，ServletResponse封装的是响应数据，后期我们可以通过这两个参数实现前端的数据交互

小结

介绍完Servlet的执行流程，需要大家掌握两个问题：

1. Servlet由谁创建?Servlet方法由谁调用?

Servlet由web服务器创建，Servlet方法由web服务器调用

2. 服务器怎么知道Servlet中一定有service方法?

因为我们自定义的Servlet,必须实现Servlet接口并复写其方法，而Servlet接口中有service方法

4.4 生命周期

介绍完Servlet的执行流程后，我们知道Servlet是由Tomcat Web服务器帮我们创建的。

接下来咱们再来思考一个问题：Tomcat什么时候创建的Servlet对象？

要想回答上述问题，我们就需要对Servlet的生命周期进行一个学习。

- 生命周期: 对象的生命周期指一个对象从被创建到被销毁的整个过程。
- Servlet运行在Servlet容器(web服务器)中，其生命周期由容器来管理，分为4个阶段：
 1. 加载和实例化：默认情况下，当Servlet第一次被访问时，由容器创建Servlet对象

```
1 | 默认情况，Servlet会在第一次访问被容器创建，但是如果创建Servlet比较耗时的话，那么第一个访问的人等待的时间就比较长，用户的体验就比较差，那么我们能不能把Servlet的创建放到服务器启动的时候来创建，具体如何来配置？
2 |
3 | @WebServlet(urlPatterns = "/demo1", loadOnStartup = 1)
4 | loadOnStartup的取值有两类情况
5 |   (1) 负整数：第一次访问时创建Servlet对象
6 |   (2) 0或正整数：服务器启动时创建Servlet对象，数字越小优先级越高
```

- 2. **初始化**：在**Servlet**实例化之后，容器将调用**Servlet**的**init()**方法初始化这个对象，完成一些如加载配置文件、创建连接等初始化的工作。该方法只调用一次
 - 3. **请求处理**：每次请求**Servlet**时，**Servlet**容器都会调用**Servlet**的**service()**方法对请求进行处理
 - 4. **服务终止**：当需要释放内存或者容器关闭时，容器就会调用**Servlet**实例的**destroy()**方法完成资源的释放。在**destroy()**方法调用之后，容器会释放这个**Servlet**实例，该实例随后会被Java的垃圾收集器所回收
- 通过案例演示下上述的生命周期

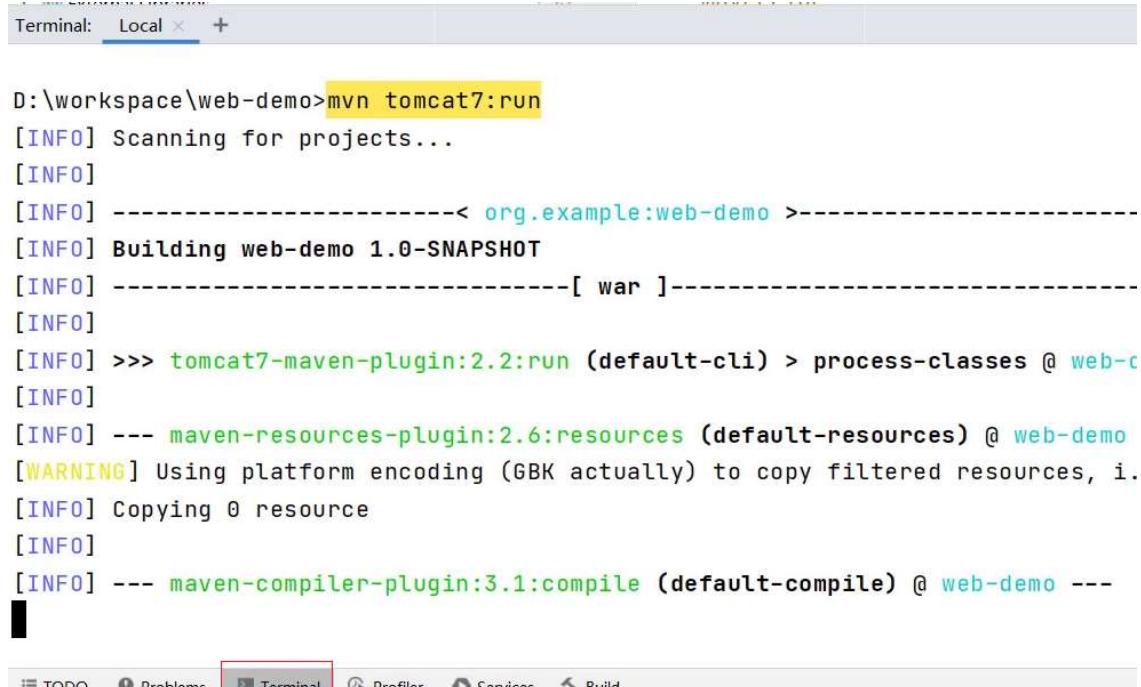
```
1 | package com.itheima.web;
2 |
3 | import javax.servlet.*;
4 | import javax.servlet.annotation.WebServlet;
5 | import java.io.IOException;
6 | /**
7 | * Servlet生命周期方法
8 | */
9 | @WebServlet(urlPatterns = "/demo2", loadOnStartup = 1)
10| public class ServletDemo2 implements Servlet {
11|
12| /**
13| * 初始化方法
14| * 1. 调用时机：默认情况下，Servlet被第一次访问时，调用
15| *           * LoadOnStartup：默认为-1，修改为0或者正整数，则会在服务器启动的时候，调用
16| *           * 2. 调用次数：1次
17| *           * @param config
18| *           * @throws ServletException
19| */
20| public void init(ServletConfig config) throws ServletException {
21|     System.out.println("init...");
22| }
23|
24| /**
25| * 提供服务
26| * 1. 调用时机：每一次Servlet被访问时，调用
27| * 2. 调用次数：多次
28| *           * @param req
29| *           * @param res
30| *           * @throws ServletException
31| *           * @throws IOException
32| */
33| public void service(ServletRequest req, ServletResponse res) throws
34| ServletException, IOException {
35|     System.out.println("servlet hello world~");
36| }
37| /**
```

```

38     * 销毁方法
39     * 1.调用时机: 内存释放或者服务器关闭的时候, Servlet对象会被销毁, 调用
40     * 2.调用次数: 1次
41     */
42     public void destroy() {
43         System.out.println("destroy...");
44     }
45     public ServletConfig getServletConfig() {
46         return null;
47     }
48
49     public String getServletInfo() {
50         return null;
51     }
52
53
54 }

```

注意:如何才能让Servlet中的destroy方法被执行?



```

Terminal: Local + 

D:\workspace\web-demo>mvn tomcat7:run
[INFO] Scanning for projects...
[INFO]
[INFO] -----< org.example:web-demo >-----
[INFO] Building web-demo 1.0-SNAPSHOT
[INFO] -----[ war ]-----
[INFO]
[INFO] >>> tomcat7-maven-plugin:2.2:run (default-cli) > process-classes @ web-d
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ web-demo
[WARNING] Using platform encoding (GBK actually) to copy filtered resources, i.
[INFO] Copying 0 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ web-demo ---

```

在Terminal命令行中, 先使用 `mvn tomcat7:run` 启动, 然后再使用 `ctrl+c` 关闭tomcat

小结

这节中需要掌握的内容是:

1. Servlet对象在什么时候被创建的?

默认是第一次访问的时候被创建, 可以使用`@WebServlet(urlPatterns = "/demo2", loadOnStartup = 1)`的`loadOnStartup`修改成在服务器启动的时候创建。

2. Servlet生命周期中涉及到的三个方法, 这三个方法是什么?什么时候被调用?调用几次?

涉及到三个方法, 分别是 `init()`、`service()`、`destroy()`

`init`方法在Servlet对象被创建的时候执行, 只执行1次

`service`方法在Servlet被访问的时候调用, 每访问1次就调用1次

`destroy`方法在Servlet对象被销毁的时候调用, 只执行1次

4.5 方法介绍

Servlet中总共有5个方法，我们已经介绍过其中的三个，剩下的两个方法作用分别是什么？

我们先来回顾下前面讲的三个方法，分别是：

- 初始化方法，在Servlet被创建时执行，只执行一次

```
1 | void init(ServletConfig config)
```

- 提供服务方法，每次Servlet被访问，都会调用该方法

```
1 | void service(ServletRequest req, ServletResponse res)
```

- 销毁方法，当Servlet被销毁时，调用该方法。在内存释放或服务器关闭时销毁Servlet

```
1 | void destroy()
```

剩下的两个方法是：

- 获取Servlet信息

```
1 | String getServletInfo()  
2 | //该方法用来返回Servlet的相关信息，没有什么太大的用处，一般我们返回一个空字符串即可  
3 | public String getServletInfo() {  
4 |     return "";  
5 | }
```

- 获取ServletConfig对象

```
1 | ServletConfig getServletConfig()
```

ServletConfig对象，在init方法的参数中有，而Tomcat Web服务器在创建Servlet对象的时候会调用init方法，必定会传入一个ServletConfig对象，我们只需要将服务器传过来的ServletConfig进行返回即可。具体如何操作？

```
1 | package com.itheima.web;  
2 |  
3 | import javax.servlet.*;  
4 | import javax.servlet.annotation.WebServlet;  
5 | import java.io.IOException;  
6 |  
7 | /**  
8 | * Servlet方法介绍  
9 | */  
10 | @WebServlet(urlPatterns = "/demo3", loadOnStartup = 1)  
11 | public class ServletDemo3 implements Servlet {  
12 |  
13 |     private ServletConfig servletConfig;  
14 |     /**  
15 |      * 初始化方法  
16 |      * 1. 调用时机：默认情况下，Servlet被第一次访问时，调用  
17 |      *          * LoadOnStartup：默认为-1，修改为0或者正整数，则会在服务器启动的时候，调  
用
```

```

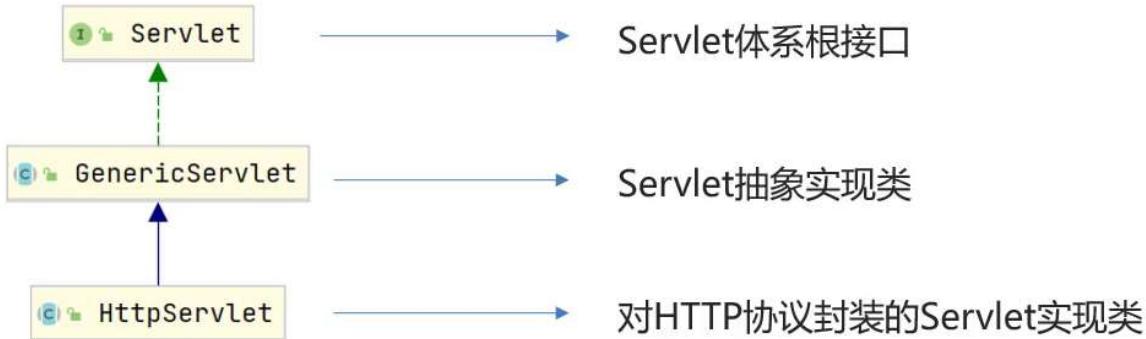
18     * 2.调用次数: 1次
19     * @param config
20     * @throws ServletException
21     */
22     public void init(ServletConfig config) throws ServletException {
23         this.servletConfig = config;
24         System.out.println("init...");
25     }
26     public ServletConfig getServletConfig() {
27         return servletConfig;
28     }
29
30 /**
31 * 提供服务
32 * 1.调用时机:每一次Servlet被访问时, 调用
33 * 2.调用次数: 多次
34 * @param req
35 * @param res
36 * @throws ServletException
37 * @throws IOException
38 */
39     public void service(ServletRequest req, ServletResponse res) throws
ServletException, IOException {
40         System.out.println("servlet hello world~");
41     }
42
43 /**
44 * 销毁方法
45 * 1.调用时机: 内存释放或者服务器关闭的时候, Servlet对象会被销毁, 调用
46 * 2.调用次数: 1次
47 */
48     public void destroy() {
49         System.out.println("destroy...");
50     }
51
52     public String getServletInfo() {
53         return "";
54     }
55 }
```

getServletInfo()和getServletConfig()这两个方法使用的不是很多, 大家了解下。

4.6 体系结构

通过上面的学习, 我们知道要想编写一个Servlet就必须要实现Servlet接口, 重写接口中的5个方法, 虽然已经能完成要求, 但是编写起来还是比较麻烦的, 因为我们更关注的其实只有service方法, 那有没有更简单方式来创建Servlet呢?

要想解决上面的问题, 我们需要先对Servlet的体系结构进行下了解:



因为我们将来开发B/S架构的web项目，都是针对HTTP协议，所以我们自定义Servlet,会通过继承 **HttpServlet**

具体的编写格式如下：

```

1  @WebServlet("/demo4")
2  public class ServletDemo4 extends HttpServlet {
3      @Override
4          protected void doGet(HttpServletRequest req, HttpServletResponse resp)
5              throws ServletException, IOException {
6                  //TODO GET 请求方式处理逻辑
7                  System.out.println("get...");}
8      @Override
9          protected void doPost(HttpServletRequest req, HttpServletResponse resp)
10             throws ServletException, IOException {
11                 //TODO Post 请求方式处理逻辑
12                 System.out.println("post...");}
13 }

```

- 要想发送一个GET请求，请求该Servlet，只需要通过浏览器发送 `http://localhost:8080/web-demo/demo4`,就能看到doGet方法被执行了
- 要想发送一个POST请求，请求该Servlet，单单通过浏览器是无法实现的，这个时候就需要编写一个form表单来发送请求，在webapp下创建一个 `a.html` 页面，内容如下：

```

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>Title</title>
6  </head>
7  <body>
8      <form action="/web-demo/demo4" method="post">
9          <input name="username"/><input type="submit"/>
10     </form>
11 </body>
12 </html>

```

启动测试，即可看到doPost方法被执行了。

Servlet的简化编写就介绍完了，接着需要思考两个问题：

1. **HttpServlet**中为什么要根据请求方式的不同，调用不同的方法？
2. 如何调用？

针对问题一，我们需要回顾之前的知识点前端发送GET和POST请求的时候，参数的位置不一致，GET请求参数在请求行中，POST请求参数在请求体中，为了能处理不同的请求方式，我们得在service方法中进行判断，然后写不同的业务处理，这样能实现，但是每个Servlet类中都将有相似的代码，针对这个问题，有什么可以优化的策略么？

```
1 package com.itheima.web;
2
3 import javax.servlet.*;
4 import javax.servlet.annotation.WebServlet;
5 import javax.servlet.http.HttpServlet;
6 import javax.servlet.http.HttpServletRequest;
7 import javax.servlet.http.HttpServletResponse;
8 import java.io.IOException;
9
10
11 @WebServlet("/demo5")
12 public class ServletDemo5 implements Servlet {
13
14     public void init(ServletConfig config) throws ServletException {
15
16     }
17
18     public ServletConfig getServletConfig() {
19         return null;
20     }
21
22     public void service(HttpServletRequest req, HttpServletResponse res) throws
23     ServletException, IOException {
24         //如何调用?
25         //获取请求方式，根据不同的请求方式进行不同的业务处理
26         HttpServletRequest request = (HttpServletRequest)req;
27         //1. 获取请求方式
28         String method = request.getMethod();
29         //2. 判断
30         if("GET".equals(method)){
31             // get方式的处理逻辑
32         }else if("POST".equals(method)){
33             // post方式的处理逻辑
34         }
35
36         public String getServletInfo() {
37             return null;
38         }
39
40         public void destroy() {
41
42     }
43 }
44 }
```

要解决上述问题，我们可以对Servlet接口进行继承封装，来简化代码开发。

```
1 package com.itheima.web;
2
3 import javax.servlet.*;
```

```

4 import javax.servlet.http.HttpServletRequest;
5 import java.io.IOException;
6
7 public class MyHttpServlet implements Servlet {
8     public void init(ServletConfig config) throws ServletException {
9
10    }
11
12    public ServletConfig getServletConfig() {
13        return null;
14    }
15
16    public void service(ServletRequest req, ServletResponse res) throws
17    ServletException, IOException {
18        HttpServletRequest request = (HttpServletRequest)req;
19        //1. 获取请求方式
20        String method = request.getMethod();
21        //2. 判断
22        if("GET".equals(method)){
23            // get方式的处理逻辑
24            doGet(req,res);
25        }else if("POST".equals(method)){
26            // post方式的处理逻辑
27            doPost(req,res);
28        }
29    }
30
31    protected void doPost(HttpServletRequest req, ServletResponse res) {
32    }
33
34    protected void doGet(HttpServletRequest req, ServletResponse res) {
35    }
36
37    public String getServletInfo() {
38        return null;
39    }
40
41    public void destroy() {
42    }
43}
44

```

有了MyHttpServlet这个类，以后我们再编写Servlet类的时候，只需要继承MyHttpServlet，重写父类中的doGet和doPost方法，就可以用来处理GET和POST请求的业务逻辑。接下来，可以把ServletDemo5代码进行改造

```

1 @WebServlet("/demo5")
2 public class ServletDemo5 extends MyHttpServlet {
3
4     @Override
5     protected void doGet(HttpServletRequest req, ServletResponse res) {
6         System.out.println("get...");
7     }
8
9     @Override
10    protected void doPost(HttpServletRequest req, ServletResponse res) {

```

```
11     System.out.println("post...");  
12 }  
13 }  
14 }
```

将来页面发送的是GET请求，则会进入到doGet方法中进行执行，如果是POST请求，则进入到doPost方法。这样代码在编写的时候就相对来说更加简单快捷。

类似MyHttpServlet这样的类Servlet中已经为我们提供好了，就是HttpServlet，翻开源码，大家可以搜索service()方法，你会发现HttpServlet做的事更多，不仅可以处理GET和POST还可以处理其他五种请求方式。

```
1 protected void service(HttpServletRequest req, HttpServletResponse resp)  
2     throws ServletException, IOException  
3 {  
4     String method = req.getMethod();  
5  
6     if (method.equals(METHOD_GET)) {  
7         long lastModified = getLastModified(req);  
8         if (lastModified == -1) {  
9             // servlet doesn't support if-modified-since, no reason  
10            // to go through further expensive logic  
11            doGet(req, resp);  
12        } else {  
13            long ifModifiedSince = req.getDateHeader(HEADER_IFMODSINCE);  
14            if (ifModifiedSince < lastModified) {  
15                // If the servlet mod time is later, call doGet()  
16                // Round down to the nearest second for a proper compare  
17                // A ifModifiedSince of -1 will always be less  
18                maybeSetLastModified(resp, lastModified);  
19                doGet(req, resp);  
20            } else {  
21                resp.setStatus(HttpServletResponse.SC_NOT_MODIFIED);  
22            }  
23        }  
24    }  
25    else if (method.equals(METHOD_HEAD)) {  
26        long lastModified = getLastModified(req);  
27        maybeSetLastModified(resp, lastModified);  
28        doHead(req, resp);  
29    }  
30    else if (method.equals(METHOD_POST)) {  
31        doPost(req, resp);  
32    }  
33    else if (method.equals(METHOD_PUT)) {  
34        doPut(req, resp);  
35    }  
36    else if (method.equals(METHOD_DELETE)) {  
37        doDelete(req, resp);  
38    }  
39    else if (method.equals(METHOD_OPTIONS)) {  
40        doOptions(req, resp);  
41    }  
42    else if (method.equals(METHOD_TRACE)) {  
43        doTrace(req, resp);  
44    }  
45 }
```

```

46         //
47         // Note that this means NO servlet supports whatever
48         // method was requested, anywhere on this server.
49         //
50
51         String errMsg =
52             IStrings.getString("http.method_not_implemented");
53         Object[] errArgs = new Object[1];
54         errArgs[0] = method;
55         errMsg = MessageFormat.format(errMsg, errArgs);
56
57         resp.sendError(HttpServletResponse.SC_NOT_IMPLEMENTED, errMsg);
58     }
59 }
```

小结

通过这一节的学习，要掌握：

1. HttpServlet的使用步骤

继承HttpServlet

重写doGet和doPost方法

2. HttpServlet原理

获取请求方式，并根据不同的请求方式，调用不同的doXXX方法

4.7 urlPattern配置

Servlet类编写好后，要想被访问到，就需要配置其访问路径（urlPattern）

- 一个Servlet,可以配置多个urlPattern

```
@WebServlet(urlPatterns = {"/demo1", "/demo2"})
```

```

1 package com.itheima.web;
2
3 import javax.servlet.ServletRequest;
4 import javax.servlet.ServletResponse;
5 import javax.servlet.annotation.WebServlet;
6
7 /**
8 * urlPattern: 一个Servlet可以配置多个访问路径
9 */
10 @webServlet(urlPatterns = {"/demo7", "/demo8"})
11 public class ServletDemo7 extends MyHttpServlet {
12
13     @Override
14     protected void doGet(ServletRequest req, ServletResponse res) {
15
16         System.out.println("demo7 get...");
17     }
18     @Override
19     protected void doPost(ServletRequest req, ServletResponse res) {
20
21 }
```

在浏览器上输入 `http://localhost:8080/web-demo/demo7`, `http://localhost:8080/web-demo/demo8` 这两个地址都能访问到ServletDemo7的doGet方法。

- **urlPattern配置规则**

- 精确匹配

- 配置路径: `@WebServlet("/user/select")`
- 访问路径: `localhost:8080/web-demo/user/select`

```
1 /**
2  * UrlPattern:
3  * * 精确匹配
4  */
5 @WebServlet(urlPatterns = "/user/select")
6 public class ServletDemo8 extends MyHttpServlet {
7
8     @Override
9     protected void doGet(ServletRequest req, ServletResponse res) {
10
11         System.out.println("demo8 get...");
12     }
13     @Override
14     protected void doPost(ServletRequest req, ServletResponse res) {
15     }
16 }
```

访问路径 `http://localhost:8080/web-demo/user/select`

- 目录匹配

- 配置路径: `@WebServlet("/user/*")`

- 访问路径: `localhost:8080/web-demo/user/aaa`

`localhost:8080/web-demo/user/bbb`

```
1 package com.itheima.web;
2
3 import javax.servlet.ServletRequest;
4 import javax.servlet.ServletResponse;
5 import javax.servlet.annotation.WebServlet;
6
7 /**
8  * UrlPattern:
9  * * 目录匹配: /user/*
10 */
11 @WebServlet(urlPatterns = "/user/*")
12 public class ServletDemo9 extends MyHttpServlet {
13
14     @Override
15     protected void doGet(ServletRequest req, ServletResponse res) {
```

```
16         System.out.println("demo9 get...");  
17     }  
18     @Override  
19     protected void doPost(HttpServletRequest req, HttpServletResponse res) {  
20     }  
21 }  
22 }
```

访问路径 `http://localhost:8080/web-demo/user/任意`

思考:

1. 访问路径 `http://localhost:8080/web-demo/user` 是否能访问到 demo9 的 doGet 方法?
2. 访问路径 `http://localhost:8080/web-demo/user/a/b` 是否能访问到 demo9 的 doGet 方法?
3. 访问路径 `http://localhost:8080/web-demo/user/select` 是否能访问到 demo9 还是 demo8 的 doGet 方法?

答案是: 能、能、demo8, 进而我们可以得到的结论是 `/user/*` 中的 `*` 代表的是零或多个层级访问目录同时精确匹配优先级要高于目录匹配。

◦ 扩展名匹配

● 配置路径: `@WebServlet("*.do")`

`localhost:8080/web-demo/aaa.do`

● 访问路径:

`localhost:8080/web-demo/bbb.do`

```
1 package com.itheima.web;  
2  
3 import javax.servlet.ServletRequest;  
4 import javax.servlet.ServletResponse;  
5 import javax.servlet.annotation.WebServlet;  
6  
7 /**  
8  * UrlPattern:  
9  * * 扩展名匹配: *.do  
10 */  
11 @WebServlet(urlPatterns = "*.do")  
12 public class ServletDemo10 extends MyHttpServlet {  
13  
14     @Override  
15     protected void doGet(ServletRequest req, ServletResponse res) {  
16  
17         System.out.println("demo10 get...");  
18     }  
19     @Override  
20     protected void doPost(ServletRequest req, ServletResponse res) {  
21     }  
22 }
```

访问路径 `http://localhost:8080/web-demo/任意.do`

注意:

1. 如果路径配置的不是扩展名，那么在路径的前面就必须要加 / 否则会报错

```
Caused by: java.lang.IllegalArgumentException: Invalid <url-pattern> user/* in servlet mapping
    at org.apache.catalina.core.StandardContext.addServletMapping(StandardContext.java:3245)
    at org.apache.catalina.core.StandardContext.addServletMapping(StandardContext.java:3220)
    at org.apache.catalina.deploy.WebXml.configureContext(WebXml.java:1367)
    at org.apache.catalina.startup.ContextConfig.webConfig(ContextConfig.java:1346)
    at org.apache.catalina.startup.ContextConfig.configureStart(ContextConfig.java:878)
    at org.apache.catalina.startup.ContextConfig.lifecycleEvent(ContextConfig.java:376)
    at org.apache.catalina.util.LifecycleSupport.fireLifecycleEvent(LifecycleSupport.java:119)
    at org.apache.catalina.util.LifecycleBase.fireLifecycleEvent(LifecycleBase.java:90)
    at org.apache.catalina.core.StandardContext.startInternal(StandardContext.java:5322)
    at org.apache.catalina.util.LifecycleBase.start(LifecycleBase.java:150)
    ... 6 more
```

2. 如果路径配置的是 *.do ,那么在*.do的前面不能加 / ,否则会报错

```
Caused by: java.lang.IllegalArgumentException: Invalid <url-pattern> /*.do in servlet mapping
    at org.apache.catalina.core.StandardContext.addServletMapping(StandardContext.java:3245)
    at org.apache.catalina.core.StandardContext.addServletMapping(StandardContext.java:3220)
    at org.apache.catalina.deploy.WebXml.configureContext(WebXml.java:1367)
    at org.apache.catalina.startup.ContextConfig.webConfig(ContextConfig.java:1346)
    at org.apache.catalina.startup.ContextConfig.configureStart(ContextConfig.java:878)
    at org.apache.catalina.startup.ContextConfig.lifecycleEvent(ContextConfig.java:376)
    at org.apache.catalina.util.LifecycleSupport.fireLifecycleEvent(LifecycleSupport.java:119)
    at org.apache.catalina.util.LifecycleBase.fireLifecycleEvent(LifecycleBase.java:90)
    at org.apache.catalina.core.StandardContext.startInternal(StandardContext.java:5322)
    at org.apache.catalina.util.LifecycleBase.start(LifecycleBase.java:150)
    ... 6 more
```

- 任意匹配

● 配置路径: `@WebServlet("/")`

● 访问路径: `localhost:8080/web-demo/hehe`

`localhost:8080/web-demo/haha`

```
1 package com.itheima.web;
2
3 import javax.servlet.ServletRequest;
4 import javax.servlet.ServletResponse;
5 import javax.servlet.annotation.WebServlet;
6
7 /**
8 * UrlPattern:
9 * * * 任意匹配: /
10 */
11 @WebServlet(urlPatterns = "/")
12 public class ServletDemo11 extends MyHttpServlet {
13
14     @Override
15     protected void doGet(ServletRequest req, ServletResponse res) {
16
17         System.out.println("demo11 get...");
```

```
19     @Override  
20     protected void doPost(HttpServletRequest req, HttpServletResponse res) {  
21     }  
22 }
```

访问路径 <http://localhost:8080/demo-web/>任意

```
1 package com.itheima.web;  
2  
3 import javax.servlet.ServletRequest;  
4 import javax.servlet.ServletResponse;  
5 import javax.servlet.annotation.WebServlet;  
6  
7 /**  
8  * UrlPattern:  
9  * * 任意匹配: /*  
10  */  
11 @WebServlet(urlPatterns = "//*")  
12 public class ServletDemo12 extends MyHttpServlet {  
13  
14     @Override  
15     protected void doGet(ServletRequest req, ServletResponse res) {  
16  
17         System.out.println("demo12 get...");  
18     }  
19     @Override  
20     protected void doPost(ServletRequest req, ServletResponse res) {  
21     }  
22 }  
23 }
```

访问路径 <http://localhost:8080/demo-web/>任意

注意: / 和 /* 的区别? (日常中使用不到这两个配置)

1. 当我们的项目中的Servlet配置了"/",会覆盖掉tomcat中的DefaultServlet,当其他的url-pattern都匹配不上时都会走这个Servlet
2. 当我们的项目中配置了"/*",意味着匹配任意访问路径
3. DefaultServlet是用来处理静态资源,如果配置了"/"会把默认的覆盖掉,就会引发请求静态资源的时候没有走默认的而是走了自定义的Servlet类,最终导致静态资源不能被访问

小结

1. urlPattern总共有四种配置方式,分别是精确匹配、目录匹配、扩展名匹配、任意匹配
2. 五种配置的优先级为 精确匹配 > 目录匹配 > 扩展名匹配 > /* > / ,无需记,以最终运行结果为准。

4.8 XML配置

前面对应Servlet的配置,我们都使用的是@WebServlet,这个是Servlet从3.0版本后开始支持注解配置,3.0版本前只支持XML配置文件的配置方法。

对于XML的配置步骤有两步:

- 编写Servlet类

```
1 package com.itheima.web;  
2
```

```
3 import javax.servlet.ServletRequest;
4 import javax.servlet.ServletResponse;
5 import javax.servlet.annotation.WebServlet;
6
7 public class ServletDemo13 extends MyHttpServlet {
8
9     @Override
10    protected void doGet(ServletRequest req, ServletResponse res) {
11
12        System.out.println("demo13 get...");}
13    }
14    @Override
15    protected void doPost(ServletRequest req, ServletResponse res) {
16    }
17 }
```

- 在web.xml中配置该Servlet

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
5         http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
6         version="4.0">
7
8
9     <!--
10        Servlet 全类名
11    -->
12    <servlet>
13        <!-- servlet的名称，名字任意-->
14        <servlet-name>demo13</servlet-name>
15        <!--servlet的类全名-->
16        <servlet-class>com.itheima.web.ServletDemo13</servlet-class>
17    </servlet>
18
19     <!--
20        Servlet 访问路径
21    -->
22    <servlet-mapping>
23        <!-- servlet的名称，要和上面的名称一致-->
24        <servlet-name>demo13</servlet-name>
25        <!-- servlet的访问路径-->
26        <url-pattern>/demo13</url-pattern>
27    </servlet-mapping>
28 </web-app>
```

这种配置方式和注解比起来，确认麻烦很多，所以建议大家使用注解来开发。但是大家要认识上面这种配置方式，因为并不是所有的项目都是基于注解开发的。

Request&Response

今日目标

- 掌握Request对象的概念与使用
- 掌握Response对象的概念与使用
- 能够完成用户登录注册案例的实现
- 能够完成SqlSessionFactory工具类的抽取

1, Request和Response的概述

Request是请求对象，Response是响应对象。这两个对象在我们使用Servlet的时候有看到：

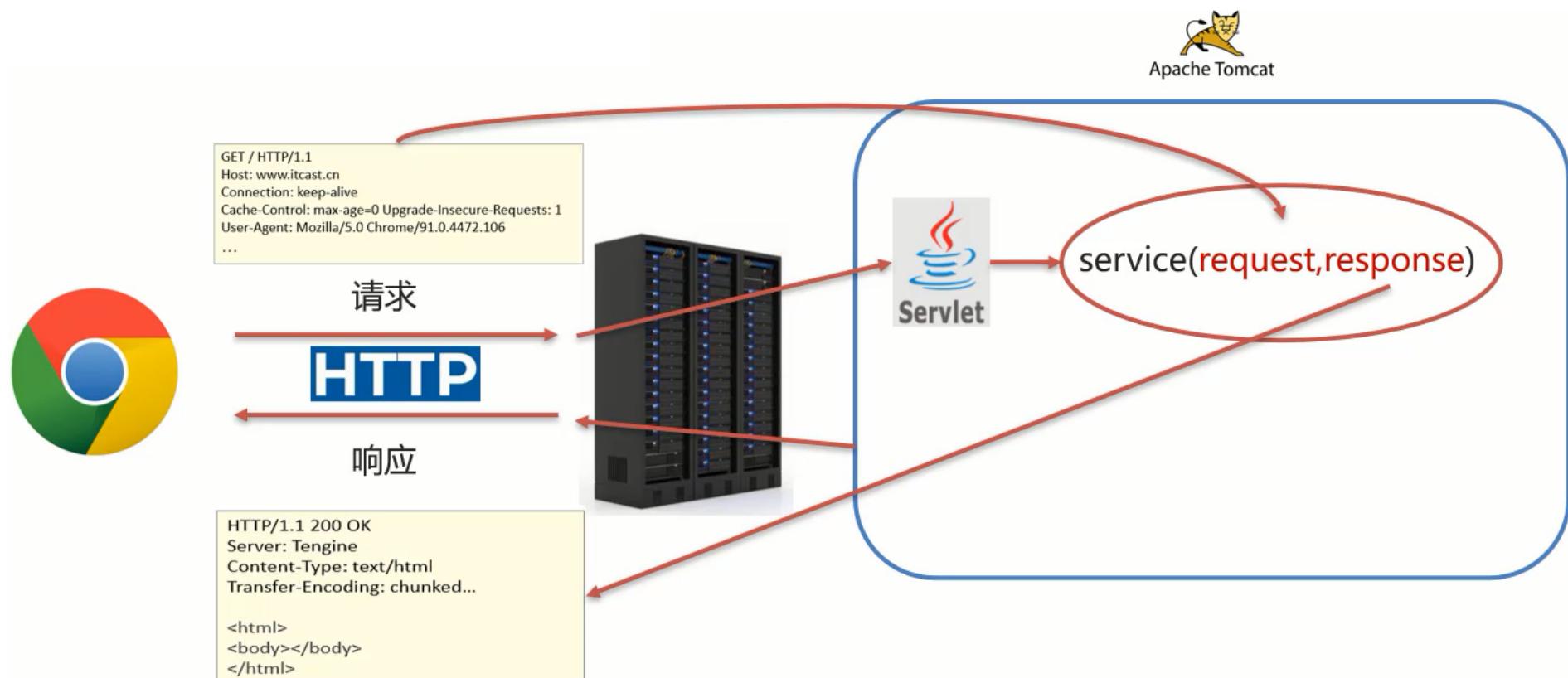
```
@WebServlet("/demo1")
public class ServletDemo1 implements Servlet{

    @Override
    public void service(ServletRequest request, ServletResponse response) throws ServletException, IOException {
        System.out.println("servlet hello ~");
    }

    @Override
    public void init(ServletConfig config) throws ServletException {

    }
}
```

此时，我们就需要思考一个问题request和response这两个参数的作用是什么？



- **request:** 获取请求数据
 - 浏览器会发送HTTP请求到后台服务器 [Tomcat]
 - HTTP的请求中会包含很多请求数据 [请求行+请求头+请求体]
 - 后台服务器 [Tomcat] 会对HTTP请求中的数据进行解析并把解析结果存入到一个对象中
 - 所存入的对象即为request对象，所以我们可以从request对象中获取请求的相关参数
 - 获取到数据后就可以继续后续的业务，比如获取用户名和密码就可以实现登录操作的相关业务
- **response:** 设置响应数据

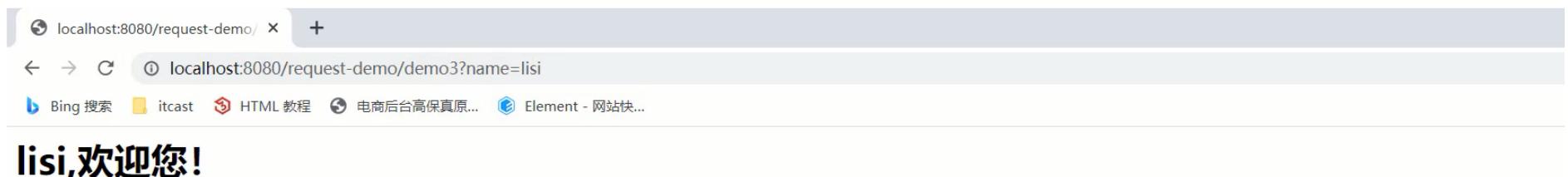
- 业务处理完后，后台就需要给前端返回业务处理的结果即响应数据
- 把响应数据封装到response对象中
- 后台服务器[Tomcat]会解析response对象，按照[响应行+响应头+响应体]格式拼接结果
- 浏览器最终解析结果，把内容展示在浏览器给用户浏览

对于上述所讲的内容，我们通过一个案例来初步体验下request和response对象的使用。

```

1 @WebServlet("/demo3")
2 public class ServletDemo3 extends HttpServlet {
3     @Override
4     protected void doGet(HttpServletRequest request, HttpServletResponse
5             response) throws ServletException, IOException {
6         //使用request对象 获取请求数据
7         String name = request.getParameter("name");//url?name=zhangsan
8
9         //使用response对象 设置响应数据
10        response.setHeader("Content-Type", "text/html; charset=utf-8");
11        response.getWriter().write("<h1>" + name + ",欢迎您! </h1>");
12    }
13
14    @Override
15    protected void doPost(HttpServletRequest request, HttpServletResponse
16             response) throws ServletException, IOException {
17        System.out.println("Post...");
18    }
19 }
```

启动成功后就可以通过浏览器来访问，并且根据传入参数的不同就可以在页面上展示不同的内容：



小结

在这节中，我们主要认识了下request对象和reponse对象：

- request对象是用来封装请求数据的对象
- response对象是用来封装响应数据的对象

目前我们只知道这两个对象是用来干什么的，那么它们具体是如何实现的，就需要我们继续深入的学习。接下来，就先从Request对象来学习，主要学习下面这些内容：

- request继承体系
- request获取请求参数
- request请求转发

2, Request对象

2.1 Request继承体系

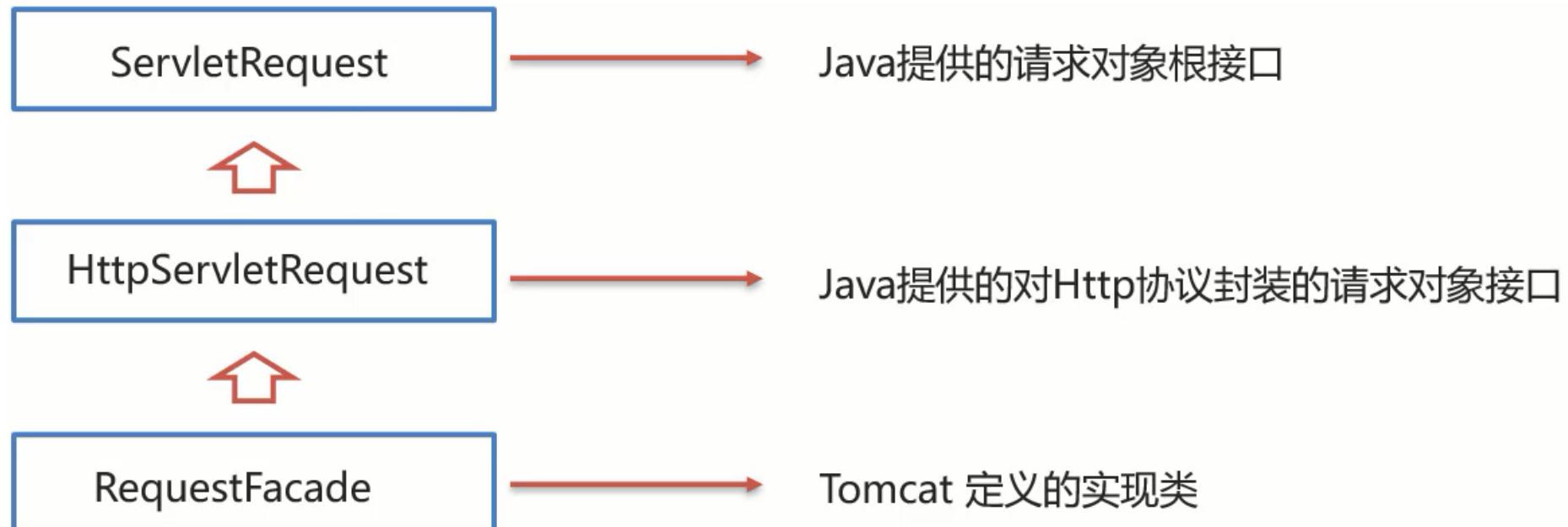
在学习这节内容之前，我们先思考一个问题，前面在介绍Request和Reponse对象的时候，比较细心的同学可能已经发现：

- 当我们的Servlet类实现的是Servlet接口的时候，service方法中的参数是ServletRequest和ServletResponse
- 当我们的Servlet类继承的是HttpServlet类的时候，doGet和doPost方法中的参数就变成HttpServletRequest和HttpServletResponse

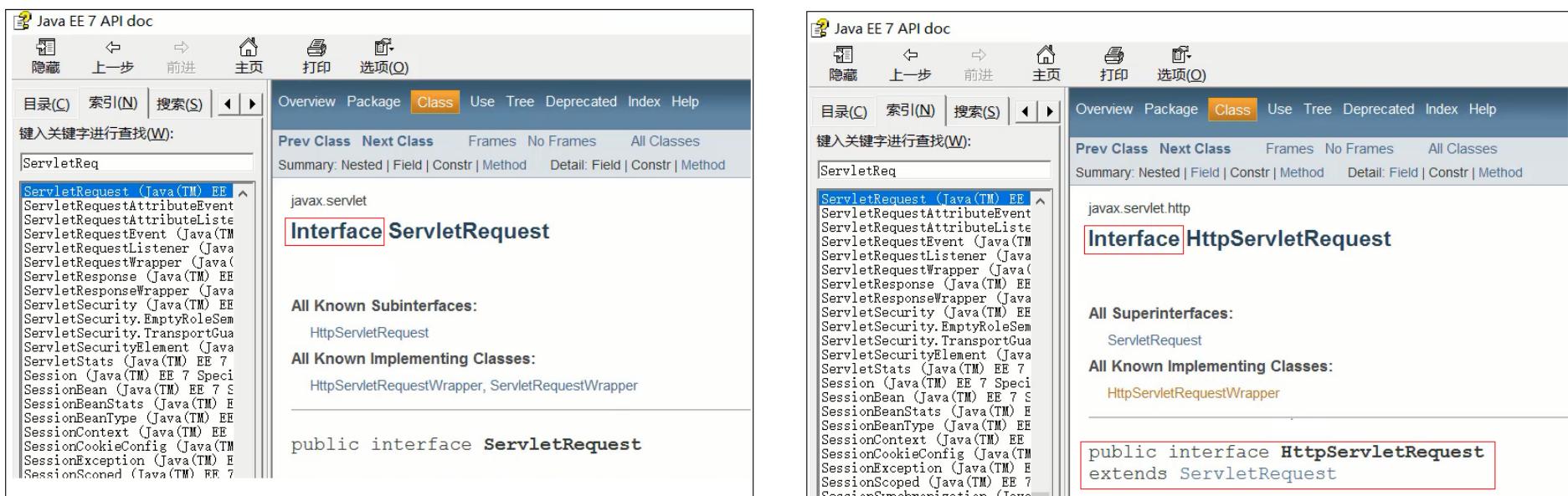
那么，

- ServletRequest和HttpServletRequest的关系是什么？
- request对象是谁来创建的？
- request提供了哪些API，这些API从哪里查？

首先，我们先来看下Request的继承体系：



从上图中可以看出，ServletRequest和HttpServletRequest都是Java提供的，所以我们可以打开JavaEE提供的API文档[参考：资料/JavaEE7-api.chm]，打开后可以看到：



所以ServletRequest和HttpServletRequest是继承关系，并且两个都是接口，接口是无法创建对象，这个时候就引发了下面这个问题：

```

public class ServletDemo1 implements Servlet {

    @Override
    public void service(ServletRequest request, ServletResponse response) throws ServletException, IOException {
        System.out.println("servlet hello~");
    }
}

@WebServlet("/demo3")
public class ServletDemo3 extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        //使用request对象 获取请求数据
        String name = request.getParameter(name: "name"); //url?name=zhangsan

        //使用response对象 设置响应数据
        response.setHeader(name: "content-type", value: "text/html;charset=utf-8");
        response.getWriter().write(s: "<h1>" + name + ", 欢迎您! </h1>");
    }
}

```

这些方法参数中的对象是由谁来创建的呢?

这个时候，我们就需要用到Request继承体系中的RequestFacade：

- 该类实现了HttpServletRequest接口，也间接实现了ServletRequest接口。
- Servlet类中的service方法、doGet方法或者是doPost方法最终都是由Web服务器[Tomcat]来调用的，所以Tomcat提供了方法参数接口的具体实现类，并完成了对象的创建
- 要想了解RequestFacade中都提供了哪些方法，我们可以直接查看JavaEE的API文档中关于ServletRequest和HttpServletRequest的接口文档，因为RequestFacade实现了其接口就需要重写接口中的方法

对于上述结论，要想验证，可以编写一个Servlet，在方法中把request对象打印下，就能看到最终的对象是不是RequestFacade，代码如下：

```

1 @WebServlet("/demo2")
2 public class ServletDemo2 extends HttpServlet {
3     @Override
4     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
5         System.out.println(request);
6     }
7
8     @Override
9     protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
10    }
11 }

```

启动服务器，运行访问<http://localhost:8080/request-demo/demo2>，得到运行结果：



小结

- Request的继承体系为ServletRequest-->HttpServletRequest-->RequestFacade
- Tomcat需要解析请求数据，封装为request对象，并且创建request对象传递到service方法
- 使用request对象，可以查阅JavaEE API文档的HttpServletRequest接口中方法说明

2.2 Request获取请求数据

HTTP请求数据总共分为三部分内容，分别是**请求行、请求头、请求体**，对于这三部分内容的数据，分别该如何获取，首先我们先来学习请求行数据如何获取？

2.2.1 获取请求行数据

请求行包含三块内容，分别是请求方式、请求资源路径、HTTP协议及版本

GET	/request-demo/req1?username=zhangsan	HTTP/1.1
请求方式	请求资源路径	HTTP协议及版本

对于这三部分内容，`request`对象都提供了对应的API方法来获取，具体如下：

- 获取请求方式：`GET`

```
1 string getMethod()
```

- 获取虚拟目录(项目访问路径)：`/request-demo`

```
1 string getContextPath()
```

- 获取URL(统一资源定位符)：`http://localhost:8080/request-demo/req1`

```
1 StringBuffer getRequestURL()
```

- 获取URI(统一资源标识符)：`/request-demo/req1`

```
1 string getRequestURI()
```

- 获取请求参数(GET方式)：`username=zhangsan&password=123`

```
1 string getQueryString()
```

介绍完上述方法后，咱们通过代码把上述方法都使用下：

```
1 /**
2  * request 获取请求数据
3 */
4 @WebServlet("/req1")
5 public class RequestDemo1 extends HttpServlet {
6     @Override
7     protected void doGet(HttpServletRequest req, HttpServletResponse resp)
8     throws ServletException, IOException {
9         // String getMethod(): 获取请求方式: GET
10        String method = req.getMethod();
11        System.out.println(method); // GET
12        // String getContextPath(): 获取虚拟目录(项目访问路径): /request-demo
```

```

12     String contextPath = req.getContextPath();
13     System.out.println(contextPath);
14     // StringBuffer getRequestURL(): 获取URL(统一资源定位符):
15     // http://localhost:8080/request-demo/req1
16     StringBuffer url = req.getRequestURL();
17     System.out.println(url.toString());
18     // String getRequestURI(): 获取URI(统一资源标识符): /request-demo/req1
19     String uri = req.getRequestURI();
20     System.out.println(uri);
21     // String getQueryString(): 获取请求参数 (GET方式) : username=zhangsan
22     String queryString = req.getQueryString();
23     System.out.println(queryString);
24 }
25 @Override
26 protected void doPost(HttpServletRequest req, HttpServletResponse resp)
27 throws ServletException, IOException {
28 }
29 }
```

启动服务器，访问 `http://localhost:8080/request-demo/req1?username=zhangsan&password=123`，获取的结果如下：

```

GET
/request-demo
http://localhost:8080/request-demo/req1
/request-demo/req1
username=zhangsan&password=123
```

2.2.2 获取请求头数据

对于请求头的数据，格式为 `key: value` 如下：

User-Agent: Mozilla/5.0 Chrome/91.0.4472.106

所以根据请求头名称获取对应值的方法为：

```
1 string getHeader(String name)
```

接下来，在代码中如果想要获取客户端浏览器的版本信息，则可以使用

```

1 /**
2  * request 获取请求数据
3 */
4 @WebServlet("/req1")
5 public class RequestDemo1 extends HttpServlet {
6     @Override
7     protected void doGet(HttpServletRequest req, HttpServletResponse resp)
8     throws ServletException, IOException {
```

```
8     //获取请求头: user-agent: 浏览器的版本信息
9     String agent = req.getHeader("user-agent");
10    System.out.println(agent);
11 }
12 @Override
13 protected void doPost(HttpServletRequest req, HttpServletResponse resp)
14 throws ServletException, IOException {
15 }
16 }
```

重新启动服务器后, `http://localhost:8080/request-demo/req1?username=zhangsan&password=123`, 获取的结果如下:

```
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/92.0.4515.131 Safari/537.36
```

2.2.3 获取请求体数据

浏览器在发送GET请求的时候是没有请求体的, 所以需要把请求方式变更为POST, 请求体中的数据格式如下:

```
username=superbaby&password=123
```

对于请求体中的数据, Request对象提供了如下两种方式来获取其中的数据, 分别是:

- 获取字节输入流, 如果前端发送的是字节数据, 比如传递的是文件数据, 则使用该方法

```
1 ServletInputStream getInputStream()
2 该方法可以获取字节
```

- 获取字符输入流, 如果前端发送的是纯文本数据, 则使用该方法

```
1 BufferedReader getReader()
```

接下来, 大家需要思考, 要想获取到请求体的内容该如何实现?

具体实现的步骤如下:

1. 准备一个页面, 在页面中添加form表单, 用来发送post请求
2. 在Servlet的doPost方法中获取请求体数据
3. 在doPost方法中使用request的getReader()或者getInputStream()来获取
4. 访问测试

1. 在项目的webapp目录下添加一个html页面, 名称为: `req.html`

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
```

```
4     <meta charset="UTF-8">
5     <title>Title</title>
6 </head>
7 <body>
8 <!--
9     action: form表单提交的请求地址
10    method: 请求方式, 指定为post
11 -->
12 <form action="/request-demo/req1" method="post">
13     <input type="text" name="username">
14     <input type="password" name="password">
15     <input type="submit">
16 </form>
17 </body>
18 </html>
```

2. 在Servlet的doPost方法中获取数据

```
1 /**
2  * request 获取请求数据
3 */
4 @WebServlet("/req1")
5 public class RequestDemo1 extends HttpServlet {
6     @Override
7     protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
8     }
9     @Override
10    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
11        //在此处获取请求体中的数据
12    }
13 }
```

3. 调用getReader()或者getInputStream()方法, 因为目前前端传递的是纯文本数据, 所以我们采用getReader()方法来获取

```
1 /**
2  * request 获取请求数据
3 */
4 @WebServlet("/req1")
5 public class RequestDemo1 extends HttpServlet {
6     @Override
7     protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
8     }
9     @Override
```

```

10 protected void doPost(HttpServletRequest req, HttpServletResponse resp)
11     throws ServletException, IOException {
12     //获取post 请求体: 请求参数
13     //1. 获取字符输入流
14     BufferedReader br = req.getReader();
15     //2. 读取数据
16     String line = br.readLine();
17     System.out.println(line);
18 }

```

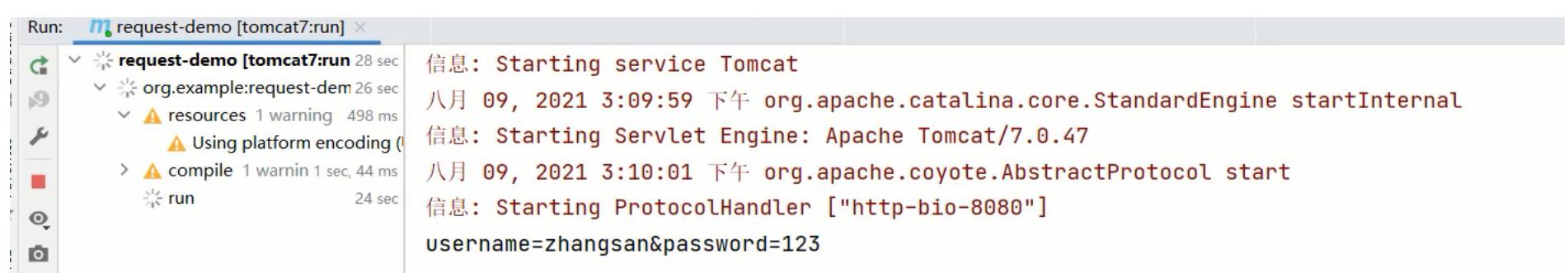
注意

BufferedReader流是通过request对象来获取的，当请求完成后request对象就会被销毁，request对象被销毁后，BufferedReader流就会自动关闭，所以此处就不需要手动关闭流了。

4. 启动服务器，通过浏览器访问 `http://localhost:8080/request-demo/req.html`



点击提交按钮后，就可以在控制台看到前端所发送的请求数据



小结

HTTP请求数据中包含了请求行、请求头和请求体，针对这三部分内容，Request对象都提供了对应的API方法来获取对应的值：

- 请求行
 - `getMethod()` 获取请求方式
 - `getContextPath()` 获取项目访问路径
 - `getRequestURL()` 获取请求URL
 - `getRequestURI()` 获取请求URI
 - `getQueryString()` 获取GET请求方式的请求参数
- 请求头
 - `getHeader(String name)` 根据请求头名称获取其对应的值
- 请求体
 - 注意： 浏览器发送的POST请求才有请求体
 - 如果是纯文本数据：`getReader()`

- 如果是字节数据如文件数据: `getInputStream()`

2.2.4 获取请求参数的通用方式

在学习下面内容之前，我们先提出两个问题：

- 什么是请求参数？
- 请求参数和请求数据的关系是什么？

1. 什么是请求参数？

为了能更好的回答上述两个问题，我们拿用户登录的例子来说明

- 想要登录网址，需要进入登录页面
- 在登录页面输入用户名和密码
- 将用户名和密码提交到后台
- 后台校验用户名和密码是否正确
- 如果正确，则正常登录，如果不正确，则提示用户名或密码错误

上述例子中，用户名和密码其实就是我们所说的请求参数。

2. 什么是请求数据？

请求数据则是包含请求行、请求头和请求体的所有数据

- 请求参数和请求数据的关系是什么？
 - 请求参数是请求数据中的部分内容
 - 如果是GET请求，请求参数在请求行中
 - 如果是POST请求，请求参数一般在请求体中

对于请求参数的获取，常用的有以下两种：

- GET方式：

```
1 string getQueryString()
```

- POST方式：

```
1 BufferedReader getReader();
```

有了上述的知识储备，我们来实现一个案例需求：

- (1) 发送一个GET请求并携带用户名，后台接收后打印到控制台
- (2) 发送一个POST请求并携带用户名，后台接收后打印到控制台

此处大家需要注意的是GET请求和POST请求接收参数的方式不一样，具体实现的代码如下：

```

1 @WebServlet("/req1")
2 public class RequestDemo1 extends HttpServlet {
3     @Override
4         protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
5
6             String result = req.getQueryString();
7             System.out.println(result);
8
9         }
10    @Override
11        protected void doPost(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
12            BufferedReader br = req.getReader();
13            String result = br.readLine();
14            System.out.println(result);
15        }
16 }

```

- 对于上述的代码，会存在什么问题呢？

```

@WebServlet("/req1")
public class RequestDemo1 extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        String result = req.getQueryString();
        System.out.println(result);
    }
    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        BufferedReader br = req.getReader();
        String result = br.readLine();
        System.out.println(result);
    }
}

```

由于获取请求参数的方式不一样，导致doGet和doPost中出现了重复代码。这行打印大家可以理解为很多相同的业务代码。

- 如何解决上述重复代码的问题呢？

```

@WebServlet("/req1")
public class RequestDemo1 extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        // 获取请求参数
        // 将请求参数进行打印控制台
    }
    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        this.doGet(req, resp);
    }
}

```

当然，也可以在doGet中调用doPost，在doPost中完成参数的获取和打印，另外需要注意的是，doGet和doPost方法都必须存在，不能删除任意一个。

GET请求和POST请求获取请求参数的方式不一样，在获取请求参数这块该如何实现呢？

要想实现，我们就需要**思考**：

GET请求方式和POST请求方式区别主要在于获取请求参数的方式不一样，是否可以提供一种统一获取请求参数的方式，从而统一doGet和doPost方法内的代码？

解决方案一：

```
1 @WebServlet("/req1")
2 public class RequestDemo1 extends HttpServlet {
3     @Override
4     protected void doGet(HttpServletRequest req, HttpServletResponse resp)
5         throws ServletException, IOException {
6         //获取请求方式
7         String method = req.getMethod();
8         //获取请求参数
9         String params = "";
10        if("GET".equals(method)){
11            params = req.getQueryString();
12        }else if("POST".equals(method)){
13            BufferedReader reader = req.getReader();
14            params = reader.readLine();
15        }
16        //将请求参数进行打印控制台
17        System.out.println(params);
18    }
19    @Override
20    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
21        throws ServletException, IOException {
22        this.doGet(req,resp);
23    }
}
```

使用request的getMethod()来获取请求方式，根据请求方式的不同分别获取请求参数值，这样就可以解决上述问题，但是以后每个Servlet都需要这样写代码，实现起来比较麻烦，这种方案我们不采用

解决方案二：

request对象已经将上述获取请求参数的方法进行了封装，并且request提供的方法实现的功能更大，以后只需要调用request提供的方法即可，在request的方法中都实现了哪些操作？

(1) 根据不同的请求方式获取请求参数，获取的内容如下：

username=zhangsan&hobby=1&hobby=2

(2) 把获取到的内容进行分割，内容如下：

username=zhangsan&hobby=1&hobby=2

username=zhangsan

hobby=1

hobby=2

username

zhangsan

hobby

1

hobby

2

(3) 把分割后端数据，存入到一个Map集合中：



Map<String, String[]>

注意：因为参数的值可能是一个，也可能有多个，所以Map的值的类型为String数组。

基于上述理论，request对象为我们提供了如下方法：

- 获取所有参数Map集合

```
1 Map<String, String[]> getParameterMap()
```

- 根据名称获取参数值（数组）

```
1 String[] getParameterValues(String name)
```

- 根据名称获取参数值（单个值）

```
1 String getParameter(String name)
```

接下来，我们通过案例来把上述的三个方法进行实例演示：

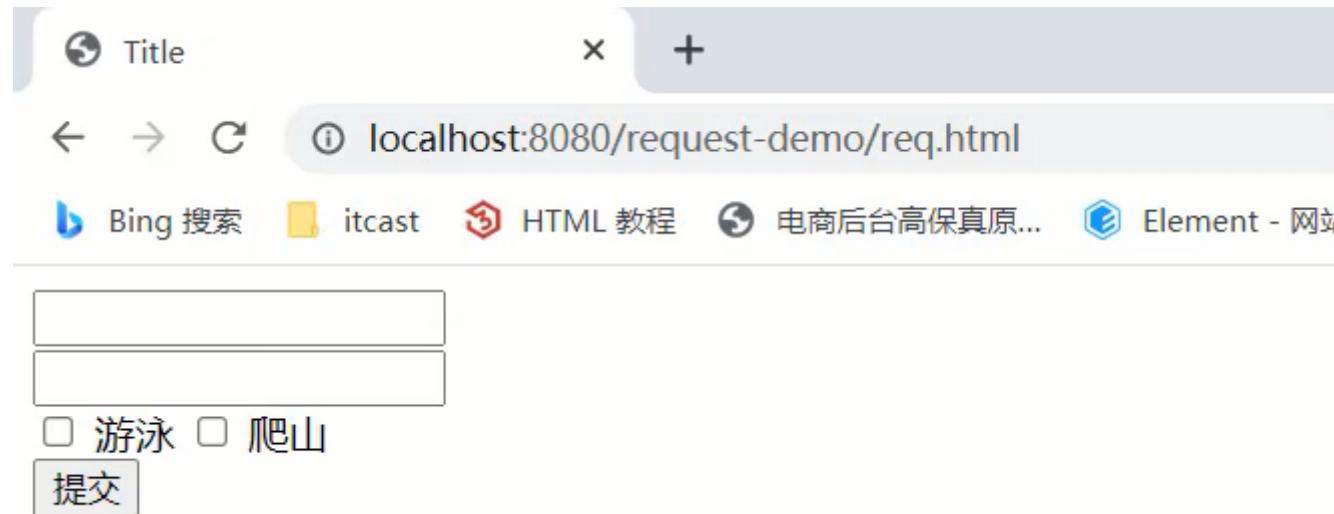
1. 修改req.html页面，添加爱好选项，爱好可以同时选多个

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>Title</title>
6 </head>
7 <body>
```

```

8 <form action="/request-demo/req2" method="get">
9   <input type="text" name="username"><br>
10  <input type="password" name="password"><br>
11  <input type="checkbox" name="hobby" value="1"> 游泳
12  <input type="checkbox" name="hobby" value="2"> 爬山 <br>
13  <input type="submit">
14
15 </form>
16 </body>
17 </html>

```



2. 在Servlet代码中获取页面传递GET请求的参数值

2.1 获取GET方式的所有请求参数

```

1 /**
2  * request 通用方式获取请求参数
3 */
4 @WebServlet("/req2")
5 public class RequestDemo2 extends HttpServlet {
6     @Override
7     protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
8         //GET请求逻辑
9         System.out.println("get....");
10        //1. 获取所有参数的Map集合
11        Map<String, String[]> map = req.getParameterMap();
12        for (String key : map.keySet()) {
13            // username:zhangsan lisi
14            System.out.print(key+":");
15
16            //获取值
17            String[] values = map.get(key);
18            for (String value : values) {
19                System.out.print(value + " ");
20            }
21
22            System.out.println();

```

```
23     }
24 }
25
26     @Override
27     protected void doPost(HttpServletRequest req, HttpServletResponse resp)
28         throws ServletException, IOException {
29 }
```

获取的结果为：

```
get....
username:zhangsan
password:123
hobby:1 2
```

2.2 获取GET请求参数中的爱好，结果是数组值

```
1 /**
2  * request 通用方式获取请求参数
3 */
4 @WebServlet("/req2")
5 public class RequestDemo2 extends HttpServlet {
6     @Override
7     protected void doGet(HttpServletRequest req, HttpServletResponse resp)
8         throws ServletException, IOException {
9         //GET请求逻辑
10        //...
11        System.out.println("-----");
12        String[] hobbies = req.getParameterValues("hobby");
13        for (String hobby : hobbies) {
14            System.out.println(hobby);
15        }
16    }
17    @Override
18    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
19        throws ServletException, IOException {
20 }
```

获取的结果为：

```
-----
1
2
```

2.3 获取GET请求参数中的用户名和密码，结果是单个值

```
1 /**
2  * request 通用方式获取请求参数
3 */
4 @WebServlet("/req2")
5 public class RequestDemo2 extends HttpServlet {
6     @Override
7     protected void doGet(HttpServletRequest req, HttpServletResponse resp)
8     throws ServletException, IOException {
9         //GET请求逻辑
10        //...
11        String username = req.getParameter("username");
12        String password = req.getParameter("password");
13        System.out.println(username);
14        System.out.println(password);
15    }
16    @Override
17    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
18    throws ServletException, IOException {
19 }
```

获取的结果为：

```
zhangsan
123
```

3. 在Servlet代码中获取页面传递POST请求的参数值

3.1 将req.html页面form表单的提交方式改成post

3.2 将doGet方法中的内容复制到doPost方法中即可

小结

- req.getParameter()方法使用的频率会比较高
- 以后我们再写代码的时候，就只需要按照如下格式来编写：

```
1 public class RequestDemo1 extends HttpServlet {  
2     @Override  
3     protected void doGet(HttpServletRequest req, HttpServletResponse resp)  
throws ServletException, IOException {  
4         //采用request提供的获取请求参数的通用方式来获取请求参数  
5         //编写其他的业务代码...  
6     }  
7     @Override  
8     protected void doPost(HttpServletRequest req, HttpServletResponse resp)  
throws ServletException, IOException {  
9         this.doGet(req,resp);  
10    }  
11 }
```

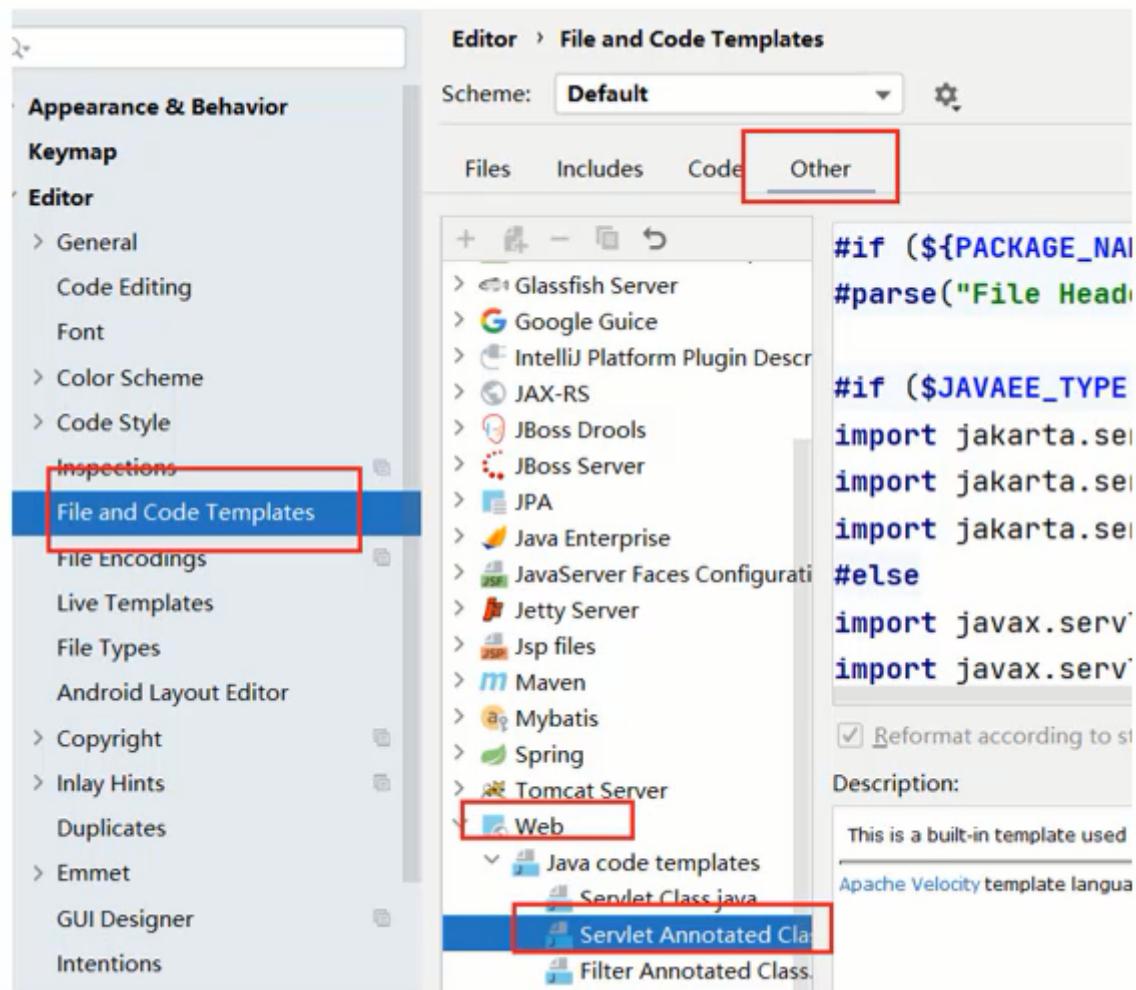
2.3 IDEA快速创建Servlet

使用通用方式获取请求参数后，屏蔽了GET和POST的请求方式代码的不同，则代码可以定义如下格式：

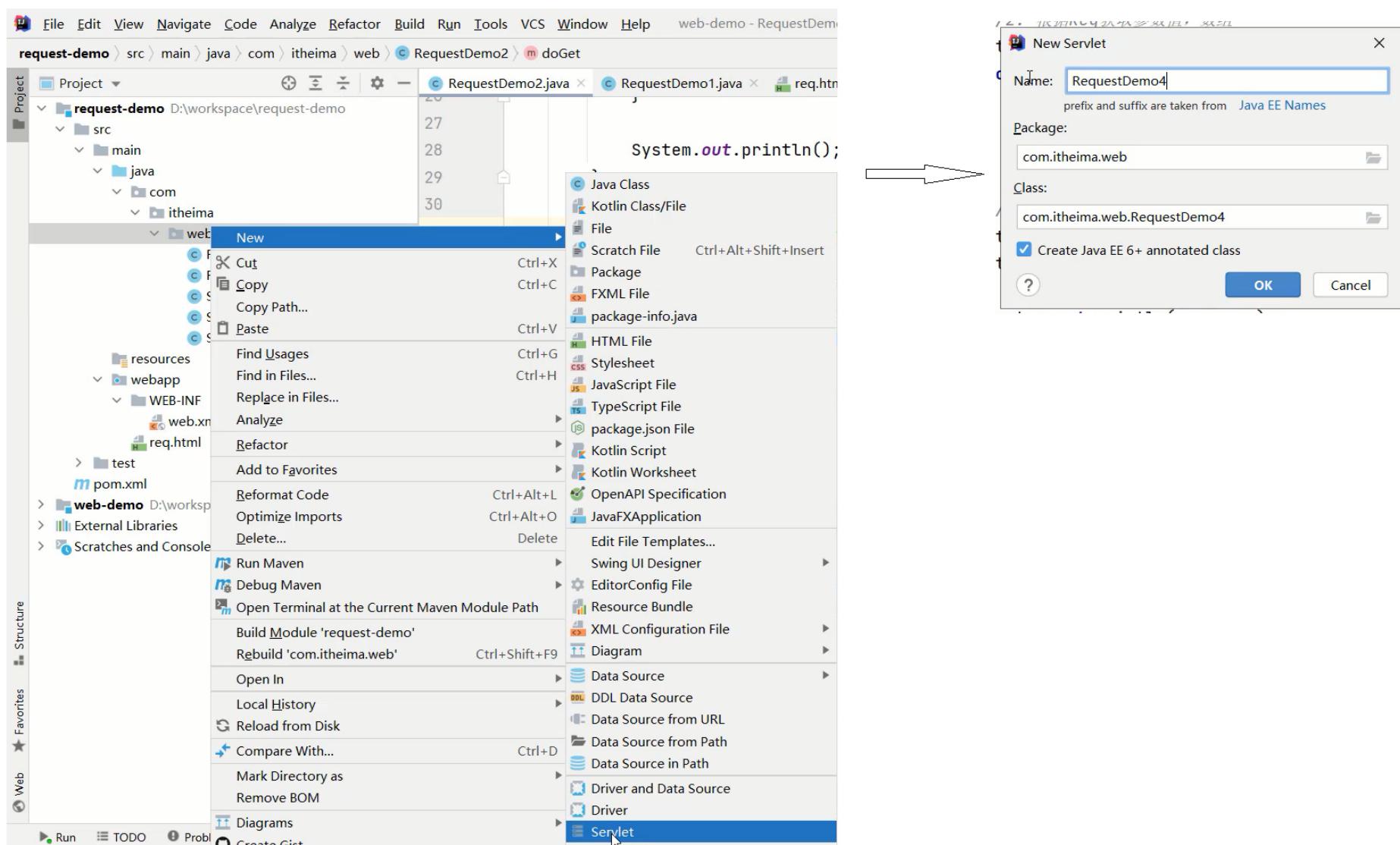
```
@WebServlet("/reqDemo3")  
public class RequestDemo3 extends HttpServlet {  
    @Override  
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) {  
        //  
    }  
    @Override  
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) {  
        this.doGet(req,resp);  
    }  
}
```

由于格式固定，所以我们可以使用IDEA提供的模板来制作一个Servlet的模板，这样我们后期在创建Servlet的时候就会更高效，具体如何实现：

- (1) 按照自己的需求，修改Servlet创建的模板内容



(2) 使用servlet模板创建Servlet类



2.4 请求参数中文乱码问题

问题展示：

(1) 将req.html页面的请求方式修改为get

```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
```

```

4   <meta charset="UTF-8">
5   <title>Title</title>
6 </head>
7 <body>
8 <form action="/request-demo/req2" method="get">
9   <input type="text" name="username"><br>
10  <input type="password" name="password"><br>
11  <input type="checkbox" name="hobby" value="1"> 游泳
12  <input type="checkbox" name="hobby" value="2"> 爬山 <br>
13  <input type="submit">
14
15 </form>
16 </body>
17 </html>

```

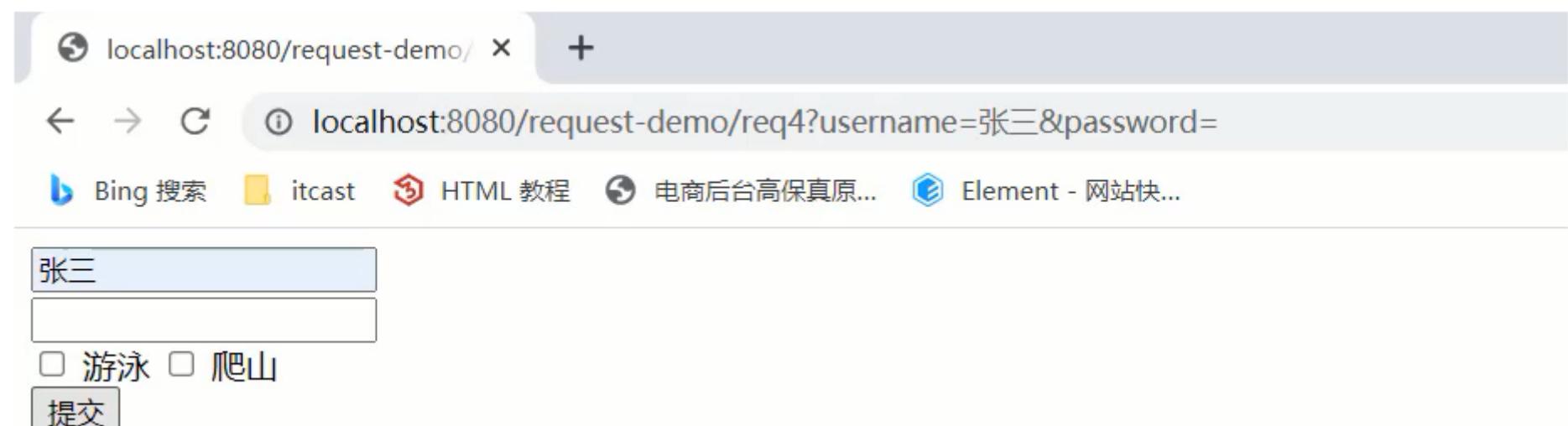
(2) 在Servlet方法中获取参数，并打印

```

1 /**
2  * 中文乱码问题解决方案
3 */
4 @WebServlet("/req4")
5 public class RequestDemo4 extends HttpServlet {
6     @Override
7     protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
8         //1. 获取username
9         String username = request.getParameter("username");
10        System.out.println(username);
11    }
12
13    @Override
14    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
15        this.doGet(request, response);
16    }
17 }

```

(3) 启动服务器，页面上输入中文参数



(4) 查看控制台打印内容

å% ä,

(5) 把req.html页面的请求方式改成post,再次发送请求和中文参数

The screenshot shows a browser window with the URL `localhost:8080/request-demo/req4`. The page has input fields for '张三' (Zhang San) and '地址' (Address). There are two checkboxes for '游泳' (Swimming) and '爬山' (Hiking). A '提交' (Submit) button is at the bottom. The browser tabs show other pages like Bing 搜索 and HTML 教程.

(6) 查看控制台打印内容，依然为乱码

å% ä,

通过上面的案例，会发现，不管是GET还是POST请求，在发送的请求参数中如果有中文，在后台接收的时候，都会出现中文乱码的问题。具体该如何解决呢？

2.4.1 POST请求解决方案

- 分析出现中文乱码的原因：

- POST的请求参数是通过`request.getReader()`来获取流中的数据
- TOMCAT在获取流的时候采用的编码是ISO-8859-1
- ISO-8859-1编码是不支持中文的，所以会出现乱码

- 解决方案：

- 页面设置的编码格式为UTF-8
- 把TOMCAT在获取流数据之前的编码设置为UTF-8
- 通过`request.setCharacterEncoding("UTF-8")`设置编码，UTF-8也可以写成小写

修改后的代码为：

```
1 /**
2  * 中文乱码问题解决方案
3 */
4 @WebServlet("/req4")
5 public class RequestDemo4 extends HttpServlet {
6     @Override
7     protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
8         //1. 解决乱码：POST getReader()
9         //设置字符输入流的编码，设置的字符集要和页面保持一致
10        request.setCharacterEncoding("UTF-8");
11        //2. 获取username
```

```

12     String username = request.getParameter("username");
13     System.out.println(username);
14 }
15
16 @Override
17 protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
18     this.doGet(request, response);
19 }
20 }
```

重新发送POST请求，就会在控制台看到正常展示的中文结果。

至此POST请求中文乱码的问题就已经解决，但是这种方案不适用于GET请求，这个原因是什么呢，咱们下面再分析。

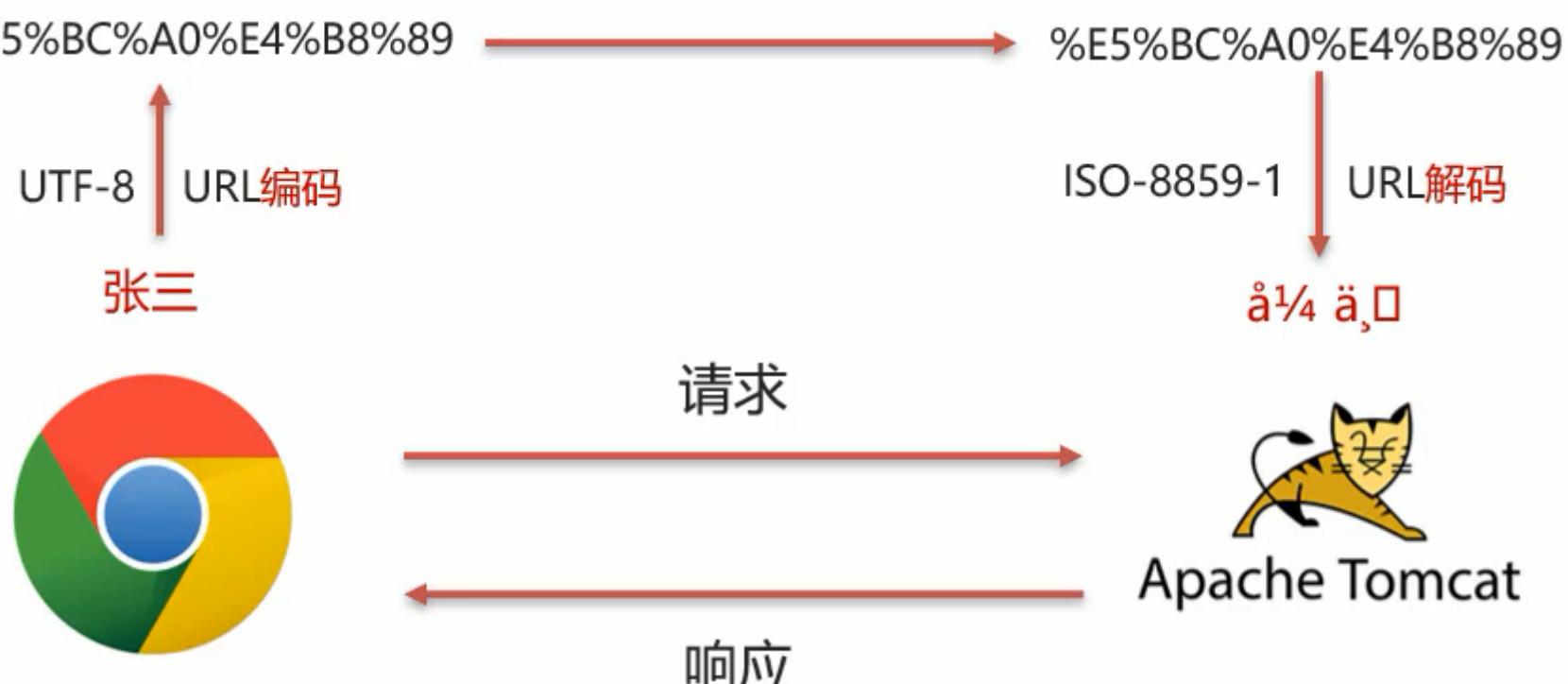
2.4.2 GET请求解决方案

刚才提到一个问题就是POST请求的中文乱码解决方案为什么不适合GET请求？

- GET请求获取请求参数的方式是`request.getQueryString()`
- POST请求获取请求参数的方式是`request.getReader()`
- `request.setCharacterEncoding("utf-8")`是设置`request`处理流的编码
- `getQueryString`方法并没有通过流的方式获取数据

所以GET请求不能用设置编码的方式来解决中文乱码问题，那问题又来了，如何解决GET请求的中文乱码呢？

1. 首先我们需要先分析下GET请求出现乱码的原因：



(1) 浏览器通过HTTP协议发送请求和数据给后台服务器（Tomcat）

(2) 浏览器在发送HTTP的过程中会对中文数据进行URL**编码**

(3) 在进行URL编码的时候会采用页面`<meta>`标签指定的UTF-8的方式进行编码，张三编码后的结果为`%E5%BC%A0%E4%B8%89`

(4) 后台服务器(Tomcat)接收到%E5%BC%A0%E4%B8%89后会默认按照ISO-8859-1进行URL解码

(5) 由于前后编码与解码采用的格式不一样，就会导致后台获取到的数据为乱码。

思考：如果把req.html页面的<meta>标签的charset属性改成ISO-8859-1，后台不做操作，能解决中文乱码问题么？

答案是否定的，因为ISO-8859-1本身是不支持中文展示的，所以改了标签的charset属性后，会导致页面上的中文内容都无法正常展示。

分析完上面的问题后，我们会发现，其中有两个我们不熟悉的内容就是URL编码和URL解码，什么是URL编码，什么又是URL解码呢？

URL编码

这块知识我们只需要了解下即可，具体编码过程分两步，分别是：

(1) 将字符串按照编码方式转为二进制

(2) 每个字节转为2个16进制数并在前边加上%

张三按照UTF-8的方式转换成二进制的结果为：

```
1 1110 0101 1011 1100 1010 0000 1110 0100 1011 1000 1000 1001
```

这个结果是如何计算的？

使用http://www.mytju.com/classcode/tools/encode_utf8.asp，输入张三

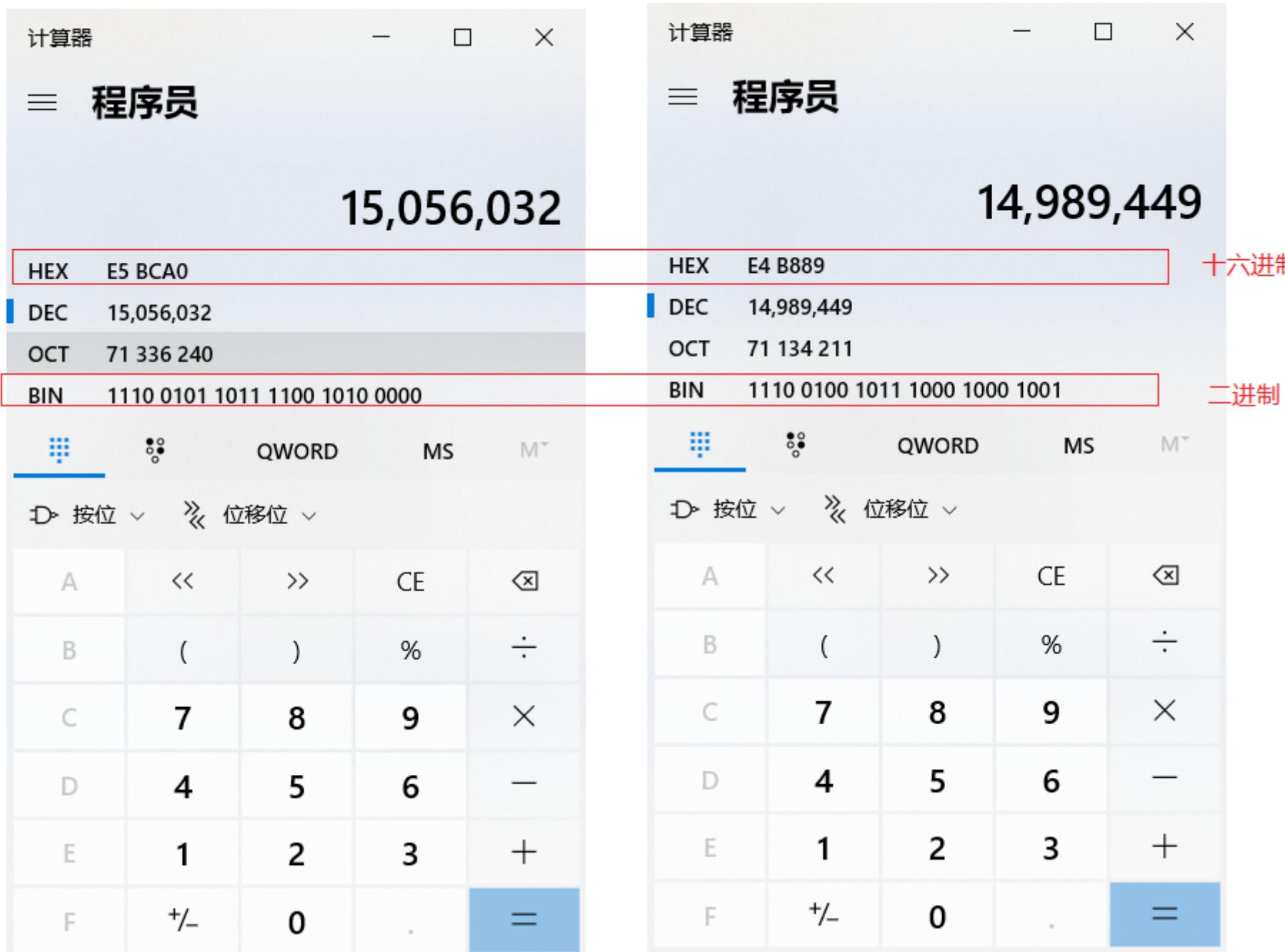
查看字符编码 (UTF-8)

请输入文字 (可以输入多个) :
如：“春眠不觉晓，处处闻啼鸟。”

请输入编码 (只可输入一个) :
如：“&HB4BA”，“65”

字符	编码10进制	编码16进制	Unicode编码10进制	Unicode编码16进制
张	15056032	E5BCA0	24352	5F20
三	14989449	E4B889	19977	4E09

就可以获取张和三分别对应的10进制，然后在使用计算器，选择程序员模式，计算出对应的二进制数据结果：



在计算的十六进制结果中，每两位前面加一个%，就可以获取到%E5%BC%A0%E4%B8%89。

当然你从上面所提供的网站中就已经能看到编码16进制的结果了：

查看字符编码 (UTF-8)

请输入文字 (可以输入多个) :	<input type="text" value="张三"/>	<input type="button" value="查看编码"/>		
如：“春眠不觉晓，处处闻啼鸟。”				
请输入编码 (只可输入一个) :	<input type="text" value="E5B889"/>	<input type="button" value="查看文字"/>		
如：“&HB4BA”，“65”				
字符	编码10进制	编码16进制	Unicode编码10进制	Unicode编码16进制
张	15056032	E5BCA0	24352	5F20
三	14989449	E4B889	19977	4E09

但是对于上面的计算过程，如果没有工具，纯手工计算的话，相对来说还是比较复杂的，我们也不需要进行手动计算，在Java中已经为我们提供了编码和解码的API工具类可以让我们更快速的进行编码和解码：

编码：

```
1 java.net.URLEncoder.encode("需要被编码的内容", "字符集(UTF-8)")
```

解码：

```
1 java.net.URLDecoder.decode("需要被解码的内容", "字符集(UTF-8)")
```

接下来咱们对张三来进行编码和解码

```
1 public class URLEmo {
```

```

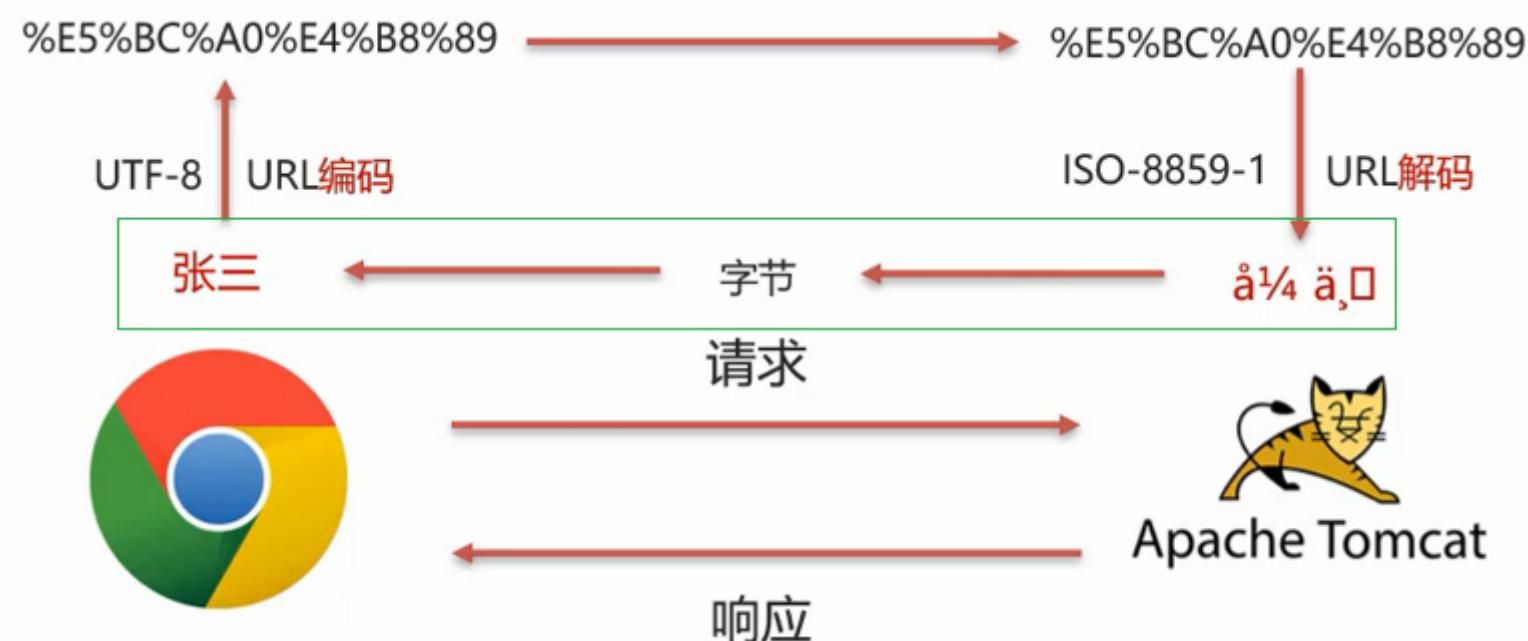
2
3     public static void main(String[] args) throws UnsupportedEncodingException
{
4         String username = "张三";
5         //1. URL编码
6         String encode = URLEncoder.encode(username, "utf-8");
7         System.out.println(encode); //打印:%E5%BC%A0%E4%B8%89
8
9         //2. URL解码
10        //String decode = URLDecoder.decode(encode, "utf-8"); //打印:张三
11        String decode = URLDecoder.decode(encode, "ISO-8859-1"); //打印:`å¼ ä, `
12        System.out.println(decode);
13    }
14 }
15

```

到这，我们就可以分析出GET请求中文参数出现乱码的原因了，

- 浏览器把中文参数按照UTF-8进行URL编码
- Tomcat对获取到的内容进行了ISO-8859-1的URL解码
- 在控制台就会出现类似`å¼ ä, `的乱码，最后一位是个空格

2. 清楚了出现乱码的原因，接下来我们就需要想办法进行解决



从上图可以看住，

- 在进行编码和解码的时候，不管使用的是哪个字符集，他们对应的%E5%BC%A0%E4%B8%89是一致的
- 那他们对应的二进制值也是一样的，为：
 - 1 1110 0101 1011 1100 1010 0000 1110 0100 1011 1000 1000 1001
- 所以我们可以考虑把`å¼ ä, `转换成字节，在把字节转换成张三，在转换的过程中是它们的编码一致，就可以解决中文乱码问题。

具体的实现步骤为：

1. 按照ISO-8859-1编码获取乱码`å¼ ä, `对应的字节数组

2. 按照UTF-8编码获取字节数组对应的字符串

实现代码如下：

```
1 public class URLDemo {  
2  
3     public static void main(String[] args) throws UnsupportedEncodingException {  
4         String username = "张三";  
5         //1. URL编码  
6         String encode = URLEncoder.encode(username, "utf-8");  
7         System.out.println(encode);  
8         //2. URL解码  
9         String decode = URLDecoder.decode(encode, "ISO-8859-1");  
10  
11         System.out.println(decode); //此处打印的是对应的乱码数据  
12  
13         //3. 转换为字节数据, 编码  
14         byte[] bytes = decode.getBytes("ISO-8859-1");  
15         for (byte b : bytes) {  
16             System.out.print(b + " ");  
17         }  
18         //此处打印的是:-27 -68 -96 -28 -72 -119  
19         //4. 将字节数组转为字符串, 解码  
20         String s = new String(bytes, "utf-8");  
21         System.out.println(s); //此处打印的是张三  
22     }  
23 }
```

说明:在第18行中打印的数据是-27 -68 -96 -28 -72 -119和张三转换成的二进制数据1110 0101 1011 1100 1010 0000 1110 0100 1011 1000 1000 1001为什么不一样呢?

其实打印出来的是十进制数据，我们只需要使用计算机换算下就能得到他们的对应关系，如下图：



至此对于GET请求中文乱码的解决方案，我们就已经分析完了，最后在代码中去实现下：

```
1 /**  
2  * 中文乱码问题解决方案  
3  */
```

```

4  @WebServlet("/req4")
5  public class RequestDemo4 extends HttpServlet {
6      @Override
7      protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
8          //1. 解决乱码: POST, getReader()
9          //request.setCharacterEncoding("UTF-8");//设置字符输入流的编码
10
11         //2. 获取username
12         String username = request.getParameter("username");
13         System.out.println("解决乱码前: "+username);
14
15         //3. GET,获取参数的方式: getQueryString
16         // 乱码原因: tomcat进行URL解码, 默认的字符集ISO-8859-1
17         /* //3.1 先对乱码数据进行编码: 转为字节数组
18         byte[] bytes = username.getBytes(StandardCharsets.ISO_8859_1);
19         //3.2 字节数组解码
20         username = new String(bytes, StandardCharsets.UTF_8);*/
21
22         username = new
23         String(username.getBytes(StandardCharsets.ISO_8859_1), StandardCharsets.UTF_8)
24 ;
25
26     }
27
28     @Override
29     protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
30         this.doGet(request, response);
31     }
32 }

```

注意

- 把`request.setCharacterEncoding("UTF-8")`代码注释掉后, 会发现GET请求参数乱码解决方案同时也可也把POST请求参数乱码的问题也解决了
- 只不过对于POST请求参数一般都会比较多, 采用这种方式解决乱码起来比较麻烦, 所以对于POST请求还是建议使用设置编码的方式进行。

另外需要说明一点的是**Tomcat8.0之后, 已将GET请求乱码问题解决, 设置默认的解码方式为UTF-8**

小结

1. 中文乱码解决方案

- POST请求和GET请求的参数中如果有中文, 后台接收数据就会出现中文乱码问题

GET请求在Tomcat8.0以后的版本就不会出现了

- POST请求解决方案是：设置输入流的编码

```
1 request.setCharacterEncoding("UTF-8");  
2 注意：设置的字符集要和页面保持一致
```

- 通用方式（GET/POST）：需要先解码，再编码

```
1 new String(username.getBytes("ISO-8859-1"), "UTF-8");
```

2. URL编码实现方式：

- 编码：

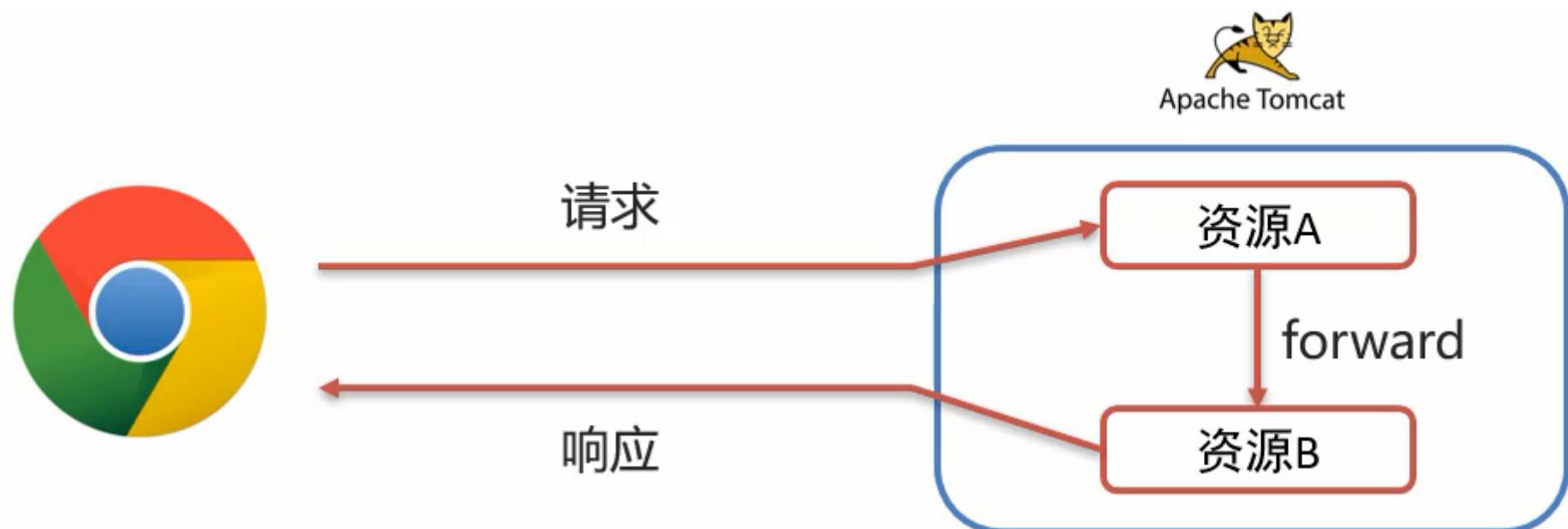
```
1 URLEncoder.encode(str, "UTF-8");
```

- 解码：

```
1 URLDecoder.decode(s, "ISO-8859-1");
```

2.5 Request请求转发

1. 请求转发（forward）：一种在服务器内部的资源跳转方式。



(1) 浏览器发送请求给服务器，服务器中对应的资源A接收到请求

(2) 资源A处理完请求后将请求发给资源B

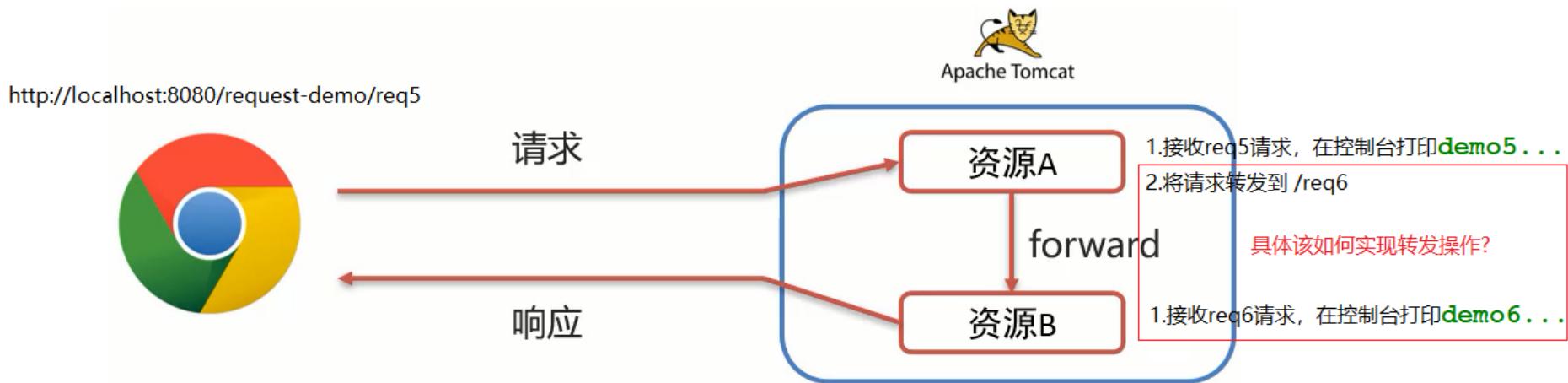
(3) 资源B处理完后将结果响应给浏览器

(4) 请求从资源A到资源B的过程就叫**请求转发**

2. 请求转发的实现方式：

```
1 req.getRequestDispatcher("资源B路径").forward(req, resp);
```

具体如何来使用，我们先来看下需求：



针对上述需求，具体的实现步骤为：

1. 创建一个 RequestDemo5 类，接收 /req5 的请求，在 doGet 方法中打印 demo5
2. 创建一个 RequestDemo6 类，接收 /req6 的请求，在 doGet 方法中打印 demo6
3. 在 RequestDemo5 的方法中使用
req.getRequestDispatcher("/req6").forward(req, resp) 进行请求转发
4. 启动测试

(1) 创建 RequestDemo5 类

```

1 /**
2  * 请求转发
3 */
4 @WebServlet("/req5")
5 public class RequestDemo5 extends HttpServlet {
6     @Override
7     protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
8         System.out.println("demo5...");
9     }
10
11    @Override
12    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
13        this.doGet(request, response);
14    }
15 }
```

(2) 创建 RequestDemo6 类

```

1 /**
2  * 请求转发
3 */
4 @WebServlet("/req6")
5 public class RequestDemo6 extends HttpServlet {
6     @Override
```

```

7     protected void doGet(HttpServletRequest request, HttpServletResponse
8         response) throws ServletException, IOException {
9             System.out.println("demo6..."); 
10        }
11    @Override
12    protected void doPost(HttpServletRequest request, HttpServletResponse
13        response) throws ServletException, IOException {
14        this.doGet(request, response);
15    }

```

(3) 在 RequestDemo5 的 doGet 方法中进行请求转发

```

1 /**
2  * 请求转发
3 */
4 @WebServlet("/req5")
5 public class RequestDemo5 extends HttpServlet {
6     @Override
7     protected void doGet(HttpServletRequest request, HttpServletResponse
8         response) throws ServletException, IOException {
9         System.out.println("demo5..."); 
10        //请求转发
11        request.getRequestDispatcher("/req6").forward(request, response);
12    }
13    @Override
14    protected void doPost(HttpServletRequest request, HttpServletResponse
15        response) throws ServletException, IOException {
16        this.doGet(request, response);
17    }

```

(4) 启动测试

访问 `http://localhost:8080/request-demo/req5`, 就可以在控制台看到如下内容:

```

demo5...
demo6...

```

说明请求已经转发到了 /req6

3. 请求转发资源间共享数据: 使用 Request 对象

此处主要解决的问题是把请求从 /req5 转发到 /req6 的时候, 如何传递数据给 /req6。

需要使用 request 对象提供的三个方法:

- 存储数据到 request 域 [范围, 数据是存储在 request 对象] 中

```
1 void setAttribute(String name, Object o);
```

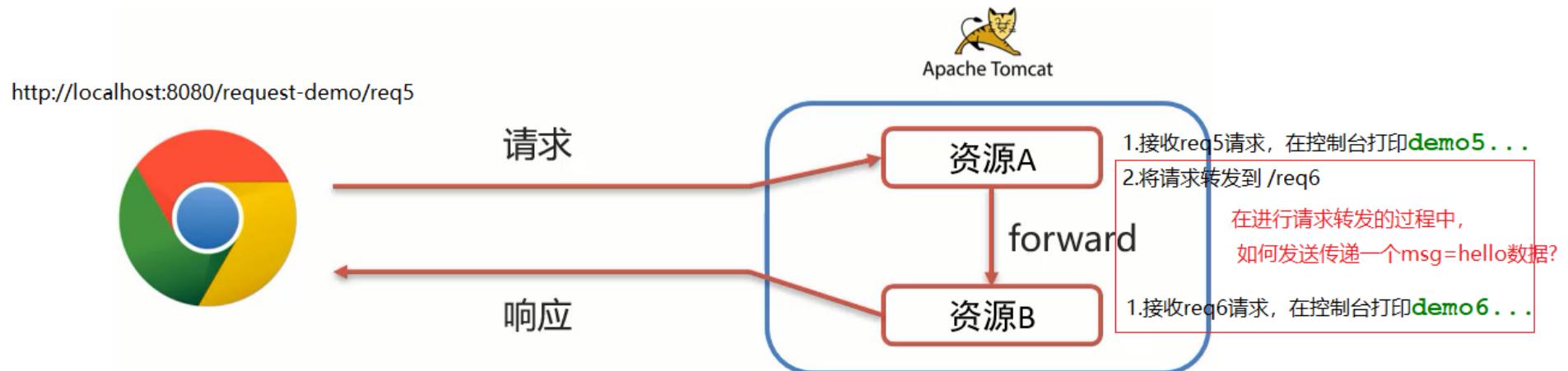
- 根据key获取值

```
1 Object getAttribute(String name);
```

- 根据key删除该键值对

```
1 void removeAttribute(String name);
```

接着上个需求来：



1. 在RequestDemo5的doGet方法中转发请求之前，将数据存入request域对象中

2. 在RequestDemo6的doGet方法从request域对象中获取数据，并将数据打印到控制台

3. 启动访问测试

(1) 修改RequestDemo5中的方法

```
1 @WebServlet("/req5")
2 public class RequestDemo5 extends HttpServlet {
3     @Override
4     protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
5         System.out.println("demo5...");
6         //存储数据
7         request.setAttribute("msg", "hello");
8         //请求转发
9         request.getRequestDispatcher("/req6").forward(request, response);
10    }
11
12    @Override
13    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
14        this.doGet(request, response);
15    }
16}
17}
```

(2) 修改RequestDemo6中的方法

```

1 /**
2  * 请求转发
3 */
4 @WebServlet("/req6")
5 public class RequestDemo6 extends HttpServlet {
6     @Override
7     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
8         System.out.println("demo6...");
9         //获取数据
10        Object msg = request.getAttribute("msg");
11        System.out.println(msg);
12    }
13
14    @Override
15    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
16        this.doGet(request, response);
17    }
18 }
19

```

(3) 启动测试

访问 `http://localhost:8080/request-demo/req5`, 就可以在控制台看到如下内容:

```

demo5...
demo6...
hello

```

此时就可以实现在转发多个资源之间共享数据。

4. 请求转发的特点

- 浏览器地址栏路径不发生变化

虽然后台从 `/req5` 转发到 `/req6`, 但是浏览器的地址一直是 `/req5`, 未发生变化



- 只能转发到当前服务器的内部资源

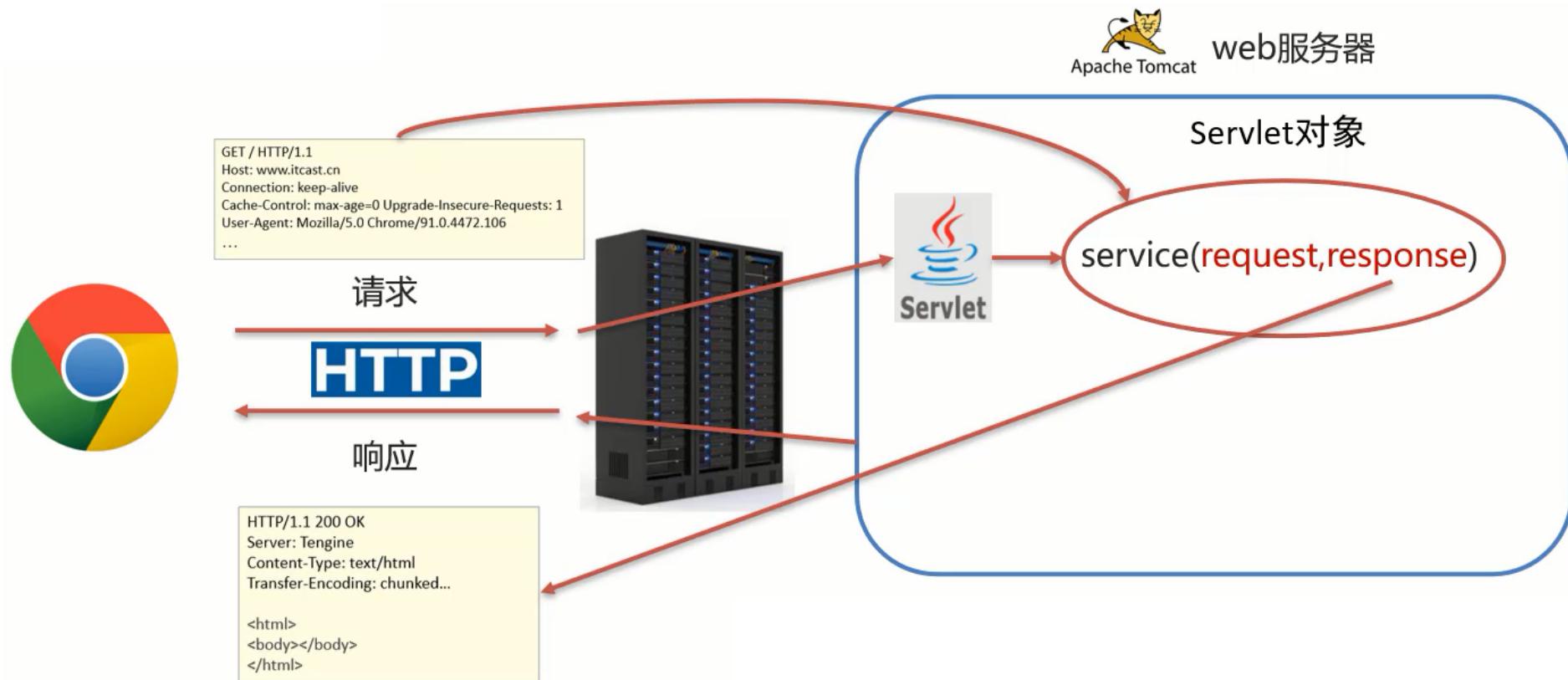
不能从一个服务器通过转发访问另一台服务器

- 一次请求, 可以在转发资源间使用 `request` 共享数据

虽然后台从 `/req5` 转发到 `/req6`, 但是这个 **只有一次请求**

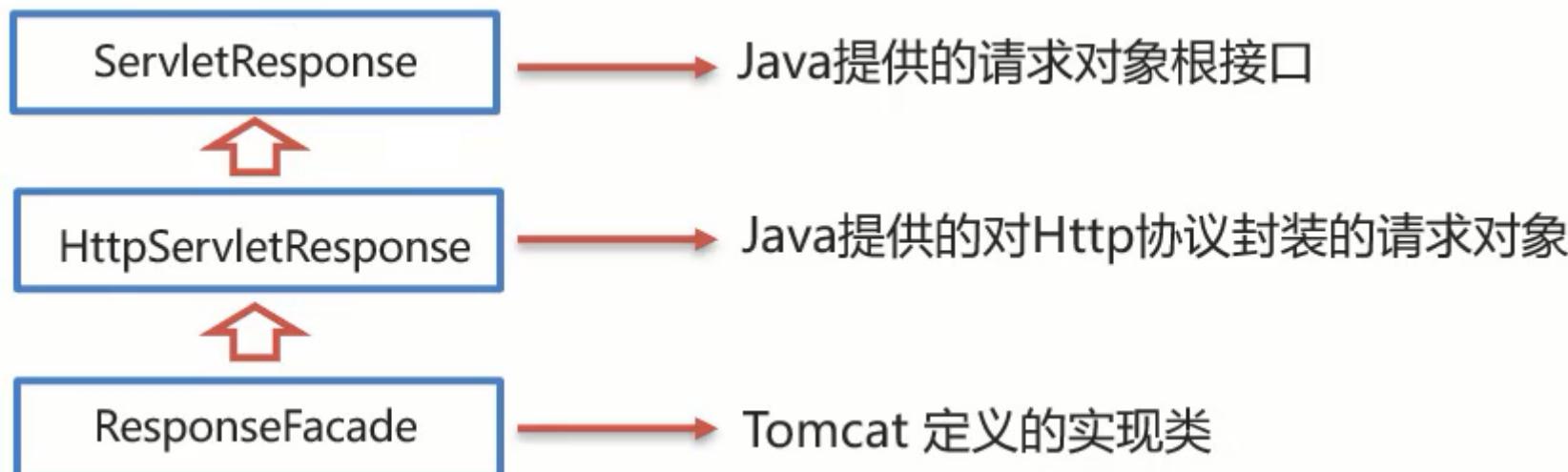
3, Response对象

前面讲解完Request对象，接下来我们回到刚开始的那张图：



- Request: 使用request对象来**获取**请求数据
- Response: 使用response对象来**设置**响应数据

Reponse的继承体系和Request的继承体系也非常相似：



介绍完Response的相关体系结构后，接下来对于Response我们需要学习如下内容：

- Response设置响应数据的功能介绍
- Response完成重定向
- Response响应字符数据
- Response响应字节数据

3.1 Response设置响应数据功能介绍

HTTP响应数据总共分为三部分内容，分别是**响应行**、**响应头**、**响应体**，对于这三部分内容的数据，responce对象都提供了哪些方法来进行设置？

1. 响应行



对于响应头，比较常用的就是设置响应状态码：

```
1 void setStatus(int sc);
```

2. 响应头

Content-Type: text/html

键 值

设置响应头键值对：

```
1 void setHeader(String name, String value);
```

3. 响应体

<html><head>head><body></body></html>

对于响应体，是通过字符、字节输出流的方式往浏览器写，

获取字符输出流：

```
1 PrintWriter getWriter();
```

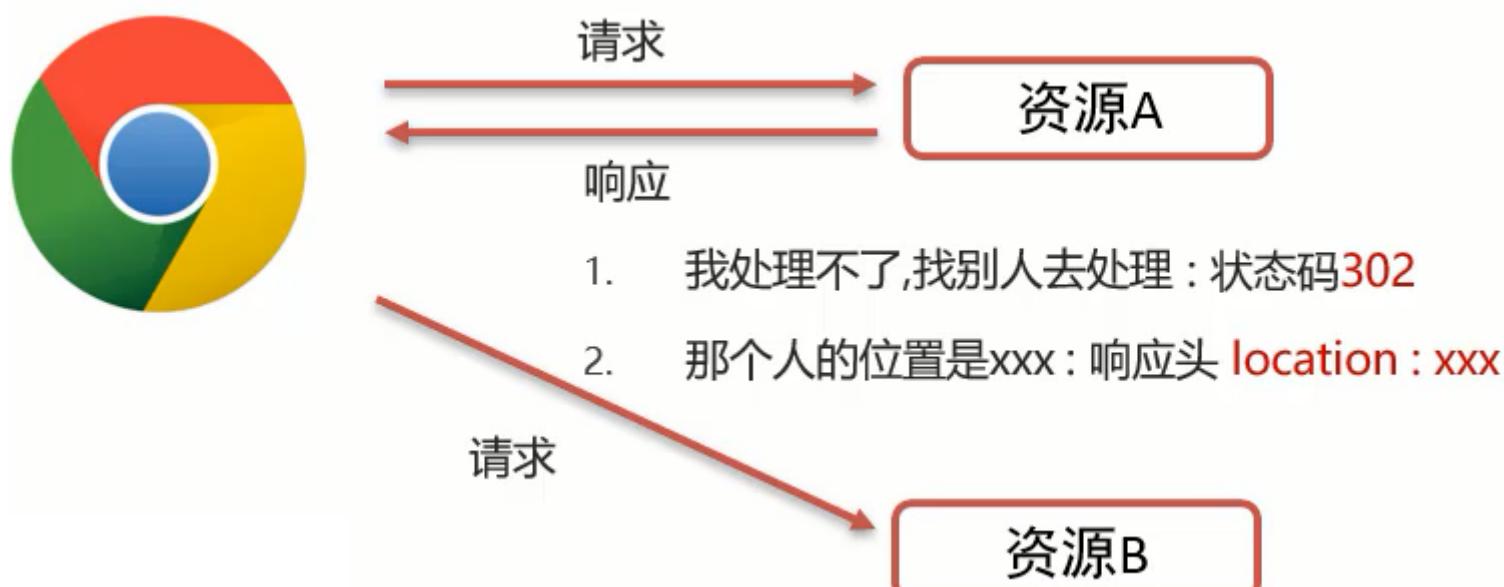
获取字节输出流

```
1 ServletOutputStream getOutputStream();
```

介绍完这些方法后，后面我们会通过案例把这些方法都用一用，首先先来完成下重定向的功能开发。

3.2 Response请求重定向

1. Response重定向 (redirect) : 一种资源跳转方式。



(1) 浏览器发送请求给服务器，服务器中对应的资源A接收到请求

(2) 资源A现在无法处理该请求，就会给浏览器响应一个302的状态码+location的一个访问资源B的路径

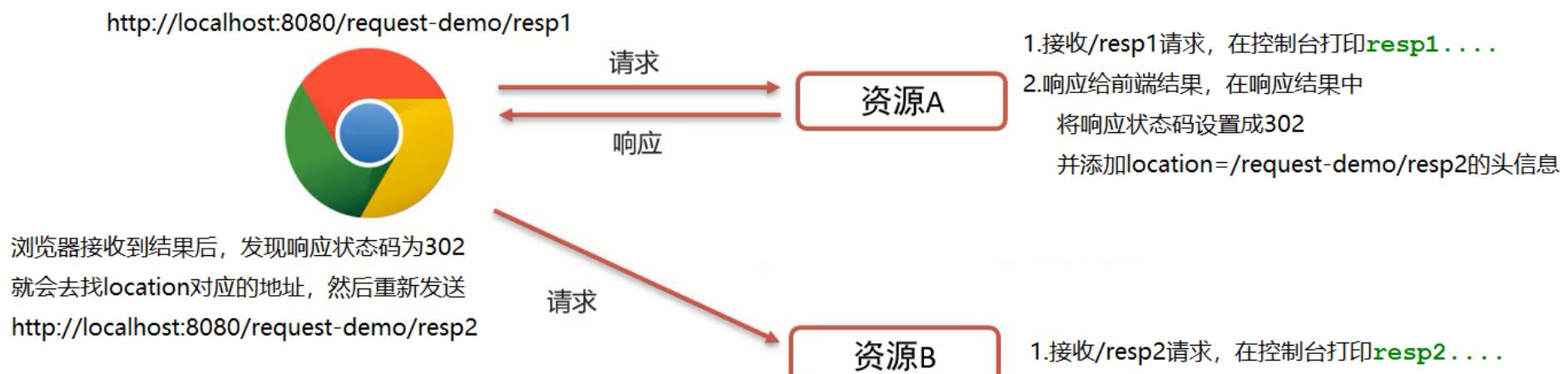
(3) 浏览器接收到响应状态码为302就会重新发送请求到location对应的访问地址去访问资源B

(4) 资源B接收到请求后进行处理并最终给浏览器响应结果，这整个过程就叫**重定向**

2. 重定向的实现方式：

```
1 resp.setStatus(302);  
2 resp.setHeader("Location", "资源B的访问路径");
```

具体如何来使用，我们先来看下需求：



针对上述需求，具体的实现步骤为：

1. 创建一个ResponseDemo1类，接收/resp1的请求，在doGet方法中打印resp1....

2. 创建一个ResponseDemo2类，接收/resp2的请求，在doGet方法中打印resp2....

3. 在ResponseDemo1的方法中使用

```
response.setStatus(302);
```

```
response.setHeader("Location", "/request-demo/resp2") 来给前端响应结果数据
```

4. 启动测试

(1) 创建ResponseDemo1类

```
1 @WebServlet("/resp1")
2 public class ResponseDemo1 extends HttpServlet {
3     @Override
4         protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
5             System.out.println("resp1....");
6         }
7
8     @Override
9         protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
10            this.doGet(request, response);
11        }
12 }
```

(2) 创建 ResponseDemo2 类

```
1 @WebServlet("/resp2")
2 public class ResponseDemo2 extends HttpServlet {
3     @Override
4         protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
5             System.out.println("resp2....");
6         }
7
8     @Override
9         protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
10            this.doGet(request, response);
11        }
12 }
```

(3) 在 ResponseDemo1 的 doGet 方法中给前端响应数据

```
1 @WebServlet("/resp1")
2 public class ResponseDemo1 extends HttpServlet {
3     @Override
4         protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
5             System.out.println("resp1....");
6             // 重定向
7             // 1. 设置响应状态码 302
8             response.setStatus(302);
9             // 2. 设置响应头 Location
10            response.setHeader("Location", "/request-demo/resp2");
11        }
12
13     @Override
```

```

14     protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
15         this.doGet(request, response);
16     }
17 }
```

(4) 启动测试

访问 `http://localhost:8080/request-demo/resp1`, 就可以在控制台看到如下内容:

```

| resp1....
| resp2....
```

说明 `/resp1` 和 `/resp2` 都被访问到了。到这重定向就已经完成了。

虽然功能已经实现，但是从设置重定向的两行代码来看，会发现除了重定向的地址不一样，其他的内容都是一模一样，所以 `request` 对象给我们提供了简化的编写方式为：

```
1 response.sendRedirect("/request-demo/resp2")
```

所以第3步中的代码就可以简化为：

```

1 @WebServlet("/resp1")
2 public class ResponseDemo1 extends HttpServlet {
3     @Override
4     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
5         System.out.println("resp1....");
6         //重定向
7         response.sendRedirect("/request-demo/resp2");
8     }
9
10    @Override
11    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
12        this.doGet(request, response);
13    }
14 }
```

3. 重定向的特点

- 浏览器地址栏路径发生变化

当进行重定向访问的时候，由于是由浏览器发送的两次请求，所以地址会发生变化



- 可以重定向到任何位置的资源(服务内容、外部均可)

因为第一次响应结果中包含了浏览器下次要跳转的路径，所以这个路径是可以任意位置资源。

- 两次请求，不能在多个资源使用request共享数据

因为浏览器发送了两次请求，是两个不同的request对象，就无法通过request对象进行共享数据

介绍完**请求重定向**和**请求转发**以后，接下来需要把这两个放在一块对比下：

● 重定向特点：	● 请求转发特点：
➤ 浏览器地址栏路径发生变化	➤ 浏览器地址栏路径不发生变化
➤ 可以重定向到任意位置的资源（服务器内部、外部均可）	➤ 只能转发到当前服务器的内部资源
➤ 两次请求，不能在多个资源使用request共享数据	➤ 一次请求，可以在转发的资源间使用request共享数据

以后到底用哪个，还是需要根据具体的业务来决定。

3.3 路径问题

- 问题1：转发的时候路径上没有加/request-demo而重定向加了，那么到底什么时候需要加，什么时候不需要加呢？

转发: `request.getRequestDispatcher(path: "/req6").forward(request, response);`

重定向: `response.setHeader(name: "Location", value: "/request-demo/resp2");`

其实判断的依据很简单，只需要记住下面的规则即可：

- 浏览器使用：需要加虚拟目录（项目访问路径）
- 服务端使用：不需要加虚拟目录

对于转发来说，因为是在服务端进行的，所以不需要加虚拟目录

对于重定向来说，路径最终是由浏览器来发送请求，就需要添加虚拟目录。

掌握了这个规则，接下来就通过一些练习来强化下知识的学习：

- ``
- `<form action='路径'>`
- `req.getRequestDispatcher("路径")`
- `resp.sendRedirect("路径")`

答案：

1. 超链接，从浏览器发送，需要加
2. 表单，从浏览器发送，需要加
3. 转发，是从服务器内部跳转，不需要加
4. 重定向，是由浏览器进行跳转，需要加。

- 问题2：在重定向的代码中，/request-demo是固定编码的，如果后期通过Tomcat插件配置了项目的访问路径，那么所有需要重定向的地方都需要重新修改，该如何优化？

```

<build>
    <plugins>
        <plugin>
            <groupId>org.apache.tomcat.maven</groupId>
            <artifactId>tomcat7-maven-plugin</artifactId>
            <version>2.2</version>
            <!-- <configuration> 配置项目的访问地址
                <path>/xxx</path>
            </configuration>-->
        </plugin>
    </plugins>
</build>

```

答案也比较简单，我们可以在代码中动态去获取项目访问的虚拟目录，具体如何获取，我们可以借助前面咱们所学习的request对象中的getContextPath()方法，修改后的代码如下：

```

1 @WebServlet("/resp1")
2 public class ResponseDemo1 extends HttpServlet {
3     @Override
4     protected void doGet(HttpServletRequest request, HttpServletResponse
5 response) throws ServletException, IOException {
6         System.out.println("resp1....");
7
8         //简化方式完成重定向
9         //动态获取虚拟目录
10        String contextPath = request.getContextPath();
11        response.sendRedirect(contextPath+"/resp2");
12    }
13
14    @Override
15    protected void doPost(HttpServletRequest request, HttpServletResponse
16 response) throws ServletException, IOException {
17        this.doGet(request, response);
18    }
19 }

```

重新启动访问测试，功能依然能够实现，此时就可以动态获取项目访问的虚拟路径，从而降低代码的耦合度。

3.4 Response响应字符数据

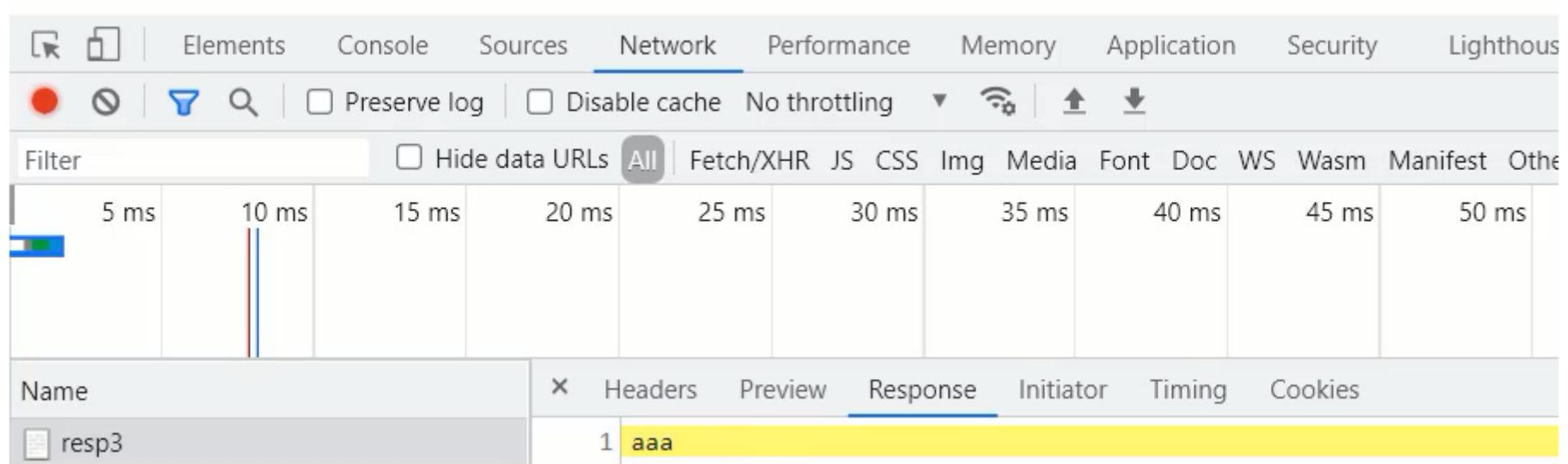
要想将字符数据写回到浏览器，我们需要两个步骤：

- 通过Response对象获取字符输出流： PrintWriter writer = resp.getWriter();
- 通过字符输出流写数据： writer.write("aaa");

接下来，我们实现通过些案例把响应字符数据给实际应用下：

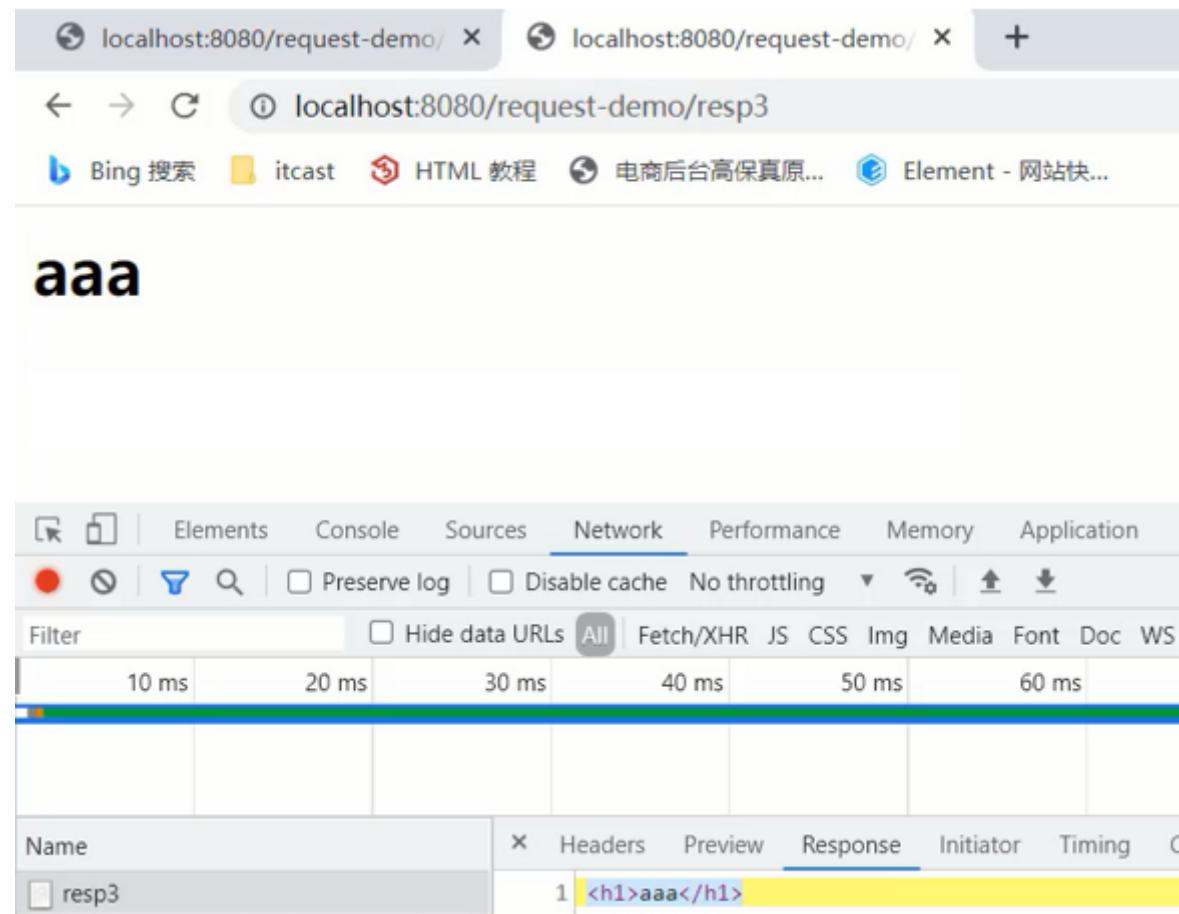
1. 返回一个简单的字符串aaa

```
1 /**
2  * 响应字符数据：设置字符数据的响应体
3 */
4 @WebServlet("/resp3")
5 public class ResponseDemo3 extends HttpServlet {
6     @Override
7     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
8         response.setContentType("text/html;charset=utf-8");
9         //1. 获取字符输出流
10        PrintWriter writer = response.getWriter();
11        writer.write("aaa");
12    }
13    @Override
14    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
15        this.doGet(request, response);
16    }
17 }
```



2. 返回一串html字符串，并且能被浏览器解析

```
1 PrintWriter writer = response.getWriter();
2 //content-type, 告诉浏览器返回的数据类型是HTML类型数据, 这样浏览器才会解析HTML标签
3 response.setHeader("content-type", "text/html");
4 writer.write("<h1>aaa</h1>");
```



注意:一次请求响应结束后, response对象就会被销毁掉, 所以不要手动关闭流。

3. 返回一个中文的字符串你好, 需要注意设置响应数据的编码为 utf-8

```
1 //设置响应的数据格式及数据的编码
2 response.setContentType("text/html;charset=utf-8");
3 writer.write("你好");
```



3.3 Response响应字节数据

要想将字节数据写回到浏览器, 我们需要两个步骤:

- 通过Response对象获取字节输出流: ServletOutputStream outputStream = resp.getOutputStream();
- 通过字节输出流写数据: outputStream.write(字节数据);

接下来, 我们实现通过些案例把响应字符数据给实际应用下:

1. 返回一个图片文件到浏览器

```
1 /**
2 * 响应字节数据: 设置字节数据的响应体
```

```
3  */
4 @webServlet("/resp4")
5 public class ResponseDemo4 extends HttpServlet {
6     @Override
7     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
8         //1. 读取文件
9         FileInputStream fis = new FileInputStream("d://a.jpg");
10        //2. 获取response字节输出流
11        ServletOutputStream os = response.getOutputStream();
12        //3. 完成流的copy
13        byte[] buff = new byte[1024];
14        int len = 0;
15        while ((len = fis.read(buff)) != -1){
16            os.write(buff, 0, len);
17        }
18        fis.close();
19    }
20
21    @Override
22    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
23        this.doGet(request, response);
24    }
25 }
```

resp4 (420×652) × + ← → C ① localhost:8080/request-demo/resp4

Bing 搜索 itcast HTML 教程 电商后台高保真原... Element - 网站快...

		2021東京奧運會	金	銀	銅	總計
1	中国	38	32	18	88	
	中国香港	1	2	3	6	
	中国台北	2	4	6	12	
2	美国	39	41	33	113	
3	日本	27	14	17	58	
4	英國	22	21	22	65	
5	ROC	20	28	23	71	
6	澳大利亚	17	7	22	46	
7	荷兰	10	12	14	36	
8	法国	10	12	11	33	
9	德国	10	11	16	37	
10	意大利	10	10	20	40	

上述代码中，对于流的copy的代码还是比较复杂的，所以我们可以使用别人提供好的方法来简化代码的开发，具体的步骤是：

(1) pom.xml添加依赖

```

1 <dependency>
2   <groupId>commons-io</groupId>
3   <artifactId>commons-io</artifactId>
4   <version>2.6</version>
5 </dependency>
```

(2) 调用工具类方法

```

1 //fis:输入流
2 //os:输出流
3 IOUtils.copy(fis,os);
```

优化后的代码：

```

1 /**
2  * 响应字节数据：设置字节数据的响应体
3 */
```

```

4  @WebServlet("/resp4")
5  public class ResponseDemo4 extends HttpServlet {
6      @Override
7      protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
8          //1. 读取文件
9          FileInputStream fis = new FileInputStream("d://a.jpg");
10         //2. 获取response字节输出流
11         ServletOutputStream os = response.getOutputStream();
12         //3. 完成流的copy
13         IOutils.copy(fis,os);
14         fis.close();
15     }
16
17     @Override
18     protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
19         this.doGet(request, response);
20     }
21 }

```

4. 用户注册登录案例

接下来我们通过两个比较常见的案例，一个是**注册**，一个是**登录**来对今天学习的内容进行一个实战演练，首先来实现用户登录。

4.1 用户登录

4.1.1 需求分析

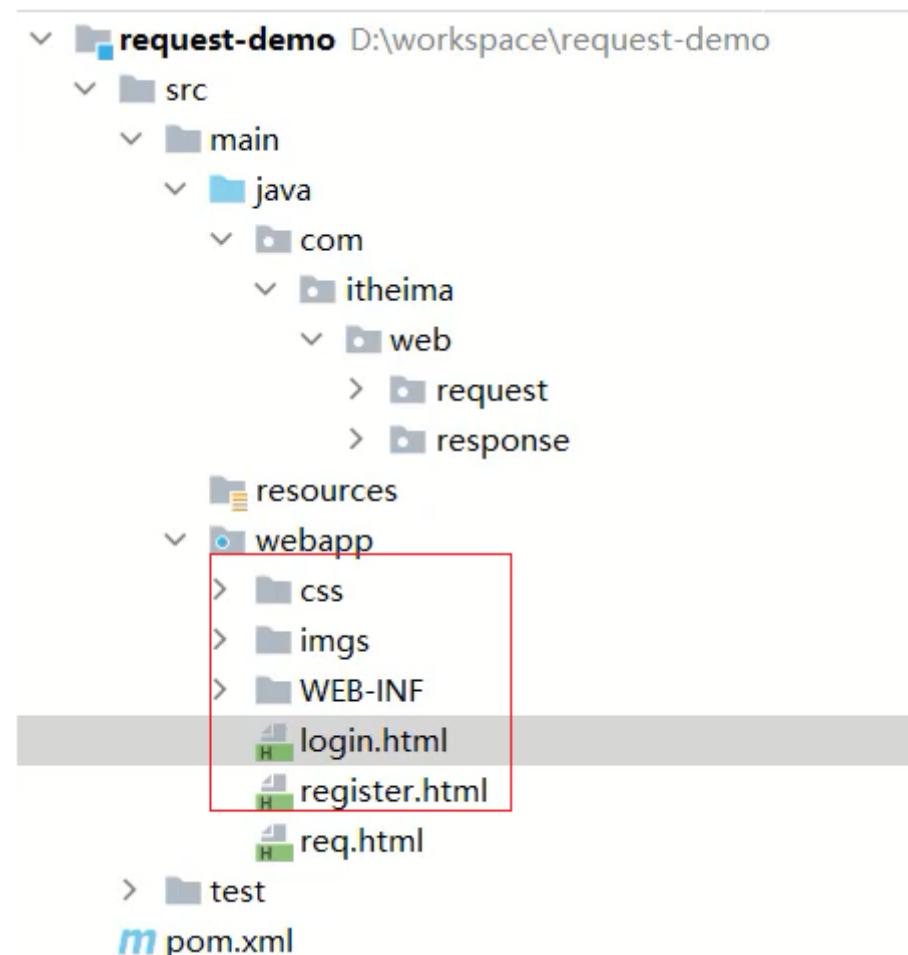


1. 用户在登录页面输入用户名和密码，提交请求给LoginServlet
2. 在LoginServlet中接收请求和数据 [用户名和密码]
3. 在LoginServlet中通过Mybatis实现调用UserMapper来根据用户名和密码查询数据库表
4. 将查询的结果封装到User对象中进行返回
5. 在LoginServlet中判断返回的User对象是否为null
6. 如果为null，说明根据用户名和密码没有查询到用户，则登录失败，返回“登录失败”数据给前端
7. 如果不为null，则说明用户存在并且密码正确，则登录成功，返回“登录成功”数据给前端

4.1.2 环境准备

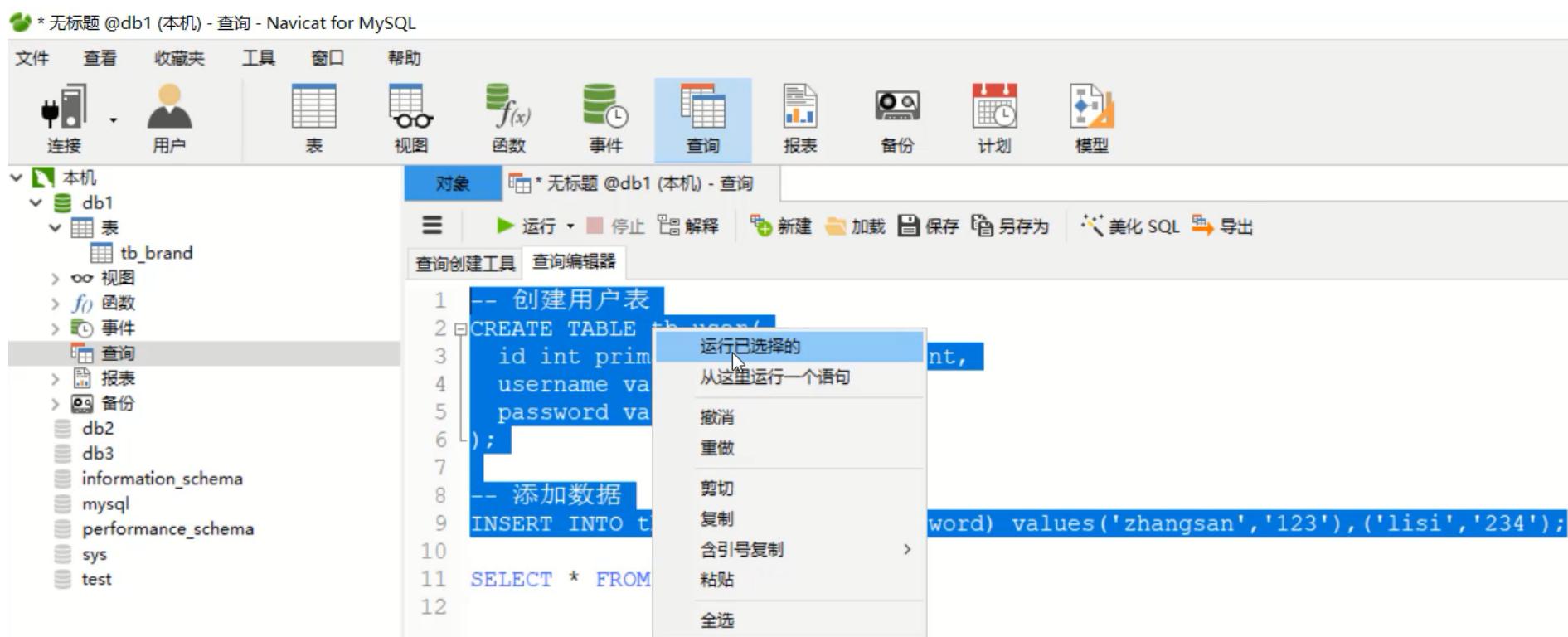
1. 复制资料中的静态页面到项目的webapp目录下

参考资料\1. 登陆注册案例\1. 静态页面，拷贝完效果如下：

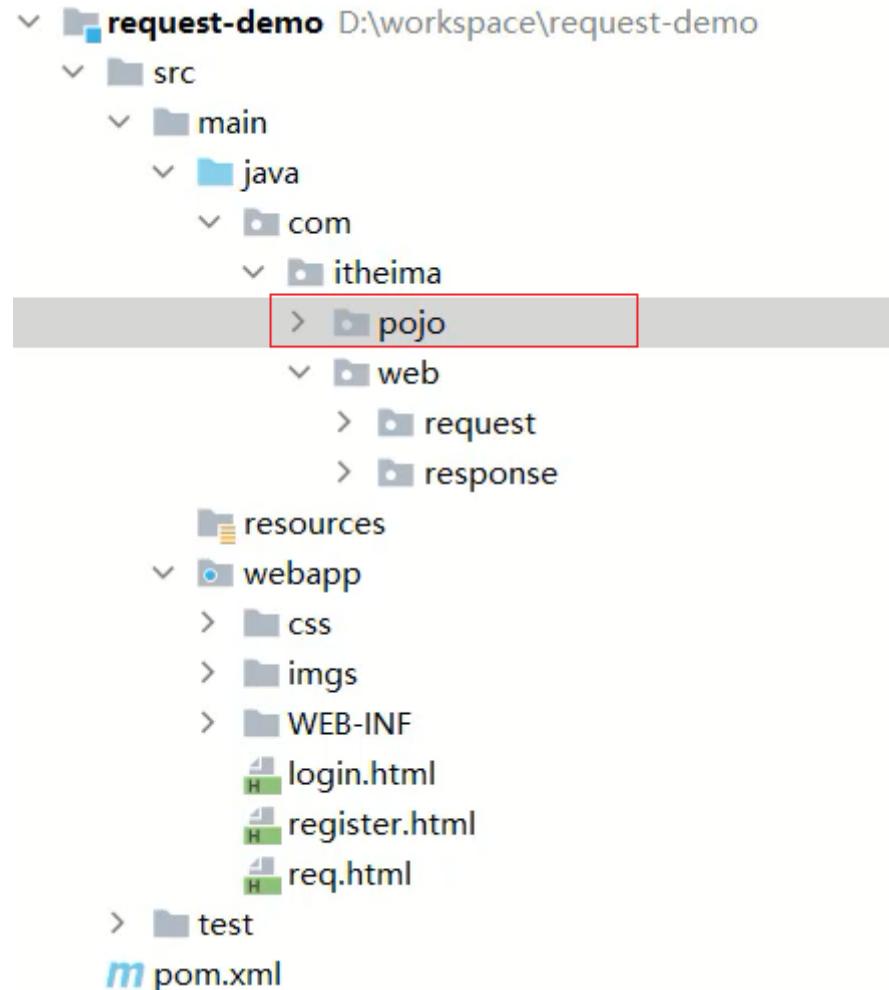


2. 创建db1数据库，创建tb_user表，创建User实体类

2.1 将资料\1. 登陆注册案例\2. MyBatis环境\tb_user.sql中的sql语句执行下：



2.2 将资料\1. 登陆注册案例\2. MyBatis环境\User.java拷贝到com.itheima.pojo



3. 在项目的pom.xml导入Mybatis和Mysql驱动坐标

```

1 <dependency>
2   <groupId>org.mybatis</groupId>
3   <artifactId>mybatis</artifactId>
4   <version>3.5.5</version>
5 </dependency>
6
7 <dependency>
8   <groupId>mysql</groupId>
9   <artifactId>mysql-connector-java</artifactId>
10  <version>5.1.34</version>
11 </dependency>
  
```

4. 创建mybatis-config.xml核心配置文件, UserMapper.xml映射文件, UserMapper接口

4.1 将资料\1. 登陆注册案例\2. MyBatis环境\mybatis-config.xml拷贝到resources目录下

```

1 <?xml version="1.0" encoding="UTF-8" ?>
2 <!DOCTYPE configuration
3   PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
4   "http://mybatis.org/dtd/mybatis-3-config.dtd">
5 <configuration>
6   <!--起别名-->
7   <typeAliases>
8     <package name="com.itheima.pojo"/>
9   </typeAliases>
10
11  <environments default="development">
12    <environment id="development">
13      <transactionManager type="JDBC"/>
  
```

```

14      <dataSource type="POOLED">
15          <property name="driver" value="com.mysql.jdbc.Driver"/>
16          <!--
17              useSSL:关闭SSL安全连接 性能更高
18              useServerPrepStmts:开启预编译功能
19              & 等同于 & ,xml配置文件中不能直接写 &符号
20          -->
21          <property name="url" value="jdbc:mysql:///db1?
useSSL=false&useServerPrepStmts=true"/>
22          <property name="username" value="root"/>
23          <property name="password" value="1234"/>
24      </dataSource>
25  </environment>
26 </environments>
27 <mappers>
28     <!--扫描mapper-->
29     <package name="com.itheima.mapper"/>
30 </mappers>
31 </configuration>

```

4.2 在com.itheima.mapper包下创建UserMapper接口

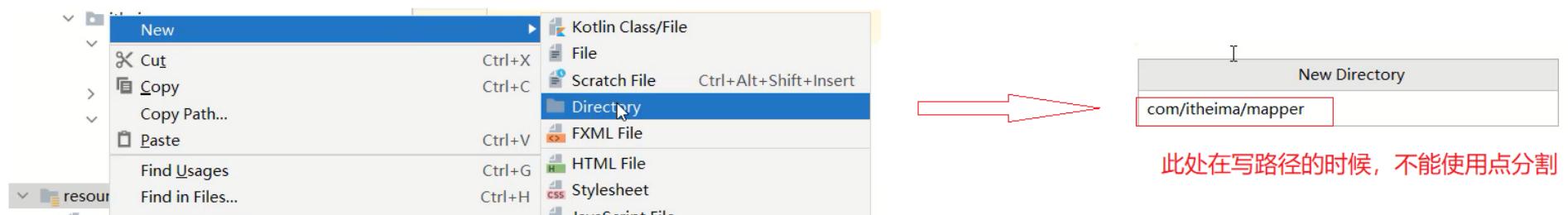
```

1 public interface UserMapper {
2
3 }

```

4.3 将资料\1. 登陆注册案例\2. MyBatis环境\UserMapper.xml拷贝到resources目录下

注意：在resources下创建UserMapper.xml的目录时，要使用/分割



至此我们所需要的环境就都已经准备好了，具体该如何实现？

4.1.3 代码实现

1. 在UserMapper接口中提供一个根据用户名和密码查询用户对象的方法

```
1 /**
2  * 根据用户名和密码查询用户对象
3  * @param username
4  * @param password
5  * @return
6 */
7 @Select("select * from tb_user where username = #{username} and password =#{password}")
8 User select(@Param("username") String username,@Param("password") String password);
```

说明

@Param注解的作用：用于传递参数，是方法的参数可以与SQL中的字段名相对应。

2. 修改login.html

```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="UTF-8">
6   <title>login</title>
7   <link href="css/login.css" rel="stylesheet">
8 </head>
9
10 <body>
11 <div id="loginDiv">
12   <form action="/request-demo/loginServlet" method="post" id="form">
13     <h1 id="loginMsg">LOGIN IN</h1>
14     <p>Username:<input id="username" name="username" type="text"></p>
15
16     <p>Password:<input id="password" name="password" type="password"></p>
17
18     <div id="subDiv">
19       <input type="submit" class="button" value="login up">
20       <input type="reset" class="button"
21         value="reset">&ampnbsp&ampnbsp&ampnbsp
22         <a href="register.html">没有账号？点击注册</a>
23     </div>
24   </form>
25 </div>
26 </body>
27 </html>
```

3. 编写LoginServlet

```

1  @WebServlet("/loginServlet")
2  public class LoginServlet extends HttpServlet {
3      @Override
4      protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
5          //1. 接收用户名和密码
6          String username = request.getParameter("username");
7          String password = request.getParameter("password");
8
9          //2. 调用MyBatis完成查询
10         //2.1 获取SqlSessionFactory对象
11         String resource = "mybatis-config.xml";
12         InputStream inputStream = Resources.getResourceAsStream(resource);
13         SqlSessionFactory sqlSessionFactory = new
14         SqlSessionFactoryBuilder().build(inputStream);
15         //2.2 获取sqlSession对象
16         SqlSession sqlSession = sqlSessionFactory.openSession();
17         //2.3 获取Mapper
18         UserMapper userMapper = sqlSession.getMapper(UserMapper.class);
19         //2.4 调用方法
20         User user = userMapper.select(username, password);
21         //2.5 释放资源
22         sqlSession.close();
23
24         //获取字符输出流，并设置content type
25         response.setContentType("text/html;charset=utf-8");
26         PrintWriter writer = response.getWriter();
27         //3. 判断user释放为null
28         if(user != null){
29             // 登陆成功
30             writer.write("登陆成功");
31         }else {
32             // 登陆失败
33             writer.write("登陆失败");
34         }
35     }
36
37     @Override
38     protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
39         this.doGet(request, response);
40     }
41 }
```

4. 启动服务器测试

4.1 如果用户名和密码输入错误，则



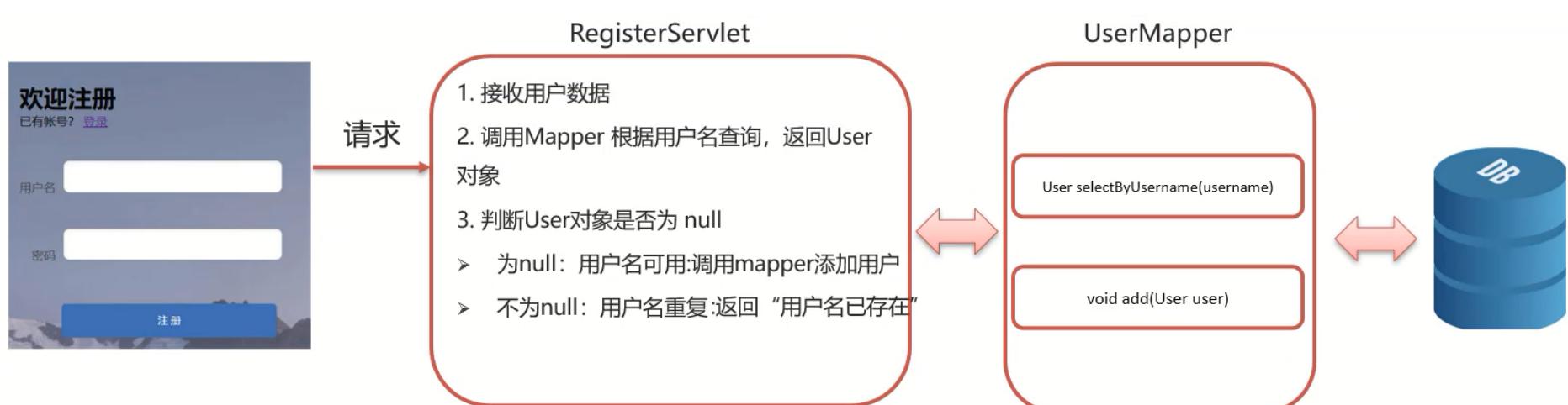
4.2 如果用户名和密码输入正确，则



至此用户的登录功能就已经完成了~

4.2 用户注册

4.2.1 需求分析



1. 用户在注册页面输入用户名和密码，提交请求给RegisterServlet
2. 在RegisterServlet中接收请求和数据 [用户名和密码]
3. 在RegisterServlet中通过Mybatis实现调用UserMapper来根据用户名查询数据库表
4. 将查询的结果封装到User对象中进行返回
5. 在RegisterServlet中判断返回的User对象是否为null
6. 如果为null，说明根据用户名可用，则调用UserMapper来实现添加用户
7. 如果不为null，则说明用户名不可以，返回"用户名已存在"数据给前端

4.2.2 代码编写

1. 编写UserMapper提供根据用户名查询用户数据方法和添加用户方法

```
1 /**
2 * 根据用户名查询用户对象
3 * @param username
4 * @return
5 */
```

```
6 @Select("select * from tb_user where username = #{username}")
7 User selectByUsername(String username);
8
9 /**
10 * 添加用户
11 * @param user
12 */
13 @Insert("insert into tb_user values(null,#{username},#{password})")
14 void add(User user);
```

2. 修改register.html

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <title>欢迎注册</title>
6     <link href="css/register.css" rel="stylesheet">
7 </head>
8 <body>
9
10 <div class="form-div">
11     <div class="reg-content">
12         <h1>欢迎注册</h1>
13         <span>已有帐号? </span> <a href="login.html">登录</a>
14     </div>
15     <form id="reg-form" action="/request-demo/registerServlet" method="post">
16
17         <table>
18
19             <tr>
20                 <td>用户名</td>
21                 <td class="inputs">
22                     <input name="username" type="text" id="username">
23                     <br>
24                     <span id="username_err" class="err_msg" style="display:
none">用户名不太受欢迎</span>
25                 </td>
26
27             </tr>
28
29             <tr>
30                 <td>密码</td>
31                 <td class="inputs">
32                     <input name="password" type="password" id="password">
33                     <br>
34                     <span id="password_err" class="err_msg" style="display:
none">密码格式有误</span>
```

```

35         </td>
36     </tr>
37
38 </table>
39
40 <div class="buttons">
41     <input value="注 册" type="submit" id="reg_btn">
42 </div>
43 <br class="clear">
44 </form>
45
46 </div>
47 </body>
48 </html>

```

3. 创建RegisterServlet类

```

1 @WebServlet("/registerServlet")
2 public class RegisterServlet extends HttpServlet {
3     @Override
4     protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
5         //1. 接收用户数据
6         String username = request.getParameter("username");
7         String password = request.getParameter("password");
8
9         //封装用户对象
10        User user = new User();
11        user.setUsername(username);
12        user.setPassword(password);
13
14        //2. 调用mapper 根据用户名查询用户对象
15        //2.1 获取SqlSessionFactory对象
16        String resource = "mybatis-config.xml";
17        InputStream inputStream = Resources.getResourceAsStream(resource);
18        SqlSessionFactory sqlSessionFactory = new
SqlSessionFactoryBuilder().build(inputStream);
19        //2.2 获取SqlSession对象
20        SqlSession sqlSession = sqlSessionFactory.openSession();
21        //2.3 获取Mapper
22        UserMapper userMapper = sqlSession.getMapper(UserMapper.class);
23
24        //2.4 调用方法
25        User u = userMapper.selectByUsername(username);
26
27        //3. 判断用户对象释放为null
28        if( u == null){
29            // 用户名不存在, 添加用户

```

```

30         userMapper.add(user);
31
32         // 提交事务
33         sqlSession.commit();
34         // 释放资源
35         sqlSession.close();
36     }else {
37         // 用户名存在，给出提示信息
38         response.setContentType("text/html;charset=utf-8");
39         response.getWriter().write("用户名已存在");
40     }
41 }
42
43
44     @Override
45     protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
46         this doGet(request, response);
47     }
48 }
```

4. 启动服务器进行测试

4.1 如果测试成功，则在数据库中就能查看到新注册的数据

4.2 如果用户已经存在，则在页面上展示 `用户名已存在` 的提示信息

4.3 SqlSessionFactory工具类抽取

上面两个功能已经实现，但是在写Servlet的时候，因为需要使用Mybatis来完成数据库的操作，所以对于Mybatis的基础操作就出现了些重复代码，如下

```

1 String resource = "mybatis-config.xml";
2 InputStream inputStream = Resources.getResourceAsStream(resource);
3 SqlSessionFactory sqlSessionFactory = new
4     SqlSessionFactoryBuilder().build(inputStream);
```

有了这些重复代码就会造成一些问题：

- 重复代码不利于后期的维护
- SqlSessionFactory工厂类进行重复创建
 - 就相当于每次买手机都需要重新创建一个手机生产工厂来给你制造一个手机一样，资源消耗非常大但性能却非常低。所以这么做是不允许的。

那如何来优化呢？

- 代码重复可以抽取工具类
- 对指定代码只需要执行一次可以使用静态代码块

有了这两个方向后，代码具体该如何编写？

```
1 public class SqlSessionFactoryUtils {  
2  
3     private static SqlSessionFactory sqlSessionFactory;  
4  
5     static {  
6         //静态代码块会随着类的加载而自动执行，且只执行一次  
7         try {  
8             String resource = "mybatis-config.xml";  
9             InputStream inputStream =  
Resources.getResourceAsStream(resource);  
10            sqlSessionFactory = new  
SqlSessionFactoryBuilder().build(inputStream);  
11        } catch (IOException e) {  
12            e.printStackTrace();  
13        }  
14    }  
15  
16  
17    public static SqlSessionFactory getSqlSessionFactory(){  
18        return sqlSessionFactory;  
19    }  
20 }
```

工具类抽取以后，以后在对Mybatis的SqlSession进行操作的时候，就可以直接使用

```
1 SqlSessionFactory sqlSessionFactory  
=SqlSessionFactoryUtils.getSqlSessionFactory();
```

这样就可以很好的解决上面所说的代码重复和重复创建工厂导致性能低的问题了。

JSP

今日目标：

- 理解 JSP 及 JSP 原理
- 能在 JSP 中使用 EL 表达式 和 JSTL 标签
- 理解 MVC 模式 和 三层架构
- 能完成品牌数据的增删改查功能

1. JSP 概述

JSP (全称：Java Server Pages)：Java 服务端页面。是一种动态的网页技术，其中既可以定义 HTML、JS、CSS 等静态内容，还可以定义 Java 代码的动态内容，也就是 `JSP = HTML + Java`。如下就是 jsp 代码

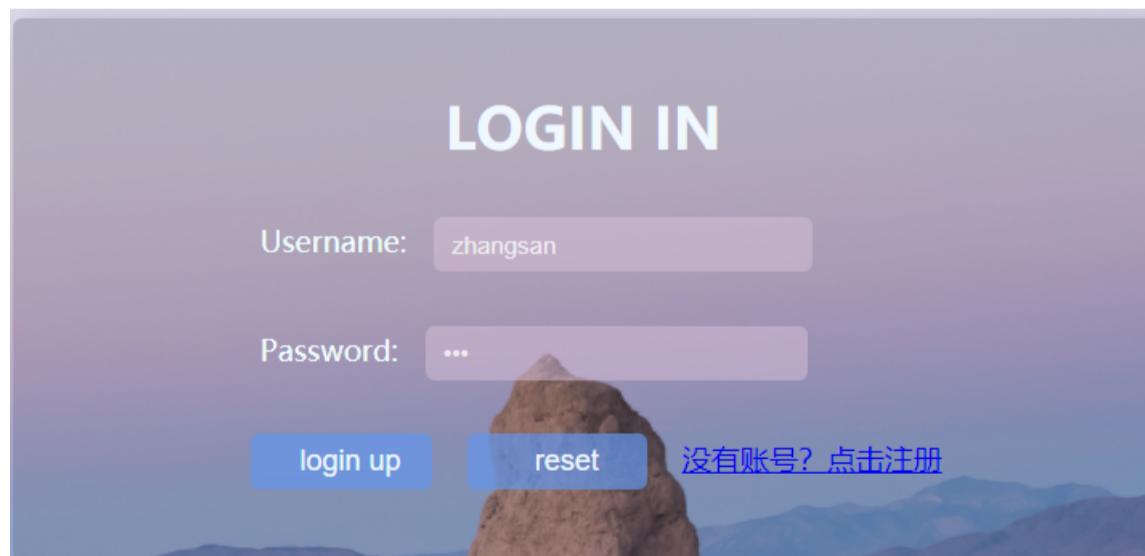
```
1 <html>
2   <head>
3     <title>Title</title>
4   </head>
5   <body>
6     <h1>JSP,Hello world</h1>
7     <%
8       System.out.println("hello,jsp~");
9     %>
10    </body>
11 </html>
```

上面代码 `h1` 标签内容是展示在页面上，而 Java 的输出语句是输出在 idea 的控制台。

那么，JSP 能做什么呢？现在我们只用 `servlet` 实现功能，看存在什么问题。如下图所示，当我们登陆成功后，需要在页面上展示用户名



上图的用户名是动态展示，也就是谁登陆就展示谁的用户名。只用 `servlet` 如何实现呢？在今天的资料里已经提供好了一个 `LoginServlet`，该 `servlet` 就是实现这个功能的，现将资料中的 `LoginServlet.java` 拷贝到 `request-demo` 项目中来演示。接下来启动服务器并访问登陆页面



输入了 `zhangsan` 用户的登陆信息后点击 `登陆` 按钮，就能看到如下图效果



当然如果是 `lisi` 登陆的，在该页面展示的就是 `lisi,欢迎您`，动态的展示效果就实现了。那么 `LoginServlet` 到底是如何实现的，我们看看它里面的内容

```
>LoginServlet.java
247 writer.write( s: "</head>\r\n");
248 writer.write( s: "\r\n");
249 writer.write( s: "<body class=\"index\">\r\n");
250 writer.write( s: "<div class=\"mod_container\">\r\n");
251 writer.write( s: "    \r\n");
252 writer.write( s: "    <div id=\"J_accessibility\"></div>\r\n");
253 writer.write( s: "    <!--顶通占位 -->\r\n");
254 writer.write( s: "    <div id=\"J_promotional-top\">\r\n");
255 writer.write( s: "        </div>\r\n");
256 writer.write( s: "        <h1 align=\"center\">+username+, 欢迎您</h1>";
257 writer.write( s: "        <div id=\"shortcut\">\r\n");
258 writer.write( s: "            <div class=\"w\">\r\n");
259 writer.write( s: "                <ul class=\"f1\" clstag=\"h|keycount|head|topbar_01\">\r\n");
260 writer.write( s: "                    <li class=\"dropdown\" id=\"ttbar-mycity\"></li>\r\n");
261 writer.write( s: "                </ul>\r\n");
262 writer.write( s: "\r\n");
```

上面的代码有大量使用到 `writer` 对象向页面写标签内容，这样我们的代码就显得很麻烦；将来如果展示的效果出现了问题，排错也显得有点力不从心。而 JSP 是如何解决这个问题的呢？在资料中也提供了一个 `login.jsp` 页面，该页面也能实现该功能，现将该页面拷贝到项目的 `webapp` 下，需要修改 `login.html` 中表单数据提交的路径为下图

```
<div id="loginDiv">
    <form action="/request-demo/login.jsp" method="post" id="form">
        <h1 id="loginMsg">LOGIN IN</h1>
        <p>Username:<input id="username" name="username" type="text"></p>
```

重新启动服务器并进行测试，发现也可以实现同样的功能。那么 `login.jsp` 又是如何实现的呢？那我们来看看 `login.jsp` 的代码

```
login.jsp
239 <body class="index">
240     <div class="mod_container">
241         <!--无障碍占位-->
242         <div id="J_accessibility"></div>
243         <!--顶通占位 -->
244         <div id="J_promotional-top">
245             </div>
246             <h1 align="center"><%=username%>, 欢迎您</h1>
247             <div id="shortcut">
248
249                 <div class="w">
250                     <ul class="f1" clstag="h|keycount|head|topbar_01">
251                         <li class="dropdown" id="ttbar-mycity"></li>
252                     </ul>
```

上面代码可以看到里面基本都是 `HTML` 标签，而动态数据使用 `Java` 代码进行展示；这样操作看起来要比用 `servlet` 实现要舒服很多。

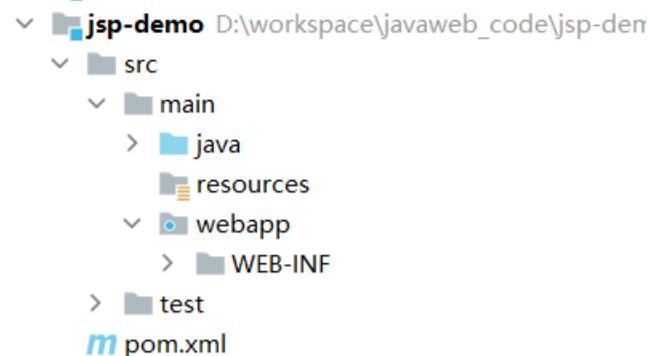
JSP 作用：简化开发，避免了在 `Servlet` 中直接输出 `HTML` 标签。

2, JSP 快速入门

接下来我们做一个简单的快速入门代码。

2.1 搭建环境

创建一个maven的 web 项目，项目结构如下：



`pom.xml` 文件内容如下：

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
5     http://maven.apache.org/xsd/maven-4.0.0.xsd">
6     <modelVersion>4.0.0</modelVersion>
7
8     <groupId>org.example</groupId>
9     <artifactId>jsp-demo</artifactId>
10    <version>1.0-SNAPSHOT</version>
11    <packaging>war</packaging>
12
13    <properties>
14      <maven.compiler.source>8</maven.compiler.source>
15      <maven.compiler.target>8</maven.compiler.target>
16    </properties>
17
18    <dependencies>
19      <dependency>
20        <groupId>javax.servlet</groupId>
21        <artifactId>javax.servlet-api</artifactId>
22        <version>3.1.0</version>
23        <scope>provided</scope>
24      </dependency>
25    </dependencies>
26
27    <build>
28      <plugins>
29        <plugin>
30          <groupId>org.apache.tomcat.maven</groupId>
31          <artifactId>tomcat7-maven-plugin</artifactId>
32          <version>2.2</version>
33        </plugin>
34      </plugins>
35    </build>
36  </project>
```

2.2 导入 JSP 依赖

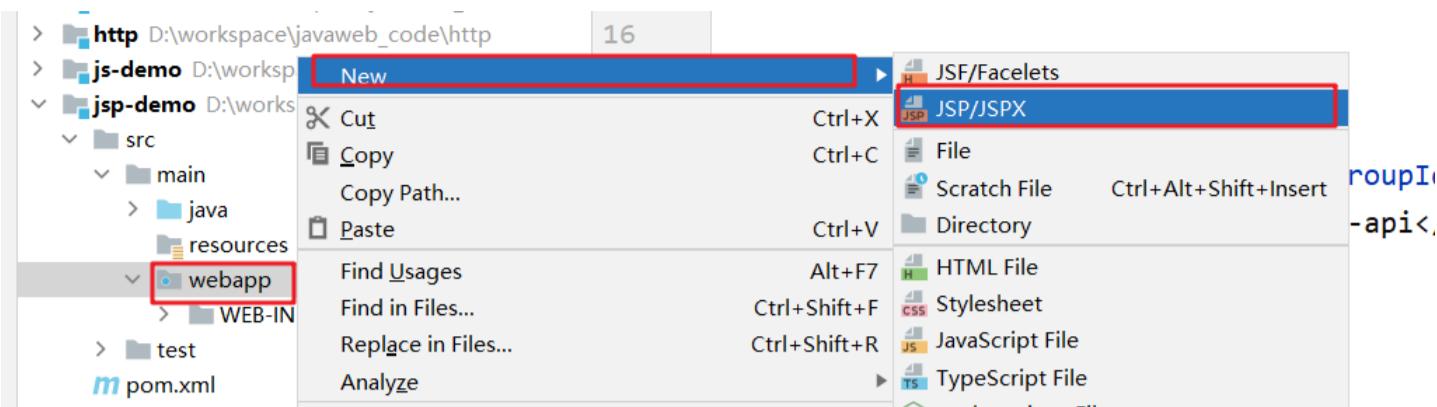
在 `dependencies` 标签中导入 JSP 的依赖，如下

```
1 <dependency>
2   <groupId>javax.servlet.jsp</groupId>
3   <artifactId>jsp-api</artifactId>
4   <version>2.2</version>
5   <scope>provided</scope>
6 </dependency>
```

该依赖的 `scope` 必须设置为 `provided`，因为 tomcat 中有这个jar包了，所以在打包时我们是不希望将该依赖打进到我们工程的war包中。

2.3 创建 jsp 页面

在项目的 webapp 下创建 jsp 页面



通过上面方式创建一个名为 `hello.jsp` 的页面。

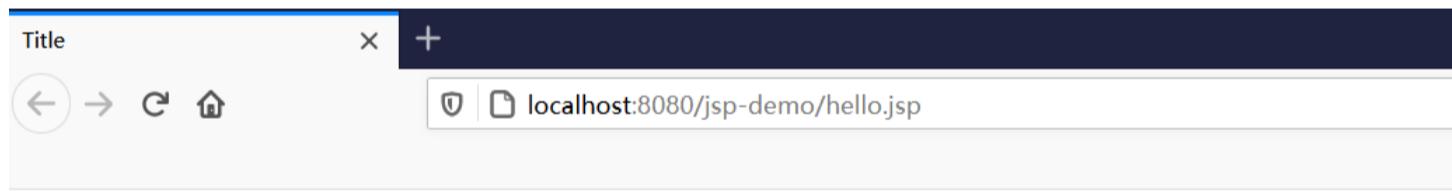
2.4 编写代码

在 `hello.jsp` 页面中书写 `HTML` 标签和 `Java` 代码，如下

```
1 <%@ page contentType="text/html; charset=UTF-8" language="java" %>
2 <html>
3 <head>
4     <title>Title</title>
5 </head>
6 <body>
7     <h1>hello jsp</h1>
8
9     <%
10        System.out.println("hello,jsp~");
11    %>
12 </body>
13 </html>
```

2.5 测试

启动服务器并在浏览器地址栏输入 `http://localhost:8080/jsp-demo/hello.jsp`，我们可以在页面上看到如下内容

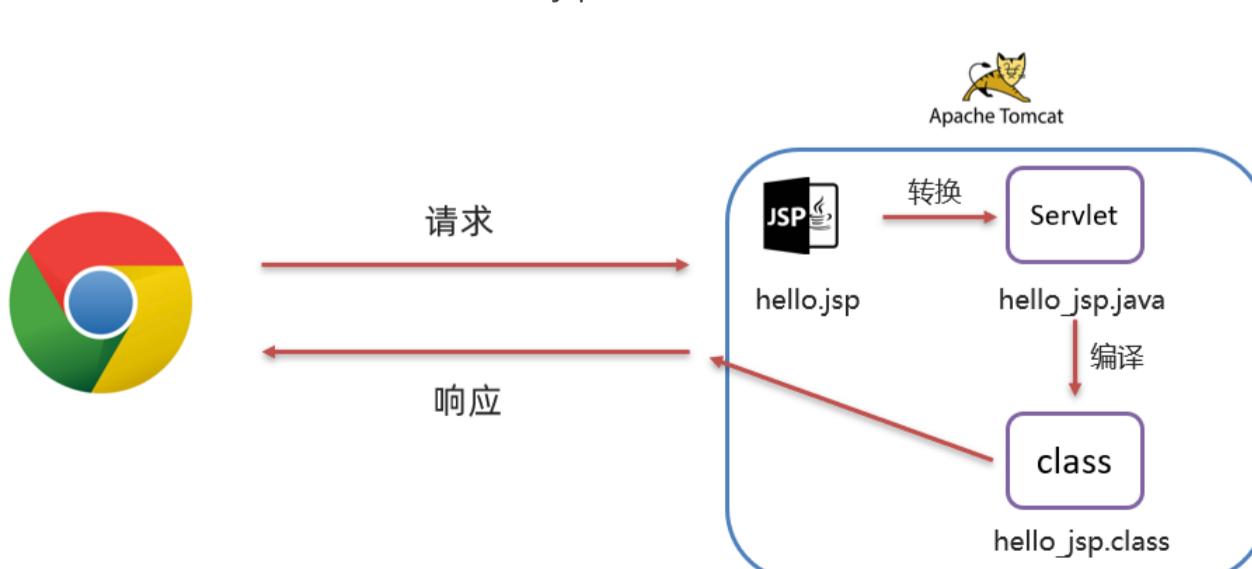


同时也可以看到在 idea 的控制台看到输出的 `hello,jsp~` 内容。

3, JSP 原理

我们之前说 JSP 就是一个页面，那么在 JSP 中写 `html` 标签，我们能理解，但是为什么还可以写 `Java` 代码呢？

因为 **JSP 本质上就是一个 Servlet**。接下来我们聊聊访问 jsp 时的流程



1. 浏览器第一次访问 `hello.jsp` 页面

2. tomcat 会将 hello.jsp 转换为名为 hello_jsp.java 的一个 servlet
3. tomcat 再将转换的 servlet 编译成字节码文件 hello_jsp.class
4. tomcat 会执行该字节码文件，向外提供服务

我们可以到项目所在磁盘目录下找 target\tomcat\work\Tomcat\localhost\jsp-demo\org\apache\jsp 目录，而这个目录下就能看到转换后的 servlet

hello_jsp.class	2021/8/18 11:00	CLASS 文件	4 KB
hello_jsp.java	2021/8/18 11:00	JAVA 文件	4 KB

打开 hello_jsp.java 文件，来查看里面的代码

```
public final class hello_jsp extends org.apache.jasper.runtime.HttpJspBase
    implements org.apache.jasper.runtime.JspSourceDependent {
```

由上面的类的继承关系可以看到继承了名为 HttpJspBase 这个类，那我们在看该类的继承关系。到资料中的找如下目录：
资料\tomcat源码\apache-tomcat-8.5.68-src\java\org\apache\jasper\runtime，该目录下就有 HttpJspBase 类，查看该类的继承关系

```
public abstract class HttpJspBase extends HttpServlet implements HttpJspPage {
```

可以看到该类继承了 HttpServlet；那么 hello_jsp 这个类就间接的继承了 HttpServlet，也就说明 hello_jsp 是一个 servlet。

继续阅读 hello_jsp 类的代码，可以看到有一个名为 _jspService() 的方法，该方法就是每次访问 jsp 时自动执行的方法，和 servlet 中的 service 方法一样。

而在 _jspService() 方法中可以看到往浏览器写标签的代码：

```
out.write("\r\n");
out.write("<html>\r\n");
out.write("<head>\r\n");
out.write("    <title>Title</title>\r\n");
out.write("</head>\r\n");
out.write("<body>\r\n");
out.write("\r\n");
out.write("    <h1>hello jsp</h1>\r\n");
out.write("\r\n");
out.write("    ");
System.out.println("hello,jsp~");
int i = 3;

out.write("\r\n");
out.write("\r\n");
out.write("</body>\r\n");
out.write("</html>\r\n");
```

以前我们自己写 servlet 时，这部分代码是由我们自己来写，现在有了 jsp 后，由tomcat完成这部分功能。

4, JSP 脚本

JSP脚本用于在 JSP页面内定义 Java代码。在之前的入门案例中我们就在 JSP 页面定义的 Java 代码就是 JSP 脚本。

4.1 JSP 脚本分类

JSP 脚本有如下三个分类：

- <%...%>：内容会直接放到 _jspService()方法之中
- <%=...%>：内容会放到out.print()中，作为out.print()的参数
- <%!...%>：内容会放到 _jspService()方法之外，被类直接包含

代码演示：

在 hello.jsp 中书写

```
1 <%
2     System.out.println("hello,jsp~");
3     int i = 3;
4 %>
```

通过浏览器访问 `hello.jsp` 后，查看转换的 `hello_jsp.java` 文件，`i` 变量定义在了 `jspService()` 方法中

```
System.out.println("hello,jsp~");
int i = 3;
```

在 `hello.jsp` 中书写

```
1 <%="hello"%>
2 <%=i%>
```

通过浏览器访问 `hello.jsp` 后，查看转换的 `hello_jsp.java` 文件，该脚本的内容被放在了 `out.print()` 中，作为参数

```
out.print("hello");
out.write("\r\n");
out.write("    ");
out.print(i);
```

在 `hello.jsp` 中书写

```
1 <%
2     void show(){}
3     String name = "zhangsan";
4 %>
```

通过浏览器访问 `hello.jsp` 后，查看转换的 `hello_jsp.java` 文件，该脚本的内容被放在了成员位置

```
public final class hello_jsp extends org.apache.jasper.runtime.HttpJspBase
    implements org.apache.jasper.runtime.JspSourceDependent {

    void show(){}
    String name = "zhangsan";
```

4.2 案例

4.2.1 需求

使用JSP脚本展示品牌数据

新增

序号	品牌名称	企业名称	排序	品牌介绍	状态	操作
1	三只松鼠	三只松鼠	100	三只松鼠，好吃不上火	启用	修改 删除
2	优衣库	优衣库	10	优衣库，服适人生	禁用	修改 删除
3	小米	小米科技有限公司	1000	为发烧而生	启用	修改 删除

说明：

- 在资料 `资料\1. JSP案例素材` 中提供了 `brand.html` 静态页面
- 在该案例中数据不从数据库中查询，而是在 JSP 页面上写死

4.2.2 实现

- 将资料 `资料\1. JSP案例素材` 中的 `Brand.java` 文件放置到项目的 `com.itheima.pojo` 包下
- 在项目的 `webapp` 中创建 `brand.jsp`，并将 `brand.html` 页面中的内容拷贝过来。`brand.jsp` 内容如下

```
1 <%@ page contentType="text/html; charset=UTF-8" language="java" %>
2 <!DOCTYPE html>
3 <html lang="en">
4 <head>
5     <meta charset="UTF-8">
```

```

6      <title>Title</title>
7  </head>
8  <body>
9  <input type="button" value="新增"><br>
10 <hr>
11   <table border="1" cellspacing="0" width="800">
12     <tr>
13       <th>序号</th>
14       <th>品牌名称</th>
15       <th>企业名称</th>
16       <th>排序</th>
17       <th>品牌介绍</th>
18       <th>状态</th>
19       <th>操作</th>
20
21     </tr>
22     <tr align="center">
23       <td>1</td>
24       <td>三只松鼠</td>
25       <td>三只松鼠</td>
26       <td>100</td>
27       <td>三只松鼠, 好吃不上火</td>
28       <td>启用</td>
29       <td><a href="#">修改</a> <a href="#">删除</a></td>
30     </tr>
31
32     <tr align="center">
33       <td>2</td>
34       <td>优衣库</td>
35       <td>优衣库</td>
36       <td>10</td>
37       <td>优衣库, 服适人生</td>
38       <td>禁用</td>
39
40       <td><a href="#">修改</a> <a href="#">删除</a></td>
41     </tr>
42
43     <tr align="center">
44       <td>3</td>
45       <td>小米</td>
46       <td>小米科技有限公司</td>
47       <td>1000</td>
48       <td>为发烧而生</td>
49       <td>启用</td>
50
51       <td><a href="#">修改</a> <a href="#">删除</a></td>
52     </tr>
53   </table>
54 </body>
55 </html>

```

现在页面中的数据都是假数据。

- 在 `brand.jsp` 中准备一些数据

```

1 <%
2   // 查询数据库
3   List<Brand> brands = new ArrayList<Brand>();
4   brands.add(new Brand(1, "三只松鼠", "三只松鼠", 100, "三只松鼠, 好吃不上火", 1));
5   brands.add(new Brand(2, "优衣库", "优衣库", 200, "优衣库, 服适人生", 0));
6   brands.add(new Brand(3, "小米", "小米科技有限公司", 1000, "为发烧而生", 1));
7 %>

```

注意：这里的类是需要导包的

- 将 brand.jsp 页面中的 table 标签中的数据改为动态的

```

1 <table border="1" cellspacing="0" width="800">
2   <tr>
3     <th>序号</th>
4     <th>品牌名称</th>
5     <th>企业名称</th>
6     <th>排序</th>
7     <th>品牌介绍</th>
8     <th>状态</th>
9     <th>操作</th>
10
11   </tr>
12
13 <%
14   for (int i = 0; i < brands.size(); i++) {
15     //获取集合中的 每一个 Brand 对象
16     Brand brand = brands.get(i);
17   }
18 >
19   <tr align="center">
20     <td>1</td>
21     <td>三只松鼠</td>
22     <td>三只松鼠</td>
23     <td>100</td>
24     <td>三只松鼠, 好吃不上火</td>
25     <td>启用</td>
26     <td><a href="#">修改</a> <a href="#">删除</a></td>
27   </tr>
28 </table>
```

上面的for循环需要将 tr 标签包裹起来，这样才能实现循环的效果，代码改进为

```

1 <table border="1" cellspacing="0" width="800">
2   <tr>
3     <th>序号</th>
4     <th>品牌名称</th>
5     <th>企业名称</th>
6     <th>排序</th>
7     <th>品牌介绍</th>
8     <th>状态</th>
9     <th>操作</th>
10
11   </tr>
12
13 <%
14   for (int i = 0; i < brands.size(); i++) {
15     //获取集合中的 每一个 Brand 对象
16     Brand brand = brands.get(i);
17   }
18 >
19   <tr align="center">
20     <td>1</td>
21     <td>三只松鼠</td>
22     <td>三只松鼠</td>
23     <td>100</td>
24     <td>三只松鼠, 好吃不上火</td>
25     <td>启用</td>
26     <td><a href="#">修改</a> <a href="#">删除</a></td>
27   </tr>
28 <%
29   }
30 >
31 </table>
```

注意: <%%> 里面写的是 Java 代码, 而外边写的是 HTML 标签

上面代码中的 `td` 标签中的数据都需要是动态的, 所以还需要改进

```
1 <table border="1" cellspacing="0" width="800">
2   <tr>
3     <th>序号</th>
4     <th>品牌名称</th>
5     <th>企业名称</th>
6     <th>排序</th>
7     <th>品牌介绍</th>
8     <th>状态</th>
9     <th>操作</th>
10
11   </tr>
12
13   <%
14   for (int i = 0; i < brands.size(); i++) {
15     // 获取集合中的 每一个 Brand 对象
16     Brand brand = brands.get(i);
17   %>
18   <tr align="center">
19     <td><%=brand.getId()%></td>
20     <td><%=brand.getBrandName()%></td>
21     <td><%=brand.getCompanyName()%></td>
22     <td><%=brand.getOrdered()%></td>
23     <td><%=brand.getDescription()%></td>
24     <td><%=brand.getStatus() == 1 ? "启用": "禁用"%></td>
25     <td><a href="#">修改</a> <a href="#">删除</a></td>
26   </tr>
27   <%
28   }
29   %>
30
31 </table>
```

4.2.3 成品代码

```
1 <%@ page import="com.itheima.pojo.Brand" %>
2 <%@ page import="java.util.List" %>
3 <%@ page import="java.util.ArrayList" %>
4 <%@ page contentType="text/html;charset=UTF-8" language="java" %>
5
6 <%
7   // 查询数据库
8   List<Brand> brands = new ArrayList<Brand>();
9   brands.add(new Brand(1, "三只松鼠", "三只松鼠", 100, "三只松鼠, 好吃不上火", 1));
10  brands.add(new Brand(2, "优衣库", "优衣库", 200, "优衣库, 服适人生", 0));
11  brands.add(new Brand(3, "小米", "小米科技有限公司", 1000, "为发烧而生", 1));
12
13 %>
14
15
16 <!DOCTYPE html>
17 <html lang="en">
18 <head>
19   <meta charset="UTF-8">
20   <title>Title</title>
21 </head>
22 <body>
23   <input type="button" value="新增"><br>
24   <hr>
25   <table border="1" cellspacing="0" width="800">
26     <tr>
```

```

27     <th>序号</th>
28     <th>品牌名称</th>
29     <th>企业名称</th>
30     <th>排序</th>
31     <th>品牌介绍</th>
32     <th>状态</th>
33     <th>操作</th>
34   </tr>
35   <%
36     for (int i = 0; i < brands.size(); i++) {
37       Brand brand = brands.get(i);
38     %}
39
40   <tr align="center">
41     <td><%=brand.getId()%></td>
42     <td><%=brand.getBrandName()%></td>
43     <td><%=brand.getCompanyName()%></td>
44     <td><%=brand.getOrdered()%></td>
45     <td><%=brand.getDescription()%></td>
46     <td><%=brand.getStatus() == 1 ? "启用" : "禁用"%></td>
47     <td><a href="#">修改</a> <a href="#">删除</a></td>
48   </tr>
49
50   <%
51   }
52   %>
53 </table>
54 </body>
55 </html>

```

4.2.4 测试

在浏览器地址栏输入 `http://localhost:8080/jsp-demo/brand.jsp`，页面展示效果如下

The screenshot shows a web browser window with the URL `localhost:8080/jsp-demo/brand.jsp` in the address bar. The page contains a table with the following data:

序号	品牌名称	企业名称	排序	品牌介绍	状态	操作
1	三只松鼠	三只松鼠	100	三只松鼠，好吃不上火	启用	修改 删除
2	优衣库	优衣库	200	优衣库，服适人生	禁用	修改 删除
3	小米	小米科技有限公司	1000	为发烧而生	启用	修改 删除

4.3 JSP 缺点

通过上面的案例，我们可以看到 JSP 的很多缺点。

由于 JSP 页面内，既可以定义 HTML 标签，又可以定义 Java 代码，造成了以下问题：

- 书写麻烦：特别是复杂的页面

既要写 HTML 标签，还要写 Java 代码

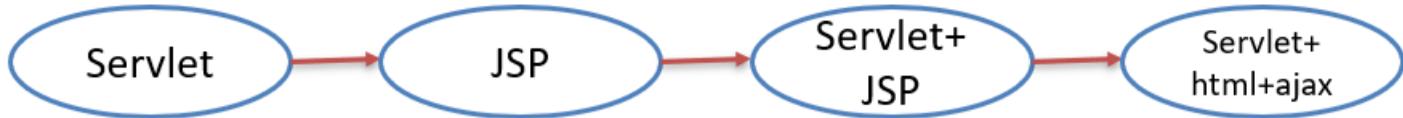
- 阅读麻烦

上面案例的代码，相信你后期再看这段代码时还需要花费很长的时间去梳理

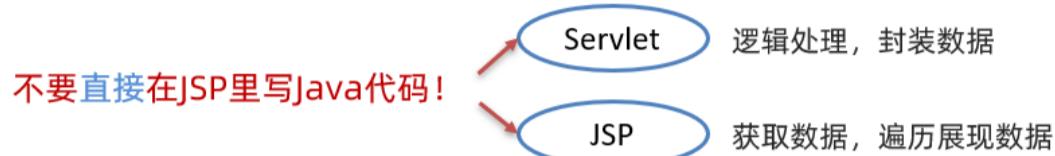
- 复杂度高：运行需要依赖于各种环境，JRE，JSP 容器，JavaEE...
- 占内存和磁盘：JSP 会自动生成 .java 和 .class 文件占磁盘，运行的是 .class 文件占内存
- 调试困难：出错后，需要找到自动生成的 .java 文件进行调试
- 不利于团队协作：前端人员不会 Java，后端人员不精 HTML

如果页面布局发生变化，前端工程师对静态页面进行修改，然后再交给后端工程师，由后端工程师再将该页面改为 JSP 页面

由于上述的问题，**JSP 已逐渐退出历史舞台**，以后开发更多的是使用 **HTML + Ajax** 来替代。Ajax 是我们后续会重点学习的技术。有个这个技术后，前端工程师负责前端页面开发，而后端工程师只负责前端代码开发。下来对技术的发展进行简单的说明



1. 第一阶段：使用 `servlet` 即实现逻辑代码编写，也对页面进行拼接。这种模式我们之前也接触过
2. 第二阶段：随着技术的发展，出现了 `JSP`，人们发现 `JSP` 使用起来比 `servlet` 方便很多，但是还是要在 `JSP` 中嵌套 `Java` 代码，也不利于后期的维护
3. 第三阶段：使用 `servlet` 进行逻辑代码开发，而使用 `JSP` 进行数据展示



4. 第四阶段：使用 `servlet` 进行后端逻辑代码开发，而使用 `HTML` 进行数据展示。而这里面就存在问题，`HTML` 是静态页面，怎么进行动态数据展示呢？这就是 `ajax` 的作用了。

既然 JSP 已经逐渐的退出历史舞台，那我们为什么还要学习 `JSP` 呢？原因有两点：

- 一些公司可能有些老项目还在用 `JSP`，所以要求我们必须学 `JSP`
- 我们如果不经历这些复杂的过程，就不能体现后面阶段开发的简单

接下来我们来学习第三阶段，使用 `EL表达式` 和 `JSTL 标签库` 替换 `JSP` 中的 `Java` 代码。

5, EL 表达式

5.1 概述

EL (全称Expression Language) 表达式语言，用于简化 JSP 页面内的 Java 代码。

EL 表达式的主要作用是 **获取数据**。其实就是从域对象中获取数据，然后将数据展示在页面上。

而 EL 表达式的语法也比较简单， `${expression}`。例如： `${brands}` 就是获取域中存储的 key 为 brands 的数据。

5.2 代码演示

- 定义 `servlet`，在 `servlet` 中封装一些数据并存储到 `request` 域对象中并转发到 `el-demo.jsp` 页面。

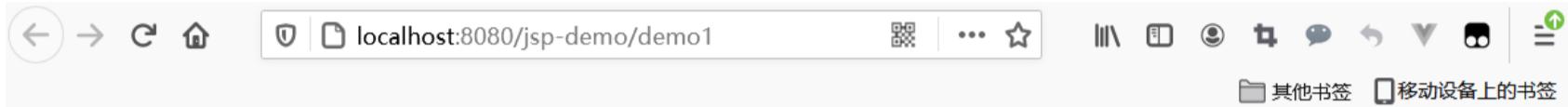
```
1 @WebServlet("/demo1")
2 public class ServletDemo1 extends HttpServlet {
3     @Override
4     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
5         //1. 准备数据
6         List<Brand> brands = new ArrayList<Brand>();
7         brands.add(new Brand(1, "三只松鼠", "三只松鼠", 100, "三只松鼠, 好吃不上火", 1));
8         brands.add(new Brand(2, "优衣库", "优衣库", 200, "优衣库, 服适人生", 0));
9         brands.add(new Brand(3, "小米", "小米科技有限公司", 1000, "为发烧而生", 1));
10
11         //2. 存储到request域中
12         request.setAttribute("brands", brands);
13
14         //3. 转发到 el-demo.jsp
15         request.getRequestDispatcher("/el-demo.jsp").forward(request, response);
16     }
17
18     @Override
19     protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
20         this.doGet(request, response);
21     }
22 }
```

注意：此处需要用转发，因为转发才可以使用 request 对象作为域对象进行数据共享

- 在 `el-demo.jsp` 中通过 EL 表达式 获取数据

```
1 <%@ page contentType="text/html; charset=UTF-8" language="java" %>
2 <html>
3 <head>
4     <title>Title</title>
5 </head>
6 <body>
7     ${brands}
8 </body>
9 </html>
```

- 在浏览器的地址栏输入 `http://localhost:8080/jsp-demo/demo1`，页面效果如下：

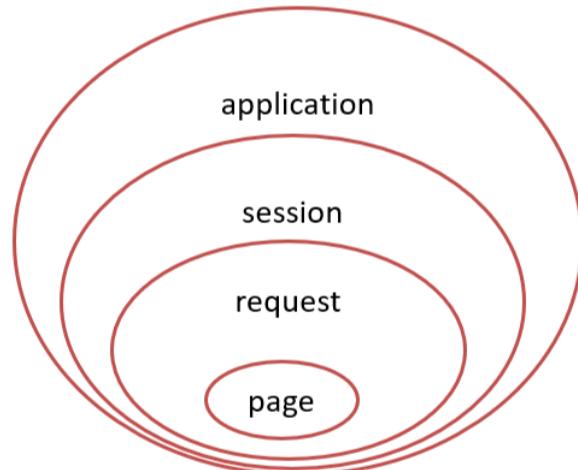


5.3 域对象

JavaWeb中有四大域对象，分别是：

- page：当前页面有效
- request：当前请求有效
- session：当前会话有效
- application：当前应用有效

el 表达式获取数据，会依次从这4个域中寻找，直到找到为止。而这四个域对象的作用范围如下图所示



例如：\${brands}，el 表达式获取数据，会先从page域对象中获取数据，如果没有再到 request 域对象中获取数据，如果没有再到 session 域对象中获取，如果还没有才会到 application 中获取数据。

6，JSTL标签

6.1 概述

JSP标准标签库(Jsp Standarded Tag Library)，使用标签取代JSP页面上的Java代码。如下代码就是JSTL标签

```
1 <c:if test="${flag == 1}">
2     男
3 </c:if>
4 <c:if test="${flag == 2}">
5     女
6 </c:if>
```

上面代码看起来是不是比 JSP 中嵌套 Java 代码看起来舒服好了。而且前端工程师对标签是特别敏感的，他们看到这段代码是能看懂的。

JSTL 提供了很多标签，如下图

标签	描述
<code><c:out></code>	用于在JSP中显示数据，就像 <code><%= ... %></code>
<code><c:set></code>	用于保存数据
<code><c:remove></code>	用于删除数据
<code><c:catch></code>	用来处理产生错误的异常状况，并且将错误信息储存起来
<code><c:if></code>	与我们在一般程序中用的if一样
<code><c:choose></code>	本身只当做 <code><c:when></code> 和 <code><c:otherwise></code> 的父标签
<code><c:when></code>	<code><c:choose></code> 的子标签，用来判断条件是否成立
<code><c:otherwise></code>	<code><c:choose></code> 的子标签，接在 <code><c:when></code> 标签后，当 <code><c:when></code> 标签判断为false时被执行
<code><c:import></code>	检索一个绝对或相对 URL，然后将其内容暴露给页面
<code><c:forEach></code>	基础迭代标签，接受多种集合类型
<code><c:forTokens></code>	根据指定的分隔符来分隔内容并迭代输出
<code><c:param></code>	用来给包含或重定向的页面传递参数
<code><c:redirect></code>	重定向至一个新的URL
<code><c:url></code>	使用可选的查询参数来创造一个URL

我们只对两个最常用的标签进行讲解，`<c:forEach>` 标签和 `<c:if>` 标签。

JSTL 使用也是比较简单的，分为如下步骤：

- 导入坐标

```

1 <dependency>
2   <groupId>jstl</groupId>
3   <artifactId>jstl</artifactId>
4   <version>1.2</version>
5 </dependency>
6 <dependency>
7   <groupId>taglibs</groupId>
8   <artifactId>standard</artifactId>
9   <version>1.1.2</version>
10 </dependency>

```

- 在JSP页面上引入JSTL标签库

```
1 <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
```

- 使用标签

6.2 if 标签

`<c:if>`：相当于 if 判断

- 属性：test，用于定义条件表达式

```

1 <c:if test="${flag == 1}">
2   男
3 </c:if>
4 <c:if test="${flag == 2}">
5   女
6 </c:if>

```

代码演示：

- 定义一个 `servlet`，在该 `servlet` 中向 `request` 域对象中添加键是 `status`，值为 `1` 的数据

```

1  @WebServlet("/demo2")
2  public class ServletDemo2 extends HttpServlet {
3      @Override
4      protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
5          //1. 存储数据到request域中
6          request.setAttribute("status", 1);
7
8          //2. 转发到 jstl-if.jsp
9          DataRequest.getRequestDispatcher("/jstl-if.jsp").forward(request, response);
10     }
11
12     @Override
13     protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
14         this.doGet(request, response);
15     }
16 }

```

- 定义 `jstl-if.jsp` 页面，在该页面使用 `<c:if>` 标签

```

1  <%@ page contentType="text/html; charset=UTF-8" language="java" %>
2  <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
3  <html>
4  <head>
5      <title>Title</title>
6  </head>
7  <body>
8      <%-- 
9          c:if: 来完成逻辑判断，替换java if else
10         --%>
11         <c:if test="${status ==1}">
12             启用
13         </c:if>
14
15         <c:if test="${status ==0}">
16             禁用
17         </c:if>
18     </body>
19 </html>

```

注意：在该页面已经要引入 JSTL核心标签库

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
```

6.3 forEach 标签

`<c:forEach>`：相当于 for 循环。java中有增强for循环和普通for循环，JSTL 中的 `<c:forEach>` 也有两种用法

6.3.1 用法一

类似于 Java 中的增强for循环。涉及到的 `<c:forEach>` 中的属性如下

- items: 被遍历的容器
- var: 遍历产生的临时变量
- varStatus: 遍历状态对象

如下代码，是从域对象中获取名为 brands 数据，该数据是一个集合；遍历遍历，并给该集合中的每一个元素起名为 `brand`，是 Brand对象。在循环里面使用 EL表达式获取每一个Brand对象的属性值

```

1 <c:forEach items="${brands}" var="brand">
2   <tr align="center">
3     <td>${brand.id}</td>
4     <td>${brand.brandName}</td>
5     <td>${brand.companyName}</td>
6     <td>${brand.description}</td>
7   </tr>
8 </c:forEach>

```

代码演示：

- `servlet` 还是使用之前的名为 `ServletDemo1`。
- 定义名为 `jstl-foreach.jsp` 页面，内容如下：

```

1 <%@ page contentType="text/html;charset=UTF-8" language="java" %>
2 <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
3
4 <!DOCTYPE html>
5 <html lang="en">
6 <head>
7   <meta charset="UTF-8">
8   <title>Title</title>
9 </head>
10 <body>
11 <input type="button" value="新增"><br>
12 <hr>
13 <table border="1" cellspacing="0" width="800">
14   <tr>
15     <th>序号</th>
16     <th>品牌名称</th>
17     <th>企业名称</th>
18     <th>排序</th>
19     <th>品牌介绍</th>
20     <th>状态</th>
21     <th>操作</th>
22   </tr>
23
24   <c:forEach items="${brands}" var="brand" varStatus="status">
25     <tr align="center">
26       <%--<td>${brand.id}</td>--%>
27       <td>${status.count}</td>
28       <td>${brand.brandName}</td>
29       <td>${brand.companyName}</td>
30       <td>${brand.ordered}</td>
31       <td>${brand.description}</td>
32       <c:if test="${brand.status == 1}">
33         <td>启用</td>
34       </c:if>
35       <c:if test="${brand.status != 1}">
36         <td>禁用</td>
37       </c:if>
38       <td><a href="#">修改</a> <a href="#">删除</a></td>
39     </tr>
40   </c:forEach>
41 </table>
42 </body>
43 </html>

```

6.3.2 用法二

类似于 Java 中的普通for循环。涉及到的 `<c:forEach>` 中的属性如下

- begin: 开始数
- end: 结束数
- step: 步长

实例代码：

从0循环到10，变量名是 `i`，每次自增1

```
1 <c:forEach begin="0" end="10" step="1" var="i">
2   ${i}
3 </c:forEach>
```

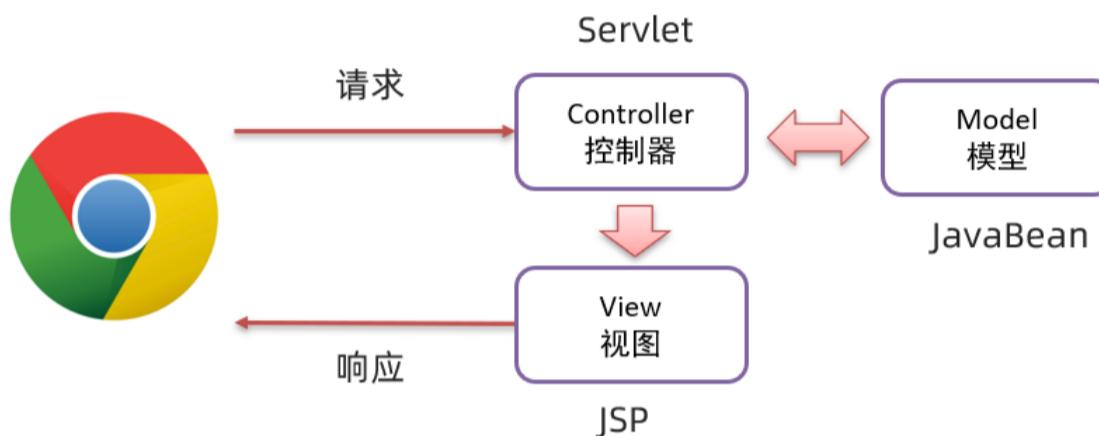
7. MVC模式和三层架构

MVC 模式和三层架构是一些理论的知识，将来我们使用了它们进行代码开发会让我们代码维护性和扩展性更好。

7.1 MVC模式

MVC 是一种分层开发的模式，其中：

- M: Model, 业务模型，处理业务
- V: View, 视图，界面展示
- C: Controller, 控制器，处理请求，调用模型和视图



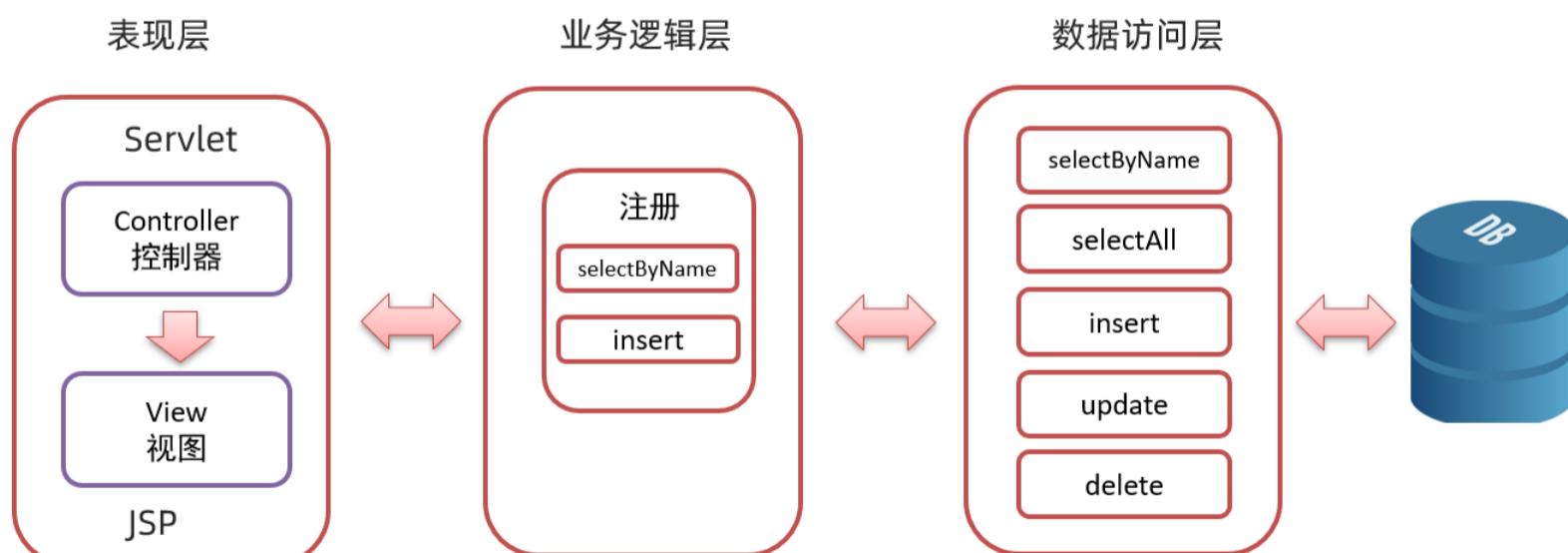
控制器（Servlet）用来接收浏览器发送过来的请求，控制器调用模型（JavaBean）来获取数据，比如从数据库查询数据；控制器获取到数据后再交由视图（JSP）进行数据展示。

MVC 好处：

- 职责单一，互不影响。每个角色做它自己的事，各司其职。
- 有利于分工协作。
- 有利于组件重用

7.2 三层架构

三层架构是将我们的项目分成了三个层面，分别是 表现层、业务逻辑层、数据访问层。



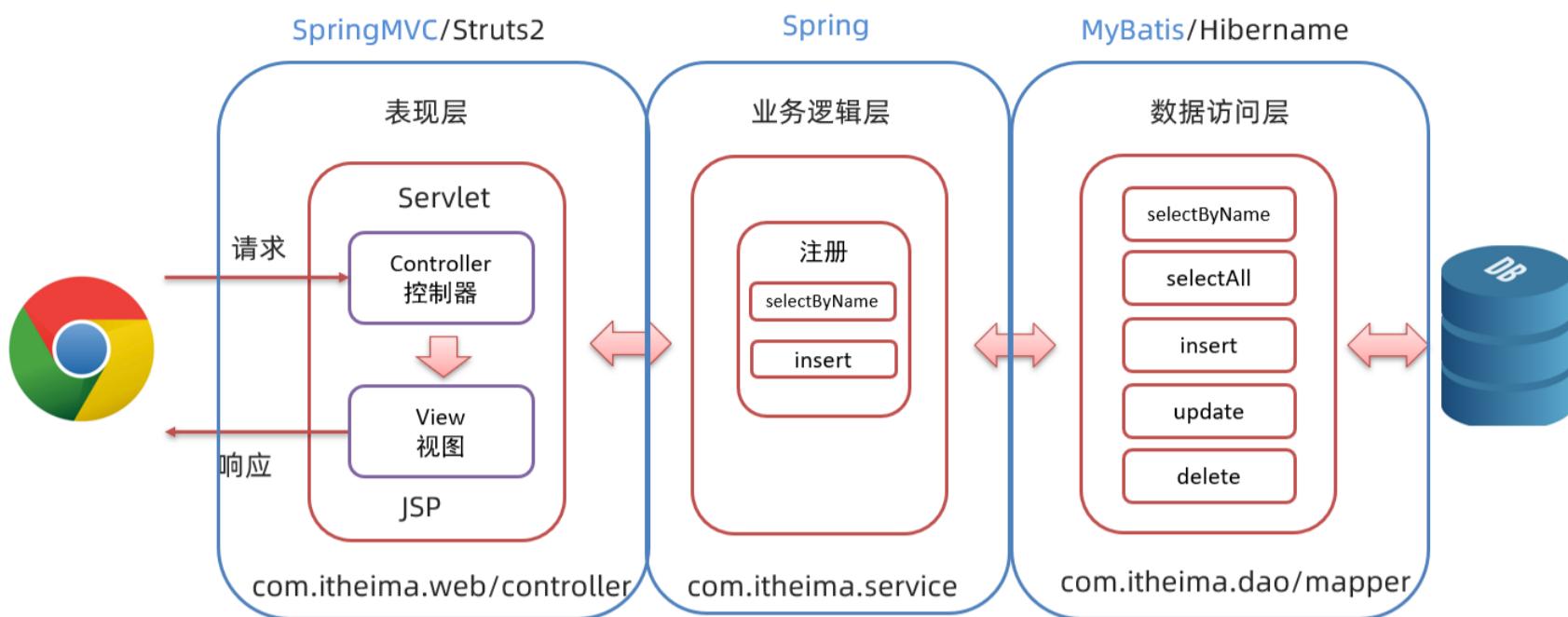
- 数据访问层：对数据库的CRUD基本操作
- 业务逻辑层：对业务逻辑进行封装，组合数据访问层层中基本功能，形成复杂的业务逻辑功能。例如 `注册业务功能`，我们会先调用 `数据访问层` 的 `selectByName()` 方法判断该用户名是否存在，如果不存在再调用 `数据访问层` 的 `insert()` 方法进行数据的添加操作
- 表现层：接收请求，封装数据，调用业务逻辑层，响应数据

而整个流程是，浏览器发送请求，表现层的Servlet接收请求并调用业务逻辑层的方法进行业务逻辑处理，而业务逻辑层方法调用数据访问层方法进行数据的操作，依次返回到Servlet，然后Servlet将数据交由 JSP 进行展示。

三层架构的每一层都有特有的包名称：

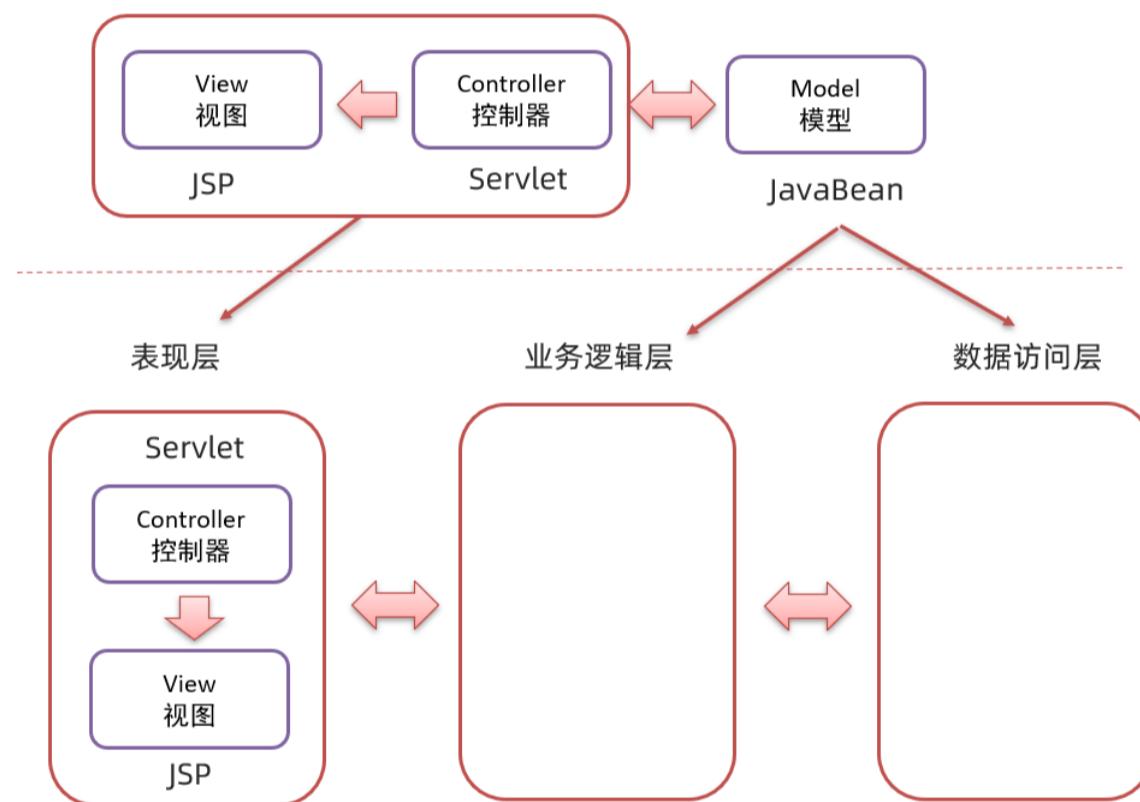
- 表现层: `com.itheima.controller` 或者 `com.itheima.web`
- 业务逻辑层: `com.itheima.service`
- 数据访问层: `com.itheima.dao` 或者 `com.itheima.mapper`

后期我们还会学习一些框架，不同的框架是对不同层进行封装的



7.3 MVC 和 三层架构

通过 MVC 和 三层架构 的学习，有些人肯定混淆了。那他们有什么区别和联系？



如上图上半部分是 MVC 模式，上图下半部分是三层架构。`MVC 模式` 中的 C (控制器) 和 V (视图) 就是 `三层架构` 中的表现层，而 `MVC 模式` 中的 M (模型) 就是 `三层架构` 中的 业务逻辑层 和 数据访问层。

可以将 `MVC 模式` 理解成是一个大的概念，而 `三层架构` 是对 `MVC 模式` 实现架构的思想。那么我们以后按照要求将不同层的代码写在不同的包下，每一层里功能职责做到单一，将来如果将表现层的技术换掉，而业务逻辑层和数据访问层的代码不需要发生变化。

8, 案例

需求：完成品牌数据的增删改查操作

序号	品牌名称	企业名称	排序	品牌介绍	状态	操作
1	三只松鼠	三只松鼠	100	三只松鼠，好吃不上火	启用	修改 删除
2	优衣库	优衣库	10	优衣库，服适人生	禁用	修改 删除
3	小米	小米科技有限公司	1000	为发烧而生	启用	修改 删除

这个功能我们之前一直在做，而这个案例是将今天学习的所有内容（包含 MVC模式 和 三层架构）进行应用，并将整个流程贯穿起来。

8.1 环境准备

环境准备工作，我们分以下步骤实现：

- 创建新的模块 brand_demo，引入坐标
- 创建三层架构的包结构
- 数据库表 tb_brand
- 实体类 Brand
- MyBatis 基础环境
 - Mybatis-config.xml
 - BrandMapper.xml
 - BrandMapper接口

8.1.1 创建工程

创建新的模块 brand_demo，引入坐标。我们只要分析出要用到哪儿些技术，那么需要哪儿些坐标也就明确了

- 需要操作数据库。mysql的驱动包
- 要使用mybatis框架。mybatis的依赖包
- web项目需要用到servlet和jsp。servlet和jsp的依赖包
- 需要使用 jstl 进行数据展示。jstl的依赖包

pom.xml 内容如下：

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
5     http://maven.apache.org/xsd/maven-4.0.0.xsd">
6   <modelVersion>4.0.0</modelVersion>
7   <groupId>org.example</groupId>
8   <artifactId>brand-demo</artifactId>
9   <version>1.0-SNAPSHOT</version>
10  <packaging>war</packaging>
11
12  <properties>
13    <maven.compiler.source>8</maven.compiler.source>
14    <maven.compiler.target>8</maven.compiler.target>
15  </properties>
16
17  <dependencies>
18    <!-- mybatis -->
19    <dependency>
20      <groupId>org.mybatis</groupId>
21      <artifactId>mybatis</artifactId>
22      <version>3.5.5</version>
23    </dependency>
24
25    <!--mysql-->
26    <dependency>
27      <groupId>mysql</groupId>
28      <artifactId>mysql-connector-java</artifactId>
29      <version>5.1.34</version>
30    </dependency>
31
32    <!--servlet-->
33    <dependency>
34      <groupId>javax.servlet</groupId>
35      <artifactId>javax.servlet-api</artifactId>
36      <version>3.1.0</version>
37      <scope>provided</scope>
38    </dependency>
39
40    <!--jsp-->
```

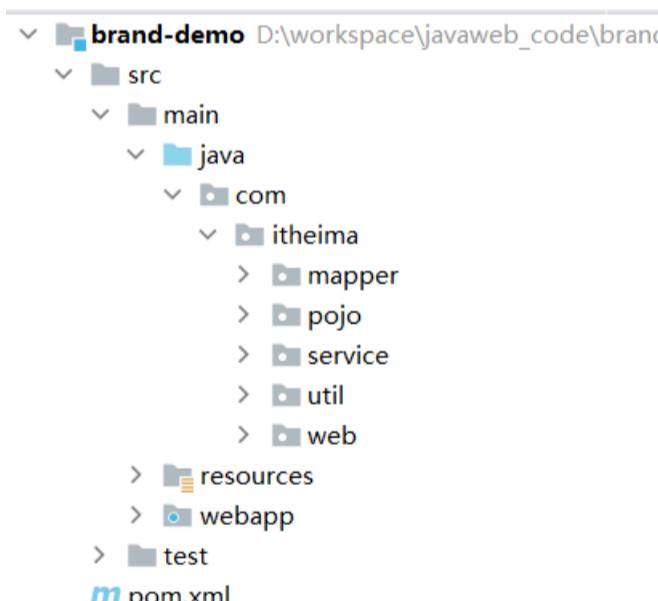
```

40     <dependency>
41         <groupId>javax.servlet.jsp</groupId>
42         <artifactId>jsp-api</artifactId>
43         <version>2.2</version>
44         <scope>provided</scope>
45     </dependency>
46
47     <!--jstl-->
48     <dependency>
49         <groupId>jstl</groupId>
50         <artifactId>jstl</artifactId>
51         <version>1.2</version>
52     </dependency>
53     <dependency>
54         <groupId>taglibs</groupId>
55         <artifactId>standard</artifactId>
56         <version>1.1.2</version>
57     </dependency>
58 </dependencies>
59
60 <build>
61     <plugins>
62         <plugin>
63             <groupId>org.apache.tomcat.maven</groupId>
64             <artifactId>tomcat7-maven-plugin</artifactId>
65             <version>2.2</version>
66         </plugin>
67     </plugins>
68 </build>
69 </project>

```

8.1.2 创建包

创建不同的包结构，用来存储不同的类。包结构如下



8.1.3 创建表

```

1 -- 删除tb_brand表
2 drop table if exists tb_brand;
3 -- 创建tb_brand表
4 create table tb_brand
5 (
6     -- id 主键
7     id      int primary key auto_increment,
8     -- 品牌名称
9     brand_name  varchar(20),
10    -- 企业名称
11    company_name varchar(20),
12    -- 排序字段
13    ordered      int,
14    -- 描述信息

```

```

15     description  varchar(100),
16     -- 状态: 0: 禁用  1: 启用
17     status      int
18 );
19 -- 添加数据
20 insert into tb_brand (brand_name, company_name, ordered, description, status)
21 values ('三只松鼠', '三只松鼠股份有限公司', 5, '好吃不上火', 0),
22         ('华为', '华为技术有限公司', 100, '华为致力于把数字世界带入每个人、每个家庭、每个组织，构建万物互联的智
23         能世界', 1),
24         ('小米', '小米科技有限公司', 50, 'are you ok', 1);

```

8.1.4 创建实体类

在 `pojo` 包下创建名为 `Brand` 的类。

```

1 public class Brand {
2     // id 主键
3     private Integer id;
4     // 品牌名称
5     private String brandName;
6     // 企业名称
7     private String companyName;
8     // 排序字段
9     private Integer ordered;
10    // 描述信息
11    private String description;
12    // 状态: 0: 禁用  1: 启用
13    private Integer status;
14
15
16    public Brand() {
17    }
18
19    public Brand(Integer id, String brandName, String companyName, String description) {
20        this.id = id;
21        this.brandName = brandName;
22        this.companyName = companyName;
23        this.description = description;
24    }
25
26    public Brand(Integer id, String brandName, String companyName, Integer ordered, String
description, Integer status) {
27        this.id = id;
28        this.brandName = brandName;
29        this.companyName = companyName;
30        this.ordered = ordered;
31        this.description = description;
32        this.status = status;
33    }
34
35    public Integer getId() {
36        return id;
37    }
38
39    public void setId(Integer id) {
40        this.id = id;
41    }
42
43    public String getBrandName() {
44        return brandName;
45    }
46
47    public void setBrandName(String brandName) {
48        this.brandName = brandName;
49    }

```

```

50
51     public String getCompanyName() {
52         return companyName;
53     }
54
55     public void setCompanyName(String companyName) {
56         this.companyName = companyName;
57     }
58
59     public Integer getordered() {
60         return ordered;
61     }
62
63     public void setOrdered(Integer ordered) {
64         this.ordered = ordered;
65     }
66
67     public String getDescription() {
68         return description;
69     }
70
71     public void setDescription(String description) {
72         this.description = description;
73     }
74
75     public Integer getStatus() {
76         return status;
77     }
78
79     public void setStatus(Integer status) {
80         this.status = status;
81     }
82
83     @Override
84     public String toString() {
85         return "Brand{" +
86                 "id=" + id +
87                 ", brandName='" + brandName + '\'' +
88                 ", companyName='" + companyName + '\'' +
89                 ", ordered=" + ordered +
90                 ", description='" + description + '\'' +
91                 ", status=" + status +
92                 '}';
93     }
94 }
95

```

8.1.5 准备mybatis环境

定义核心配置文件 `Mybatis-config.xml`，并将该文件放置在 `resources` 下

```

1 <?xml version="1.0" encoding="UTF-8" ?>
2 <!DOCTYPE configuration
3     PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
4     "http://mybatis.org/dtd/mybatis-3-config.dtd">
5 <configuration>
6     <!--起别名-->
7     <typeAliases>
8         <package name="com.itheima.pojo"/>
9     </typeAliases>
10
11    <environments default="development">
12        <environment id="development">
13            <transactionManager type="JDBC"/>
14            <dataSource type="POOLED">

```

```

15         <property name="driver" value="com.mysql.jdbc.Driver"/>
16         <property name="url" value="jdbc:mysql:///db1?
useSSL=false&useServerPrepStmts=true"/>
17         <property name="username" value="root"/>
18         <property name="password" value="1234"/>
19     </dataSource>
20   </environment>
21 </environments>
22 <mappers>
23   <!--扫描mapper-->
24   <package name="com.itheima.mapper"/>
25 </mappers>
26 </configuration>

```

在 `resources` 下创建放置映射配置文件的目录结构 `com/itheima/mapper`，并在该目录下创建映射配置文件 `BrandMapper.xml`

```

1 <?xml version="1.0" encoding="UTF-8" ?>
2 <!DOCTYPE mapper
3   PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
4   "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
5 <mapper namespace="com.itheima.mapper.BrandMapper">
6
7 </mapper>

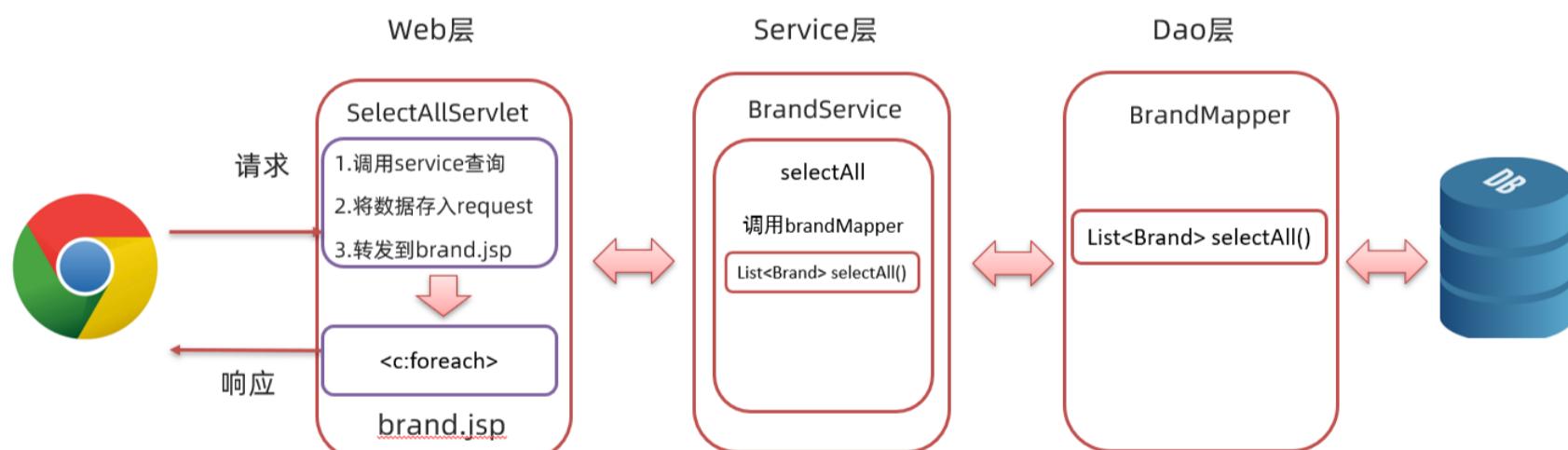
```

8.2 查询所有



当我们点击 `index.html` 页面中的 `查询所有` 这个超链接时，就能查询到上图右半部分的数据。

对于上述的功能，点击 `查询所有` 超链接是需要先调用后端的 `servlet`，由 `servlet` 跳转到对应的页面进行数据的动态展示。而整个流程如下图：



8.2.1 编写BrandMapper

在 `mapper` 包下创建 `BrandMapper` 接口，在接口中定义 `selectAll()` 方法

```

1 /**
2  * 查询所有
3  * @return
4 */
5 @Select("select * from tb_brand")
6 List<Brand> selectAll();

```

8.2.2 编写工具类

在 `com.itheima` 包下创建 `utils` 包，并在该包下创建名为 `SqlSessionFactoryUtils` 工具类

```

1 public class SqlSessionFactoryUtils {
2

```

```

3     private static SqlSessionFactory sqlSessionFactory;
4
5     static {
6         //静态代码块会随着类的加载而自动执行，且只执行一次
7         try {
8             String resource = "mybatis-config.xml";
9             InputStream inputStream = Resources.getResourceAsStream(resource);
10            sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);
11        } catch (IOException e) {
12            e.printStackTrace();
13        }
14    }
15
16    public static SqlSessionFactory getSqlSessionFactory(){
17        return sqlSessionFactory;
18    }
19}

```

8.2.3 编写BrandService

在 `service` 包下创建 `BrandService` 类

```

1  public class BrandService {
2      SqlSessionFactory factory = SqlSessionFactoryUtils.getSqlSessionFactory();
3
4      /**
5       * 查询所有
6       * @return
7       */
8      public List<Brand> selectAll(){
9          //调用BrandMapper.selectAll()
10
11         //2. 获取SqlSession
12         SqlSession sqlSession = factory.openSession();
13         //3. 获取BrandMapper
14         BrandMapper mapper = sqlSession.getMapper(BrandMapper.class);
15
16         //4. 调用方法
17         List<Brand> brands = mapper.selectAll();
18
19         sqlSession.close();
20
21         return brands;
22     }
23 }

```

8.2.4 编写Servlet

在 `web` 包下创建名为 `SelectAllServlet` 的 `servlet`，该 `servlet` 的逻辑如下：

- 调用 `BrandService` 的 `selectAll()` 方法进行业务逻辑处理，并接收返回的结果
- 将上一步返回的结果存储到 `request` 域对象中
- 跳转到 `brand.jsp` 页面进行数据的展示

具体的代码如下：

```

1  @WebServlet("/selectAllServlet")
2  public class SelectAllServlet extends HttpServlet {
3      private BrandService service = new BrandService();
4
5      @Override
6      protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
7          ServletException, IOException {
8          //1. 调用BrandService完成查询

```

```

9      List<Brand> brands = service.selectAll();
10     //2. 存入request域中
11     request.setAttribute("brands",brands);
12     //3. 转发到brand.jsp
13     request.getRequestDispatcher("/brand.jsp").forward(request,response);
14   }
15
16   @Override
17   protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
18   ServletException, IOException {
19     this.doGet(request, response);
20   }

```

8.2.5 编写brand.jsp页面

将资料 [资料\2. 品牌增删改查案例\静态页面] 下的 `brand.html` 页面拷贝到项目的 `webapp` 目录下，并将该页面改成 `brand.jsp` 页面，而 `brand.jsp` 页面在表格中使用 `JSTL` 和 `EL表达式` 从 `request` 域对象中获取名为 `brands` 的集合数据并展示出来。页面内容如下：

```

1 <%@ page contentType="text/html; charset=UTF-8" language="java" %>
2 <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
3
4 <!DOCTYPE html>
5 <html lang="en">
6 <head>
7   <meta charset="UTF-8">
8   <title>Title</title>
9 </head>
10 <body>
11 <hr>
12 <table border="1" cellspacing="0" width="80%">
13   <tr>
14     <th>序号</th>
15     <th>品牌名称</th>
16     <th>企业名称</th>
17     <th>排序</th>
18     <th>品牌介绍</th>
19     <th>状态</th>
20     <th>操作</th>
21   </tr>
22
23   <c:forEach items="${brands}" var="brand" varStatus="status">
24     <tr align="center">
25       <%--<td>${brand.id}</td>--%>
26       <td>${status.count}</td>
27       <td>${brand.brandName}</td>
28       <td>${brand.companyName}</td>
29       <td>${brand.ordered}</td>
30       <td>${brand.description}</td>
31       <c:if test="${brand.status == 1}">
32         <td>启用</td>
33       </c:if>
34       <c:if test="${brand.status != 1}">
35         <td>禁用</td>
36       </c:if>
37       <td><a href="/brand-demo/selectByIdServlet?id=${brand.id}">修改</a> <a href="#">删除</a></td>
38     </tr>
39   </c:forEach>
40 </table>
41 </body>
42 </html>

```

8.2.6 测试

启动服务器，并在浏览器输入 `http://localhost:8080/brand-demo/index.html`，看到如下 `查询所有` 的超链接，点击该链接就可以查询出所有的品牌数据

序号	品牌名称	企业名称	排序	品牌介绍	状态	操作
1			5	好吃不上火	禁用	修改 删除
2			100	华为致力于把数字世界带入每个人、每个家庭、每个组织，构建万物互联的智能世界	启用	修改 删除
3			50	are you ok	启用	修改 删除

为什么出现这个问题呢？是因为查询到的字段名和实体类对象的属性名没有一一对应。相比看到这大家一定会解决了，就是在映射配置文件中使用 `resultMap` 标签定义映射关系。映射配置文件内容如下：

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <!DOCTYPE mapper
3     PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
4     "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
5 <mapper namespace="com.itheima.mapper.BrandMapper">
6
7     <resultMap id="brandResultMap" type="brand">
8         <result column="brand_name" property="brandName"></result>
9         <result column="company_name" property="companyName"></result>
10    </resultMap>
11 </mapper>
```

并且在 `BrandMapper` 接口中使用 `@ResultMap` 注解指定使用该映射

```
1 /**
2  * 查询所有
3  * @return
4  */
5 @Select("select * from tb_brand")
6 @ResultMap("brandResultMap")
7 List<Brand> selectAll();
```

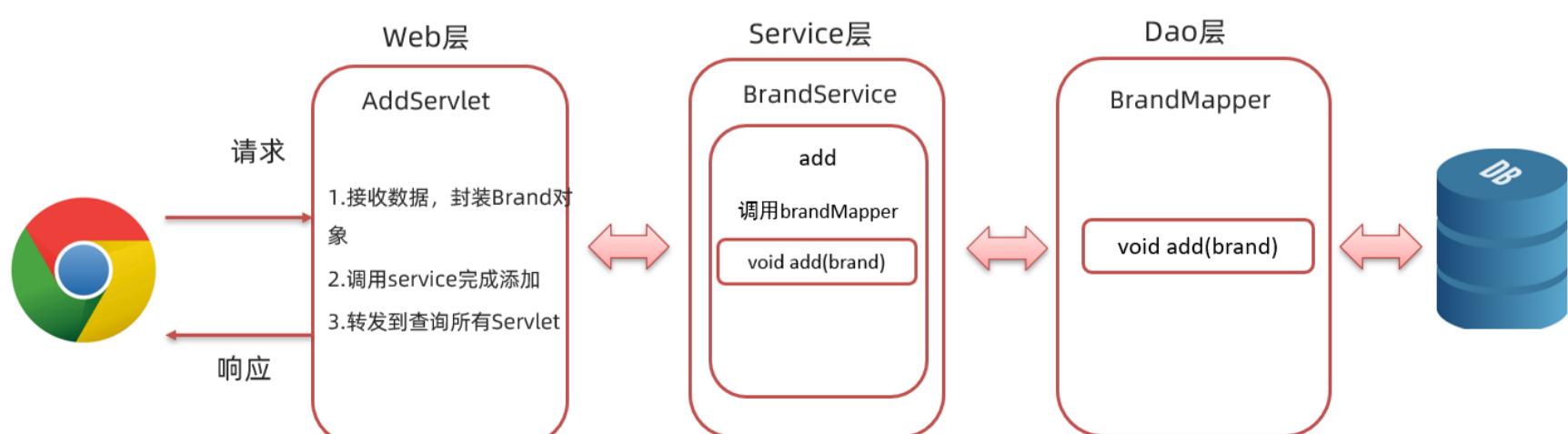
重启服务器，再次访问就能看到我们想要的数据了

序号	品牌名称	企业名称	排序	品牌介绍	状态	操作
1	三只松鼠	三只松鼠股份有限公司	5	好吃不上火	禁用	修改 删除
2	华为	华为技术有限公司	100	华为致力于把数字世界带入每个人、每个家庭、每个组织，构建万物互联的智能世界	启用	修改 删除
3	小米	小米科技有限公司	50	are you ok	启用	修改 删除

8.3 添加



上图是做添加功能流程。点击 `新增` 按钮后，会先跳转到 `addBrand.jsp` 新增页面，在该页面输入要添加的数据，输入完毕后点击 `提交` 按钮，需要将数据提交到后端，而后端进行数据添加操作，并重新将所有的数据查询出来。整个流程如下：



接下来我们根据流程来实现功能：

8.3.1 编写BrandMapper方法

在 `BrandMapper` 接口，在接口中定义 `add(Brand brand)` 方法

```
1 @Insert("insert into tb_brand values(null,#{brandName},#{companyName},#{ordered},#  
2 {description},#{status})")  
2 void add(Brand brand);
```

8.3.2 编写BrandService方法

在 `BrandService` 类中定义添加品牌数据方法 `add(Brand brand)`

```
1 /**
2  * 添加
3  * @param brand
4  */
5 public void add(Brand brand){  
6  
7     //2. 获取sqlsession  
8     SqlSession sqlSession = factory.openSession();  
9     //3. 获取BrandMapper  
10    BrandMapper mapper = sqlSession.getMapper(BrandMapper.class);  
11  
12    //4. 调用方法  
13    mapper.add(brand);  
14  
15    //提交事务  
16    sqlSession.commit();  
17    //释放资源  
18    sqlSession.close();  
19}
```

8.3.3 改进brand.jsp页面

我们需要在该页面表格的上面添加 `新增` 按钮

```
1 <input type="button" value="新增" id="add"><br>
```

并给该按钮绑定单击事件，当点击了该按钮需要跳转到 `brand.jsp` 添加品牌数据的页面

```
1 <script>  
2     document.getElementById("add").onclick = function (){  
3         location.href = "/brand-demo/addBrand.jsp";  
4     }  
5 </script>
```

注意：该 `script` 标签建议放在 `body` 结束标签前面。

8.3.4 编写addBrand.jsp页面

从资料 `资料\2. 品牌增删改查案例\静态页面` 中将 `addBrand.html` 页面拷贝到项目的 `webapp` 下，并改成 `addBrand.jsp` 动态页面

```
1 <%@ page contentType="text/html; charset=UTF-8" language="java" %>  
2 <!DOCTYPE html>  
3 <html lang="en">  
4  
5 <head>  
6     <meta charset="UTF-8">  
7     <title>添加品牌</title>  
8 </head>  
9 <body>
```

```

10 <h3>添加品牌</h3>
11 <form action="/brand-demo/addServlet" method="post">
12   品牌名称: <input name="brandName"><br>
13   企业名称: <input name="companyName"><br>
14   排序: <input name="ordered"><br>
15   描述信息: <textarea rows="5" cols="20" name="description"></textarea><br>
16   状态:
17   <input type="radio" name="status" value="0">禁用
18   <input type="radio" name="status" value="1">启用<br>
19
20   <input type="submit" value="提交">
21 </form>
22 </body>
23 </html>

```

8.3.5 编写servlet

在 web 包下创建 AddServlet 的 servlet，该 servlet 的逻辑如下：

- 设置处理post请求乱码的字符集
- 接收客户端提交的数据
- 将接收到的数据封装到 Brand 对象中
- 调用 BrandService 的 add() 方法进行添加的业务逻辑处理
- 跳转到 selectAllServlet 资源重新查询数据

具体的代码如下：

```

1 @WebServlet("/addServlet")
2 public class AddServlet extends HttpServlet {
3   private BrandService service = new BrandService();
4
5
6   @Override
7   protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
8   ServletException, IOException {
9
10   //处理POST请求的乱码问题
11   request.setCharacterEncoding("utf-8");
12
13   //1. 接收表单提交的数据，封装为一个Brand对象
14   String brandName = request.getParameter("brandName");
15   String companyName = request.getParameter("companyName");
16   String ordered = request.getParameter("ordered");
17   String description = request.getParameter("description");
18   String status = request.getParameter("status");
19
20   //封装为一个Brand对象
21   Brand brand = new Brand();
22   brand.setBrandName(brandName);
23   brand.setCompanyName(companyName);
24   brand.setOrdered(Integer.parseInt(ordered));
25   brand.setDescription(description);
26   brand.setStatus(Integer.parseInt(status));
27
28   //2. 调用service 完成添加
29   service.add(brand);
30
31   //3. 转发到查询所有Servlet
32   request.getRequestDispatcher("/selectAllServlet").forward(request, response);
33 }
34
35   @Override
36   protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
37   ServletException, IOException {
38     this.doGet(request, response);

```

```
37 }  
38 }
```

8.3.6 测试

点击 `brand.jsp` 页面的 `新增` 按钮，会跳转到 `addBrand.jsp` 页面

添加品牌

品牌名称: 鸿星尔克
企业名称: 鸿星尔克
排序: 10000
to be no.1

描述信息:

状态: 禁用 启用

提交

点击 `提交` 按钮，就能看到如下页面，里面就包含我们刚添加的数据

序号	品牌名称	企业名称	排序	品牌介绍	状态	操作
1	三只松鼠	三只松鼠股份有限公司	5	好吃不上火	禁用	修改 删除
2	华为	华为技术有限公司	100	华为致力于把数字世界带入每个人、每个家庭、每个组织，构建万物互联的智能世界	启用	修改 删除
3	小米	小米科技有限公司	50	are you ok	启用	修改 删除
4	鸿星尔克	鸿星尔克	10000	to be no.1	启用	修改 删除

8.4 修改

电商后台原型

商品管理

订单管理 物流管理 促销管理 文章管理 权限管理 VIP原型

新增

序号	品牌LOGO	品牌名称	企业名称	排序	当前状态	操作
010		三只松鼠	这里是企业名称	2	<input checked="" type="checkbox"/>	删除 编辑 查看详情
009		优衣库	这里是企业名称	1	<input checked="" type="checkbox"/>	删除 编辑 查看详情
008		小米	这里是企业名称	6	<input checked="" type="checkbox"/>	删除 编辑 查看详情
007		阿迪达斯	这里是企业名称	6	<input checked="" type="checkbox"/>	删除 编辑 查看详情
006		百草味	这里是企业名称	4	<input type="checkbox"/>	删除 编辑 查看详情

点击每条数据后面的 `编辑` 按钮会跳转到修改页面，如下图：

电商后台原型

商品管理

订单管理 物流管理 促销管理 文章管理 权限管理 VIP原型

商品管理

新增商品

商品评价

商品回收站

商品属性

商品分类

新增分类

商品品牌

新增品牌

商品列表

首页

品牌LOGO

+ 上傳logo
支持JPG、PNG、GIF格式的图片，大小请小于200K，尺寸200*120px

* 品牌名称: zara旗舰店

* 企业名称: 这里是企业名称 E

排序: 20
排序为空时，默认按新增时间倒序排在最前面

备注信息
在逻辑清晰的基础上，用相对简洁的语言进行描述。每句话尽量不要超过15个字，如果一句话超过15个字，尽量从中间寻找断句的地方。在逻辑清晰的基础上，用相对简洁的语言进行描述。

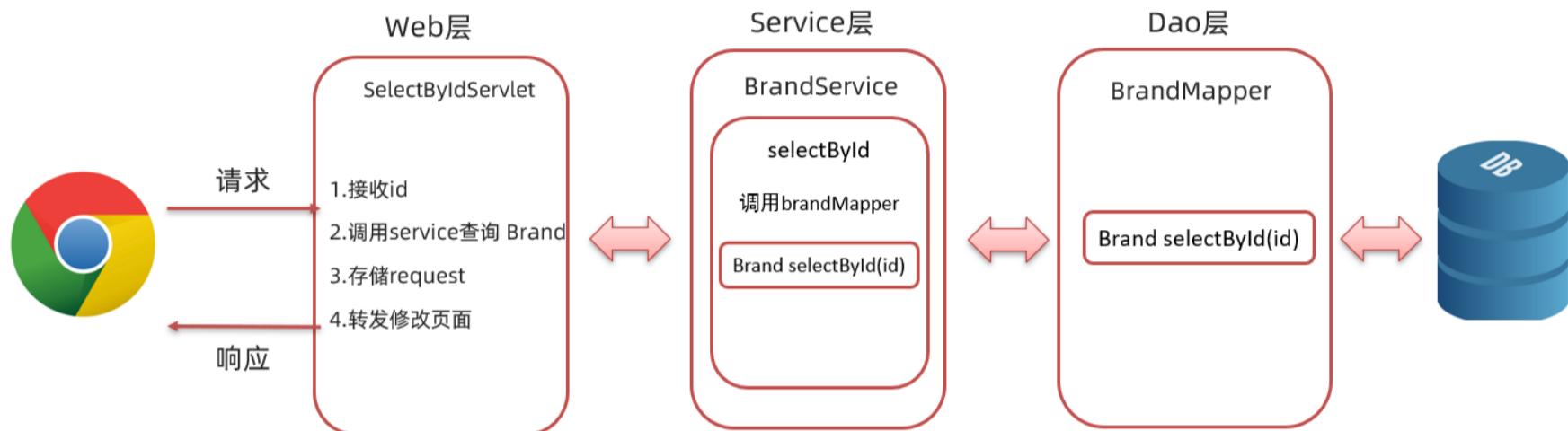
在该修改页面我们可以看到将 `编辑` 按钮所在行的数据 **回显** 到表单，然后需要修改那个数据在表单中进行修改，然后点击 `提交` 的按钮将数据提交到后端，后端再将数据存储到数据库中。

从上面的例子我们知道 `修改` 功能需要从两方面进行实现，数据回显和修改操作。

8.4.1 回显数据



上图就是回显数据的效果。要实现这个效果，那当点击 `修改` 按钮时不能直接跳转到 `update.jsp` 页面，而是需要先带着当前行数据的 `id` 请求后端程序，后端程序根据 `id` 查询数据，将数据存储到域对象中跳转到 `update.jsp` 页面进行数据展示。整体流程如下



8.4.1.1 编写BrandMapper方法

在 `BrandMapper` 接口中定义 `selectById(int id)` 方法

```
1 /**
2  * 根据id查询
3  * @param id
4  * @return
5 */
6 @Select("select * from tb_brand where id = #{id}")
7 @ResultMap("brandResultMap")
8 Brand selectById(int id);
```

8.4.1.2 编写BrandService方法

在 `BrandService` 类中定义根据id查询数据方法 `selectById(int id)`

```
1 /**
2  * 根据id查询
3  * @return
4 */
5 public Brand selectById(int id){
6     //调用BrandMapper.selectAll()
7     //2. 获取SqlSession
8     SqlSession sqlSession = factory.openSession();
9     //3. 获取BrandMapper
10    BrandMapper mapper = sqlSession.getMapper(BrandMapper.class);
11    //4. 调用方法
12    Brand brand = mapper.selectById(id);
13    sqlSession.close();
14    return brand;
15 }
```

8.4.1.3 编写servlet

在 `web` 包下创建 `SelectByIdServlet` 的 `servlet`，该 `servlet` 的逻辑如下：

- 获取请求数据 `id`
- 调用 `BrandService` 的 `selectById()` 方法进行数据查询的业务逻辑

- 将查询到的数据存储到 request 域对象中
- 跳转到 `update.jsp` 页面进行数据真实

具体代码如下：

```

1  @WebServlet("/selectByIdServlet")
2  public class SelectByIdServlet extends HttpServlet {
3      private BrandService service = new BrandService();
4
5      @Override
6      protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
7          //1. 接收id
8          String id = request.getParameter("id");
9          //2. 调用service查询
10         Brand brand = service.selectById(Integer.parseInt(id));
11         //3. 存储到request中
12         request.setAttribute("brand", brand);
13         //4. 转发到update.jsp
14         request.getRequestDispatcher("/update.jsp").forward(request, response);
15     }
16
17     @Override
18     protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
19         this.doGet(request, response);
20     }
21 }
```

8.4.1.4 编写update.jsp页面

拷贝 `addBrand.jsp` 页面，改名为 `update.jsp` 并做出以下修改：

- `title` 标签内容改为 `修改品牌`
- `form` 标签的 `action` 属性值改为 `/brand-demo/updateServlet`
- `input` 标签要进行数据回显，需要设置 `value` 属性

```

1 品牌名称: <input name="brandName" value="${brand.brandName}"><br>
2 企业名称: <input name="companyName" value="${brand.companyName}"><br>
3 排序: <input name="ordered" value="${brand.ordered}"><br>
```

- `textarea` 标签要进行数据回显，需要在标签体中使用 `EL表达式`

```

1 描述信息: <textarea rows="5" cols="20" name="description">${brand.description} </textarea>
<br>
```

- 单选框使用 `if` 标签需要判断 `brand.status` 的值是 1 还是 0 在指定的单选框上使用 `checked` 属性，表示被选中状态

```

1 状态:
2 <c:if test="${brand.status == 0}">
3     <input type="radio" name="status" value="0" checked>禁用
4     <input type="radio" name="status" value="1">启用<br>
5 </c:if>
6
7 <c:if test="${brand.status == 1}">
8     <input type="radio" name="status" value="0" >禁用
9     <input type="radio" name="status" value="1" checked>启用<br>
10 </c:if>
```

综上，`update.jsp` 代码如下：

```

1 <%@ page contentType="text/html; charset=UTF-8" language="java" %>
2 <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
3 <!DOCTYPE html>
```

```

4 <html lang="en">
5   <head>
6     <meta charset="UTF-8">
7     <title>修改品牌</title>
8   </head>
9   <body>
10  <h3>修改品牌</h3>
11  <form action="/brand-demo/updateServlet" method="post">
12
13    品牌名称: <input name="brandName" value="${brand.brandName}"><br>
14    企业名称: <input name="companyName" value="${brand.companyName}"><br>
15    排序: <input name="ordered" value="${brand.ordered}"><br>
16    描述信息: <textarea rows="5" cols="20" name="description">${brand.description} </textarea>
17    <br>
18    状态:
19    <c:if test="${brand.status == 0}">
20      <input type="radio" name="status" value="0" checked>禁用
21      <input type="radio" name="status" value="1">启用<br>
22    </c:if>
23
24    <c:if test="${brand.status == 1}">
25      <input type="radio" name="status" value="0" >禁用
26      <input type="radio" name="status" value="1" checked>启用<br>
27    </c:if>
28
29    <input type="submit" value="提交">
30  </form>
31 </body>
32 </html>

```

8.4.2 修改数据

做完回显数据后，接下来我们要做修改数据了，而下图是修改数据的效果：

The screenshot shows the '修改品牌' (Modify Brand) page on the left and a database table on the right.

修改品牌 (Left):

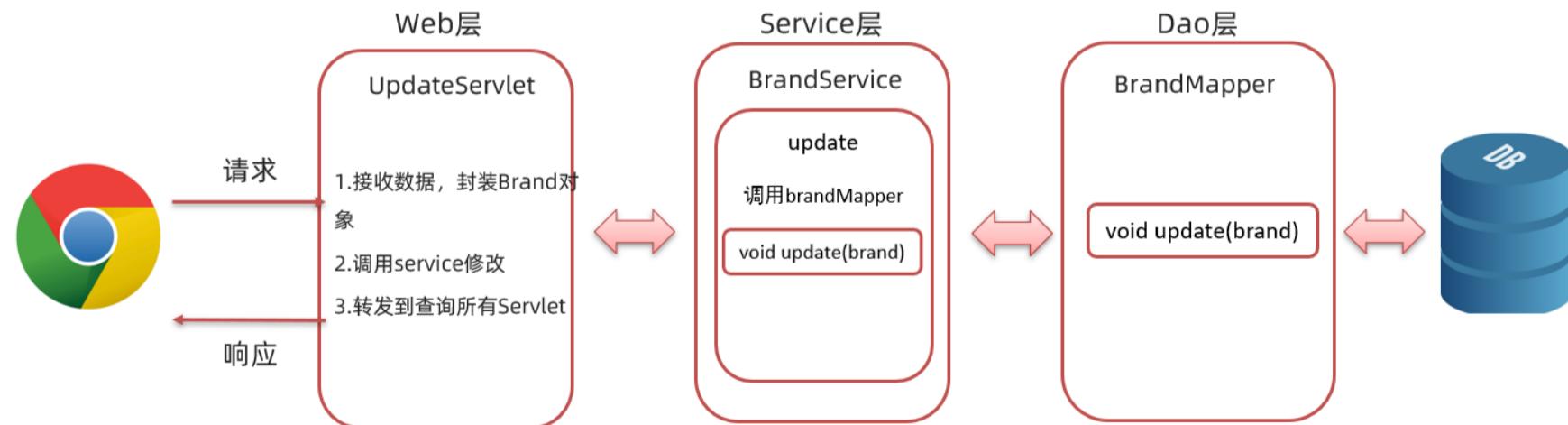
- Form fields:
 - 品牌名称: 三只松鼠
 - 企业名称: 三只松鼠
 - 排序: 100
 - 描述信息: 好吃不上火
 - 状态: 启用
- Submit button: 提交

数据库表 (Right):

序号	品牌名称	企业名称	排序	品牌介绍	状态	操作
1	三只松鼠	三只松鼠	100	三只松鼠, 好吃不上火	启用	修改 删除
2	优衣库	优衣库	10	优衣库, 服适人生	禁用	修改 删除
3	小米	小米科技有限公司	1000	为发烧而生	启用	修改 删除

update.jsp

在修改页面进行数据修改，点击 提交 按钮，会将数据提交到后端程序，后端程序会对表中的数据进行修改操作，然后重新进行数据的查询操作。整体流程如下：



8.4.2.1 编写BrandMapper方法

在 **BrandMapper** 接口中，在接口中定义 **update(Brand brand)** 方法

```
1 /**
2  * 修改
3  * @param brand
4 */
5 @Update("update tb_brand set brand_name = #{brandName},company_name = #{companyName},ordered =
6 void update(Brand brand);
```

8.4.2.2 编写BrandService方法

在 `BrandService` 类中定义根据id查询数据方法 `update(Brand brand)`

```
1 /**
2  * 修改
3  * @param brand
4 */
5 public void update(Brand brand){
6     //2. 获取SqlSession
7     SqlSession sqlSession = factory.openSession();
8     //3. 获取BrandMapper
9     BrandMapper mapper = sqlSession.getMapper(BrandMapper.class);
10    //4. 调用方法
11    mapper.update(brand);
12    //提交事务
13    sqlSession.commit();
14    //释放资源
15    sqlSession.close();
16 }
```

8.4.2.3 编写servlet

在 `web` 包下创建 `AddServlet` 的 `servlet`, 该 `servlet` 的逻辑如下:

- 设置处理post请求乱码的字符集
- 接收客户端提交的数据
- 将接收到的数据封装到 `Brand` 对象中
- 调用 `BrandService` 的 `update()` 方法进行添加的业务逻辑处理
- 跳转到 `selectAllServlet` 资源重新查询数据

具体的代码如下:

```
1 @WebServlet("/updateServlet")
2 public class UpdateServlet extends HttpServlet {
3     private BrandService service = new BrandService();
4
5     @Override
6     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
7     ServletException, IOException {
8
9         //处理POST请求的乱码问题
10        request.setCharacterEncoding("utf-8");
11        //1. 接收表单提交的数据, 封装为一个Brand对象
12        String id = request.getParameter("id");
13        String brandName = request.getParameter("brandName");
14        String companyName = request.getParameter("companyName");
15        String ordered = request.getParameter("ordered");
16        String description = request.getParameter("description");
17        String status = request.getParameter("status");
18
19        //封装为一个Brand对象
20        Brand brand = new Brand();
21        brand.setId(Integer.parseInt(id));
22        brand.setBrandName(brandName);
23        brand.setCompanyName(companyName);
24        brand.setOrdered(Integer.parseInt(ordered));
```

```

24     brand.setDescription(description);
25     brand.setStatus(Integer.parseInt(status));
26
27     //2. 调用service 完成修改
28     service.update(brand);
29
30     //3. 转发到查询所有Servlet
31     request.getRequestDispatcher("/selectAllServlet").forward(request, response);
32 }
33
34 @Override
35 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
36     this doGet(request, response);
37 }
38 }
```

存在问题：update.jsp 页面提交数据时是没有携带主键数据的，而后台修改数据需要根据主键进行修改。

针对这个问题，我们不希望页面将主键id展示给用户看，但是又希望在提交数据时能将主键id提交到后端。此时我们就想到了在学习 HTML 时学习的隐藏域，在 update.jsp 页面的表单中添加如下代码：

```

1 <%--隐藏域，提交id--%>
2 <input type="hidden" name="id" value="${brand.id}">
```

update.jsp 页面的最终代码如下：

```

1 <%@ page contentType="text/html; charset=UTF-8" language="java" %>
2 <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
3 <!DOCTYPE html>
4 <html lang="en">
5 <head>
6     <meta charset="UTF-8">
7     <title>修改品牌</title>
8 </head>
9 <body>
10 <h3>修改品牌</h3>
11 <form action="/brand-demo/updateServlet" method="post">
12
13     <%--隐藏域，提交id--%>
14     <input type="hidden" name="id" value="${brand.id}">
15
16     品牌名称: <input name="brandName" value="${brand.brandName}"><br>
17     企业名称: <input name="companyName" value="${brand.companyName}"><br>
18     排序: <input name="ordered" value="${brand.ordered}"><br>
19     描述信息: <textarea rows="5" cols="20" name="description">${brand.description} </textarea>
<br>
20     状态:
21     <c:if test="${brand.status == 0}">
22         <input type="radio" name="status" value="0" checked>禁用
23         <input type="radio" name="status" value="1">启用<br>
24     </c:if>
25
26     <c:if test="${brand.status == 1}">
27         <input type="radio" name="status" value="0" >禁用
28         <input type="radio" name="status" value="1" checked>启用<br>
29     </c:if>
30     <input type="submit" value="提交">
31 </form>
32 </body>
33 </html>
```


会话技术

今日目标

- 理解什么是会话跟踪技术
- 掌握Cookie的使用
- 掌握Session的使用
- 完善用户登录注册案例的功能

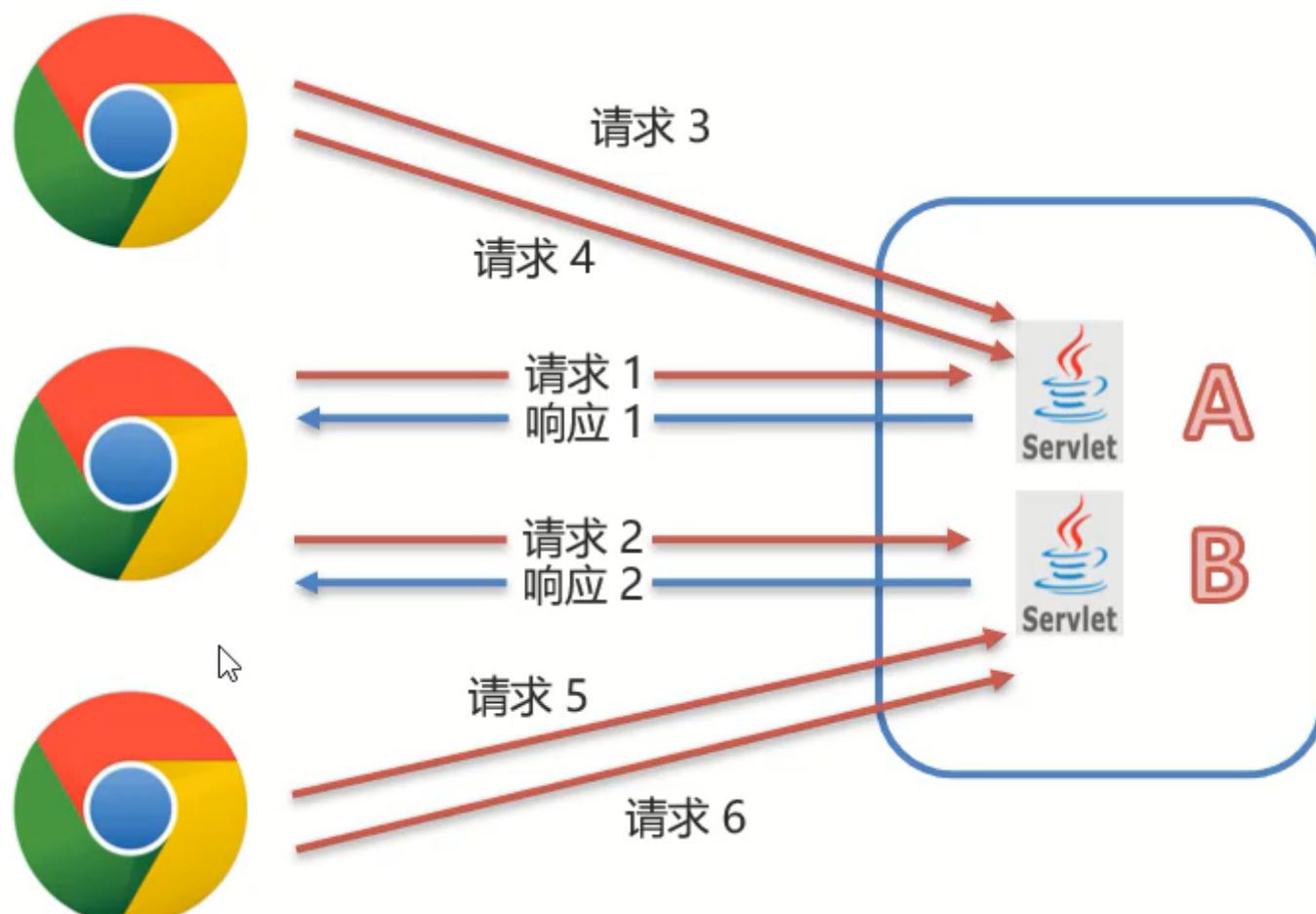
1. 会话跟踪技术的概述

对于会话跟踪这四个词，我们需要拆开来理解，首先要理解什么是会话，然后再去理解什么是会话跟踪：

- 会话：用户打开浏览器，访问web服务器的资源，会话建立，直到有一方断开连接，会话结束。在一次会话中可以包含多次请求和响应。
 - 从浏览器发出请求到服务端响应数据给前端之后，一次会话（在浏览器和服务端之间）就被建立了
 - 会话被建立后，如果浏览器或服务端都没有被关闭，则会话就会持续建立着
 - 浏览器和服务端就可以继续使用该会话进行请求发送和响应，上述的整个过程就称之为会话。

用实际场景来理解下会话，比如在我们访问京东的时候，当打开浏览器进入京东首页后，浏览器和京东的服务器之间就建立了一次会话，后面的搜索商品，查看商品的详情，加入购物车等都是在这一次会话中完成。

思考：下图中总共建立了几个会话？



每个浏览器都会与服务端建立了一个会话，加起来总共是3个会话。

- 会话跟踪：一种维护浏览器状态的方法，服务器需要识别多次请求是否来自于同一浏览器，以便在同一次会话的多次请求间**共享数据**。

- 服务器会收到多个请求，这多个请求可能来自多个浏览器，如上图中的6个请求来自3个浏览器
- 服务器需要用来识别请求是否来自同一个浏览器
- 服务器用来识别浏览器的过程，这个过程就是**会话跟踪**
- 服务器识别浏览器后就可以在同一个会话中多次请求之间来共享数据

那么我们又有一个问题需要思考，一个会话中的多次请求为什么要共享数据呢？有了这个数据共享功能后能实现哪些功能呢？

- 购物车：加入购物车和去购物车结算是两次请求，但是后面这次请求要想展示前一次请求所添加的商品，就需要用到数据共享。

The screenshot shows the JD.com shopping cart interface. A Redmi 9A smartphone is listed with a price of ¥599.00, reduced by -¥10.00. The cart summary at the bottom indicates 1 item totaling ¥589.00.

商品	单价	数量	小计	操作
Redmi 9A 5000mAh大电量 1300万 AI相机 八核处理器 人脸解锁 砂石黑 4GB+64GB	¥599.00	1	¥599.00	删除 移入关注

- 页面展示用户登录信息：很多网站，登录后访问多个功能发送多次请求后，浏览器上都会有当前登录用户的信息[用户名]，比如百度、京东、码云等。

The screenshot shows a GitHub profile page for the user 'superbaby'. The profile picture is highlighted with a red box. The page displays the user's activity: 1 repository, 0 pull requests, and 0 tasks.

- 网站登录页面的记住我功能：当用户登录成功后，勾选记住我按钮后下次再登录的时候，网站就会自动填充用户名和密码，简化用户的登录操作，多次登录就会有多个请求，他们之间也涉及到共享数据



- 登录页面的验证码功能：生成验证码和输入验证码点击注册这也是两次请求，这两次请求的数据之间要进行对比，相同则允许注册，不同则拒绝注册，该功能的实现也需要在同一次会话中共享数据。



通过这几个例子的讲解，相信大家对会话追踪技术已经有了一定的理解，该技术在实际开发中也非常重要。那么接下来我们就需要去学习下会话跟踪技术，在学习这些技术之前，我们需要思考：为什么现在浏览器和服务器不支持数据共享呢？

- 浏览器和服务器之间使用的是HTTP请求来进行数据传输
- HTTP协议是**无状态**的，每次浏览器向服务器请求时，服务器都会将该请求视为**新的**请求
- HTTP协议设计成无状态的目的是让每次请求之间相互独立，互不影响
- 请求与请求之间独立后，就无法实现多次请求之间的数据共享

分析完具体的原因后，那么该如何实现会话跟踪技术呢？具体的实现方式有：

- (1) 客户端会话跟踪技术：**Cookie**
- (2) 服务端会话跟踪技术：**Session**

这两个技术都可以实现会话跟踪，它们之间最大的区别：**Cookie是存储在浏览器端而Session是存储在服务器端**

具体的学习思路为：

- Cookie的基本使用、原理、使用细节
- Session的基本使用、原理、使用细节
- Cookie和Session的综合案例

小结

在这节中，我们主要介绍了什么是会话和会话跟踪技术，需要注意的是：

- HTTP协议是无状态的，靠HTTP协议是无法实现会话跟踪
- 想要实现会话跟踪，就需要用到Cookie和Session

这个Cookie和Session具体该如何使用，接下来就先从Cookie来学起。

2, Cookie

学习Cookie，我们主要解决下面几个问题：

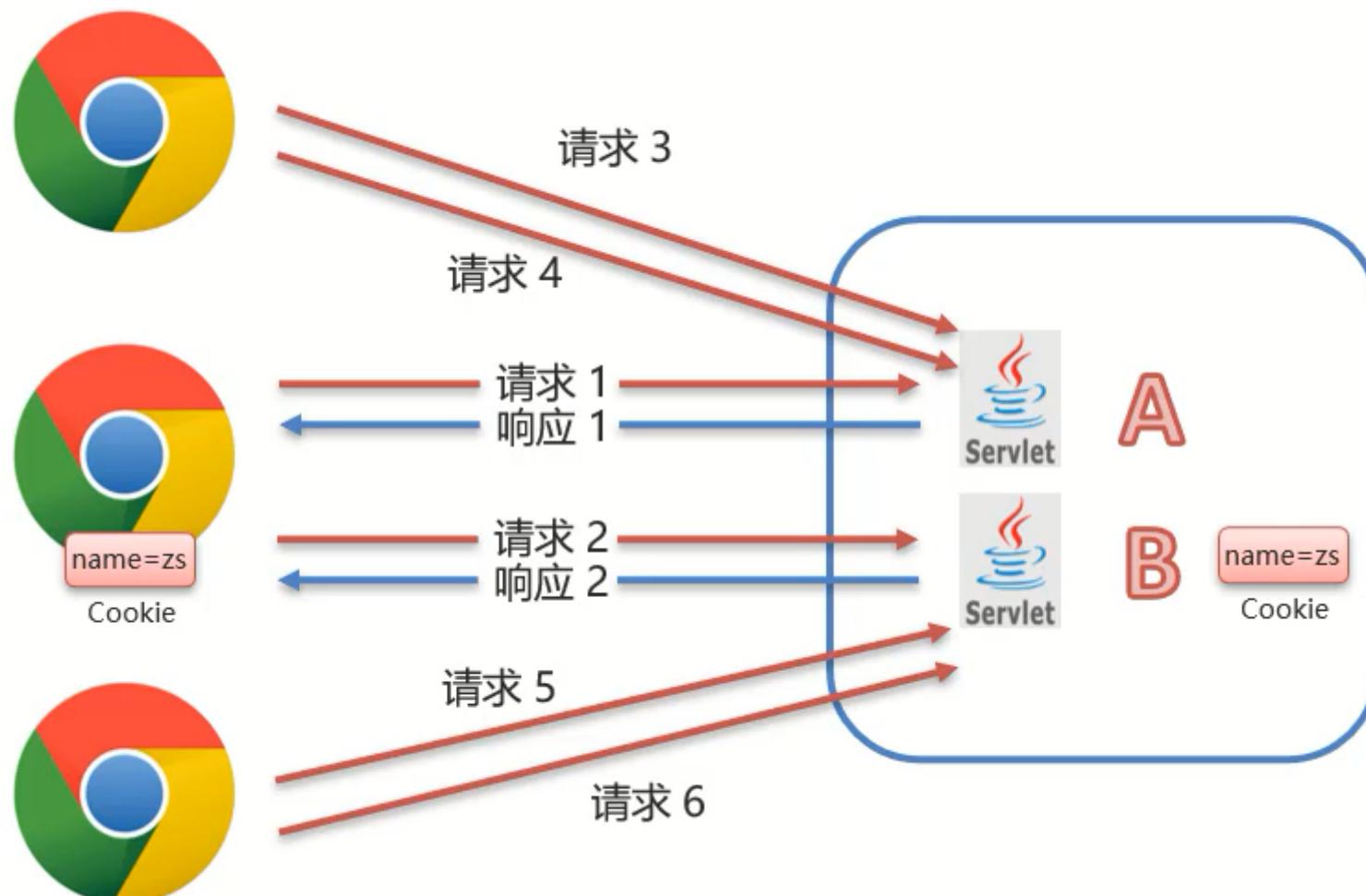
- 什么是Cookie？
- Cookie如何来使用？
- Cookie是如何实现的？
- Cookie的使用注意事项有哪些？

2.1 Cookie的基本使用

1. 概念

Cookie：客户端会话技术，将数据保存到客户端，以后每次请求都携带Cookie数据进行访问。

2. Cookie的工作流程



- 服务端提供了两个Servlet，分别是ServletA和ServletB
- 浏览器发送HTTP请求1给服务端，服务端ServletA接收请求并进行业务处理
- 服务端ServletA在处理的过程中可以创建一个Cookie对象并将name=zs的数据存入Cookie
- 服务端ServletA在响应数据的时候，会把Cookie对象响应给浏览器
- 浏览器接收到响应数据，会把Cookie对象中的数据存储在浏览器内存中，此时浏览器和服务端就建立了一次会话

- 在同一次会话中浏览器再次发送HTTP请求2给服务端ServletB，浏览器会携带Cookie对象中的所有数据
- ServletB接收到请求和数据后，就可以获取到存储在Cookie对象中的数据，这样同一个会话中的多次请求之间就实现了数据共享

3. Cookie的基本使用

对于Cookie的使用，我们更关注的应该是后台代码如何操作Cookie，对于Cookie的操作主要分两大类，分别是**发送Cookie**和**获取Cookie**，对于上面这两块内容，分别该如何实现呢？

3.1 发送Cookie

- 创建Cookie对象，并设置数据

```
1 cookie cookie = new cookie("key", "value");
```

- 发送Cookie到客户端：使用**response**对象

```
1 response.addCookie(cookie);
```

介绍完发送Cookie对应的步骤后，接下面通过一个案例来完成Cookie的发送，具体实现步骤为：

需求：在Servlet中生成Cookie对象并存入数据，然后将数据发送给浏览器

1. 创建Maven项目，项目名称为cookie-demo，并在pom.xml添加依赖
2. 编写Servlet类，名称为AServlet
3. 在AServlet中创建Cookie对象，存入数据，发送给前端
4. 启动测试，在浏览器查看Cookie对象中的值

(1) 创建Maven项目cookie-demo，并在pom.xml添加依赖

```
1 <properties>
2   <maven.compiler.source>8</maven.compiler.source>
3   <maven.compiler.target>8</maven.compiler.target>
4 </properties>
5
6 <dependencies>
7   <!--servlet-->
8   <dependency>
9     <groupId>javax.servlet</groupId>
10    <artifactId>javax.servlet-api</artifactId>
11    <version>3.1.0</version>
12    <scope>provided</scope>
13  </dependency>
14  <!--jsp-->
15  <dependency>
16    <groupId>javax.servlet.jsp</groupId>
```

```

17     <artifactId>jsp-api</artifactId>
18     <version>2.2</version>
19     <scope>provided</scope>
20   </dependency>
21   <!--jstl-->
22   <dependency>
23     <groupId>jstl</groupId>
24     <artifactId>jstl</artifactId>
25     <version>1.2</version>
26   </dependency>
27   <dependency>
28     <groupId>taglibs</groupId>
29     <artifactId>standard</artifactId>
30     <version>1.1.2</version>
31   </dependency>
32 </dependencies>
33 <build>
34   <plugins>
35     <plugin>
36       <groupId>org.apache.tomcat.maven</groupId>
37       <artifactId>tomcat7-maven-plugin</artifactId>
38       <version>2.2</version>
39     </plugin>
40   </plugins>
41 </build>

```

(2) 编写Servlet类，名称为AServlet

```

1 @WebServlet("/aservlet")
2 public class AServlet extends HttpServlet {
3     @Override
4     protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
5
6     }
7
8     @Override
9     protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
10         this.doGet(request, response);
11     }
12 }

```

(3) 在Servlet中创建Cookie对象，存入数据，发送给前端

```

1 @WebServlet("/aservlet")
2 public class AServlet extends HttpServlet {
3     @Override

```

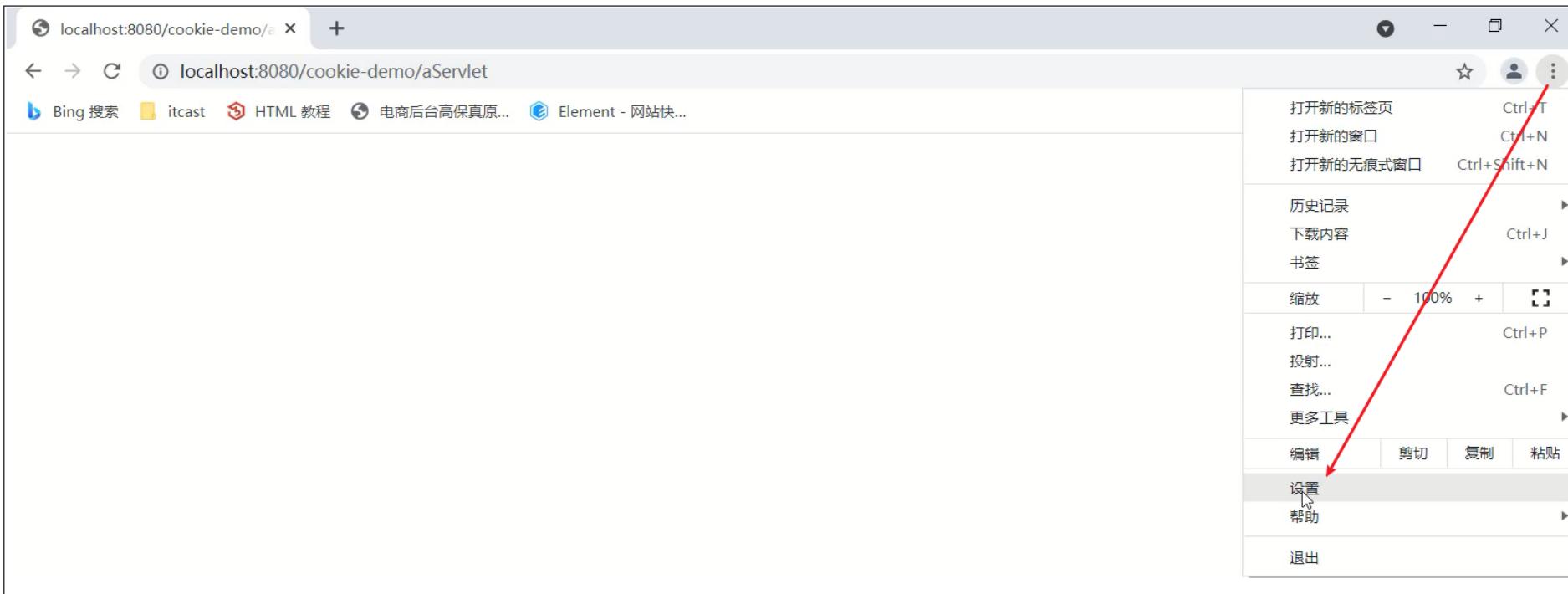
```
4  protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
5      //发送Cookie
6      //1. 创建Cookie对象
7      Cookie cookie = new Cookie("username", "zs");
8      //2. 发送Cookie, response
9      response.addCookie(cookie);
10 }
11
12 @Override
13 protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
14     this.doGet(request, response);
15 }
16 }
```

(4) 启动测试，在浏览器查看Cookie对象中的值

访问<http://localhost:8080/cookie-demo/aServlet>

chrome浏览器查看Cookie的值，有两种方式，分布式：

方式一：



The screenshot shows the 'Privacy and security' section of the Chrome settings. On the left, there's a sidebar with various settings categories. A red arrow points from the sidebar to the 'Cookie and other site data' section in the main content area. The main content area also includes sections for 'Clear browsing data', 'Security', 'Site settings', and 'Privacy sandbox'.

This screenshot shows the detailed settings for 'Cookie and other site data'. It includes four toggle switches: 'Clear cookies and site data when I close all windows', 'Send a "Do Not Track" request with every browsing flow', 'Preload pages to make browsing faster', and 'View all cookies and site data'. The sidebar on the left remains the same as the previous screenshot.

The screenshot displays the 'All cookies and site data' page. It features a search bar at the top right with the text 'loca'. Below the search bar is a button labeled 'Remove all displayed cookies'. A list of cookies is shown, with one entry for 'localhost' which has '5 cookies' listed under it. The sidebar on the left is identical to the previous screenshots.

This is a screenshot of the 'Cookie detail' page, showing a single cookie entry for 'localhost'. The cookie details are not fully visible in the screenshot.

The screenshot shows the 'Settings' page in Google Chrome. On the left, there's a sidebar with various options like '您与 Google', '自动填充', '安全检查', etc. The main area is titled 'localhost 本地存储的数据' and lists several items:

名称	内容
Idea-5fd74a63	
name	
o2State	
username	zs

At the top right, there's a '全部删除' (Delete All) button.

方式二：选中打开开发者工具或者 使用快捷键F12 或者 Ctrl+Shift+I

The screenshot shows a browser window with the URL 'localhost:8080/cookie-demo/aServlet'. A context menu is open on the right side, with the 'Developer Tools' option highlighted. The menu also includes other options like '打开新的标签页' (Open New Tab), '历史记录' (History), and '更多工具' (More Tools).

The screenshot shows the 'Application' tab in the Chrome DevTools. On the left, there's a sidebar with sections for 'Application', 'Storage', and 'Cookies'. The 'Cookies' section is expanded, showing a table of stored cookies:

Name	Value	Domain	Path	Exp...	Size	Htt
Idea-83cb...	4bbf347c-5f03-4205-9f0...	localhost	/	203...	49	
username	zs	localhost	/cookie-...	Ses...	10	

The 'username' cookie is selected and highlighted with a red border.

3.2 获取Cookie

- 获取客户端携带的所有Cookie，使用request对象

```
1 Cookie[] cookies = request.getCookies();
```

- 遍历数组，获取每一个Cookie对象：for
- 使用Cookie对象方法获取数据

```
1 cookie.getName();
2 cookie.getValue();
```

介绍完获取Cookie对应的步骤后，接下面再通过一个案例来完成Cookie的获取，具体实现步骤为：

需求：在Servlet中获取前一个案例存入在Cookie对象中的数据

1. 编写一个新Servlet类，名称为BServlet
2. 在BServlet中使用request对象获取Cookie数组，遍历数组，从数据中获取指定名称对应的值
3. 启动测试，在控制台打印出获取的值

(1) 编写一个新Servlet类，名称为BServlet

```
1 @WebServlet("/bServlet")
2 public class BServlet extends HttpServlet {
3     @Override
4     protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
5
6     }
7
8     @Override
9     protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
10         this.doGet(request, response);
11     }
12 }
```

(2) 在BServlet中使用request对象获取Cookie数组，遍历数组，从数据中获取指定名称对应的值

```
1 @WebServlet("/bServlet")
2 public class BServlet extends HttpServlet {
3     @Override
4     protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
5         //获取Cookie
```

```

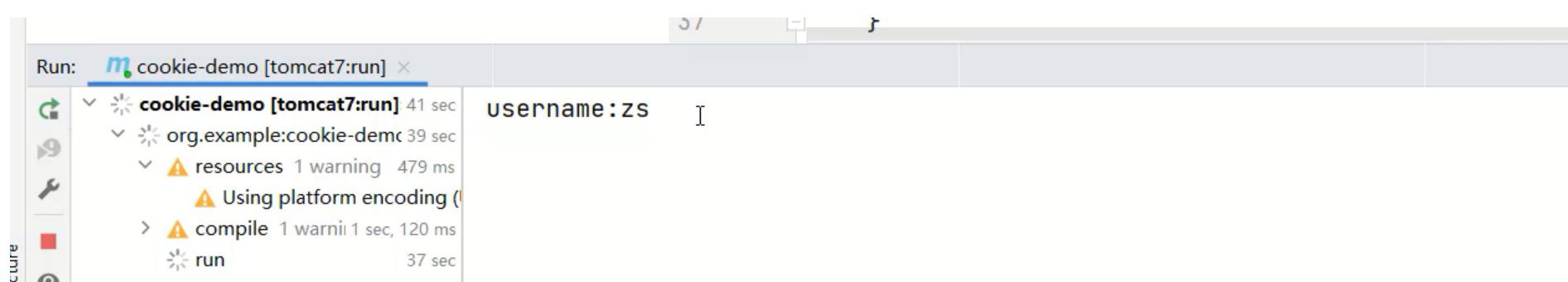
6     //1. 获取Cookie数组
7     Cookie[] cookies = request.getCookies();
8     //2. 遍历数组
9     for (Cookie cookie : cookies) {
10         //3. 获取数据
11         String name = cookie.getName();
12         if("username".equals(name)){
13             String value = cookie.getValue();
14             System.out.println(name+":"+value);
15             break;
16         }
17     }
18 }
19
20
21 @Override
22 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
23     this doGet(request, response);
24 }
25 }
```

(3) 启动测试，在控制台打印出获取的值

访问<http://localhost:8080/cookie-demo/bServlet>



在IDEA控制台就能看到输出的结果：



思考：测试的时候

- 在访问AServlet和BServlet的中间把关闭浏览器，重启浏览器后访问BServlet能否获取到Cookie中的数据？

这个问题，我们会在Cookie的使用细节中讲，大家可以动手先试下。

小结

在这节中，我们主要讲解了Cookie的基本使用，包含两部分内容

- **发送Cookie：**

- 创建Cookie对象，并设置值：Cookie cookie = new Cookie("key", "value");
- 发送Cookie到客户端使用的是Response对象：response.addCookie(cookie);

- **获取Cookie：**

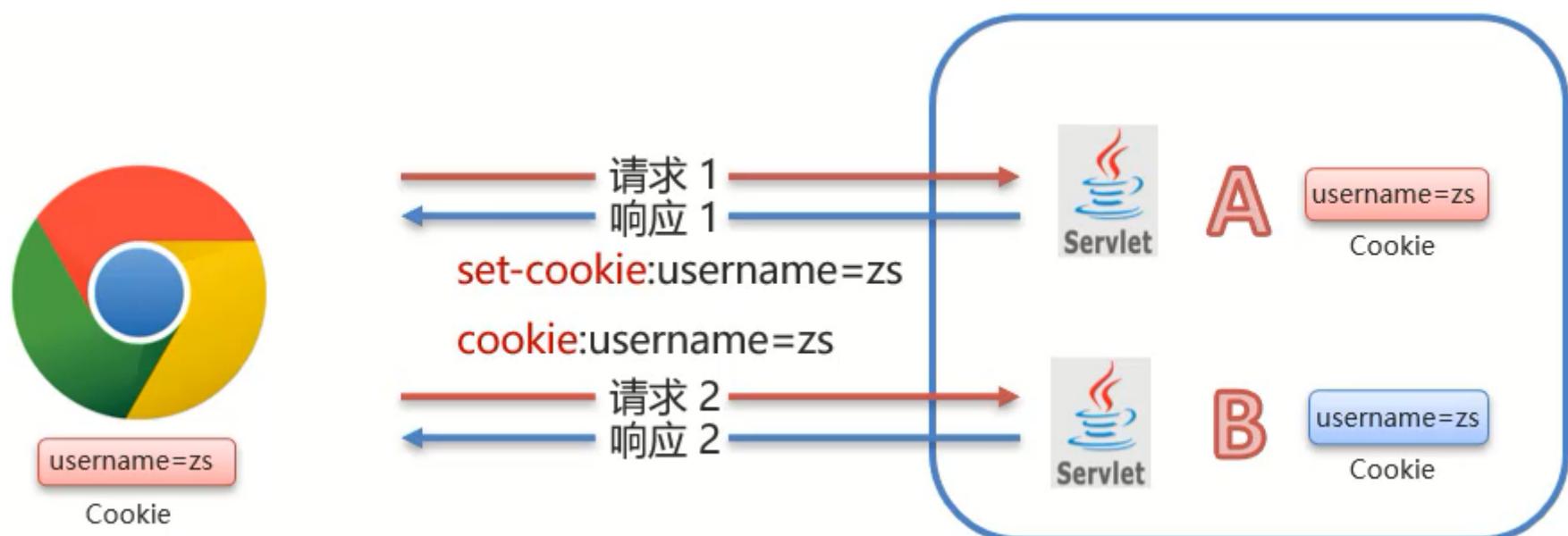
- 使用Request对象获取Cookie数组：Cookie[] cookies = request.getCookies();
- 遍历数组
- 获取数组中每个Cookie对象的值：cookie.getName() 和 cookie.getValue()

介绍完Cookie的基本使用之后，那么Cookie的底层到底是如何实现一次会话两次请求之间的数据共享呢？

2.2 Cookie的原理分析

对于Cookie的实现原理是基于HTTP协议的，其中设计到HTTP协议中的两个请求头信息：

- **响应头：**set-cookie
- **请求头：** cookie



- 前面的案例中已经能够实现，AServlet给前端发送Cookie，BServlet从request中获取Cookie的功能
- 对于AServlet响应数据的时候，Tomcat服务器都是基于HTTP协议来响应数据
- 当Tomcat发现后端要返回的是一个Cookie对象之后，Tomcat就会在响应头中添加一行数据 **Set-Cookie:username=zs**
- 浏览器获取到响应结果后，从响应头中就可以获取到set-Cookie对应值username=zs，并将数据存储在浏览器的内存中
- 浏览器再次发送请求给BServlet的时候，浏览器会自动在请求头中添加**Cookie: username=zs**发送给服务端BServlet
- Request对象会把请求头中cookie对应的值封装成一个个Cookie对象，最终形成一个数组
- BServlet通过Request对象获取到Cookie[]后，就可以从中获取自己需要的数据

接下来，使用刚才的案例，把上述结论验证下：

(1) 访问AServlet对应的地址 <http://localhost:8080/cookie-demo/aServlet>

使用Chrom浏览器打开开发者工具 (F12或Crtl+Shift+I) 进行查看**响应头**中的数据

The screenshot shows the Chrome DevTools Network tab with a single request listed: "aServlet". The "Response Headers" section is expanded, showing the following details:

- Request URL: <http://localhost:8080/cookie-demo/aServlet>
- Request Method: GET
- Status Code: 200 OK
- Remote Address: [::1]:8080
- Referrer Policy: strict-origin-when-cross-origin
- Response Headers** (highlighted with a red border):
 - Content-Length: 0
 - Date: Tue, 17 Aug 2021 08:04:57 GMT
 - Server: Apache-Coyote/1.1
 - Set-Cookie: username=zs** (highlighted with a red border)
- Request Headers

At the bottom of the Network tab, it shows: 2 requests | 226 R transferred | 0 R resol

(2) 访问BServlet对应的地址 <http://localhost:8080/cookie-demo/bServlet>

使用Chrom浏览器打开开发者工具 (F12或Crtl+Shift+I) 进行查看**请求头**中的数据

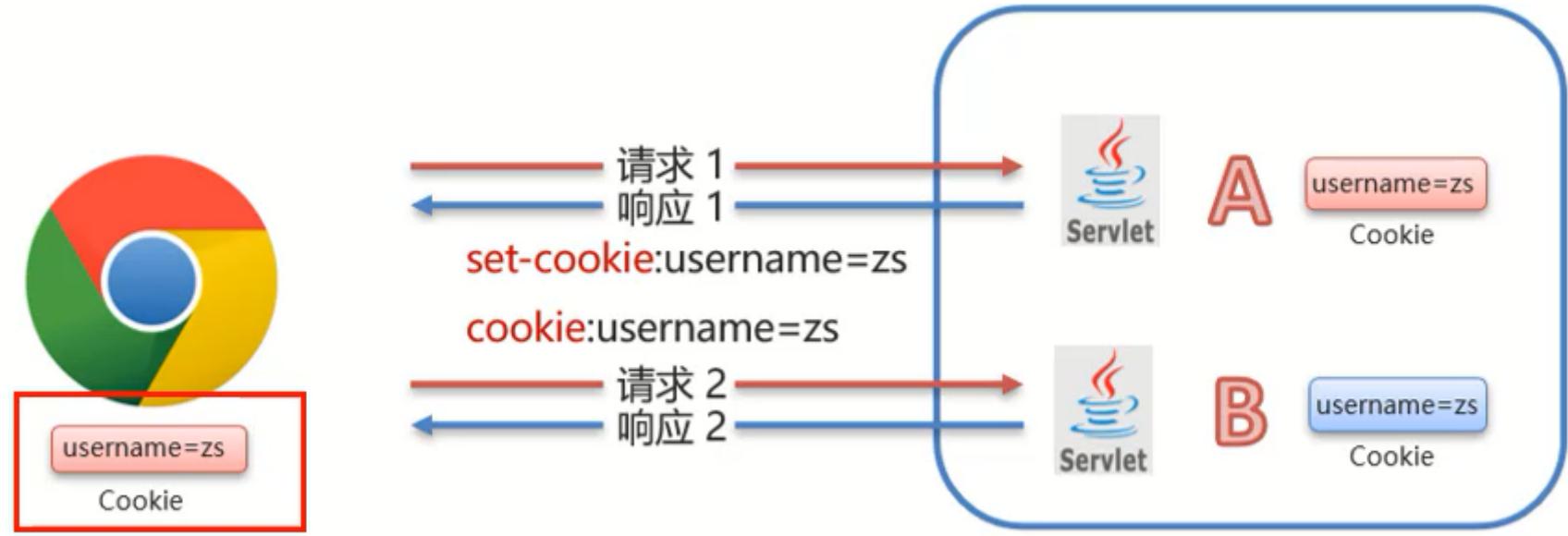
The screenshot shows a browser window with the URL `localhost:8080/cookie-demo/bServlet`. Below the browser is the Chrome DevTools Network tab. A request to `bServlet` is selected. In the Headers section, the `Cookie` header is highlighted with a red border. The value of the `Cookie` header is `name=zs; username=zs; Idea-5fd74a63=115284b8-5c3c-475a-81aa-4ba6ISVFJSLNK273Y6RDNPZIQP5VBA`.

2.3 Cookie的使用细节

在这节我们主要讲解两个知识，第一个是Cookie的存活时间，第二个是Cookie如何存储中文，首先来学习下Cookie的存活时间。

2.3.1 Cookie的存活时间

前面让大家思考过一个问题：



- (1) 浏览器发送请求给AServlet, AServlet会响应一个存有username=zs的Cookie对象给浏览器
- (2) 浏览器接收到响应数据将cookie存入到浏览器内存中
- (3) 当浏览器再次发送请求给BServlet, BServle就可使用Request对象获取到Cookie数据
- (4) 在发送请求到BServlet之前, 如果把浏览器关闭再打开进行访问, BServle能否获取到Cookie数据?

注意：浏览器关闭再打开不是指打开一个新的显卡，而且必须是先关闭再打开，顺序不能变。

针对上面这个问题，通过演示，会发现，BServle中无法再获取到Cookie数据，这是为什么呢？

- 默认情况下，Cookie存储在浏览器内存中，当浏览器关闭，内存释放，则Cookie被销毁

这个结论就印证了上面的演示效果，但是如果使用这种默认情况下的Cookie，有些需求就无法实现，比如：



上面这个网站的登录页面上有一个记住我的功能，这个功能大家都比较熟悉

- 第一次输入用户名和密码并勾选记住我然后进行登录
- 下次再登陆的时候，用户名和密码就会被自动填充，不需要再重新输入登录
- 比如记住我这个功能需要记住用户名和密码一个星期，那么使用默认情况下的Cookie就会出现问题
- 因为默认情况，浏览器一关，Cookie就会从浏览器内存中删除，对于记住我功能就无法实现

所以我们现在就遇到一个难题是如何将Cookie持久化存储？

Cookie其实已经为我们提供好了对应的API来完成这件事，这个API就是**setMaxAge**，

- 设置Cookie存活时间

```
1 setMaxAge(int seconds)
```

参数值为：

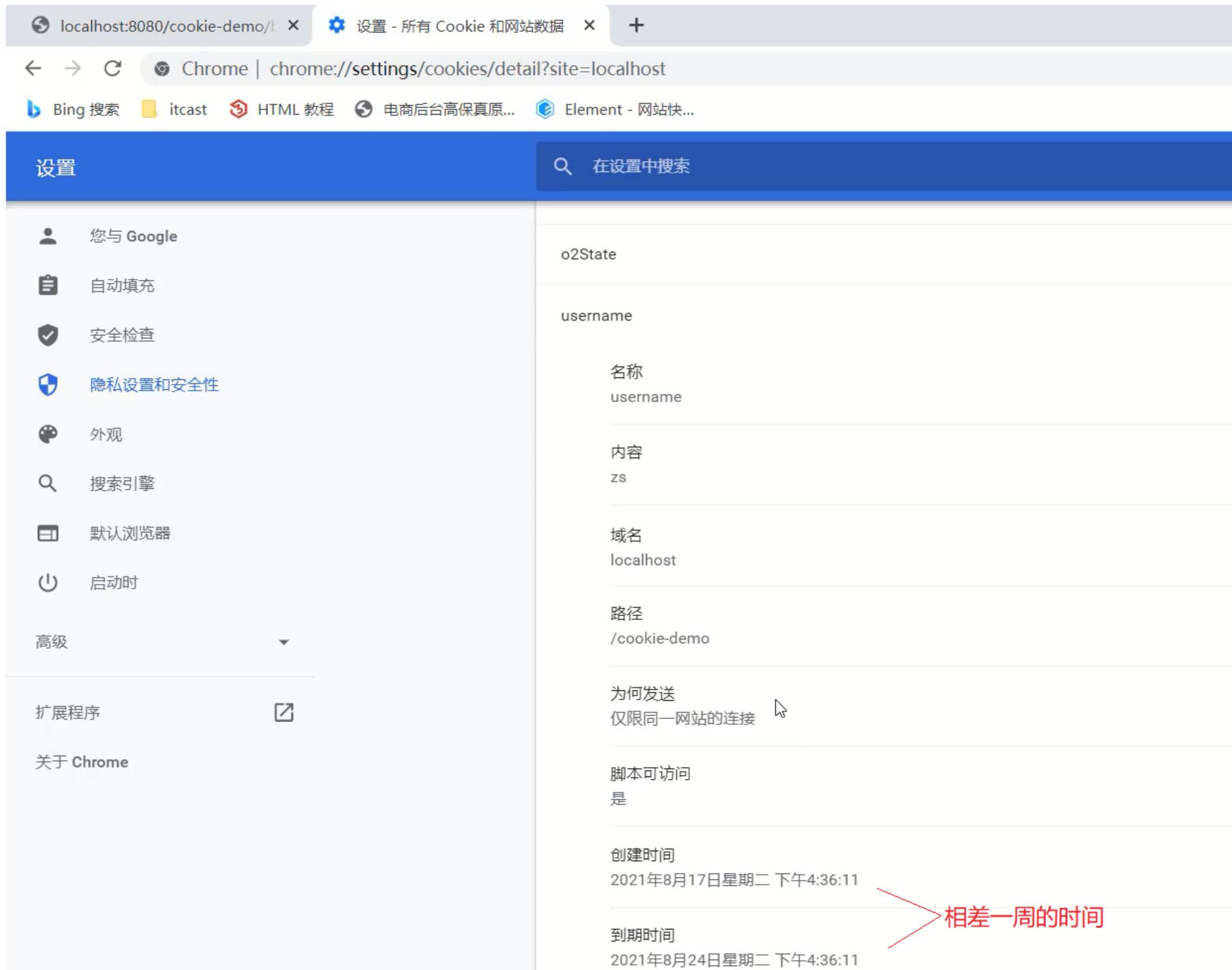
1. 正数：将Cookie写入浏览器所在电脑的硬盘，持久化存储。到时间自动删除
2. 负数：默认值，Cookie在当前浏览器内存中，当浏览器关闭，则Cookie被销毁
3. 零：删除对应Cookie

接下来，咱们就在AServlet中去设置Cookie的存活时间。

```
1 @WebServlet("/aServlet")
2 public class AServlet extends HttpServlet {
3     @Override
4     protected void doGet(HttpServletRequest request, HttpServletResponse
5             response) throws ServletException, IOException {
6         //发送Cookie
7         //1. 创建Cookie对象
8         Cookie cookie = new Cookie("username", "zs");
9         //设置存活时间    , 1周 7天
10        cookie.setMaxAge(60*60*24*7); //易阅读, 需程序计算
11        //cookie.setMaxAge(604800); //不易阅读(可以使用注解弥补), 程序少进行一次计算
12        //2. 发送Cookie, response
13        response.addCookie(cookie);
14    }
15
16    @Override
17    protected void doPost(HttpServletRequest request, HttpServletResponse
18            response) throws ServletException, IOException {
19        this.doGet(request, response);
20    }
21 }
```

修改完代码后，启动测试，访问<http://localhost:8080/cookie-demo/aServlet>

- 访问一个AServlet后，把浏览器关闭重启后，再去访问<http://localhost:8080/cookie-demo/bServlet>，能在控制台打印出username:zs，说明Cookie没有随着浏览器关闭而被销毁
- 通过浏览器查看Cookie的内容，会发现Cookie的相关信息



2.3.2 Cookie存储中文

首先，先来演示一个效果，将之前 `username=zs` 的值改成 `username=张三`，把汉字张三存入到 Cookie 中，看是什么效果：

```
1 @WebServlet("/aServlet")
2 public class AServlet extends HttpServlet {
3     @Override
4     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
5         //发送Cookie
6         String value = "张三";
7         Cookie cookie = new Cookie("username", value);
8         //设置存活时间    , 1周 7天
9         cookie.setMaxAge(60*60*24*7);
10        //2. 发送Cookie, response
11        response.addCookie(cookie);
12    }
13
14    @Override
15    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
```

```
16     this.doGet(request, response);
17 }
18 }
```

启动访问测试，访问`http://localhost:8080/cookie-demo/aServlet`会发现浏览器会提示错误信息

Apache Tomcat/7.0.47 - Error +
localhost:8080/cookie-demo/aServlet
Bing 搜索 itcast HTML 教程 电商后台高保真原... Element - 网站快...

HTTP Status 500 - Control character in cookie value or attribute.

type Exception report

message Control character in cookie value or attribute.

description The server encountered an internal error that prevented it from fulfilling this request.

exception

```
java.lang.IllegalArgumentException: Control character in cookie value or attribute.
    org.apache.tomcat.util.http.CookieSupport.isHttpSeparator(CookieSupport.java:193)
    org.apache.tomcat.util.http.CookieSupport.isHttpToken(CookieSupport.java:217)
    org.apache.tomcat.util.http.ServerCookie.appendCookieValue(ServerCookie.java:186)
    org.apache.catalina.connector.Response.generateCookieString(Response.java:1032)
    org.apache.catalina.connector.Response.addCookie(Response.java:974)
    org.apache.catalina.connector.ResponseFacade.addCookie(ResponseFacade.java:381)
    com.itheima.web.AServlet doGet(AServlet.java:22)
    javax.servlet.http.HttpServlet.service(HttpServlet.java:621)
    javax.servlet.http.HttpServlet.service(HttpServlet.java:728)
    org.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter.java:51)
```

note The full stack trace of the root cause is available in the Apache Tomcat/7.0.47 logs.

Apache Tomcat/7.0.47

通过上面的案例演示，我们得到一个结论：

- Cookie不能直接存储中文

Cookie不能存储中文，但是如果有这方面的需求，这个时候该如何解决呢？

这个时候，我们可以使用之前学过的一个知识点叫URL编码，所以如果需要存储中文，就需要进行转码，具体的实现思路为：

1. 在AServlet中对中文进行URL编码，采用URLEncoder.encode()，将编码后的值存入Cookie中
2. 在BServlet中获取Cookie中的值，获取的值为URL编码后的值
3. 将获取的值在进行URL解码，采用URLDecoder.decode()，就可以获取到对应的中文值

(1) 在AServlet中对中文进行URL编码

```
1 @WebServlet("/aServlet")
2 public class AServlet extends HttpServlet {
3     @Override
```

```

4  protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
5      //发送cookie
6      String value = "张三";
7      //对中文进行URL编码
8      value = URLEncoder.encode(value, "UTF-8");
9      System.out.println("存储数据: "+value);
10     //将编码后的值存入cookie中
11     Cookie cookie = new Cookie("username",value);
12     //设置存活时间 , 1周 7天
13     cookie.setMaxAge(60*60*24*7);
14     //2. 发送Cookie, response
15     response.addCookie(cookie);
16 }
17
18 @Override
19 protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
20     this.doGet(request, response);
21 }
22 }
```

(2) 在BServlet中获取值，并对值进行解码

```

1 @WebServlet("/bServlet")
2 public class BServlet extends HttpServlet {
3     @Override
4     protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
5         //获取Cookie
6         //1. 获取Cookie数组
7         Cookie[] cookies = request.getCookies();
8         //2. 遍历数组
9         for (Cookie cookie : cookies) {
10             //3. 获取数据
11             String name = cookie.getName();
12             if("username".equals(name)){
13                 String value = cookie.getValue(); //获取的是URL编码后的值
14                 //URL解码
15                 value = URLDecoder.decode(value, "UTF-8");
16                 System.out.println(name+":"+value); //value解码后为 张三
17                 break;
18             }
19         }
20     }
21 }
22 }
```

```

23     @Override
24     protected void doPost(HttpServletRequest request, HttpServletResponse
25             response) throws ServletException, IOException {
26         this.doGet(request, response);
27     }

```

至此，我们就可以将中文存入Cookie中进行使用。

小结

Cookie的使用细节中，我们讲了Cookie的存活时间和存储中文：

- 存活时间，需要掌握setMaxAge() API的使用
- 存储中文，需要掌握URL编码和解码的使用

3, Session

Cookie已经能完成一次会话多次请求之间的数据共享，之前我们还提到过Session也可以实现，那么：

- 什么是Session？
- Session如何来使用？
- Session是如何实现的？
- Session的使用注意事项有哪些？

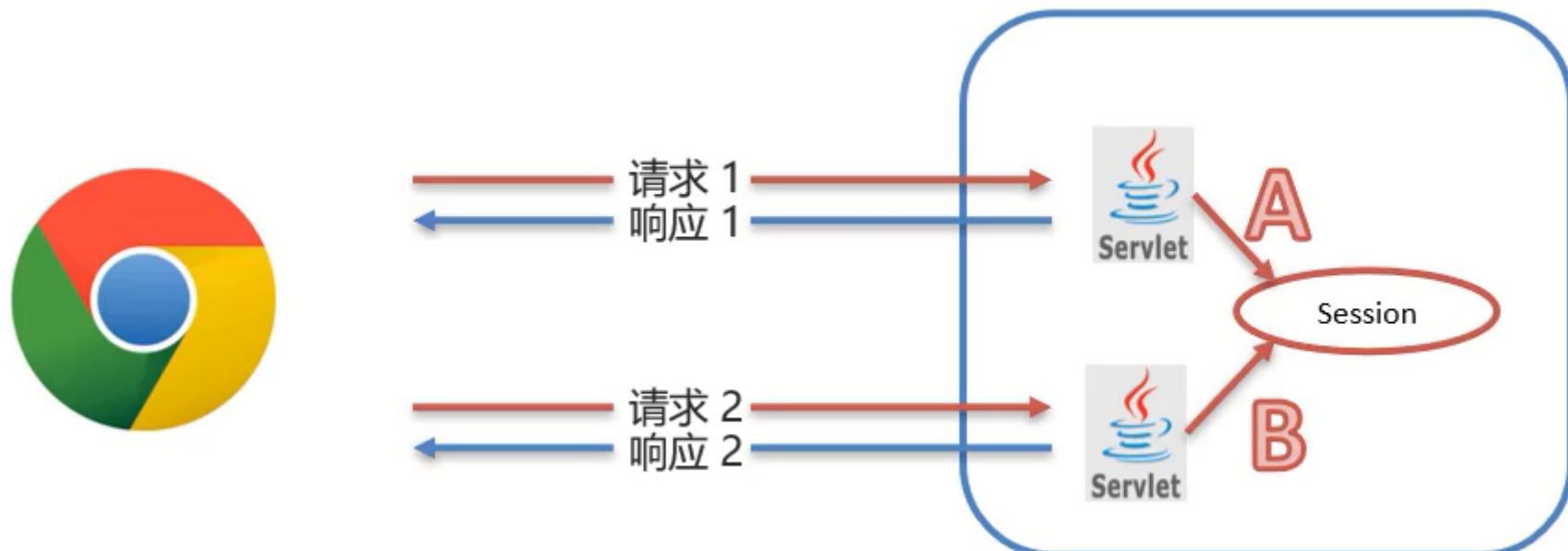
3.1 Session的基本使用

1. 概念

Session: 服务端会话跟踪技术：将数据保存到服务端。

- Session是存储在服务端而Cookie是存储在客户端
- 存储在客户端的数据容易被窃取和截获，存在很多不安全的因素
- 存储在服务端的数据相比于客户端来说就更安全

2. Session的工作流程



- 在服务端的AServlet获取一个Session对象，把数据存入其中

- 在服务端的BServlet获取到相同的Session对象，从中取出数据
- 就可以实现一次会话中多次请求之间的数据共享了
- 现在最大的问题是如何保证AServlet和BServlet使用的是同一个Session对象（在原理分析会讲解）？

3. Session的基本使用

在JavaEE中提供了HttpSession接口，来实现一次会话的多次请求之间数据共享功能。

具体的使用步骤为：

- 获取Session对象，使用的是request对象

```
1 HttpSession session = request.getSession();
```

- Session对象提供的功能：

- 存储数据到 session 域中

```
1 void setAttribute(String name, Object o)
```

- 根据 key，获取值

```
1 Object getAttribute(String name)
```

- 根据 key，删除该键值对

```
1 void removeAttribute(String name)
```

介绍完Session相关的API后，接下来通过一个案例来完成对Session的使用，具体实现步骤为：

需求：在一个Servlet中往Session中存入数据，在另一个Servlet中获取Session中存入的数据

1. 创建名为SessionDemo1的Servlet类
2. 创建名为SessionDemo2的Servlet类
3. 在SessionDemo1的方法中：获取Session对象、存储数据
4. 在SessionDemo2的方法中：获取Session对象、获取数据
5. 启动测试

(1) 创建名为SessionDemo1的Servlet类

```
1 @WebServlet("/demo1")
2 public class SessionDemo1 extends HttpServlet {
3     @Override
4         protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
5
6     }
7
8     @Override
9         protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
10             this.doGet(request, response);
11         }
12 }
```

(2) 创建名为SessionDemo2的Servlet类

```
1 @WebServlet("/demo2")
2 public class SessionDemo2 extends HttpServlet {
3     @Override
4         protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
5
6     }
7
8     @Override
9         protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
10             this.doGet(request, response);
11         }
12 }
```

(3) SessionDemo1: 获取Session对象、存储数据

```
1 @WebServlet("/demo1")
2 public class SessionDemo1 extends HttpServlet {
3     @Override
4         protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
5             //存储到Session中
6             //1. 获取Session对象
7             HttpSession session = request.getSession();
8             //2. 存储数据
9             session.setAttribute("username", "zs");
10        }
11
12     @Override
```

```

13     protected void doPost(HttpServletRequest request, HttpServletResponse
14         response) throws ServletException, IOException {
15         this.doGet(request, response);
16     }

```

(4) SessionDemo2: 获取Session对象、获取数据

```

1 @WebServlet("/demo2")
2 public class SessionDemo2 extends HttpServlet {
3     @Override
4     protected void doGet(HttpServletRequest request, HttpServletResponse
5         response) throws ServletException, IOException {
6         //获取数据, 从session中
7         //1. 获取Session对象
8         HttpSession session = request.getSession();
9         //2. 获取数据
10        Object username = session.getAttribute("username");
11        System.out.println(username);
12    }
13
14    @Override
15    protected void doPost(HttpServletRequest request, HttpServletResponse
16        response) throws ServletException, IOException {
17        this.doGet(request, response);
18    }
19
20 }

```

(5) 启动测试,

- 先访问 `http://localhost:8080/cookie-demo/demo1`, 将数据存入Session
- 在访问 `http://localhost:8080/cookie-demo/demo2`, 从Session中获取数据
- 查看控制台

```

m cookie-demo [tomcat7:run] ×
cookie-demo [tomcat7:run] 17 sec
  org.example:cookie-demo 15 sec
    resources 1 warning 613 ms
      Using platform encoding (.)
    compile 1 warning 1 sec, 142 ms
    run 13 sec
信息: TLD skipped. URI: http://java.sun.com/jstl/xml
八月 17, 2021 7:19:27 下午 org.apache.catalina.startu
信息: TLD skipped. URI: http://java.sun.com/jstl/xml
八月 17, 2021 7:19:27 下午 org.apache.catalina.startu
信息: TLD skipped. URI: http://java.sun.com/jsp/jstl
八月 17, 2021 7:19:27 下午 org.apache.coyote.Abstract
信息: Starting ProtocolHandler ["http-bio-8080"]
八月 17, 2021 7:19:29 下午 org.apache.tomcat.util.htt
信息: Cookies: Invalid cookie. Value not a token or
Note: further occurrences of Cookie errors will be

```

通过案例的效果，能看到Session是能够在一次会话中两次请求之间共享数据。

小结

至此Session的基本使用就已经完成了，重点要掌握的是：

- Session的获取

```
1 HttpSession session = request.getSession();
```

- Session常用方法的使用

```
1 void setAttribute(String name, Object o)  
2 Object getAttribute(String name)
```

注意：Session中可以存储的是一个Object类型的数据，也就是说Session中可以存储任意数据类型。

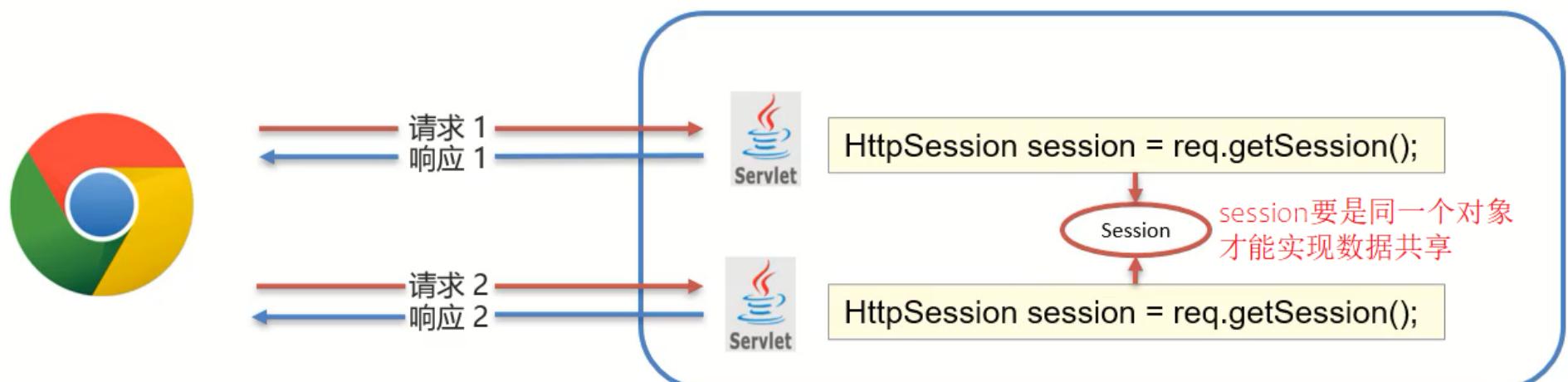
介绍完Session的基本使用之后，那么Session的底层到底是如何实现一次会话两次请求之间的数据共享呢？

3.2 Session的原理分析

- Session是基于Cookie实现的

这句话其实不太能详细的说明Session的底层实现，接下来，咱们一步步来分析下Session的具体实现原理：

(1) 前提条件



Session要想实现一次会话多次请求之间的数据共享，就必须要保证多次请求获取Session的对象是同一个。

那么它们是一个对象么？要验证这个结论也很简单，只需要在上面案例中的两个Servlet中分别打印下Session对象

SessionDemo1

```
1 @WebServlet("/demo1")  
2 public class SessionDemo1 extends HttpServlet {  
3     @Override
```

```
4     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
5         //存储到Session中
6         //1. 获取Session对象
7         HttpSession session = request.getSession();
8         System.out.println(session);
9         //2. 存储数据
10        session.setAttribute("username", "zs");
11    }
12
13    @Override
14    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
15        this.doGet(request, response);
16    }
17 }
```

SessionDemo2

```
1 @WebServlet("/demo2")
2 public class SessionDemo2 extends HttpServlet {
3     @Override
4     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
5         //获取数据，从Session中
6         //1. 获取Session对象
7         HttpSession session = request.getSession();
8         System.out.println(session);
9         //2. 获取数据
10        Object username = session.getAttribute("username");
11        System.out.println(username);
12    }
13
14    @Override
15    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
16        this.doGet(request, response);
17    }
18 }
```

启动测试，分别访问

<http://localhost:8080/cookie-demo/demo1>

<http://localhost:8080/cookie-demo/demo2>

```

Run: cookie-demo [tomcat7:run] ×
cookie-demo [tomcat7:run] 26 sec
org.example:cookie-demo 24 sec
  resources 1 warning 487 ms
    Using platform encoding (UTF-8)
  compile 1 warning 1 sec, 190 ms
    run 23 sec

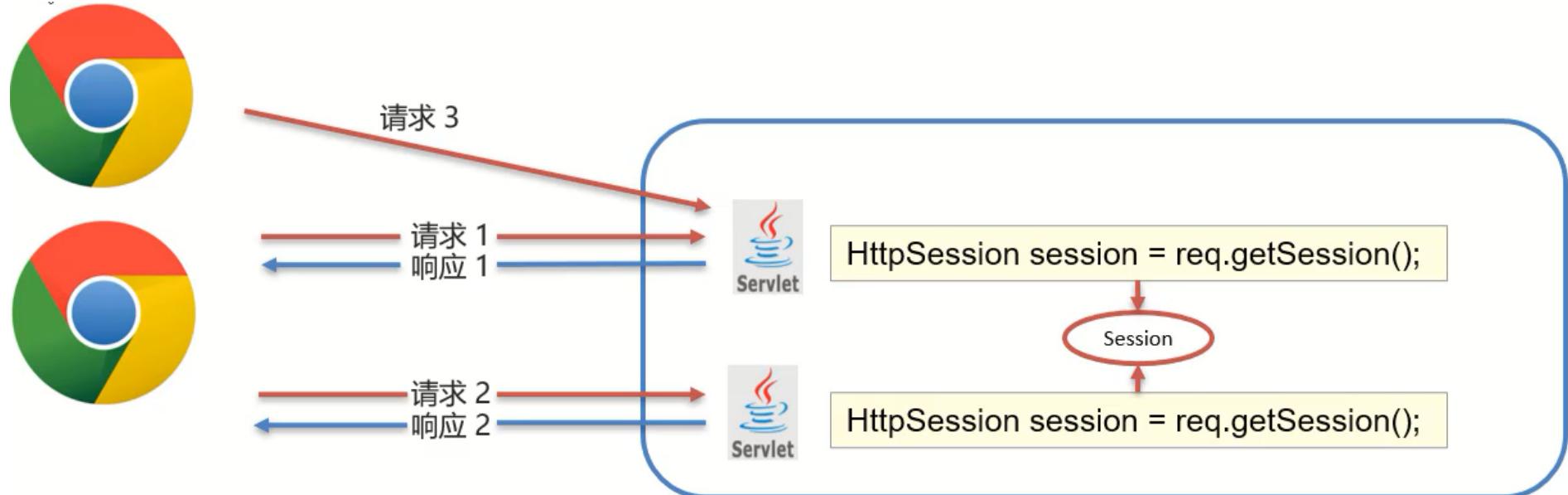
八月 17, 2021 7:46:26 下午 org.apache.catalina.startup.TaglibUriRule body
信息: TLD skipped. URI: http://java.sun.com/jstl/xml is already defined
八月 17, 2021 7:46:26 下午 org.apache.catalina.startup.TaglibUriRule body
信息: TLD skipped. URI: http://java.sun.com/jsp/jstl/xml is already defined
八月 17, 2021 7:46:26 下午 org.apache.coyote.AbstractProtocol start
信息: Starting ProtocolHandler ["http-bio-8080"]
八月 17, 2021 7:46:31 下午 org.apache.tomcat.util.http.Cookies processCookie
信息: Cookies: Invalid cookie. Value not a token or quoted value
Note: further occurrences of Cookie errors will be logged at DEBUG level
org.apache.catalina.session.StandardSessionFacade@4a3cf497
org.apache.catalina.session.StandardSessionFacade@4a3cf497

```

通过打印可以得到如下结论：

- 两个Servlet类中获取的Session对象是同一个
- 把demo1和demo2请求刷新多次，控制台最终打印的结果都是同一个

那么问题又来了，如果新开一个浏览器，访问demo1或者demo2，打印在控制台的Session还是同一个对象么？



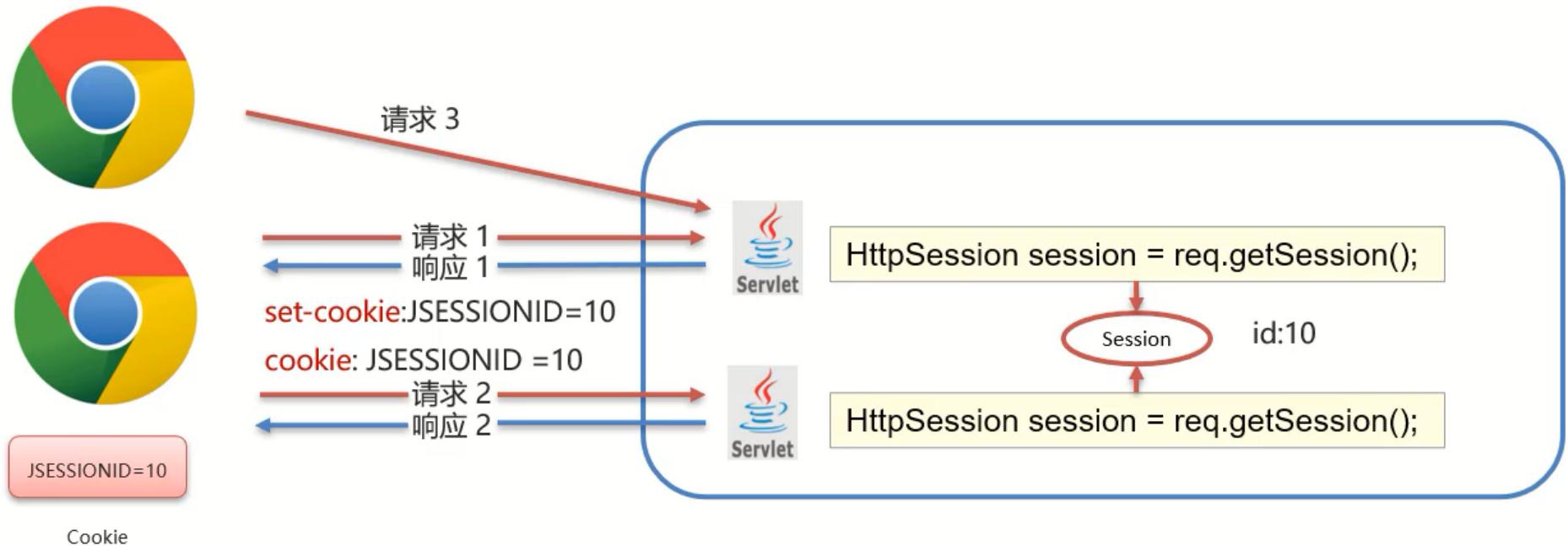
注意：在一台电脑上演示的时候，如果是相同的浏览器必须要把浏览器全部关掉重新打开，才算新开的一个浏览器。

当然也可以使用不同的浏览器进行测试，就不需要把之前的浏览器全部关闭。

测试的结果：如果是不同浏览器或者重新打开浏览器后，打印的Session就不一样了。

所以Session实现的也是一次会话中的多次请求之间的数据共享。

那么最主要的问题就来了，Session是如何保证在一次会话中获取的Session对象是同一个呢？



- (1) demo1在第一次获取session对象的时候，session对象会有一个唯一的标识，假如是 id:10
- (2) demo1在session中存入其他数据并处理完成所有业务后，需要通过Tomcat服务器响应结果给浏览器
- (3) Tomcat服务器发现业务处理中使用了session对象，就会把session的唯一标识 id:10当做一个cookie，添加 set-Cookie:JSESSIONID=10 到响应头中，并响应给浏览器
- (4) 浏览器接收到响应结果后，会把响应头中的 cookie 数据存储到浏览器的内存中
- (5) 浏览器在同一会话中访问demo2的时候，会把 cookie 中的数据按照 cookie: JSESSIONID=10 的格式添加到请求头中并发送给服务器Tomcat
- (6) demo2获取到请求后，从请求头中就读取cookie中的JSESSIONID值为10，然后就会到服务器内存中寻找 id:10 的session对象，如果找到了，就直接返回该对象，如果没有则新创建一个session对象
- (7) 关闭打开浏览器后，因为浏览器的cookie已被销毁，所以就没有 JSESSIONID 的数据，服务端获取到的 session 就是一个全新的 session 对象

至此， session 是基于 cookie 来实现的。这说明，我们就解释完了，接下来通过实例来演示下：

- (1) 使用 chrome 浏览器访问 <http://localhost:8080/cookie-demo/demo1>，打开开发者模式 (F12 或 Ctrl+Shift+I)，查看 **响应头 (Response Headers)** 数据：

The screenshot shows the Chrome DevTools Network tab. The URL in the address bar is `localhost:8080/cookie-demo/demo1`. The Network tab is selected. A request for `demo1` is listed under the Name column. The Headers section is expanded, showing the following details:

- Request URL: `http://localhost:8080/cookie-demo/demo1`
- Request Method: GET
- Status Code: 200 OK
- Remote Address: `[::1]:8080`
- Referrer Policy: strict-origin-when-cross-origin

The Response Headers section is also expanded, showing:

- Content-Length: 0
- Date: Tue, 17 Aug 2021 11:51:25 GMT
- Server: Apache-Coyote/1.1
- Set-Cookie: JSESSIONID=5B8C524F0BD885BF277CD281724E7DC5; Path=/cookie-demo/; HttpOnly**

(2) 使用chrome浏览器再次访问`http://localhost:8080/cookie-demo/demo2`, 查看**请求头(Request Headers)**数据:

The screenshot shows the Network tab in the Chrome DevTools. A request for 'demo2' is selected. In the 'Cookies' section, the 'JSESSIONID' cookie is highlighted with a red box. The cookie value is: JSESSIONID=5B8C524F0BD885BF277CD281724E7DC5; name=zs; username=%E5%BC%A0%E4%B8%89; Idea-5fd74a63=115284b8-5KHRTVDFJRKXWU67NR04ZY6HVYHPJI632VBOYUKKZH2IQOYJROGXPTQVGULRISVFJSLNK273Y6RDNPZIQP5VBA

小结

介绍完Session的原理，我们只需要记住

- Session是基于Cookie来实现的

3.3 Session的使用细节

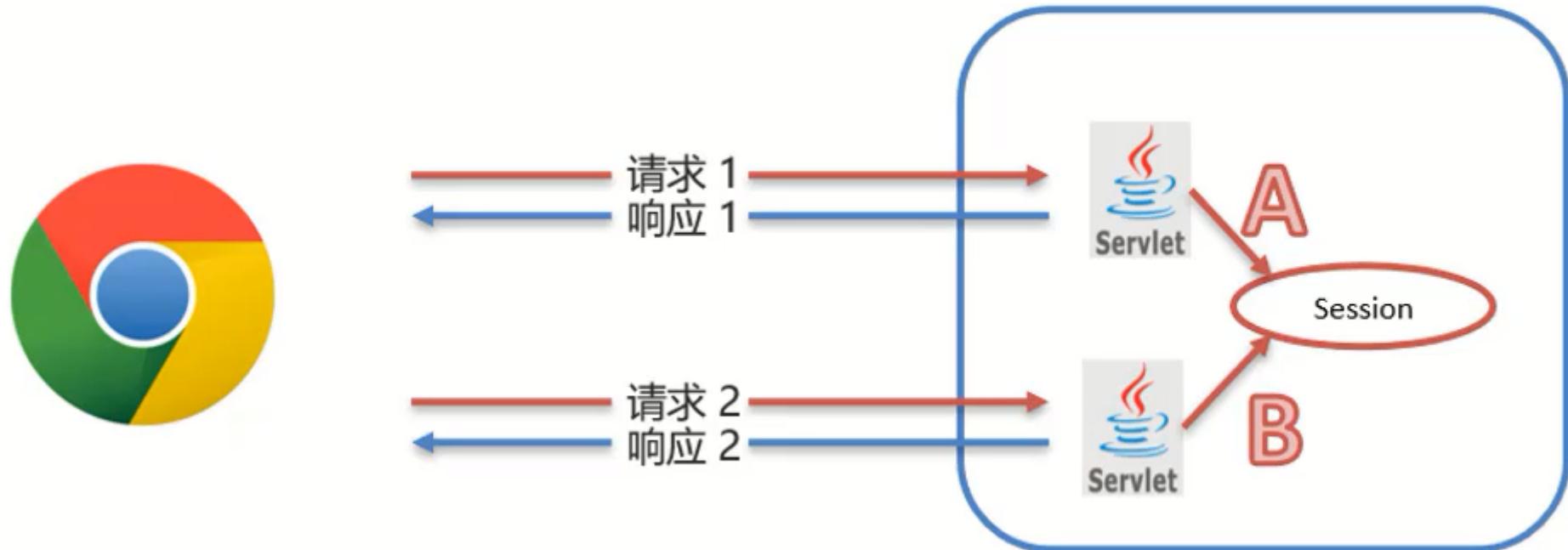
这节我们会主要讲解两个知识，第一个是Session的钝化和活化，第二个是Session的销毁，首先来学习什么是Session的钝化和活化？

3.3.1 Session钝化与活化

首先需要大家思考的问题是：

- 服务器重启后，Session中的数据是否还在？

要想回答这个问题，我们可以先看下下面这幅图，



(1) 服务器端AServlet和BServlet共用的session对象应该是存储在服务器的内存中

(2) 服务器重新启动后，内存中的数据应该是已经被释放，对象也应该都销毁了

所以session数据应该也已经不存在了。但是如果session不存在会引发什么问题呢？

举个例子说明下，

(1) 用户把需要购买的商品添加到购物车，因为要实现同一个会话多次请求数据共享，所以假设把数据存入Session对象中

(2) 用户正要付钱的时候接到一个电话，付钱的动作就搁浅了

(3) 正在用户打电话的时候，购物网站因为某些原因需要重启

(4) 重启后session数据被销毁，购物车中的商品信息也就会随之而消失

(5) 用户想再次发起支付，就会出为问题

所以说对于session的数据，我们应该做到就算服务器重启了，也应该能把数据保存下来才对。

分析了这么多，那么Tomcat服务器在重启的时候，session数据到底会不会保存以及是如何保存的，我们可以通过实际案例来演示下：

注意：这里所说的关闭和启动应该要确保是正常的关闭和启动。

那如何才是正常关闭Tomcat服务器呢？

需要使用命令行的方式来启动和停止Tomcat服务器：

启动：进入到项目pom.xml所在目录，执行 `tomcat7:run`

```
Terminal: Local +  
(c) Microsoft Corporation. 保留所有权利。  
D:\workspace\web-demo>cd ..  
D:\workspace>cd cookie-demo  
D:\workspace\cookie-demo>mvn tomcat7:run  
[INFO] Scanning for projects...  
[INFO]  
[INFO] -----< org.example:cookie-demo >-----  
[INFO] Building cookie-demo 1.0-SNAPSHOT  
[INFO] -----[ war ]-----
```

停止: 在启动的命令行界面, 输入 **ctrl+c**

```
Terminal: Local +  
八月 17, 2021 8:35:41 下午 org.apache.tomcat.util.http.Cookies processCookieHeader  
信息: Cookies: Invalid cookie. Value not a token or quoted value  
Note: further occurrences of Cookie errors will be logged at DEBUG level.  
org.apache.catalina.session.StandardSessionFacade@790e1576  
★ 终止批处理操作吗(Y/N)? y  
D:\workspace\cookie-demo>
```

有了上述两个正常启动和关闭的方式后, 接下来的测试流程是:

- (1) 先启动Tomcat服务器
- (2) 访问 <http://localhost:8080/cookie-demo/demo1> 将数据存入session中
- (3) 正确停止Tomcat服务器
- (4) 再次重新启动Tomcat服务器
- (5) 访问 <http://localhost:8080/cookie-demo/demo2> 查看是否能获取到session中的数据

```
Terminal: Local +  
信息: Starting ProtocolHandler ["http-bio-8080"]  
八月 17, 2021 8:36:35 下午 org.apache.tomcat.util.http.Cookies processCookieHeader  
信息: Cookies: Invalid cookie. Value not a token or quoted value  
Note: further occurrences of Cookie errors will be logged at DEBUG level.  
org.apache.catalina.session.StandardSessionFacade@436cae83  
ZS
```

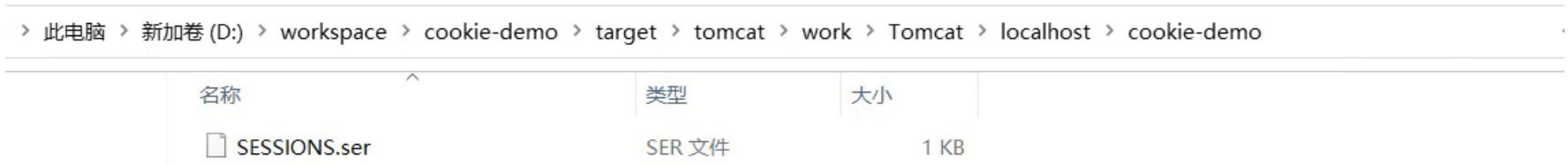
经过测试, 会发现只要服务器是正常关闭和启动, session中的数据是可以被保存下来的。

那么Tomcat服务器到底是如何做到的呢?

具体的原因就是: Session的钝化和活化:

- 钝化：在服务器正常关闭后，Tomcat会自动将Session数据写入硬盘的文件中

- 钝化的数据路径为：项目目录\target\tomcat\work\Tomcat\localhost\项目名称\SESSIONS.ser



- 活化：再次启动服务器后，从文件中加载数据到Session中

- 数据加载到Session中后，路径中的SESSIONS.ser文件会被删除掉

对于上述的整个过程，大家只需要了解下即可。因为所有的过程都是Tomcat自己完成的，不需要我们参与。

小结

Session的钝化和活化介绍完后，需要我们注意的是：

- session数据存储在服务端，服务器重启后，session数据会被保存
- 浏览器被关闭启动后，重新建立的连接就已经是一个全新的会话，获取的session数据也是一个新的对象
- session的数据要想共享，浏览器不能关闭，所以session数据不能长期保存数据
- cookie是存储在客户端，是可以长期保存

3.3.2 Session销毁

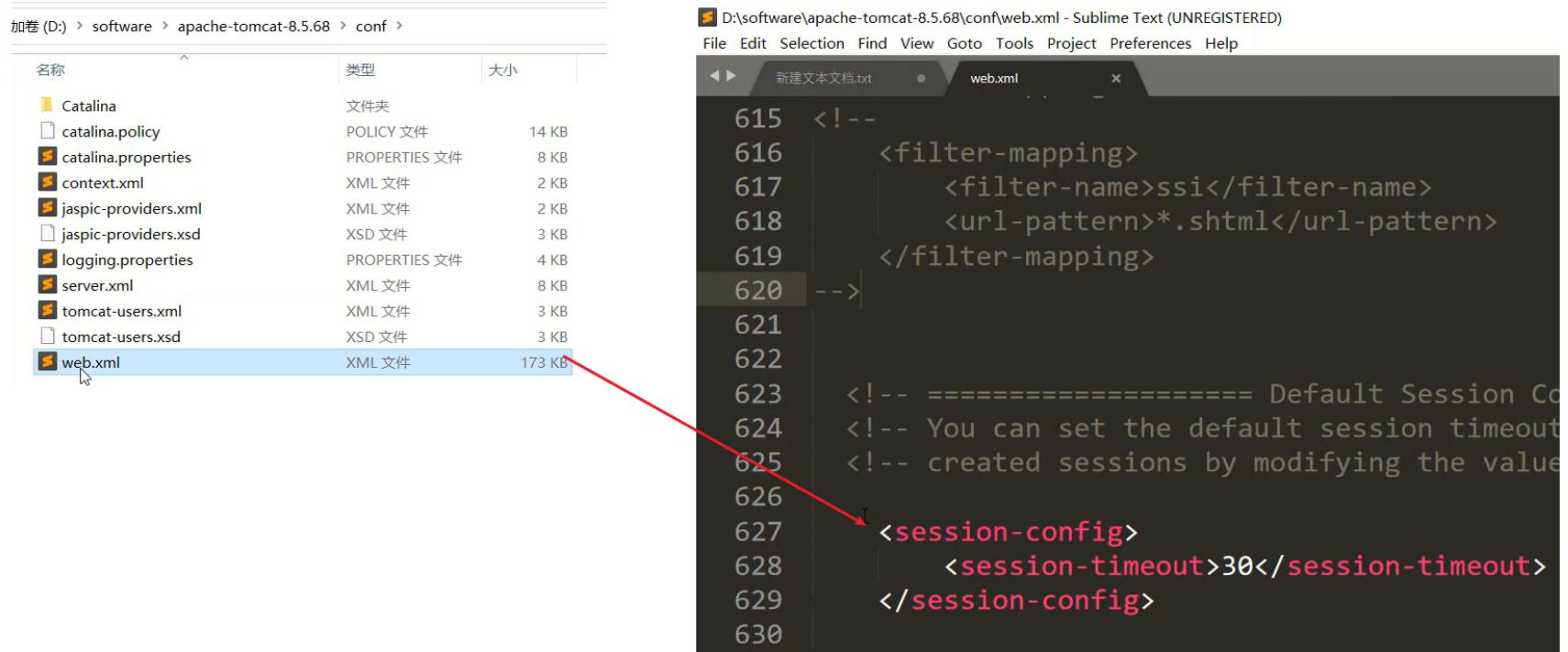
session的销毁会有两种方式：

- 默认情况下，无操作，30分钟自动销毁
 - 对于这个失效时间，是可以通过配置进行修改的
 - 在项目的web.xml中配置

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
5     http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
6   version="3.1">
7
8   <session-config>
9     <session-timeout>100</session-timeout>
10  </session-config>
11 </web-app>
```

- 如果没有配置，默认是30分钟，默认值是在Tomcat的web.xml配置文件中写死的



- 调用Session对象的`invalidate()`进行销毁

- 在`SessionDemo2`类中添加session销毁的方法

```

1 @WebServlet("/demo2")
2 public class SessionDemo2 extends HttpServlet {
3     @Override
4     protected void doGet(HttpServletRequest request, HttpServletResponse
5             response) throws ServletException, IOException {
6         //获取数据，从session中
7
8         //1. 获取Session对象
9         HttpSession session = request.getSession();
10        System.out.println(session);
11
12        // 销毁
13        session.invalidate();
14
15        //2. 获取数据
16        Object username = session.getAttribute("username");
17        System.out.println(username);
18    }
19
20    @Override
21    protected void doPost(HttpServletRequest request,
22                          HttpServletResponse response) throws ServletException, IOException {
23        this.doGet(request, response);
24    }
25
26 }

```

- 启动访问测试，先访问`demo1`将数据存入到`session`，再次访问`demo2`从`session`中获取数据

The screenshot shows a browser window with two tabs: 'localhost:8080/cookie-demo/' and 'Apache Tomcat/7.0.47 - Error'. The main content is an 'HTTP Status 500 - getAttribute: Session already invalidated' page. It includes an 'Exception report' section with details like type (Exception report), message (getAttribute: Session already invalidated), and description (The server encountered an internal error that prevented it from fulfilling this request). It also shows the full stack trace and a note about the root cause in the logs. Below this is a 'Apache Tomcat/7.0.47' section.

type Exception report

message getAttribute: Session already invalidated

description The server encountered an internal error that prevented it from fulfilling this request.

exception

```
java.lang.IllegalStateException: getAttribute: Session already invalidated
    org.apache.catalina.session.StandardSession.getAttribute(StandardSession.java:1167)
    org.apache.catalina.session.StandardSessionFacade.getAttribute(StandardSessionFacade.java:122)
    com.itheima.web.session.SessionDemo2 doGet(SessionDemo2.java:24)
    javax.servlet.http.HttpServlet.service(HttpServlet.java:621)
    javax.servlet.http.HttpServlet.service(HttpServlet.java:728)
    org.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter.java:51)
```

note The full stack trace of the root cause is available in the Apache Tomcat/7.0.47 logs.

- 该销毁方法一般会在用户退出的时候，需要将session销毁掉。

Cookie和Session小结

- Cookie 和 Session 都是用来完成一次会话内多次请求间**数据共享**的。

所需两个对象放在一块，就需要思考：

Cookie和Session的区别是什么？

Cookie和Session的应用场景分别是什么？

- 区别：**
 - 存储位置：Cookie 是将数据存储在客户端，Session 将数据存储在服务端
 - 安全性：Cookie不安全，Session安全
 - 数据大小：Cookie最大3KB，Session无大小限制
 - 存储时间：Cookie可以通过setMaxAge()长期存储，Session默认30分钟
 - 服务器性能：Cookie不占服务器资源，Session占用服务器资源
- 应用场景：**
 - 购物车：使用Cookie来存储
 - 以登录用户的名称展示：使用Session来存储
 - 记住我功能：使用Cookie来存储
 - 验证码：使用session来存储
- 结论**
 - Cookie是用来保证用户在未登录情况下的身份识别
 - Session是用来保存用户登录后的数据

介绍完Cookie和Session以后，具体用哪个还是需要根据具体的业务进行具体分析。

4. 用户登录注册案例

4.1 需求分析

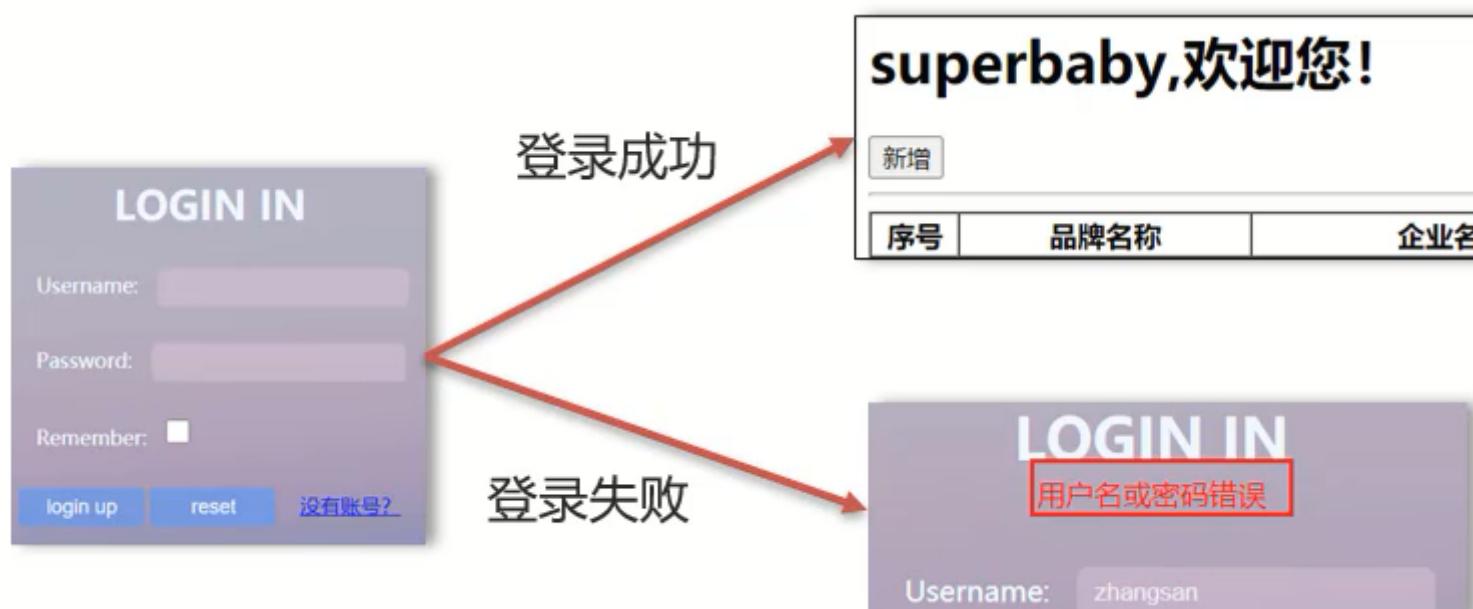
需求说明：

- 完成用户登录功能，如果用户勾选“记住用户”，则下次访问登录页面**自动**填充用户名密码
- 完成注册功能，并实现**验证码**功能



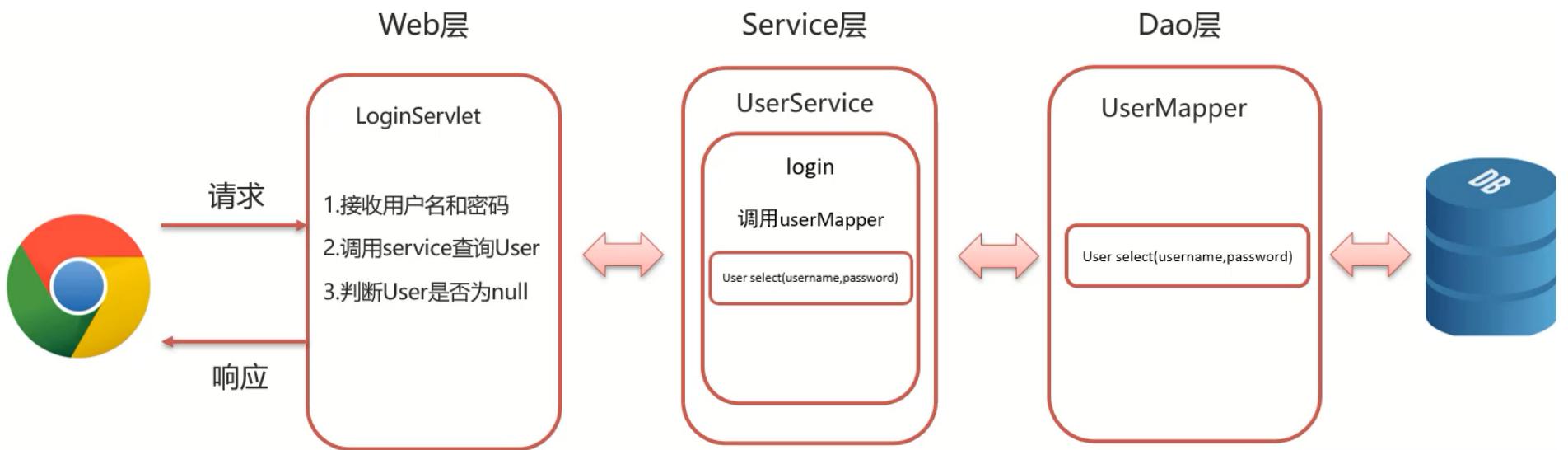
4.2 用户登录功能

- 需求：



- 用户登录成功后，跳转到列表页面，并在页面上展示当前登录的用户名称
- 用户登录失败后，跳转回登录页面，并在页面上展示对应的错误信息

- 实现流程分析



- (1) 前端通过表单发送请求和数据给Web层的LoginServlet
- (2) 在LoginServlet中接收请求和数据 [用户名和密码]
- (3) LoginServlet接收到请求和数据后，调用Service层完成根据用户名和密码查询用户对象
- (4) 在Service层需要编写UserService类，在类中实现login方法，方法中调用Dao层的UserMapper
- (5) 在UserMapper接口中，声明一个根据用户名和密码查询用户信息的方法
- (6) Dao层把数据查询出来以后，将返回数据封装到User对象，将对象交给Service层
- (7) Service层将数据返回给Web层
- (8) Web层获取到User对象后，判断User对象，如果为Null，则将错误信息响应给登录页面，如果不为Null，则跳转到列表页面，并把当前登录用户的信息存入Session携带到列表页面。

3. 具体实现

- (1) 完成Dao层的代码编写

(1.1) 将04-资料\1. 登录注册案例\2. MyBatis环境\UserMapper.java放到com.itheima.mapper包下：

```

1 public interface UserMapper {
2     /**
3      * 根据用户名和密码查询用户对象
4      * @param username
5      * @param password
6      * @return
7      */
8     @Select("select * from tb_user where username = #{username} and password
= #{password}")
9     User select(@Param("username") String username,@Param("password") String
password);
10
11    /**
12     * 根据用户名查询用户对象
13     * @param username
14     * @return

```

```
15     */
16     @Select("select * from tb_user where username = #{username}")
17     User selectByUsername(String username);
18
19     /**
20      * 添加用户
21      * @param user
22      */
23     @Insert("insert into tb_user values(null,#{username},#{password})")
24     void add(User user);
25 }
```

(1.2) 将04-资料\1. 登录注册案例\2. MyBatis环境\User.java放到com.itheima.pojo包下:

```
1 public class User {
2
3     private Integer id;
4     private String username;
5     private String password;
6
7     public Integer getId() {
8         return id;
9     }
10
11    public void setId(Integer id) {
12        this.id = id;
13    }
14
15    public String getUsername() {
16        return username;
17    }
18
19    public void setUsername(String username) {
20        this.username = username;
21    }
22
23    public String getPassword() {
24        return password;
25    }
26
27    public void setPassword(String password) {
28        this.password = password;
29    }
30
31    @Override
32    public String toString() {
33        return "User{" +
34                "id=" + id +
```

```
35             ", username='\" + username + '\" +  
36             ", password='\" + password + '\" +  
37             '}';  
38     }  
39 }
```

(1.3) 将 04-资料\1. 登录注册案例\2. MyBatis 环境\UserMapper.xml 放入到 resources/com/itheima/mapper 目录下：

```
1 <?xml version="1.0" encoding="UTF-8" ?>  
2 <!DOCTYPE mapper  
3         PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  
4         "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  
5 <mapper namespace="com.itheima.mapper.UserMapper">  
6  
7 </mapper>
```

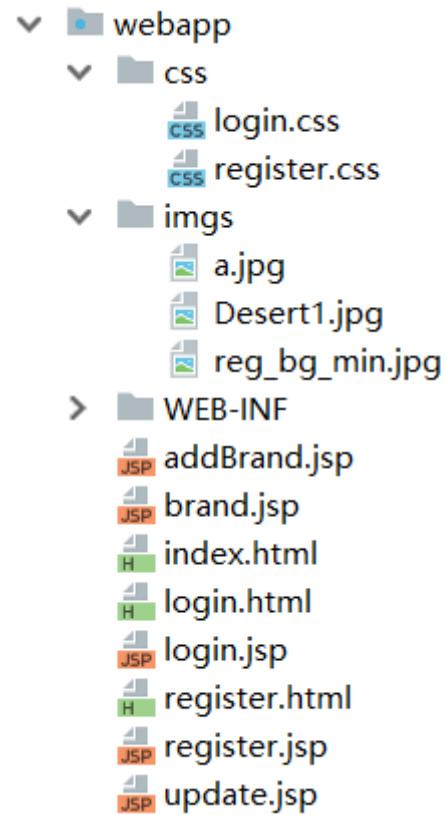
(2) 完成 Service 层的代码编写

(2.1) 在 com.itheima.service 包下，创建 UserService 类

```
1 public class UserService {  
2     //1. 使用工具类获取SqlSessionFactory  
3     SqlSessionFactory factory =  
4         SqlSessionFactoryUtils.getSqlSessionFactory();  
5     /**  
6      * 登录方法  
7      * @param username  
8      * @param password  
9      * @return  
10     */  
11     public User login(String username, String password){  
12         //2. 获取sqlSession  
13         sqlSession = factory.openSession();  
14         //3. 获取UserMapper  
15         UserMapper mapper = sqlSession.getMapper(UserMapper.class);  
16         //4. 调用方法  
17         User user = mapper.select(username, password);  
18         //释放资源  
19         sqlSession.close();  
20         return user;  
21     }  
22 }
```

(3) 完成页面和 Web 层的代码编写

(3.1) 将 04-资料\1. 登录注册案例\1. 静态页面拷贝到项目的 webapp 目录下：



(3.2) 将login.html内容修改成login.jsp

```
1 <%@ page contentType="text/html; charset=UTF-8" language="java" %>
2 <!DOCTYPE html>
3 <html lang="en">
4
5 <head>
6     <meta charset="UTF-8">
7     <title>login</title>
8     <link href="css/login.css" rel="stylesheet">
9 </head>
10
11 <body>
12 <div id="loginDiv" style="height: 350px">
13     <form action="/brand-demo/loginServlet" method="post" id="form">
14         <h1 id="loginMsg">LOGIN IN</h1>
15         <div id="errorMsg">用户名或密码不正确</div>
16         <p>Username:<input id="username" name="username" type="text"></p>
17         <p>Password:<input id="password" name="password" type="password"></p>
18         <p>Remember:<input id="remember" name="remember" type="checkbox"></p>
19         <div id="subDiv">
20             <input type="submit" class="button" value="login up">
21             <input type="reset" class="button"
22               value="reset">&ampnbsp&ampnbsp&ampnbsp
23             <a href="register.html">没有账号? </a>
24         </div>
25     </form>
26 </div>
27 </body>
```

(3.3) 创建LoginServlet类

```

1  @WebServlet("/loginServlet")
2  public class LoginServlet extends HttpServlet {
3      private UserService service = new UserService();
4
5      @Override
6      protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
7          //1. 获取用户名和密码
8          String username = request.getParameter("username");
9          String password = request.getParameter("password");
10
11         //2. 调用service查询
12         User user = service.login(username, password);
13
14         //3. 判断
15         if(user != null){
16             //登录成功，跳转到查询所有的BrandServlet
17
18             //将登陆成功后的user对象，存储到session
19             HttpSession session = request.getSession();
20             session.setAttribute("user",user);
21
22             String contextPath = request.getContextPath();
23             response.sendRedirect(contextPath+"/selectAllServlet");
24         }else {
25             // 登录失败,
26             // 存储错误信息到request
27             request.setAttribute("login_msg","用户名或密码错误");
28             // 跳转到login.jsp
29
30             request.getRequestDispatcher("/login.jsp").forward(request,response);
31         }
32     }
33
34     @Override
35     protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
36         this.doGet(request, response);
37     }
38 }
```

(3.4) 在brand.jsp中标签下添加欢迎当前用户的提示信息：

```
1 <h1>${user.username},欢迎您</h1>
```

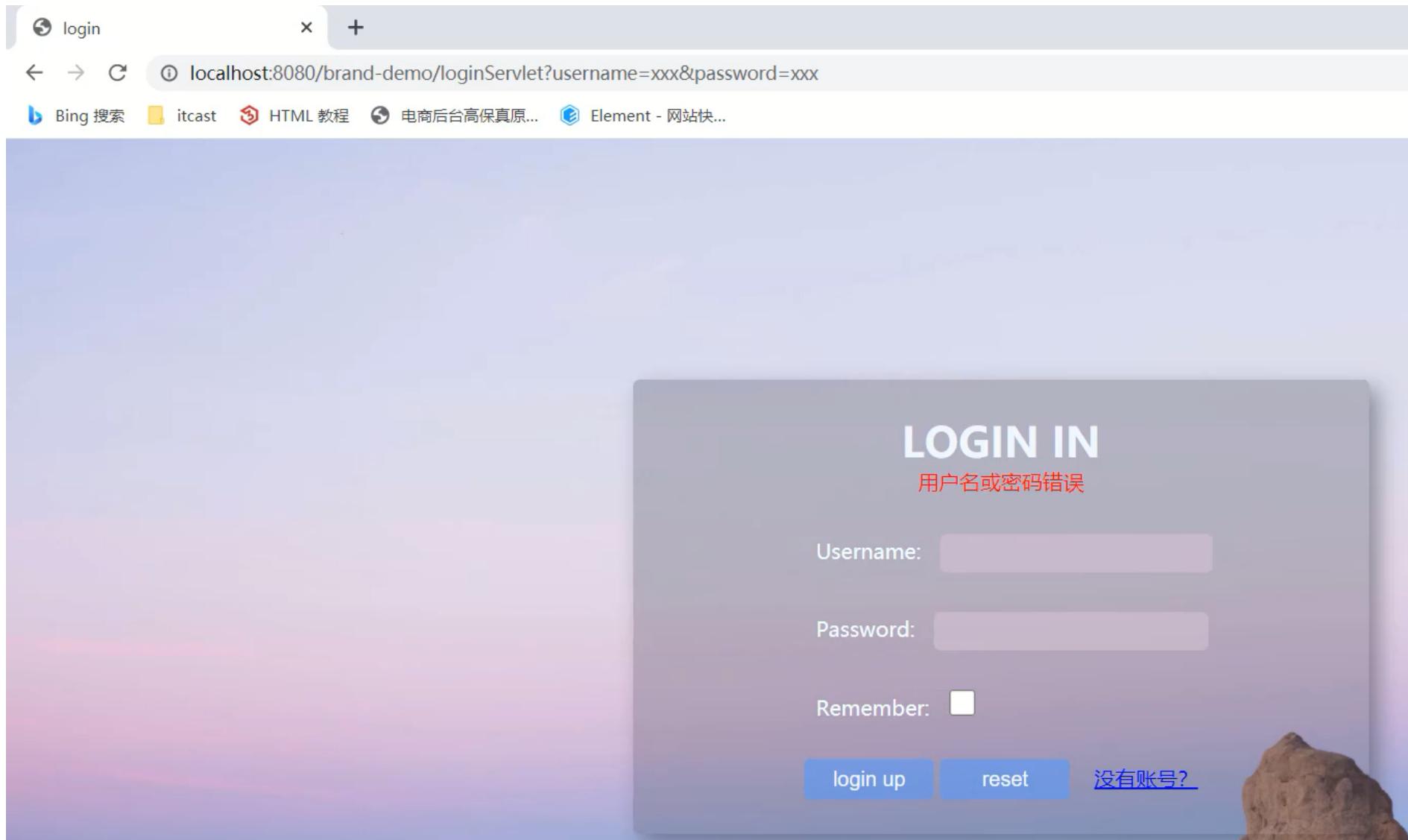
(3.5) 修改login.jsp, 将错误信息使用EL表达式来获取

1 修改前内容: <div id="errorMsg">用户名或密码不正确</div>

2 修改后内容: <div id="errorMsg">\${login_msg}</div>

(4) 启动，访问测试

(4.1) 进入登录页面，输入错误的用户名或密码



(4.2) 输入正确的用户名和密码信息



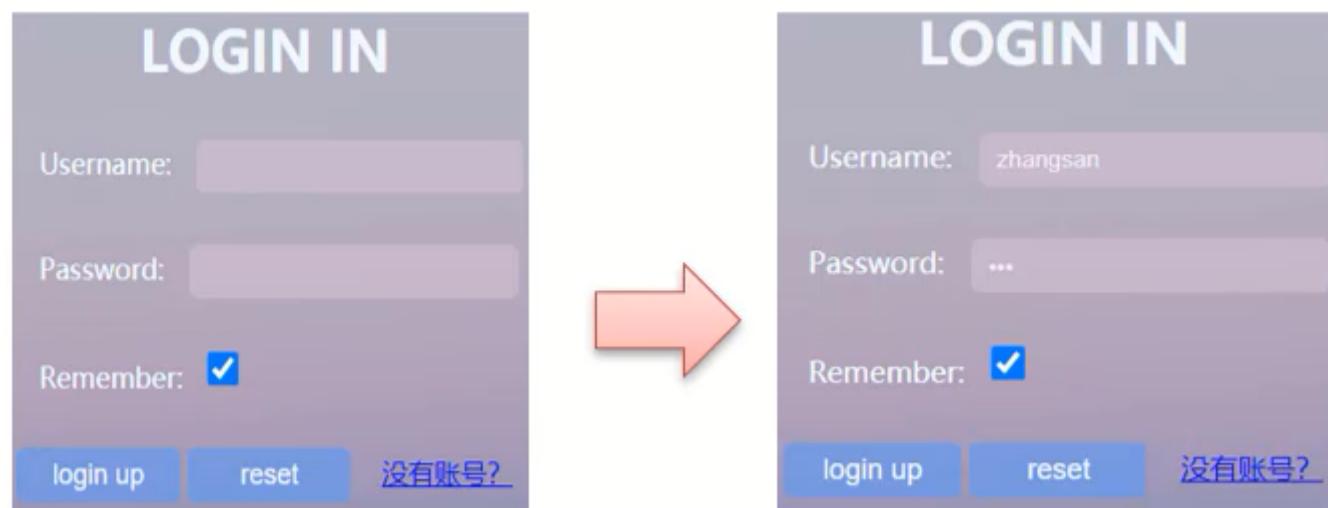
小结

- 在LoginServlet中，将登录成功的用户数据存入session中，方法在列表页面中获取当前登录用户信息进行展示
- 在LoginServlet中，将登录失败的错误信息存入到request中，如果存入到session中就会出现这次会话的所有请求都有登录失败的错误信息，这个是不需要的，所以不用存入到session中

4.3 记住我-设置Cookie

1. 需求：

如果用户勾选“记住用户”，则下次访问登陆页面自动填充用户名密码。这样可以提升用户的体验。

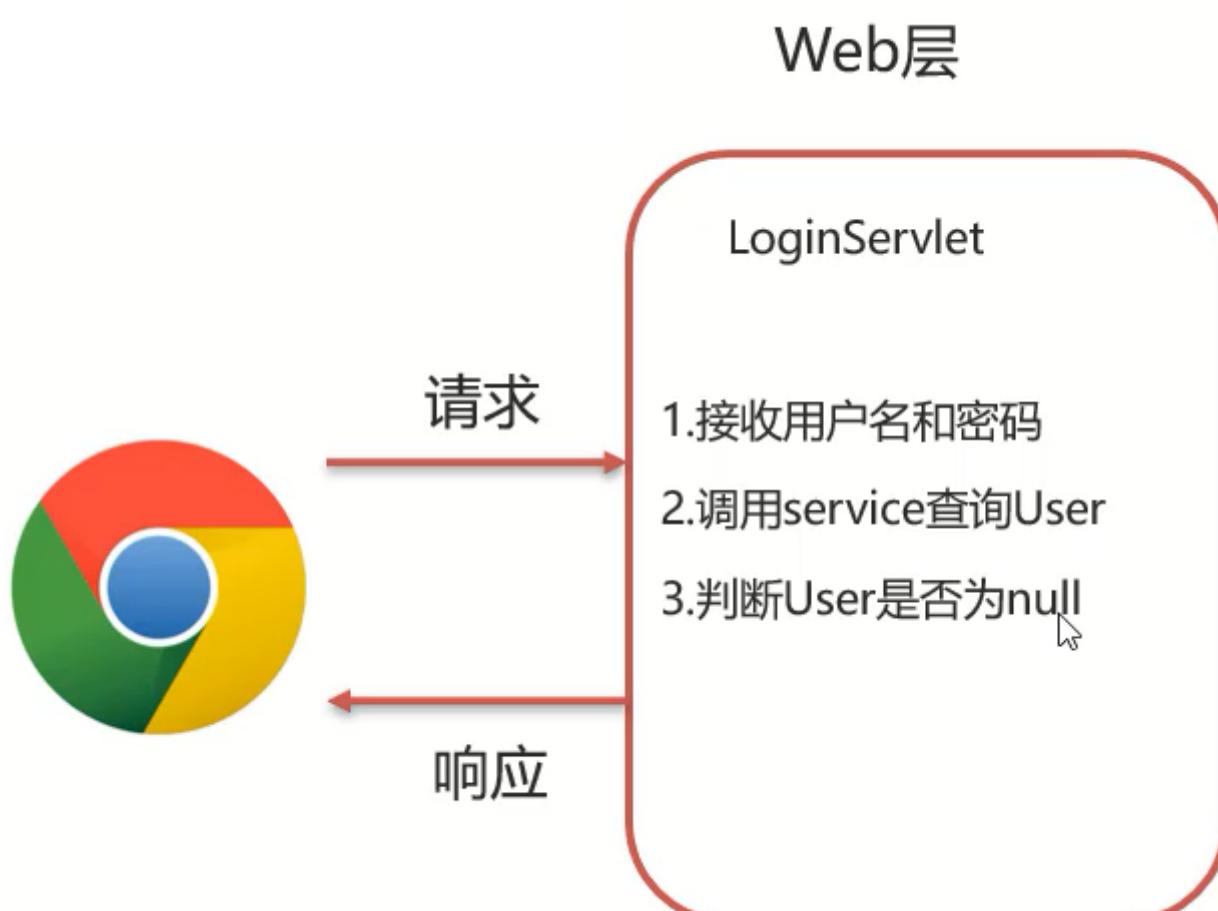


对应上面这个需求，最大的问题就是：如何自动填充用户名和密码？

2. 实现流程分析

因为记住我功能要实现的效果是，就算用户把浏览器关闭过几天再来访问也能自动填充，所以需要将登陆信息存入一个可以长久保存，并且能够在浏览器关闭重新启动后依然有效的地方，就是我们前面讲的Cookie，所以：

- 将用户名和密码写入Cookie中，并且持久化存储Cookie，下次访问浏览器会自动携带Cookie
- 在页面获取Cookie数据后，设置到用户名和密码框中
- 何时写入Cookie?
 - 用户必须登陆成功后才需要写
 - 用户必须在登录页面勾选了记住我的复选框



(1) 前端需要在发送请求和数据的时候，多携带一个用户是否勾选 Remember 的数据

(2) LoginServlet 获取到数据后，调用 Service 完成用户名和密码的判定

(3) 登录成功，并且用户在前端勾选了记住我，需要往Cookie中写入用户名和密码的数据，并设置Cookie存活时间

(4) 设置成功后，将数据响应给前端

3. 具体实现

(1) 在login.jsp为复选框设置值

```
1 <%@ page contentType="text/html; charset=UTF-8" language="java" %>
2 <!DOCTYPE html>
3 <html lang="en">
4
5 <head>
6   <meta charset="UTF-8">
7   <title>login</title>
8   <link href="css/login.css" rel="stylesheet">
9 </head>
10
11 <body>
12 <div id="loginDiv" style="height: 350px">
13   <form action="/brand-demo/loginServlet" method="post" id="form">
14     <h1 id="loginMsg">LOGIN IN</h1>
15     <div id="errorMsg">${login_msg}</div>
16     <p>Username:<input id="username" name="username" type="text"></p>
17     <p>Password:<input id="password" name="password" type="password"></p>
18     <p>Remember:<input id="remember" name="remember" value="1"
19       type="checkbox"></p>
20     <div id="subDiv">
21       <input type="submit" class="button" value="login up">
22       <input type="reset" class="button"
23         value="reset">&ampnbsp&ampnbsp&ampnbsp
24       <a href="register.html">没有账号? </a>
25     </div>
26   </form>
27 </div>
28 </body>
29 </html>
```

(2) 在LoginServlet获取复选框的值并在登录成功后进行设置Cookie

```
1 @WebServlet("/loginServlet")
2 public class LoginServlet extends HttpServlet {
3   private UserService service = new UserService();
4
5   @Override
6   protected void doGet(HttpServletRequest request, HttpServletResponse
7   response) throws ServletException, IOException {
8     //1. 获取用户名和密码
```

```
8     String username = request.getParameter("username");
9     String password = request.getParameter("password");
10
11    //获取复选框数据
12    String remember = request.getParameter("remember");
13
14    //2. 调用service查询
15    User user = service.login(username, password);
16
17    //3. 判断
18    if(user != null){
19        //登录成功，跳转到查询所有的BrandServlet
20
21        //判断用户是否勾选记住我，字符串写前面是为了避免出现空指针异常
22        if("1".equals(remember)){
23            //勾选了，发送cookie
24            //1. 创建Cookie对象
25            Cookie c_username = new Cookie("username",username);
26            Cookie c_password = new Cookie("password",password);
27            // 设置Cookie的存活时间
28            c_username.setMaxAge( 60 * 60 * 24 * 7);
29            c_password.setMaxAge( 60 * 60 * 24 * 7);
29            //2. 发送
30            response.addCookie(c_username);
31            response.addCookie(c_password);
32        }
33
34
35        //将登陆成功的user对象，存储到session
36        HttpSession session = request.getSession();
37        session.setAttribute("user",user);
38
39        String contextPath = request.getContextPath();
40        response.sendRedirect(contextPath+"/selectAllServlet");
41    }else {
42        // 登录失败,
43
44        // 存储错误信息到request
45        request.setAttribute("login_msg","用户名或密码错误");
46
47        // 跳转到login.jsp
48
49        request.getRequestDispatcher("/login.jsp").forward(request,response);
50    }
51
52
53    @Override
```

```

54     protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
55         this.doGet(request, response);
56     }
57 }

```

(3) 启动访问测试,

只有当前用户名和密码输入正确，并且勾选了Remember的复选框，在响应头中才可以看得cookie的相关数据

序号	品牌名称	企业名称	排序	品牌介绍

Network

Filter: All

5 ms	10 ms	15 ms	20 ms	25 ms	30 ms	35 ms	40 ms	45 ms	50 ms	55 ms
------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------

Name: loginServlet?username=zhangsan&password=123

General

Request URL: http://localhost:8080/brand-demo/loginServlet?username=zhangsan&password=123

Request Method: GET

Status Code: 302 Found

Remote Address: [::1]:8080

Referrer Policy: strict-origin-when-cross-origin

Response Headers

Content-Length: 0

Date: Wed, 18 Aug 2021 13:05:40 GMT

Location: http://localhost:8080/brand-demo/selectAllServlet

Server: Apache-Coyote/1.1

Set-Cookie: username=zhangsan; Expires=Wed, 25-Aug-2021 13:05:40 GMT

Set-Cookie: password=123; Expires=Wed, 25-Aug-2021 13:05:40 GMT

Request Headers

4.4 记住我-获取Cookie

1. 需求

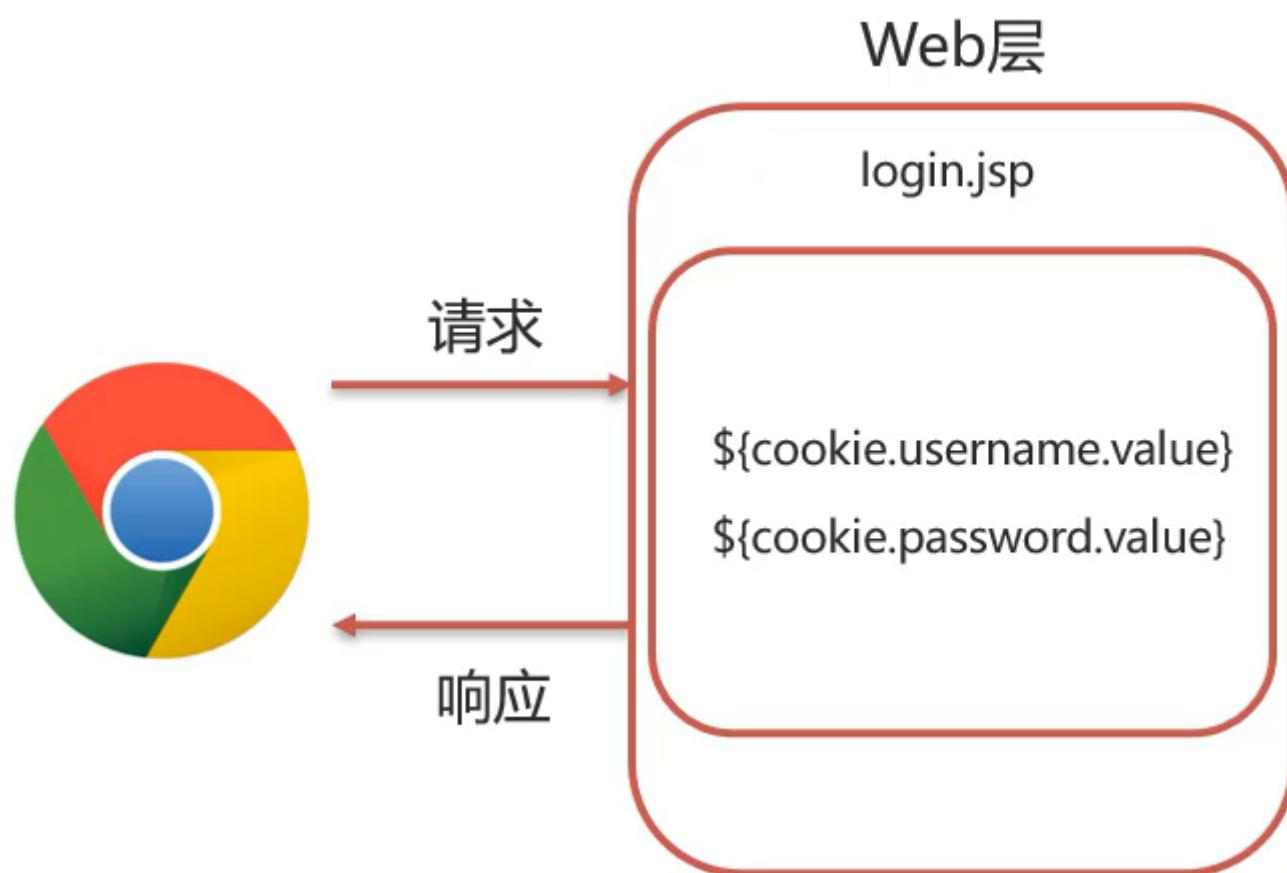
登录成功并勾选了Remember后，后端返回给前端的Cookie数据就已经存储好了，接下来就需要在页面获取Cookie中的数据，并把数据设置到登录页面的用户名和密码框中。

如何在页面直接获取Cookie中的值呢？

2. 实现流程分析

在页面可以使用EL表达式， `${cookie.key.value}`

key：指的是存储在cookie中的键名称



(1) 在login.jsp用户名的表单输入框使用value值给表单元素添加默认值，value可以使用 `${cookie.username.value}`

(2) 在login.jsp密码的表单输入框使用value值给表单元素添加默认值，value可以使用 `${cookie.password.value}`

3. 具体实现

(1) 修改login.jsp页面

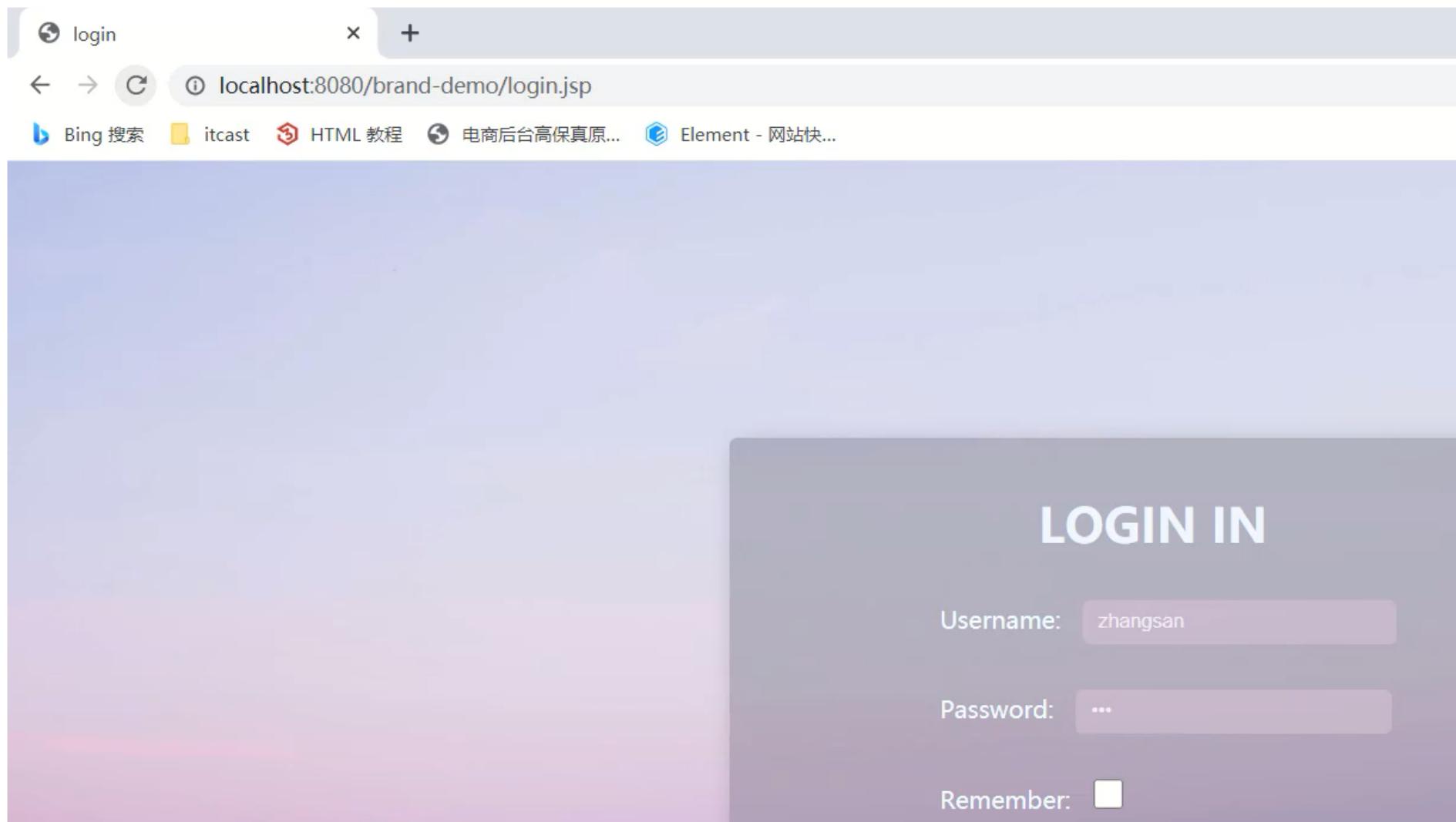
```

1 <%@ page contentType="text/html; charset=UTF-8" language="java" %>
2 <!DOCTYPE html>
3 <html lang="en">
4
5 <head>
6   <meta charset="UTF-8">

```

```
7 <title>login</title>
8 <link href="css/login.css" rel="stylesheet">
9 </head>
10
11 <body>
12 <div id="loginDiv" style="height: 350px">
13   <form action="/brand-demo/loginServlet" method="post" id="form">
14     <h1 id="loginMsg">LOGIN IN</h1>
15     <div id="errorMsg">${login_msg}</div>
16     <p>Username:<input id="username" name="username"
17       value="${cookie.username.value}" type="text"></p>
18
19     <p>Password:<input id="password" name="password"
20       value="${cookie.password.value}" type="password"></p>
21     <p>Remember:<input id="remember" name="remember" value="1"
22       type="checkbox"></p>
23     <div id="subDiv">
24       <input type="submit" class="button" value="login up">
25       <input type="reset" class="button"
26         value="reset">&nbsp;&nbsp;
27       <a href="register.html">没有账号? </a>
28     </div>
29   </form>
30 </div>
31 </body>
32 </html>
```

4. 访问测试，重新访问登录页面，就可以看得用户和密码已经被填充。



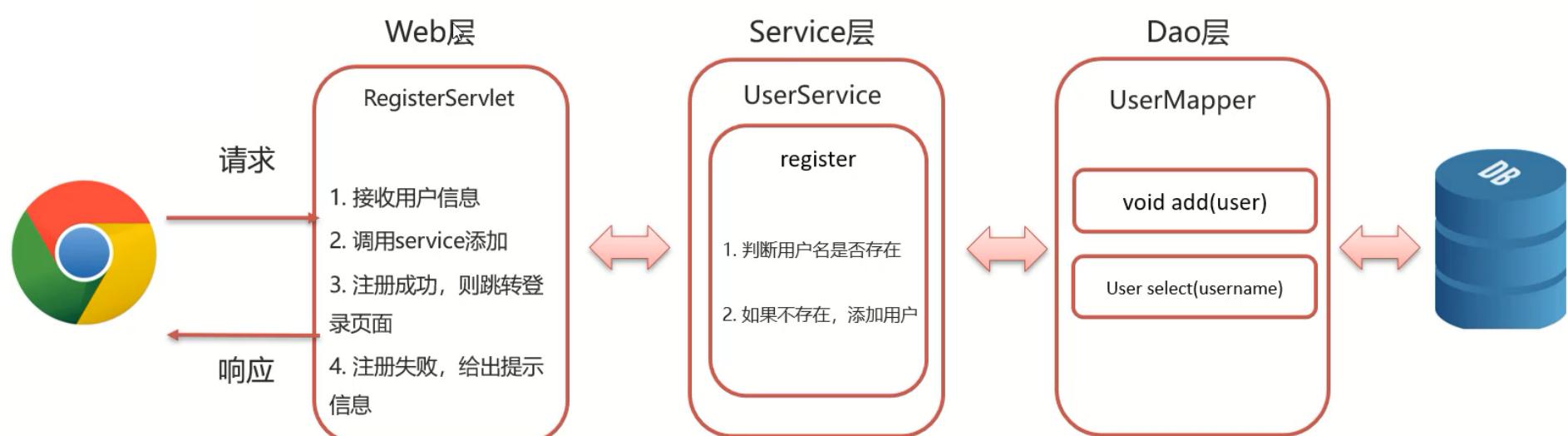
4.5 用户注册功能

1. 需求

- 注册功能：保存用户信息到数据库
- 验证码功能
 - 展示验证码：展示验证码图片，并可以点击切换
 - 校验验证码：验证码填写不正确，则注册失败



2. 实现流程分析



(1) 前端通过表单发送请求和数据给Web层的RegisterServlet

(2) 在RegisterServlet中接收请求和数据 [用户名和密码]

(3) RegisterServlet接收到请求和数据后，调用Service层完成用户信息的保存

(4) 在Service层需要编写UserService类，在类中实现register方法，需要判断用户是否已经存在，如果不存在，则完成用户数据的保存

(5) 在UserMapper接口中，声明两个方法，一个是根据用户名查询用户信息方法，另一个是保存用户信息方法

(6) 在UserService类中保存成功则返回true，失败则返回false，将数据返回给Web层

(7) Web层获取到结果后，如果返回的是true，则提示注册成功，并转发到登录页面，如果返回false则提示用户名已存在并转发到注册页面

3. 具体实现

(1) Dao层代码参考资料中的内容完成

(2) 编写Service层代码

```
1 public class UserService {
2     //1. 使用工具类获取sqlSessionFactory
3     sqlSessionFactory factory =
4         SqlSessionFactoryUtils.getSqlSessionFactory();
5     /**
6      * 注册方法
7      * @return
8      */
9
9     public boolean register(User user){
10        //2. 获取sqlSession
11        sqlSession sqlSession = factory.openSession();
12        //3. 获取UserMapper
13        UserMapper mapper = sqlSession.getMapper(UserMapper.class);
14        //4. 判断用户名是否存在
15        User u = mapper.selectByUsername(user.getUsername());
16
17        if(u == null){
18            // 用户名不存在，注册
19            mapper.add(user);
20            sqlSession.commit();
21        }
22        sqlSession.close();
23
24        return u == null;
25
26    }
27 }
```

(3) 完成页面和Web层的代码编写

(3.1) 将register.html内容修改成register.jsp

```
1 <%@ page contentType="text/html; charset=UTF-8" language="java" %>
2 <!DOCTYPE html>
3 <html lang="en">
4 <head>
5     <meta charset="UTF-8">
6     <title>欢迎注册</title>
7     <link href="css/register.css" rel="stylesheet">
8 </head>
9 <body>
10 <div class="form-div">
11     <div class="reg-content">
12         <h1>欢迎注册</h1>
13         <span>已有帐号? </span> <a href="login.html">登录</a>
14     </div>
15     <form id="reg-form" action="/brand-demo/registerServlet" method="post">
16         <table>
17             <tr>
18                 <td>用户名</td>
19                 <td class="inputs">
20                     <input name="username" type="text" id="username">
21                     <br>
22                     <span id="username_err" class="err_msg" style="display:none">用户名不太受欢迎</span>
23                     </td>
24                 </tr>
25                 <tr>
26                     <td>密码</td>
27                     <td class="inputs">
28                         <input name="password" type="password" id="password">
29                         <br>
30                         <span id="password_err" class="err_msg" style="display:none">密码格式有误</span>
31                     </td>
32                 </tr>
33                 <tr>
34                     <td>验证码</td>
35                     <td class="inputs">
36                         <input name="checkCode" type="text" id="checkCode">
37                         
38                         <a href="#" id="changeImg" >看不清? </a>
39                     </td>
40                 </tr>
41             </table>
42             <div class="buttons">
43                 <input value="注 册" type="submit" id="reg_btn">
44             </div>
45             <br class="clear">
```

```
46    </form>
47  </div>
48 </body>
49 </html>
```

(3.2) 编写RegisterServlet

```
1 @WebServlet("/registerServlet")
2 public class RegisterServlet extends HttpServlet {
3     private UserService service = new UserService();
4
5     @Override
6     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
7         //1. 获取用户名和密码数据
8         String username = request.getParameter("username");
9         String password = request.getParameter("password");
10
11         User user = new User();
12         user.setUsername(username);
13         user.setPassword(password);
14
15         //2. 调用service 注册
16         boolean flag = service.register(user);
17         //3. 判断注册成功与否
18         if(flag){
19             //注册功能，跳转登陆页面
20             request.setAttribute("register_msg","注册成功，请登录");
21
22             request.getRequestDispatcher("/login.jsp").forward(request,response);
23         }else {
24             //注册失败，跳转到注册页面
25
26             request.setAttribute("register_msg","用户名已存在");
27         }
28
29     }
30
31     @Override
32     protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
33         this.doGet(request, response);
34     }
35 }
36 }
```

(3.3) 需要在页面上展示后台返回的错误信息，需要修改register.jsp

- 1 修改前: 用户名不太受欢迎
- 2 修改后: \${register_msg}

(3.4) 如果注册成功，需要把成功信息展示在登录页面，所以也需要修改login.jsp

- 1 修改前: <div id="errorMsg">\${login_msg}</div>
- 2 修改后: <div id="errorMsg">\${login_msg} \${register_msg}</div>

(3.5) 修改login.jsp，将注册跳转地址修改为register.jsp

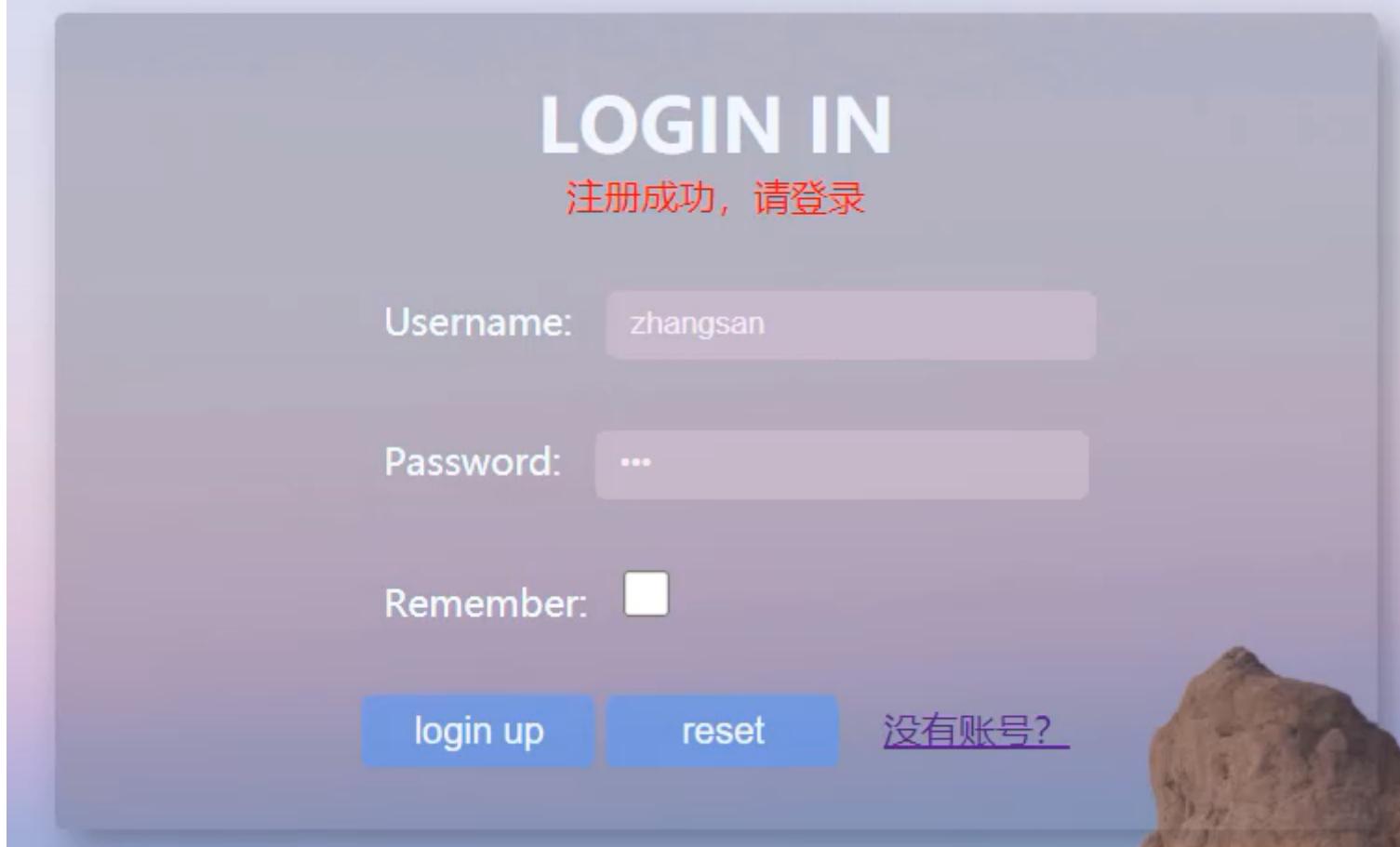
- 1 修改前: 没有账号?
- 2 修改后: 没有账号?

(3.6) 启动测试，

如果是注册的用户信息已经存在：



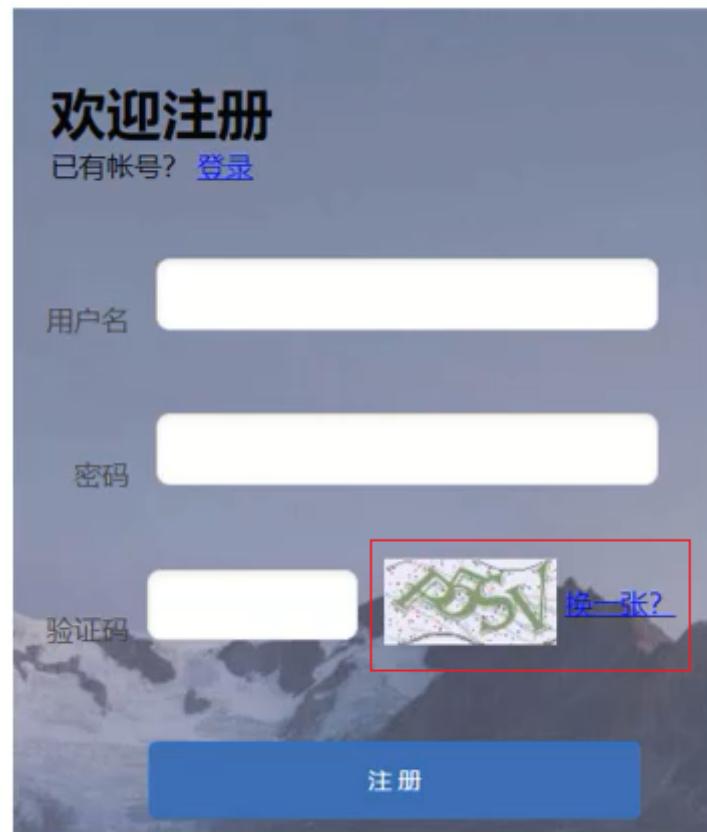
如果注册的用户信息不存在，注册成功：



4.6 验证码-展示

1. 需求分析

展示验证码：展示验证码图片，并可以点击切换



验证码的生成是通过工具类来实现的，具体的工具类参考

04-资料\1. 登录注册案例\CheckCodeUtil.java

在该工具类中编写main方法进行测试：

```

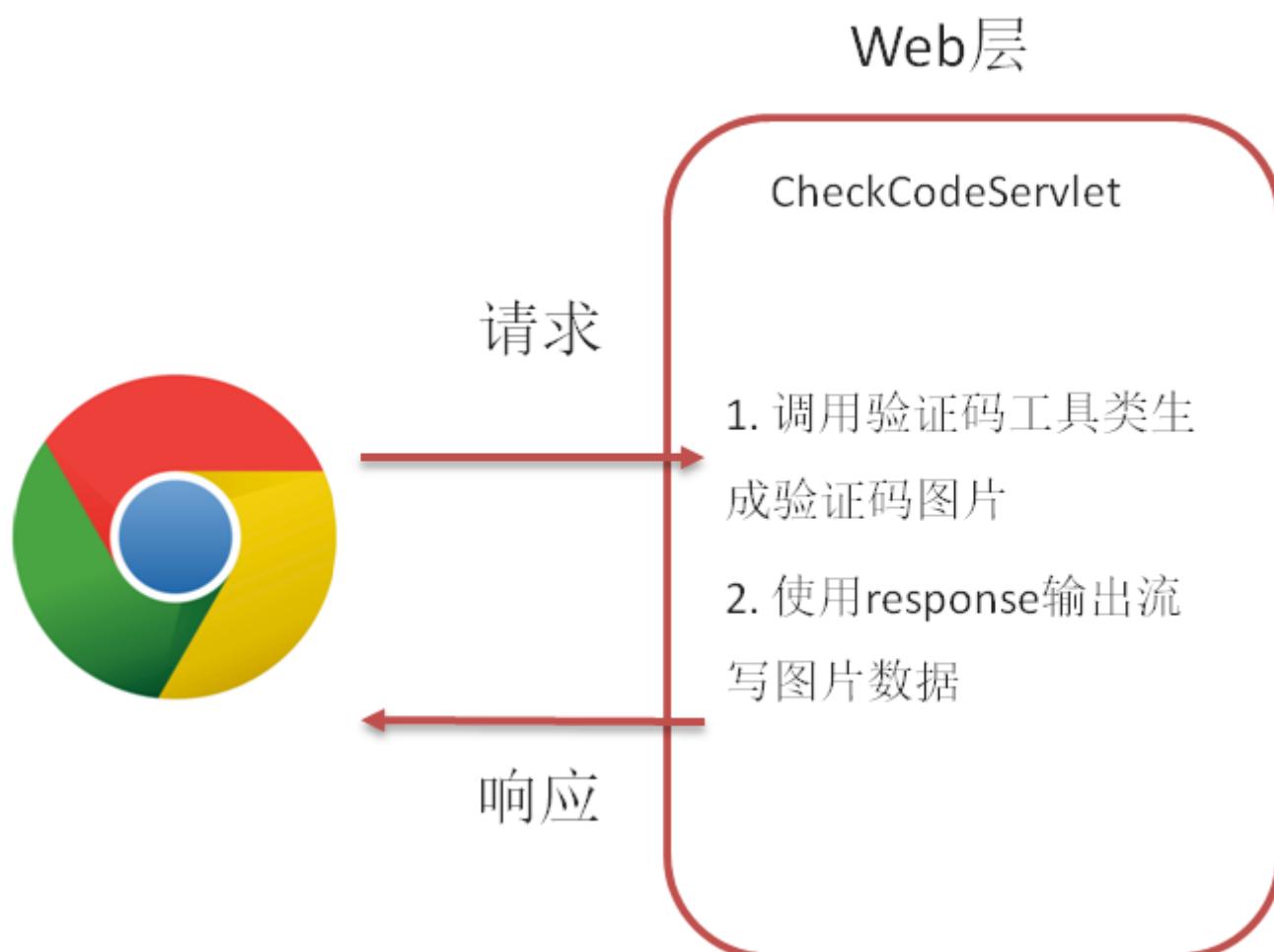
1 public static void main(String[] args) throws IOException {
2     //生成验证码的图片位置
3     OutputStream fos = new FileOutputStream("d://a.jpg");
4     //checkCode为最终验证码的数据
5     String checkCode = CheckCodeutil.outputVerifyImage(100, 50, fos, 4);
6     System.out.println(checkCode);
7 }
8

```

生成完验证码以后，我们就可以知晓：

- 验证码就是使用Java代码生成的一张图片
- 验证码的作用：防止机器自动注册，攻击服务器

2. 实现流程分析



(1) 前端发送请求给CheckCodeServlet

(2) CheckCodeServlet接收到请求后，生成验证码图片，将图片用Reponse对象的输出流写回到前端

思考：如何将图片写回到前端浏览器呢？

(1) Java中已经有工具类生成验证码图片，测试类中只是把图片生成到磁盘上 (2) 生成磁盘的过程中使用的是OutputStream流，如何把这个图片生成在页面呢？ (3) 前面在将Reponse对象的时候，它有一个方法可以获取其字节输出流，getOutputStream() (4) 综上所述，我们可以把写往磁盘的流对象更好成Response的字节流，即可完成图片响应给前端

3. 具体实现

(1) 修改Register.jsp页面，将验证码的图片从后台获取

```

1 <tr>
2     <td>验证码</td>
3         <td class="inputs">
4             <input name="checkCode" type="text" id="checkCode">
5             
6             <a href="#" id="changeImg" >看不清? </a>
7         </td>
8     </tr>
9
10 <script>
11     document.getElementById("changeImg").onclick = function () {
12         //路径后面添加时间戳的目的是避免浏览器进行缓存静态资源
13         document.getElementById("checkCodeImg").src = "/brand-
14         demo/checkCodeServlet?" + new Date().getMilliseconds();
15     }
16 </script>

```

(2) 编写CheckCodeServlet类，用来接收请求生成验证码

```

1 @WebServlet("/checkCodeServlet")
2 public class CheckCodeServlet extends HttpServlet {
3     @Override
4     protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
5         // 生成验证码
6         ServletOutputStream os = response.getOutputStream();
7         String checkCode = CheckCodeUtil.outputVerifyImage(100, 50, os, 4);
8     }
9
10    @Override
11    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
12        this.doGet(request, response);
13    }
14 }

```

4.7 验证码-校验

1. 需求

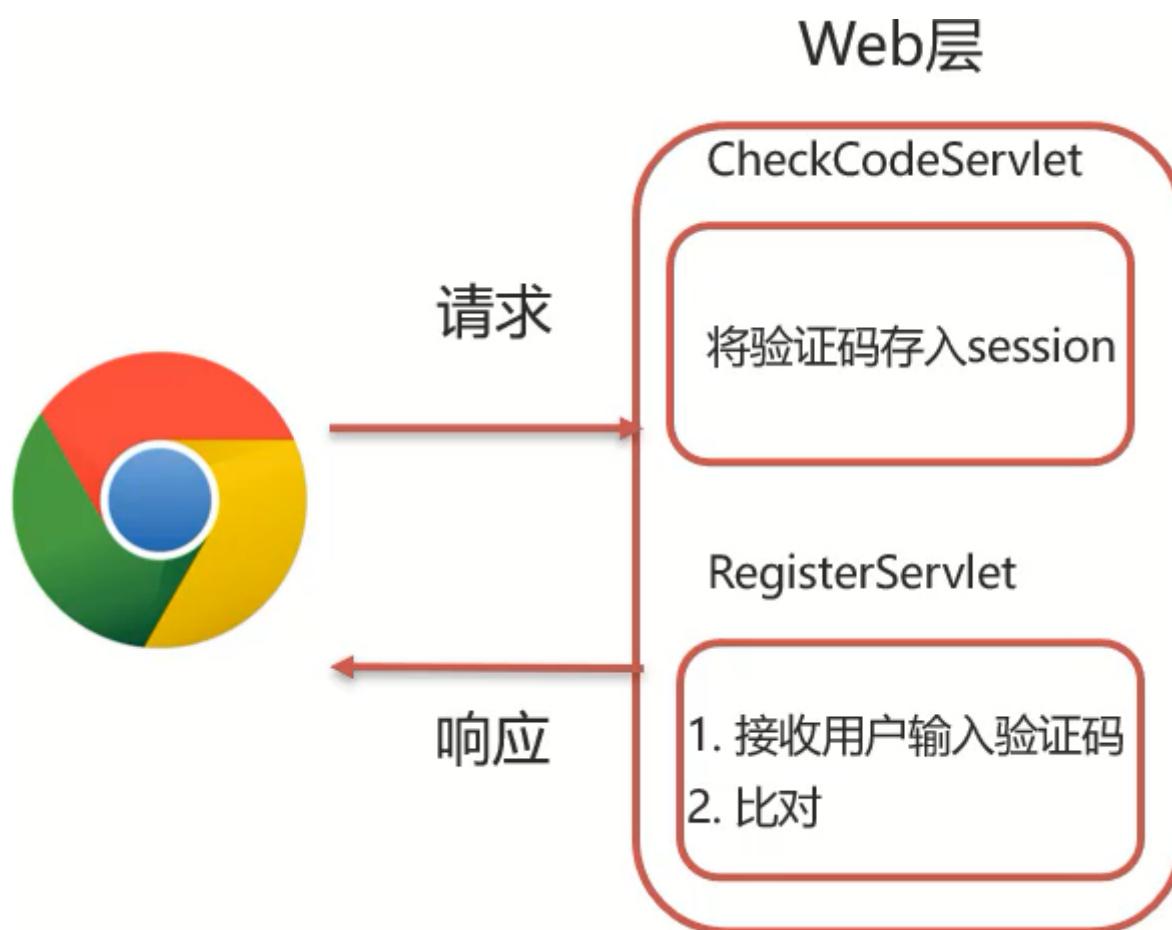
- 判断程序生成的验证码 和 用户输入的验证码 是否一样，如果不一样，则阻止注册
- 验证码图片访问和提交注册表单是**两次**请求，所以要将程序生成的验证码存入Session中



思考:为什么要把验证码数据存入到Session中呢?

- 生成验证码和校验验证码是两次请求，此处就需要在一个会话的两次请求之间共享数据
- 验证码属于安全数据类的，所以我们选中Session来存储验证码数据。

2. 实现流程分析



- (1) 在CheckCodeServlet中生成验证码的时候，将验证码数据存入Session对象
- (2) 前端将验证码和注册数据提交到后台，交给RegisterServlet类
- (3) RegisterServlet类接收到请求和数据后，其中就有验证码，和Session中的验证码进行对比
- (4) 如果一致，则完成注册，如果不一致，则提示错误信息

3. 具体实现

- (1) 修改CheckCodeServlet类，将验证码存入Session对象

```

1 @WebServlet("/checkCodeServlet")
2 public class CheckCodeServlet extends HttpServlet {
3     @Override
4         protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
5
6             // 生成验证码
7             ServletOutputStream os = response.getOutputStream();
8             String checkCode = CheckCodeUtil.outputVerifyImage(100, 50, os, 4);
9
10            // 存入session
11            HttpSession session = request.getSession();
12            session.setAttribute("checkCodeGen", checkCode);
13
14        }
15
16
17        @Override
18        protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
19            this.doGet(request, response);
20        }
21    }

```

(2) 在 RegisterServlet 中，获取页面的和 session 对象中的验证码，进行对比

```

1 package com.itheima.web;
2
3 import com.itheima.pojo.User;
4 import com.itheima.service.UserService;
5
6 import javax.servlet.ServletException;
7 import javax.servlet.annotation.WebServlet;
8 import javax.servlet.http.*;
9 import java.io.IOException;
10
11 @WebServlet("/registerServlet")
12 public class RegisterServlet extends HttpServlet {
13     private UserService service = new UserService();
14
15     @Override
16     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
17         // 1. 获取用户名和密码数据
18         String username = request.getParameter("username");
19         String password = request.getParameter("password");
20
21         User user = new User();

```

```
22     user.setUsername(username);
23     user.setPassword(password);
24
25     // 获取用户输入的验证码
26     String checkCode = request.getParameter("checkCode");
27
28     // 程序生成的验证码，从session获取
29     HttpSession session = request.getSession();
30     String checkCodeGen = (String) session.getAttribute("checkCodeGen");
31
32     // 比对
33     if(!checkCodeGen.equalsIgnoreCase(checkCode)){
34
35         request.setAttribute("register_msg", "验证码错误");
36
37         request.getRequestDispatcher("/register.jsp").forward(request, response);
38
39         // 不允许注册
40         return;
41     }
42     //2. 调用service 注册
43     boolean flag = service.register(user);
44     //3. 判断注册成功与否
45     if(flag){
46
47         //注册功能，跳转登陆页面
48
49         request.setAttribute("register_msg", "注册成功，请登录");
50
51         request.getRequestDispatcher("/login.jsp").forward(request, response);
52     }else {
53
54         //注册失败，跳转到注册页面
55
56         request.setAttribute("register_msg", "用户名已存在");
57
58     }
59
60     @Override
61     protected void doPost(HttpServletRequest request, HttpServletResponse
62     response) throws ServletException, IOException {
63         this doGet(request, response);
64     }
65 }
```

至此，用户的注册登录功能就已经完成了。

(8)

Filter&Listener&Ajax

今日目标：

- 能够使用 Filter 完成登陆状态校验功能
- 能够使用 axios 发送 ajax 请求
- 熟悉 json 格式，并能使用 Fastjson 完成 java 对象和 json 串的相互转换
- 使用 axios + json 完成综合案例

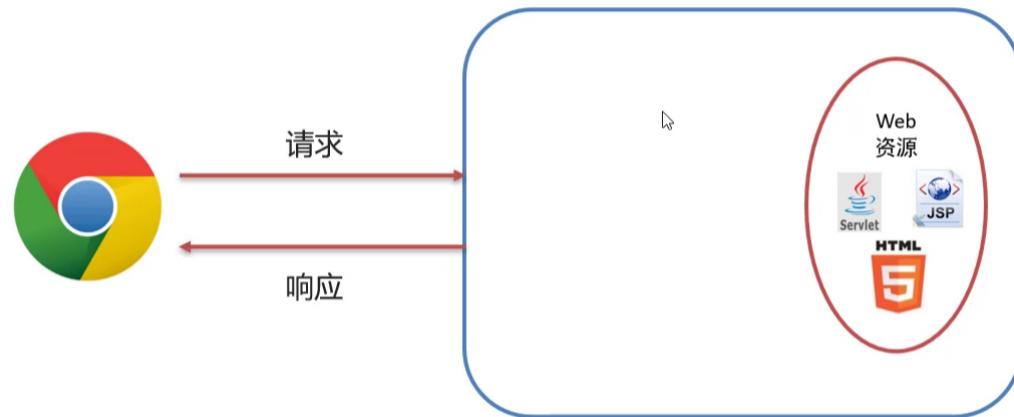
1, Filter

1.1 Filter概述

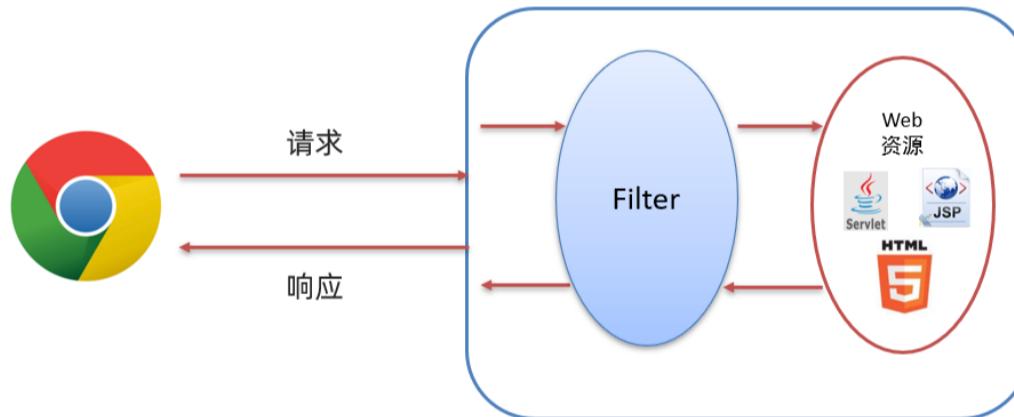
Filter 表示过滤器，是 JavaWeb 三大组件(Servlet、Filter、Listener)之一。Servlet 我们之前都已经学习过了，Filter 和 Listener 我们今天都会进行学习。

过滤器可以把对资源的请求**拦截**下来，从而实现一些特殊的功能。

如下图所示，浏览器可以访问服务器上的所有的资源 (servlet、jsp、html等)



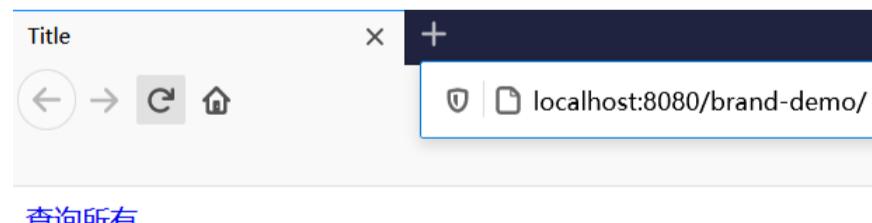
而在访问到这些资源之前可以使过滤器拦截下来，也就是说在访问资源之前会先经过 Filter，如下图



拦截器拦截到后可以做什么功能呢？

过滤器一般完成一些通用的操作。比如每个资源都要写一些代码完成某个功能，我们总不能在每个资源中写这样的代码吧，而此时我们可以将这些代码写在过滤器中，因为请求每一个资源都要经过过滤器。

我们之前做的品牌数据管理的案例中就已经做了登陆的功能，而如果我们不登录能不能访问到数据呢？我们可以在浏览器直接访问首页，可以看到 [查询所有](#) 的超链接



当我点击该按钮，居然可以看到品牌的的数据

新增

序号	品牌名称	企业名称	排序	品牌介绍	状态	操作
1	三只松鼠	三只松鼠股份有限公司	5	好吃不上火	禁用	修改 删除
2	华为	华为技术有限公司	100	华为致力于把数字世界带入每个人、每个家庭、每个组织，构建万物互联的智能世界	启用	修改 删除
3	小米	小米科技有限公司	50	are you ok	启用	修改 删除
4	鸿星尔克	鸿星尔克	10	to be no.1	禁用	修改 删除

这显然和我们的要求不符。我们希望实现的效果是用户如果登陆过了就跳转到品牌数据展示的页面；如果没有登陆就跳转到登陆页面让用户进行登陆，要实现这个效果需要在每一个资源中都写上这段逻辑，而像这种通用的操作，我们就可以放在过滤器中进行实现。这个就是**权限控制**，以后我们还会进行细粒度权限控制。过滤器还可以做**统一编码处理**、**敏感字符处理**等...等等...

1.2 Filter快速入门

1.2.1 开发步骤

进行 Filter 开发分成以下三步实现

- 定义类，实现 Filter 接口，并重写其所有方法

```
public class FilterDemo implements Filter {
    public void init(FilterConfig filterConfig) {
    }
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) {
        //放行
        chain.doFilter(request, response);
    }
    public void destroy() {
    }
}
```

- 配置 Filter 拦截资源的路径：在类上定义 @WebFilter 注解。而注解的 value 属性值 /* 表示拦截所有的资源

```
@WebFilter("/*")
public class FilterDemo implements Filter {
```

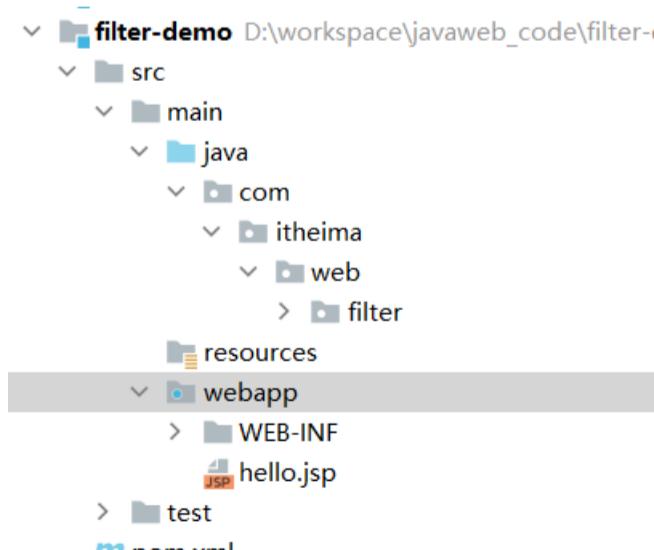
- 在 doFilter 方法中输出一句话，并放行

```
public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) {
    System.out.println("filter 被执行了...");
    //放行
    chain.doFilter(request, response);
}
```

上述代码中的 chain.doFilter(request, response); 就是放行，也就是让其访问本该访问的资源。

1.2.2 代码演示

创建一个项目，项目下有一个 hello.jsp 页面，项目结构如下：



pom.xml 配置文件内容如下：

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```

4         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
5             http://maven.apache.org/xsd/maven-4.0.0.xsd">
6             <modelVersion>4.0.0</modelVersion>
7             <groupId>org.example</groupId>
8             <artifactId>filter-demo</artifactId>
9             <version>1.0-SNAPSHOT</version>
10            <packaging>war</packaging>
11
12            <properties>
13                <maven.compiler.source>8</maven.compiler.source>
14                <maven.compiler.target>8</maven.compiler.target>
15            </properties>
16
17            <dependencies>
18                <dependency>
19                    <groupId>javax.servlet</groupId>
20                    <artifactId>javax.servlet-api</artifactId>
21                    <version>3.1.0</version>
22                    <scope>provided</scope>
23                </dependency>
24            </dependencies>
25
26            <build>
27                <plugins>
28                    <plugin>
29                        <groupId>org.apache.tomcat.maven</groupId>
30                        <artifactId>tomcat7-maven-plugin</artifactId>
31                        <version>2.2</version>
32                        <configuration>
33                            <port>80</port>
34                        </configuration>
35                    </plugin>
36                </plugins>
37            </build>
38        </project>

```

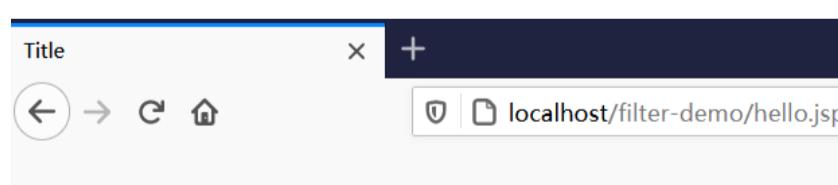
`hello.jsp` 页面内容如下：

```

1 <%@ page contentType="text/html; charset=UTF-8" language="java" %>
2 <html>
3 <head>
4     <title>Title</title>
5 </head>
6 <body>
7     <h1>hello JSP~</h1>
8 </body>
9 </html>

```

我们现在在浏览器输入 `http://localhost/filter-demo/hello.jsp` 访问 `hello.jsp` 页面，这里是可以访问到 `hello.jsp` 页面内容的。



hello JSP~

接下来编写过滤器。过滤器是 Web 三大组件之一，所以我们将 `filter` 创建在 `com.itheima.web.filter` 包下，起名为 `FilterDemo`

```

1 @WebFilter("/*")
2 public class FilterDemo implements Filter {
3
4     @Override

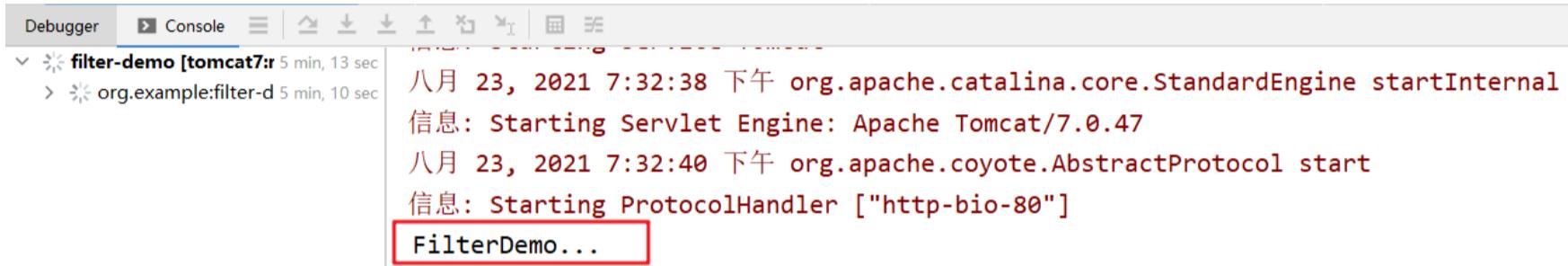
```

```
5     public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
6         throws IOException, ServletException {
7             System.out.println("FilterDemo...");
```

8 }

```
9     @Override
10    public void init(FilterConfig filterConfig) throws ServletException {
11    }
12
13    @Override
14    public void destroy() {
15    }
16}
```

重启启动服务器，再次重新访问 `hello.jsp` 页面，这次发现页面没有任何效果，但是在 `idea` 的控制台可以看到如下内容



上述效果说明 `FilterDemo` 这个过滤器的 `doFilter()` 方法执行了，但是为什么在浏览器上看不到 `hello.jsp` 页面的内容呢？这是因为在 `doFilter()` 方法中添加放行的方法才能访问到 `hello.jsp` 页面。那就在 `doFilter()` 方法中添加放行的代码

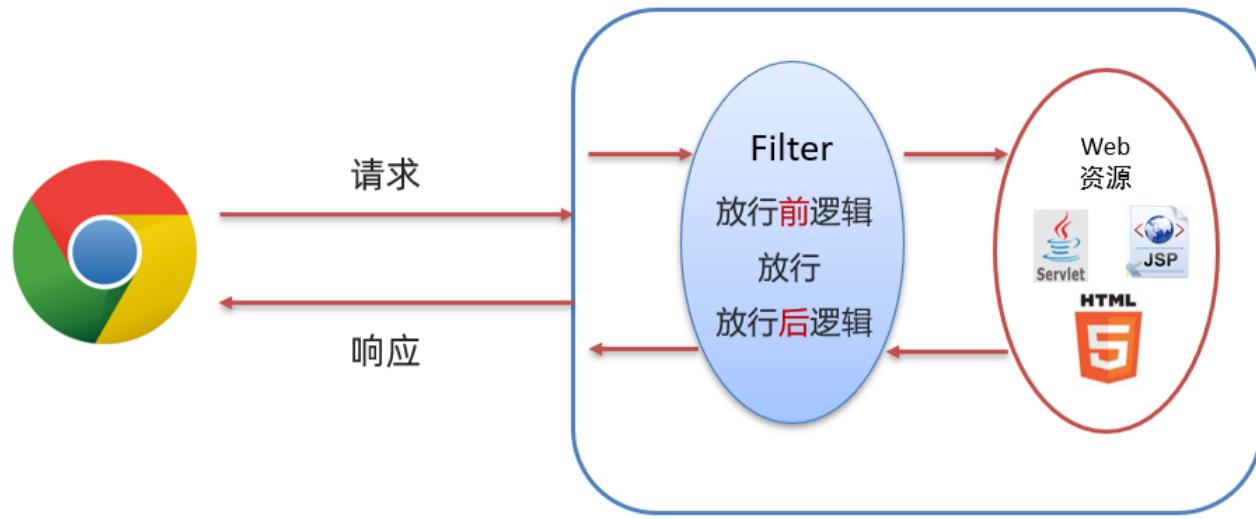
```
1 //放行  
2 chain.doFilter(request, response);
```

再次重启服务器并访问 `hello.jsp` 页面，发现这次就可以在浏览器上看到页面效果。

FilterDemo 过滤器完整代码如下：

```
1 @WebFilter("/*")
2 public class FilterDemo implements Filter {
3
4     @Override
5     public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
6 throws IOException, ServletException {
7         System.out.println("1.FilterDemo...");  
//放行
8         chain.doFilter(request, response);
9     }
10
11    @Override
12    public void init(FilterConfig filterConfig) throws ServletException {
13    }
14
15    @Override
16    public void destroy() {
17    }
18 }
```

1.3 Filter执行流程



如上图是使用过滤器的流程，我们通过以下问题来研究过滤器的执行流程：

- 放行后访问对应资源，资源访问完成后，还会回到Filter中吗？

从上图就可以看出肯定 **会** 回到Filter中

- 如果回到Filter中，是重头执行还是执行放行后的逻辑呢？

如果是重头执行的话，就意味着 **放行前逻辑** 会被执行两次，肯定不会这样设计了；所以访问完资源后，会回到 **放行后逻辑**，执行该部分代码。

通过上述的说明，我们就可以总结Filter的执行流程如下：



接下来我们通过代码验证一下，在 `doFilter()` 方法前后都加上输出语句，如下

```
@Override
public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)

    //1. 放行前，对 request 数据进行处理
    System.out.println("1.FilterDemo...");
    //放行
    chain.doFilter(request, response);
    //2. 放行后，对 Response 数据进行处理
    System.out.println("3.FilterDemo...");

}
```

同时在 `hello.jsp` 页面加上输出语句，如下

```
<body>
    <h1>hello JSP~</h1>
    <%>
        System.out.println("2.hello.jsp");
    <%>
</body>
```

执行访问该资源打印的顺序是按照我们标记的标号进行打印的话，说明我们上边总结出来的流程是没有问题的。启动服务器访问 `hello.jsp` 页面，在控制台打印的内容如下：

```
八月 23, 2021 8:01:32 下午 org.apache.coyote.AbstractProtocol start
信息: Starting ProtocolHandler ["http-bio-80"]
1.FilterDemo...
2.hello.jsp
3.FilterDemo...
```

以后我们可以将对请求进行处理的代码放在放行之前进行处理，而如果请求完资源后还要对响应的数据进行处理时可以在放行后进行逻辑处理。

1.4 Filter拦截路径配置

拦截路径表示 Filter 会对请求的哪些资源进行拦截，使用 `@webFilter` 注解进行配置。如：`@webFilter("拦截路径")`

拦截路径有如下四种配置方式：

- 拦截具体的资源：`/index.jsp`：只有访问`index.jsp`时才会被拦截

- 目录拦截：/user/*：访问/user下的所有资源，都会被拦截
- 后缀名拦截：*.jsp：访问后缀名为jsp的资源，都会被拦截
- 拦截所有：/*：访问所有资源，都会被拦截

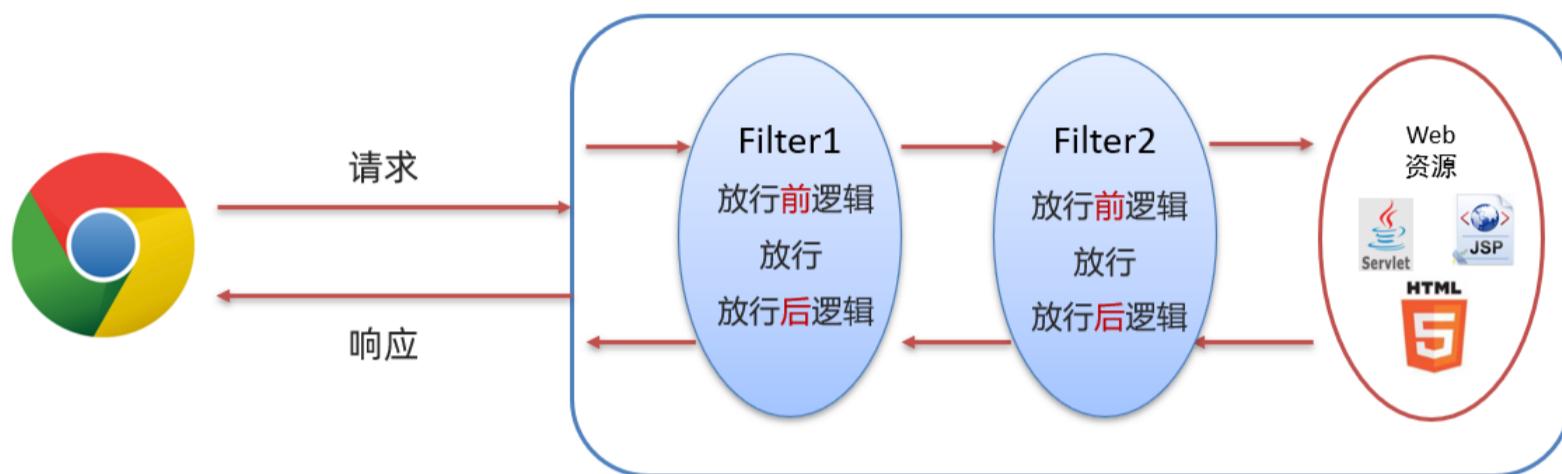
通过上面拦截路径的学习，大家会发现拦截路径的配置方式和 `Servlet` 的请求资源路径配置方式一样，但是表示的含义不同。

1.5 过滤器链

1.5.1 概述

过滤器链是指在一个Web应用，可以配置多个过滤器，这多个过滤器称为过滤器链。

如下图就是一个过滤器链，我们学习过滤器链主要是学习过滤器链执行的流程



上图中的过滤器链执行是按照以下流程执行：

1. 执行 `Filter1` 的放行前逻辑代码
2. 执行 `Filter1` 的放行代码
3. 执行 `Filter2` 的放行前逻辑代码
4. 执行 `Filter2` 的放行代码
5. 访问到资源
6. 执行 `Filter2` 的放行后逻辑代码
7. 执行 `Filter1` 的放行后逻辑代码

以上流程串起来就像一条链子，故称之为过滤器链。

1.5.2 代码演示

- 编写第一个过滤器 `FilterDemo`，配置成拦截所有资源

```

1  @webFilter("/*")
2  public class FilterDemo implements Filter {
3
4      @Override
5      public void doFilter(ServletRequest request, ServletResponse response, FilterChain
chain) throws IOException, ServletException {
6
7          //1. 放行前，对 request 数据进行处理
8          System.out.println("1.FilterDemo...");
9          //放行
10         chain.doFilter(request, response);
11         //2. 放行后，对 Response 数据进行处理
12         System.out.println("3.FilterDemo...");
13     }
14
15     @Override
16     public void init(FilterConfig filterConfig) throws ServletException {
17     }
18
19     @Override
20     public void destroy() {
21     }
22 }
```

- 编写第二个过滤器 `FilterDemo2`，配置炒年糕拦截所有资源

```

1  @webFilter("/*")
2  public class FilterDemo2 implements Filter {
3
4      @Override
5      public void doFilter(ServletRequest request, ServletResponse response, FilterChain
chain) throws IOException, ServletException {
6
7          //1. 放行前，对 request 数据进行处理
8          System.out.println("2.FilterDemo...");
9          //放行
10         chain.doFilter(request, response);
11         //2. 放行后，对 Response 数据进行处理
12         System.out.println("4.FilterDemo...");
13     }
14
15     @Override
16     public void init(FilterConfig filterConfig) throws ServletException {
17     }
18
19     @Override
20     public void destroy() {
21     }
22 }
23

```

- 修改 `hello.jsp` 页面中脚本的输出语句

```

1  <%@ page contentType="text/html; charset=UTF-8" language="java" %>
2  <html>
3  <head>
4      <title>Title</title>
5  </head>
6  <body>
7      <h1>hello JSP~</h1>
8      <%
9          System.out.println("3.hello jsp");
10     %>
11 </body>
12 </html>

```

- 启动服务器，在浏览器输入 `http://localhost/filter-demo/hello.jsp` 进行测试，在控制台打印内容如下

```

八月 23, 2021 10:11:36 下午 org.apache.coyote.AbstractProtocol start
信息: Starting ProtocolHandler ["http-bio-80"]
1.FilterDemo...
2.FilterDemo...
3.hello jsp
4.FilterDemo...
5.FilterDemo...

```

从结果可以看到确实是按照我们之前说的执行流程进行执行的。

1.5.3 问题

上面代码中为什么是先执行 `FilterDemo`，后执行 `FilterDemo2` 呢？

我们现在使用的是注解配置Filter，而这种配置方式的优先级是按照过滤器类名(字符串)的自然排序。

比如有如下两个名称的过滤器：`BFilterDemo` 和 `AFilterDemo`。那一定是 `AFilterDemo` 过滤器先执行。

1.6 案例

1.6.1 需求

访问服务器资源时，需要先进行登录验证，如果没有登录，则自动跳转到登录页面

1.6.2 分析

我们要实现该功能是在每一个资源里加入登陆状态校验的代码吗？显然是不需要的，只需要写一个 `Filter`，在该过滤器中进行登陆状态校验即可。而在该 `Filter` 中逻辑如下：



1.6.3 代码实现

1.6.3.1 创建Filter

在 `brand-demo` 工程创建 `com.itheima.web.filter` 包，在该下创建名为 `LoginFilter` 的过滤器

```
1 @WebFilter("/*")
2 public class LoginFilter implements Filter {
3     @Override
4     public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
5         throws ServletException, IOException {
6
7
8         public void init(FilterConfig config) throws ServletException {
9
10
11     public void destroy() {
12
13 }
```

1.6.3.2 编写逻辑代码

在 `doFilter()` 方法中编写登陆状态校验的逻辑代码。

我们首先需要从 `session` 对象中获取用户信息，但是 `ServletRequest` 类型的 `request` 对象没有获取 `session` 对象的方法，所以此时需要将 `request` 对象强转成 `HttpServletRequest` 对象。

```
1 HttpServletRequest req = (HttpServletRequest) request;
```

然后完成以下逻辑

- 获取 `Session` 对象
- 从 `Session` 对象中获取名为 `user` 的数据
- 判断获取到的数据是否是 `null`
 - 如果不是，说明已经登陆，放行
 - 如果是，说明尚未登陆，将提示信息存储到域对象中并跳转到登陆页面

代码如下：

```
1 @WebFilter("/*")
2 public class LoginFilter implements Filter {
3     @Override
```

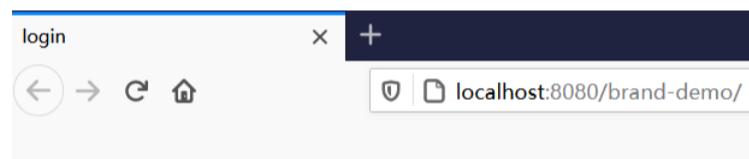
```

4     public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
5         throws ServletException, IOException {
6             HttpServletRequest req = (HttpServletRequest) request;
7
8             //1. 判断session中是否有user
9             HttpSession session = req.getSession();
10            Object user = session.getAttribute("user");
11
12            if(user != null){
13                // 登录过了
14                //放行
15                chain.doFilter(request, response);
16            }else {
17                // 没有登陆, 存储提示信息, 跳转到登录页面
18
19                req.setAttribute("login_msg", "您尚未登陆!");
20                req.getRequestDispatcher("/login.jsp").forward(req, response);
21            }
22        }
23
24    public void init(FilterConfig config) throws ServletException {
25    }
26
27    public void destroy() {
28    }
29 }

```

1.6.3.3 测试并抛出问题

在浏览器上输入 `http://localhost:8080/brand-demo/`, 可以看到如下页面效果



LOGIN IN

您尚未登陆!

Username:

Password:

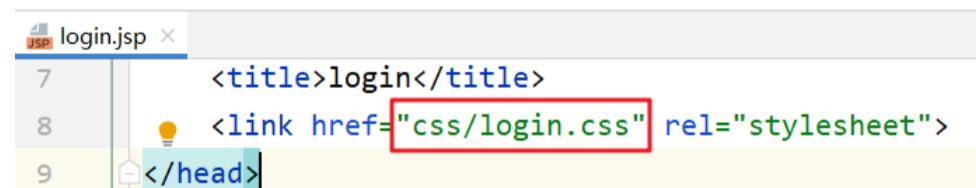
Remember:

[login up](#) [reset](#) [没有账号?](#)

从上面效果可以看出没有登陆确实是跳转到登陆页面了，但是登陆页面为什么展示成这种效果了呢？

1.6.3.4 问题分析及解决

因为登陆页面需要 `css/login.css` 这个文件进行样式的渲染，下图是登陆页面引入的css文件图解



而在请求这个css资源时被过滤器拦截，就相当于没有加载到样式文件导致的。解决这个问题，只需要对所有的登陆相关的资源进行放行即可。还有一种情况就是当我没有用户信息时需要进行注册，而注册时也希望被过滤器放行。

综上，我们需要在判断session中是否包含用户信息之前，应该加上对登陆及注册相关资源放行的逻辑处理

```

1 //判断访问资源路径是否和登录注册相关
2 //1,在数组中存储登陆和注册相关的资源路径
3 String[] urls =
4     {"/login.jsp", "/imgs/", "/css/", "/loginServlet", "/register.jsp", "/registerServlet", "/checkCodes
5     ervlet"};
6 //2,获取当前访问的资源路径
7 String url = req.getRequestURL().toString();

```

```

6
7 //3, 遍历数组, 获取到每一个需要放行的资源路径
8 for (String u : urls) {
9     //4, 判断当前访问的资源路径字符串是否包含要放行的的资源路径字符串
10    /*
11        比如当前访问的资源路径是 /brand-demo/login.jsp
12        而字符串 /brand-demo/login.jsp 包含了 字符串 /login.jsp , 所以这个字符串就需要放行
13    */
14    if(url.contains(u)){
15        //找到了, 放行
16        chain.doFilter(request, response);
17        //break;
18        return;
19    }
20 }

```

1.6.3.5 过滤器完整代码

```

1 @WebFilter("/*")
2 public class LoginFilter implements Filter {
3     @Override
4     public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
5         throws ServletException, IOException {
6         HttpServletRequest req = (HttpServletRequest) request;
7
8         //判断访问资源路径是否和登录注册相关
9         //1, 在数组中存储登陆和注册相关的资源路径
10        String[] urls =
11            {"/login.jsp", "/imgs/", "/css/", "/loginServlet", "/register.jsp", "/registerServlet", "/checkCodes
12        ervlet"};
13
14        //2, 获取当前访问的资源路径
15        String url = req.getRequestURL().toString();
16
17        //3, 遍历数组, 获取到每一个需要放行的资源路径
18        for (String u : urls) {
19            //4, 判断当前访问的资源路径字符串是否包含要放行的的资源路径字符串
20            /*
21                比如当前访问的资源路径是 /brand-demo/login.jsp
22                而字符串 /brand-demo/login.jsp 包含了 字符串 /login.jsp , 所以这个字符串就需要放行
23            */
24            if(url.contains(u)){
25                //找到了, 放行
26                chain.doFilter(request, response);
27                //break;
28                return;
29            }
30        }
31
32        //1. 判断session中是否有user
33        HttpSession session = req.getSession();
34        Object user = session.getAttribute("user");
35
36        //2. 判断user是否为null
37        if(user != null){
38            // 登录过了
39            //放行
40            chain.doFilter(request, response);
41        }else {
42            // 没有登陆, 存储提示信息, 跳转到登录页面
43
44            req.setAttribute("login_msg", "您尚未登陆!");
45            req.getRequestDispatcher("/login.jsp").forward(req, response);
46        }
47    }
48 }

```

```

45     public void init(FilterConfig config) throws ServletException {
46 }
47
48     public void destroy() {
49 }
50 }
```

2, Listener

2.1 概述

- Listener 表示监听器，是 JavaWeb 三大组件(Servlet、Filter、Listener)之一。
- 监听器可以监听就是在 `application`, `session`, `request` 三个对象创建、销毁或者往其中添加修改删除属性时自动执行代码的功能组件。

`request` 和 `session` 我们学习过。而 `application` 是 `ServletContext` 类型的对象。

`ServletContext` 代表整个 web 应用，在服务器启动的时候，tomcat 会自动创建该对象。在服务器关闭时会自动销毁该对象。

2.2 分类

JavaWeb 提供了 8 个监听器：

监听器分类	监听器名称	作用
ServletContext 监听	ServletContextListener	用于对 ServletContext 对象进行监听（创建、销毁）
	ServletContextAttributeListener	对 ServletContext 对象中属性的监听（增删改属性）
Session 监听	HttpSessionListener	对 Session 对象的整体状态的监听（创建、销毁）
	HttpSessionAttributeListener	对 Session 对象中的属性监听（增删改属性）
Request 监听	HttpSessionBindingListener	监听对象于 Session 的绑定和解除
	HttpSessionActivationListener	对 Session 数据的钝化和活化的监听
Request 监听	ServletRequestListener	对 Request 对象进行监听（创建、销毁）
	ServletRequestAttributeListener	对 Request 对象中属性的监听（增删改属性）

这里面只有 `ServletContextListener` 这个监听器后期我们会接触到，`ServletContextListener` 是用来监听 `ServletContext` 对象的创建和销毁。

`ServletContextListener` 接口中有以下两个方法

- `void contextInitialized(ServletContextEvent sce)`: `ServletContext` 对象被创建了会自动执行的方法
- `void contextDestroyed(ServletContextEvent sce)`: `ServletContext` 对象被销毁时会自动执行的方法

2.3 代码演示

我们只演示一下 `ServletContextListener` 监听器

- 定义一个类，实现 `ServletContextListener` 接口
- 重写所有的抽象方法
- 使用 `@WebListener` 进行配置

代码如下：

```

1  @webListener
2  public class ContextLoaderListener implements ServletContextListener {
3      @Override
4      public void contextInitialized(ServletContextEvent sce) {
5          //加载资源
6          System.out.println("ContextLoaderListener...");
7      }
8
9      @Override
10     public void contextDestroyed(ServletContextEvent sce) {
11         //释放资源
12     }
13 }

```

启动服务器，就可以在启动的日志信息中看到 `contextInitialized()` 方法输出的内容，同时也说明了 `ServletContext` 对象在服务器启动的时候被创建了。

3, Ajax

3.1 概述

AJAX (Asynchronous JavaScript And XML): 异步的 JavaScript 和 XML.

我们先来说概念中的 `JavaScript` 和 `XML`，`JavaScript` 表明该技术和前端相关；`XML` 是指以此进行数据交换。而这两个我们之前都学习过。

3.1.1 作用

AJAX 作用有以下两方面：

- 与服务器进行数据交换：通过AJAX可以给服务器发送请求，服务器将数据直接响应回给浏览器。如下图

我们先来看之前做功能的流程，如下图：



如上图，`Servlet` 调用完业务逻辑层后将数据存储到域对象中，然后跳转到指定的 `jsp` 页面，在页面上使用 `EL表达式` 和 `JSTL` 标签库进行数据的展示。

而我们学习了AJAX 后，就可以**使用AJAX和服务器进行通信，以达到使用 HTML+AJAX来替换JSP页面**了。如下图，浏览器发送请求servlet，servlet 调用完业务逻辑层后将数据直接响应回给浏览器页面，页面使用 HTML 来进行数据展示。



- 异步交互：可以在**不重新加载整个页面**的情况下，与服务器交换数据并**更新部分网页**的技术，如：搜索联想、用户名是否可用校验，等等...



奥运

奥运会羽毛球男子双打决赛
奥运金牌榜
奥运会直播
奥运会
奥运赛程

上图所示的效果我们经常见到，在我们输入一些关键字（例如 奥运）后就会在下面联想起相关的内容，而联想起的这部分数据肯定是存储在百度的服务器上，而我们并没有看出页面重新刷新，这就是 **更新局部页面** 的效果。再如下图：

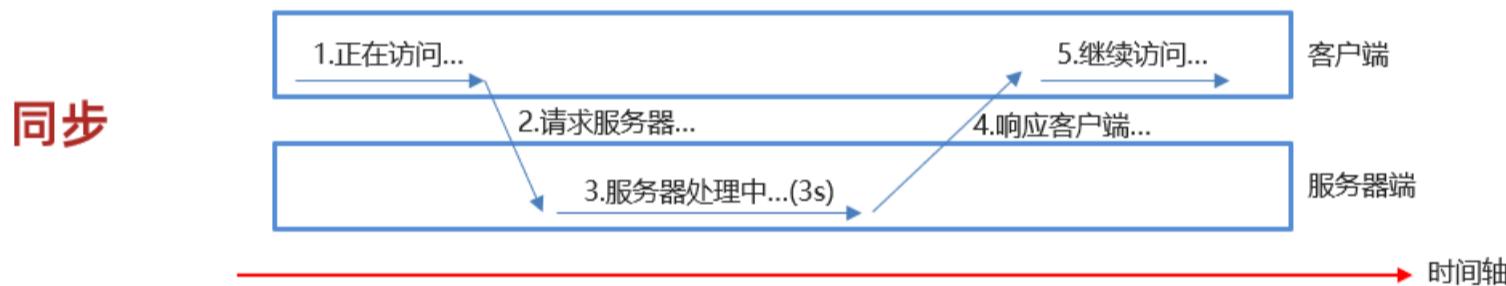


我们在用户名的输入框输入用户名，当输入框一失去焦点，如果用户名已经被占用就会在下方展示提示的信息；在这整个过程中也没有页面的刷新，只是在局部展示出了提示信息，这就是 **更新局部页面** 的效果。

3.1.2 同步和异步

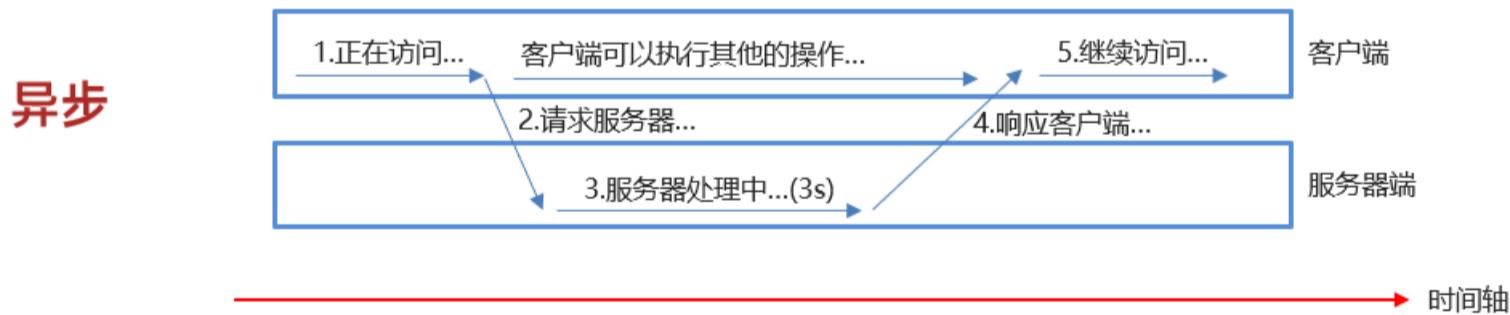
知道了局部刷新后，接下来我们再聊聊同步和异步：

- 同步发送请求过程如下



浏览器页面在发送请求给服务器，在服务器处理请求的过程中，浏览器页面不能做其他的操作。只能等到服务器响应结束后才能，浏览器页面才能继续做其他的操作。

- 异步发送请求过程如下



浏览器页面发送请求给服务器，在服务器处理请求的过程中，浏览器页面还可以做其他的操作。

3.2 快速入门

3.2.1 服务端实现

在项目的创建 `com.itheima.web.servlet`，并在该包下创建名为 `AjaxServlet` 的servlet

```

1 @webServlet("/ajaxServlet")
2 public class AjaxServlet extends HttpServlet {
3     @Override
4     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
5         ServletException, IOException {
6         //1. 响应数据
7         response.getWriter().write("hello ajax~");
8     }
9     @Override
10    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
11        ServletException, IOException {
12        this.doGet(request, response);
13    }

```

3.2.2 客户端实现

在 `webapp` 下创建名为 `01-ajax-demo1.html` 的页面，在该页面书写 `ajax` 代码

- 创建核心对象，不同的浏览器创建的对象是不同的

```

1 var xhttp;
2 if (window.XMLHttpRequest) {
3     xhttp = new XMLHttpRequest();
4 } else {
5     // code for IE6, IE5
6     xhttp = new ActiveXObject("Microsoft.XMLHTTP");
7 }

```

- 发送请求

```

1 //建立连接
2 xhttp.open("GET", "http://localhost:8080/ajax-demo/ajaxServlet");
3 //发送请求
4 xhttp.send();

```

- 获取响应

```

1 xhttp.onreadystatechange = function() {
2     if (this.readyState == 4 && this.status == 200) {
3         // 通过 this.responseText 可以获取到服务端响应的数据
4         alert(this.responseText);
5     }
6 };

```

完整代码如下：

```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <title>Title</title>
6 </head>
7 <body>
8
9 <script>
10    //1. 创建核心对象
11    var xhttp;
12    if (window.XMLHttpRequest) {
13        xhttp = new XMLHttpRequest();
14    } else {
15        // code for IE6, IE5
16        xhttp = new ActiveXObject("Microsoft.XMLHTTP");
17    }

```

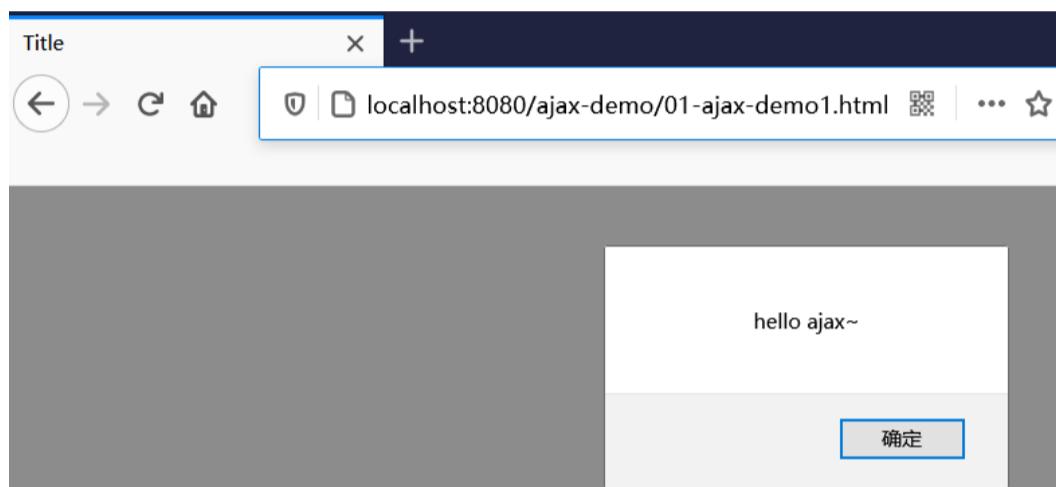
```

18     //2. 发送请求
19     xhttp.open("GET", "http://localhost:8080/ajax-demo/ajaxServlet");
20     xhttp.send();
21
22     //3. 获取响应
23     xhttp.onreadystatechange = function() {
24         if (this.readyState == 4 && this.status == 200) {
25             alert(this.responseText);
26         }
27     };
28 </script>
29 </body>
30 </html>

```

3.2.3 测试

在浏览器地址栏输入 `http://localhost:8080/ajax-demo/01-ajax-demo1.html`，在 `01-ajax-demo1.html` 加载的时候就会发送 `ajax` 请求，效果如下



我们可以通过 `开发者模式` 查看发送的 `AJAX` 请求。在浏览器上按 `F12` 快捷键

Name	Status	Type	Initiator
01-ajax-demo1.html	304	document	Other
ajaxServlet	200	xhr	01-ajax-demo1.html:22
index.js	200	script	response.js:182

这个是查看所有的请求，如果我们只是想看异步请求的话，点击上图中 `All` 旁边的 `XHR`，会发现只展示 `Type` 是 `xhr` 的请求。如下图：

Name	Status	Type	Initiator
ajaxServlet	200	xhr	01-ajax-demo1.html:22

3.3 案例

需求：在完成用户注册时，当用户名输入框失去焦点时，校验用户名是否在数据库已存在



3.3.1 分析

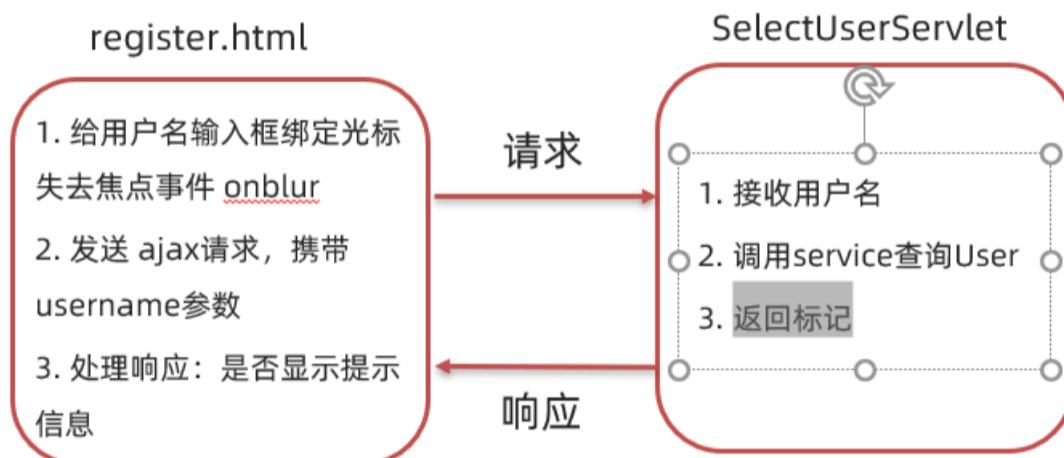
- 前端完成的逻辑

1. 给用户名输入框绑定光标失去焦点事件 `onblur`
2. 发送 ajax 请求，携带 `username` 参数
3. 处理响应：是否显示提示信息

- 后端完成的逻辑

1. 接收用户名
2. 调用 service 查询 User。此案例是为了演示前后端异步交互，所以此处我们不做业务逻辑处理
3. 返回标记

整体流程如下：



3.3.2 后端实现

在 `com.ithiema.web.servlet` 包中定义名为 `SelectUserServlet` 的 servlet。代码如下：

```

1  @WebServlet("/selectUserServlet")
2  public class SelectUserServlet extends HttpServlet {
3      @Override
4      protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
5          //1. 接收用户名
6          String username = request.getParameter("username");
7          //2. 调用service查询User对象，此处不进行业务逻辑处理，直接给 flag 赋值为 true，表明用户名占用
8          boolean flag = true;
9          //3. 响应标记
10         response.getWriter().write("+" + flag);
11     }
12
13     @Override
14     protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
15         this.doGet(request, response);
16     }
17 }
  
```

3.3.3 前端实现

将 04-资料\1. 验证用户名案例\1. 静态页面 下的文件整体拷贝到项目下 webapp 下。并在 register.html 页面的 body 结束标签前编写 script 标签，在该标签中实现如下逻辑

第一步：给用户名输入框绑定光标失去焦点事件 onblur

```
1 //1. 给用户名输入框绑定 失去焦点事件
2 document.getElementById("username").onblur = function () {
3
4 }
```

第二步：发送 ajax请求，携带username参数

在 第一步 绑定的匿名函数中书写发送 ajax 请求的代码

```
1 //2. 发送ajax请求
2 //2.1. 创建核心对象
3 var xhttp;
4 if (window.XMLHttpRequest) {
5     xhttp = new XMLHttpRequest();
6 } else {
7     // code for IE6, IE5
8     xhttp = new ActiveXObject("Microsoft.XMLHTTP");
9 }
10 //2.2. 发送请求
11 xhttp.open("GET", "http://localhost:8080/ajax-demo/selectUserServlet");
12 xhttp.send();
13
14 //2.3. 获得响应
15 xhttp.onreadystatechange = function() {
16     if (this.readyState == 4 && this.status == 200) {
17         //处理响应的结果
18     }
19 };
```

由于我们发送的是 GET 请求，所以需要在 URL 后拼接从输入框获取的用户名数据。而我们在 第一步 绑定的匿名函数中通过以下代码可以获取用户名数据

```
1 // 获取用户名的值
2 var username = this.value; //this : 给谁绑定的事件, this就代表谁
```

而携带数据需要将 URL 修改为：

```
1 xhttp.open("GET", "http://localhost:8080/ajax-demo/selectUserServlet?username="+username);
```

第三步：处理响应：是否显示提示信息

当 this.readyState == 4 && this.status == 200 条件满足时，说明已经成功响应数据了。

此时需要判断响应的数据是否是 "true" 字符串，如果是说明用户名已经占用给出错误提示；如果不是说明用户名未被占用清除错误提示。代码如下

```
1 //判断
2 if(this.responseText == "true"){
3     //用户名存在，显示提示信息
4     document.getElementById("username_err").style.display = '';
5 }else {
6     //用户名不存在，清楚提示信息
7     document.getElementById("username_err").style.display = 'none';
8 }
```

综上所述，前端完成代码如下：

```
1 //1. 给用户名输入框绑定 失去焦点事件
2 document.getElementById("username").onblur = function () {
```

```

3 //2. 发送ajax请求
4 // 获取用户名的值
5 var username = this.value;
6
7 //2.1. 创建核心对象
8 var xhttp;
9 if (window.XMLHttpRequest) {
10     xhttp = new XMLHttpRequest();
11 } else {
12     // code for IE6, IE5
13     xhttp = new ActiveXObject("Microsoft.XMLHTTP");
14 }
15 //2.2. 发送请求
16 xhttp.open("GET", "http://localhost:8080/ajax-demo/selectUserServlet?username="+username);
17 xhttp.send();
18
19 //2.3. 获取响应
20 xhttp.onreadystatechange = function() {
21     if (this.readyState == 4 && this.status == 200) {
22         //alert(this.responseText);
23         //判断
24         if(this.responseText == "true"){
25             //用户名存在，显示提示信息
26             document.getElementById("username_err").style.display = '';
27         }else {
28             //用户名不存在，清楚提示信息
29             document.getElementById("username_err").style.display = 'none';
30         }
31     }
32 };
33 }

```

4, axios

Axios 对原生的AJAX进行封装，简化书写。

Axios官网是: <https://www.axios-http.cn>

4.1 基本使用

axios 使用是比较简单的，分为以下两步：

- 引入 axios 的 js 文件

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

- 使用axios 发送请求，并获取响应结果

- 发送 get 请求

```

1 axios({
2     method:"get",
3     url:"http://localhost:8080/ajax-demo1/aJAXDemo1?username=zhangsan"
4 }).then(function (resp){
5     alert(resp.data);
6 })

```

- 发送 post 请求

```

1 axios({
2     method:"post",
3     url:"http://localhost:8080/ajax-demo1/aJAXDemo1",
4     data:"username=zhangsan"
5 }).then(function (resp){
6     alert(resp.data);
7 });

```

`axios()` 是用来发送异步请求的，小括号中使用 js 对象传递请求相关的参数：

- `method` 属性：用来设置请求方式的。取值为 `get` 或者 `post`。
- `url` 属性：用来书写请求的资源路径。如果是 `get` 请求，需要将请求参数拼接到路径的后面，格式为：`url?参数名=参数值&参数名2=参数值2`。
- `data` 属性：作为请求体被发送的数据。也就是说如果是 `post` 请求的话，数据需要作为 `data` 属性的值。

`then()` 需要传递一个匿名函数。我们将 `then()` 中传递的匿名函数称为 **回调函数**，意思是该匿名函数在发送请求时不会被调用，而是在成功响应后调用的函数。而该回调函数中的 `resp` 参数是对响应的数据进行封装的对象，通过 `resp.data` 可以获取到响应的数据。

4.2 快速入门

4.2.1 后端实现

定义一个用于接收请求的servlet，代码如下：

```
1 @WebServlet("/axiosServlet")
2 public class AxiosServlet extends HttpServlet {
3     @Override
4     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
5         System.out.println("get...");
6         //1. 接收请求参数
7         String username = request.getParameter("username");
8         System.out.println(username);
9         //2. 响应数据
10        response.getWriter().write("hello Axios~");
11    }
12
13    @Override
14    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
15        System.out.println("post...");
16        this.doGet(request, response);
17    }
18 }
```

4.2.2 前端实现

- 引入 js 文件

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

- 发送 ajax 请求

- get 请求

```
1 axios({
2     method:"get",
3     url:"http://localhost:8080/ajax-demo/axiosServlet?username=zhangsan"
4 }).then(function (resp) {
5     alert(resp.data);
6 })
```

- post 请求

```
1 axios({
2     method:"post",
3     url:"http://localhost:8080/ajax-demo/axiosServlet",
4     data:"username=zhangsan"
5 }).then(function (resp) {
6     alert(resp.data);
7 })
```

整体页面代码如下：

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <title>Title</title>
6 </head>
7 <body>
8
9 <script src="js/axios-0.18.0.js"></script>
10 <script>
11     //1. get
12     /* axios({
13         method:"get",
14         url:"http://localhost:8080/ajax-demo/axiosServlet?username=zhangsan"
15     }).then(function (resp) {
16         alert(resp.data);
17     })*/
18
19     //2. post 在js中{} 表示一个js对象，而这个js对象中有三个属性
20     axios({
21         method:"post",
22         url:"http://localhost:8080/ajax-demo/axiosServlet",
23         data:"username=zhangsan"
24     }).then(function (resp) {
25         alert(resp.data);
26     })
27 </script>
28 </body>
29 </html>
```

4.3 请求方法别名

为了方便起见，Axios 已经为所有支持的请求方法提供了别名。如下：

- `get` 请求：`axios.get(url[,config])`
- `delete` 请求：`axios.delete(url[,config])`
- `head` 请求：`axios.head(url[,config])`
- `options` 请求：`axios.option(url[,config])`
- `post` 请求：`axios.post(url[,data[,config]])`
- `put` 请求：`axios.put(url[,data[,config]])`
- `patch` 请求：`axios.patch(url[,data[,config]])`

而我们只关注 `get` 请求和 `post` 请求。

入门案例中的 `get` 请求代码可以改为如下：

```
1 axios.get("http://localhost:8080/ajax-demo/axiosServlet?username=zhangsan").then(function
2     (resp) {
3         alert(resp.data);
4     });
5 
```

入门案例中的 `post` 请求代码可以改为如下：

```
1 axios.post("http://localhost:8080/ajax-demo/axiosServlet","username=zhangsan").then(function
2     (resp) {
3         alert(resp.data);
4     });
5 
```

5, JSON

5.1 概述

概念：`JavaScript Object Notation`。JavaScript 对象表示法。

如下是 `JavaScript` 对象的定义格式：

```
1 {  
2     name:"zhangsan",  
3     age:23,  
4     city:"北京"  
5 }
```

接下来我们再看看 `JSON` 的格式：

```
1 {  
2     "name":"zhangsan",  
3     "age":23,  
4     "city":"北京"  
5 }
```

通过上面 `js` 对象格式和 `json` 格式进行对比，发现两个格式特别像。只不过 `js` 对象中的属性名可以使用引号（可以是单引号，也可以是双引号）；而 `json` 格式中的键要求必须使用双引号括起来，这是 `json` 格式的规定。`json` 格式的数据有什么作用呢？

作用：由于其语法格式简单，层次结构鲜明，现多用于作为**数据载体**，在网络中进行数据传输。如下图所示就是服务端给浏览器响应的数据，这个数据比较简单，如果现需要将 `JAVA` 对象中封装的数据响应回给浏览器的话，应该以何种数据传输呢？

```
//2. 响应数据  
response.getWriter().write( s: "hello Axios~");
```

大家还记得 `ajax` 的概念吗？是**异步的 JavaScript 和 XML**。这里的 `xml` 就是以前进行数据传递的方式，如下：

```
1 <student>  
2     <name>张三</name>  
3     <age>23</age>  
4     <city>北京</city>  
5 </student>
```

再看 `json` 描述以上数据的写法：

```
1 {  
2     "name":"张三",  
3     "age":23,  
4     "city":"北京"  
5 }
```

上面两种格式进行对比后就会发现 `json` 格式数据的简单，以及所占的字节数少等优点。

5.2 JSON 基础语法

5.2.1 定义格式

`JSON` 本质就是一个字符串，但是该字符串内容是有一定的格式要求的。定义格式如下：

```
1 var 变量名 = '{"key":value,"key":value,...}';
```

`JSON` 串的键要求必须使用双引号括起来，而值根据要表示的类型确定。`value` 的数据类型分为如下

- 数字（整数或浮点数）
- 字符串（使用双引号括起来）
- 逻辑值（`true`或者`false`）
- 数组（在方括号中）
- 对象（在花括号中）
- `null`

示例：

```
1 | var jsonStr = '{"name":"zhangsan", "age":23, "addr":["北京", "上海", "西安"]}'
```

5.2.2 代码演示

创建一个页面，在该页面的 `<script>` 标签中定义json字符串

```
1 | <!DOCTYPE html>
2 | <html lang="en">
3 | <head>
4 |   <meta charset="UTF-8">
5 |   <title>Title</title>
6 | </head>
7 | <body>
8 | <script>
9 |   //1. 定义JSON字符串
10 |   var jsonStr = '{"name":"zhangsan", "age":23, "addr":["北京", "上海", "西安"]}'
11 |   alert(jsonStr);
12 |
13 | </script>
14 | </body>
15 | </html>
```

通过浏览器打开，页面效果如下图所示



现在我们需要获取到该 `JSON` 串中的 `name` 属性值，应该怎么处理呢？

如果它是一个 js 对象，我们就可以通过 `js对象.属性名` 的方式来获取数据。JS 提供了一个对象 `JSON`，该对象有如下两个方法：

- `parse(str)`：将 JSON 串转换为 js 对象。使用方式是：`var jsObject = JSON.parse(jsonStr);`
- `stringify(obj)`：将 js 对象转换为 JSON 串。使用方式是：`var jsonStr = JSON.stringify(jsObject)`

代码演示：

```
1 | <!DOCTYPE html>
2 | <html lang="en">
3 | <head>
4 |   <meta charset="UTF-8">
5 |   <title>Title</title>
6 | </head>
7 | <body>
8 | <script>
9 |   //1. 定义JSON字符串
10 |   var jsonStr = '{"name":"zhangsan", "age":23, "addr":["北京", "上海", "西安"]}'
11 |   alert(jsonStr);
12 |
13 |   //2. 将 JSON 字符串转为 JS 对象
14 |   let jsObject = JSON.parse(jsonStr);
15 |   alert(jsObject)
16 |   alert(jsObject.name)
17 |   //3. 将 JS 对象转换为 JSON 字符串
18 |   let jsonStr2 = JSON.stringify(jsObject);
19 |   alert(jsonStr2)
20 | </script>
21 | </body>
22 | </html>
```

5.2.3 发送异步请求携带参数

后面我们使用 `axios` 发送请求时，如果要携带复杂的数据时都会以 `JSON` 格式进行传递，如下

```
1 axios({
2     method:"post",
3     url:"http://localhost:8080/ajax-demo/axiosServlet",
4     data:"username=zhangsan"
5 }).then(function (resp) {
6     alert(resp.data);
7 })
```

请求参数不可能由我们自己拼接字符串吧？肯定不用，可以提前定义一个 `js` 对象，用来封装需要提交的参数，然后使用 `JSON.stringify(js对象)` 转换为 `JSON` 串，再将该 `JSON` 串作为 `axios` 的 `data` 属性值进行请求参数的提交。如下：

```
1 var jsObject = {name:"张三"};
2
3 axios({
4     method:"post",
5     url:"http://localhost:8080/ajax-demo/axiosServlet",
6     data: JSON.stringify(jsObject)
7 }).then(function (resp) {
8     alert(resp.data);
9 })
```

而 `axios` 是一个很强大的工具。我们只需要将需要提交的参数封装成 `js` 对象，并将该 `js` 对象作为 `axios` 的 `data` 属性值进行，它会自动将 `js` 对象转换为 `JSON` 串进行提交。如下：

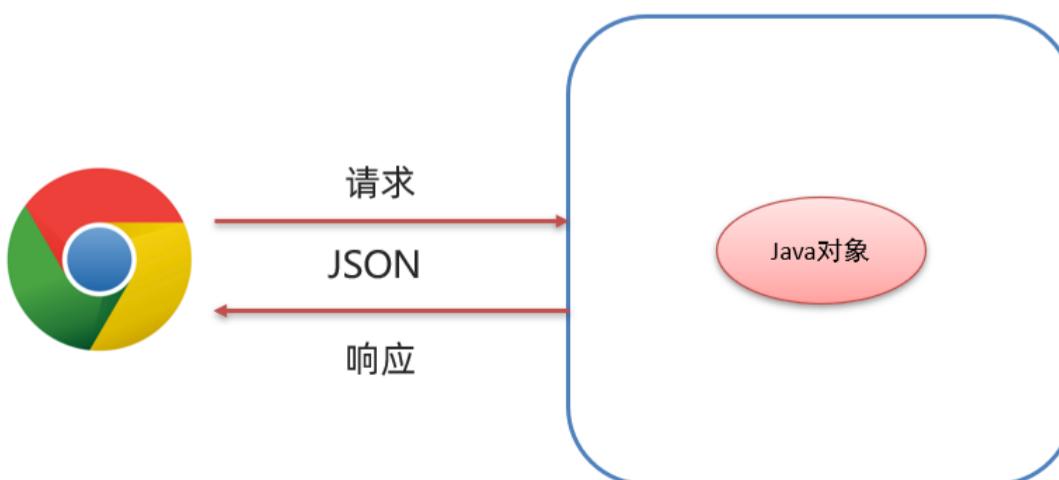
```
1 var jsObject = {name:"张三"};
2
3 axios({
4     method:"post",
5     url:"http://localhost:8080/ajax-demo/axiosServlet",
6     data:jsObject //这里 axios 会将该js对象转换为 json 串的
7 }).then(function (resp) {
8     alert(resp.data);
9 })
```

注意：

- `js` 提供的 `JSON` 对象我们只需要了解一下即可。因为 `axios` 会自动对 `js` 对象和 `JSON` 串进行想换转换。
- 发送异步请求时，如果请求参数是 `JSON` 格式，那请求方式必须是 `POST`。因为 `JSON` 串需要放在请求体中。

5.3 JSON串和Java对象的相互转换

学习完 `json` 后，接下来聊聊 `json` 的作用。以后我们会以 `json` 格式的数据进行前后端交互。前端发送请求时，如果是复杂的数据就会以 `json` 提交给后端；而后端如果需要响应一些复杂的数据时，也需要以 `json` 格式将数据响应给浏览器。



在后端我们就需要重点学习以下两部分操作：

- 请求数据：`JSON`字符串转为`Java`对象
- 响应数据：`Java`对象转为`JSON`字符串

接下来给大家介绍一套 API，可以实现上面两部分操作。这套 API 就是 `Fastjson`

5.3.1 Fastjson 概述

`Fastjson` 是阿里巴巴提供的一个 Java 语言编写的高性能功能完善的 `JSON` 库，是目前 Java 语言中最快的 `JSON` 库，可以实现 `Java` 对象和 `JSON` 字符串的相互转换。

5.3.2 Fastjson 使用

`Fastjson` 使用也是比较简单的，分为以下三步完成

1. 导入坐标

```
1 <dependency>
2   <groupId>com.alibaba</groupId>
3   <artifactId>fastjson</artifactId>
4   <version>1.2.62</version>
5 </dependency>
```

2. Java对象转JSON

```
1 String jsonStr = JSON.toJSONString(obj);
```

将 Java 对象转换为 JSON 串，只需要使用 `Fastjson` 提供的 `JSON` 类中的 `toJSONString()` 静态方法即可。

3. JSON字符串转Java对象

```
1 User user = JSON.parseObject(jsonStr, User.class);
```

将 json 转换为 Java 对象，只需要使用 `Fastjson` 提供的 `JSON` 类中的 `parseObject()` 静态方法即可。

5.3.3 代码演示

- 引入坐标
- 创建一个类，专门用来测试 Java 对象和 JSON 串的相互转换，代码如下：

```
1 public class FastJsonDemo {
2
3     public static void main(String[] args) {
4         //1. 将Java对象转为JSON字符串
5         User user = new User();
6         user.setId(1);
7         user.setUsername("zhangsan");
8         user.setPassword("123");
9
10        String jsonString = JSON.toJSONString(user);
11        System.out.println(jsonString);//{"id":1,"password":"123","username":"zhangsan"}
12
13
14        //2. 将JSON字符串转为Java对象
15        User u = JSON.parseObject(
16            "{\"id\":1,\"password\":\"123\",\"username\":\"zhangsan\"}", User.class);
17        System.out.println(u);
18    }
19}
```

6, 案例

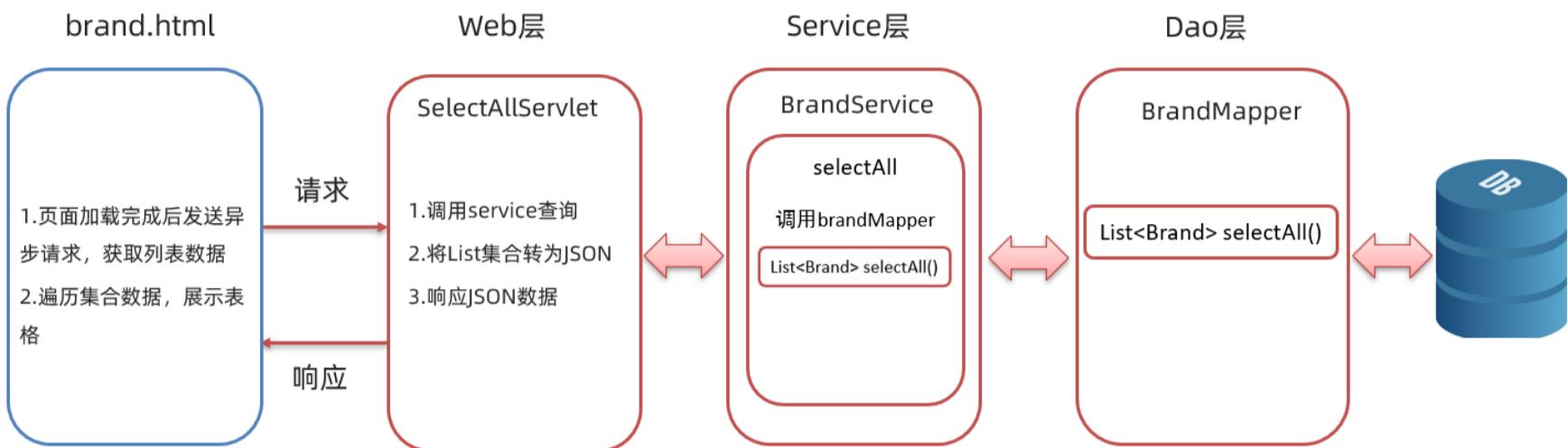
6.1 需求

使用 Axios + JSON 完成品牌列表数据查询和添加。页面效果还是下图所示：

新增

序号	品牌名称	企业名称	排序	品牌介绍	状态	操作
1	三只松鼠	三只松鼠	100	三只松鼠，好吃不上火	启用	修改 删除
2	优衣库	优衣库	10	优衣库，服适人生	禁用	修改 删除
3	小米	小米科技有限公司	1000	为发烧而生	启用	修改 删除

6.2 查询所有功能

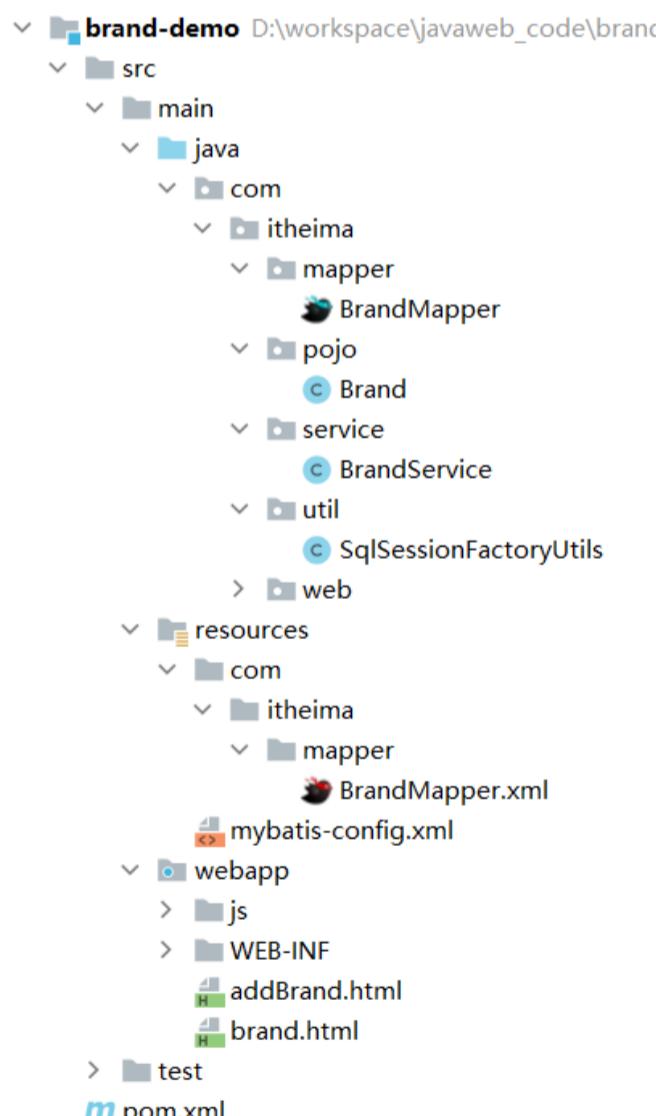


如上图所示就该功能的整体流程。前后端需以 JSON 格式进行数据的传递；由于此功能是查询所有的功能，前端发送 ajax 请求不需要携带参数，而后端响应数据需以如下格式的 json 数据

```
[{"brandName": "三只松鼠", "companyName": "三只松鼠股份有限公司", "description": "好吃不上火", "id": 1, "ordered": 5, "status": 0, "statusStr": "禁用"}, {"brandName": "华为", "companyName": "华为技术有限公司", "description": "华为致力于把数字世界带入每个人、每个家庭、每个组织，构建万物互联的智能世界", "id": 2, "ordered": 100, "status": 1, "statusStr": "启用"}, {"brandName": "小米", "companyName": "小米科技有限公司", "description": "are you ok", "id": 3, "ordered": 50, "status": 1, "statusStr": "启用"}, {"brandName": "鸿星尔克", "companyName": "鸿星尔克", "description": "to be no 1", "id": 6, "ordered": 10, "status": 0, "statusStr": "禁用"}, {"brandName": "黑马", "companyName": "黑马", "description": "发送到发大水", "id": 7, "ordered": 100, "status": 1, "statusStr": "启用"}]
```

6.2.1 环境准备

将 02-AJAX\04-资料\3. 品牌列表案例\初始工程 下的 `brand-demo` 工程拷贝到我们自己 `工作空间`，然后再将项目导入到我们自己的 Idea 中。工程目录结构如下：



注意：

- 在给定的原始工程中已经给定一些代码。而在此案例中我们只关注前后端交互代码实现
- 要根据自己的数据库环境去修改连接数据库的信息，在 `mybatis-config.xml` 核心配置文件中修改

6.2.2 后端实现

在 `com.itheima.web` 包下创建名为 `SelectAllServlet` 的 `servlet`，具体的逻辑如下：

- 调用 service 的 `selectAll()` 方法进行查询所有的逻辑处理
- 将查询到的集合数据转换为 json 数据。我们将此过程称为 **序列化**；如果是将 json 数据转换为 Java 对象，我们称之为 **反序列化**
- 将 json 数据响应回给浏览器。这里一定要设置响应数据的类型及字符集
`response.setContentType("text/json;charset=utf-8");`

`SelectAllServlet` 代码如下：

```
1 @WebServlet("/selectAllServlet")
2 public class SelectAllServlet extends HttpServlet {
3     private BrandService brandService = new BrandService();
4
5     @Override
6     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
7         ServletException, IOException {
8         //1. 调用Service查询
9         List<Brand> brands = brandService.selectAll();
10
11         //2. 将集合转换为JSON数据 序列化
12         String jsonString = JSON.toJSONString(brands);
13
14         //3. 响应数据 application/json text/json
15         response.setContentType("text/json;charset=utf-8");
16         response.getWriter().write(jsonString);
17     }
18
19     @Override
20     protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
21         ServletException, IOException {
22         this.doGet(request, response);
23     }
24 }
```

6.2.3 前端实现

1. 引入 js 文件

在 `brand.html` 页面引入 `axios` 的 js 文件

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

2. 绑定 `页面加载完毕` 事件

在 `brand.html` 页面绑定加载完毕事件，该事件是在页面加载完毕后被触发，代码如下

```
1 window.onload = function() {
2
3 }
```

3. 发送异步请求

在页面加载完毕事件绑定的匿名函数中发送异步请求，代码如下：

```
1 //2. 发送ajax请求
2 axios({
3     method: "get",
4     url: "http://localhost:8080/brand-demo/selectAllServlet"
5 }).then(function (resp) {
6
7 });
```

4. 处理响应数据

在 `then` 中的回调函数中通过 `resp.data` 可以获取响应回来的数据，而数据格式如下

```
[{
  "brandName": "三只松鼠",
  "companyName": "三只松鼠股份有限公司",
  "description": "好吃不上火",
  "id": 1,
  "ordered": 5,
  "status": 0,
  "statusStr": "禁用"
}, {
  "brandName": "华为",
  "companyName": "华为技术有限公司",
  "description": "华为致力于把数字世界带入每个人、每个家庭、每个组织，构建万物互联的智能世界",
  "id": 2,
  "ordered": 100,
  "status": 1,
  "statusStr": "启用"
}, {
```

现在我们需要拼接字符串，将下面表格中的所有的 `tr` 拼接到一个字符串中，然后使用

`document.getElementById("brandTable").innerHTML = 拼接好的字符串` 就可以动态的展示出用户想看到的数据

```
<table id="brandTable" border="1" cellspacing="0" width="100%">
  <tr>
    <th>序号</th>
    <th>品牌名称</th>
    <th>企业名称</th>
    <th>排序</th>
    <th>品牌介绍</th>
    <th>状态</th>
    <th>操作</th>
  </tr>
  <tr align="center">
    <td>1</td>
    <td>三只松鼠</td>
    <td>三只松鼠</td>
    <td>100</td>
    <td>三只松鼠，好吃不上火</td>
    <td>启用</td>
    <td><a href="#">修改</a> <a href="#">删除</a></td>
  </tr>
```

而表头行是固定的，所以先定义初始值是表头行数据的字符串，如下

```
1 //获取数据
2 let brands = resp.data;
3 let tableData = " <tr>\n" +
4   "   <th>序号</th>\n" +
5   "   <th>品牌名称</th>\n" +
6   "   <th>企业名称</th>\n" +
7   "   <th>排序</th>\n" +
8   "   <th>品牌介绍</th>\n" +
9   "   <th>状态</th>\n" +
10  "   <th>操作</th>\n" +
11  " </tr>";
```

接下来遍历响应回来的数据 `brands`，拿到每一条品牌数据

```
1 for (let i = 0; i < brands.length ; i++) {
2   let brand = brands[i];
3
4 }
```

紧接着就是从 `brand` 对象中获取数据并且拼接 `数据行`，累加到 `tableData` 字符串变量中

```

1  tableData += "\n" +
2      "      <tr align=\"center\">\n" +
3      "          <td>" + (i+1) + "</td>\n" +
4      "          <td>" + brand.brandName + "</td>\n" +
5      "          <td>" + brand.companyName + "</td>\n" +
6      "          <td>" + brand.ordered + "</td>\n" +
7      "          <td>" + brand.description + "</td>\n" +
8      "          <td>" + brand.status + "</td>\n" +
9      "\n" +
10     "          <td><a href=\"#\\">修改</a> <a href=\"#\\">删除</a></td>\n" +
11     "</tr>";

```

最后再将拼接好的字符串写到表格中

```

1 // 设置表格数据
2 document.getElementById("brandTable").innerHTML = tableData;

```

整体页面代码如下：

```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <title>Title</title>
6 </head>
7 <body>
8 <a href="addBrand.html"><input type="button" value="新增"></a><br>
9 <hr>
10 <table id="brandTable" border="1" cellspacing="0" width="100%">
11
12 </table>
13
14 <script src="js/axios-0.18.0.js"></script>
15
16 <script>
17     //1. 当页面加载完成后，发送ajax请求
18     window.onload = function () {
19         //2. 发送ajax请求
20         axios({
21             method: "get",
22             url: "http://localhost:8080/brand-demo/selectAllServlet"
23         }).then(function (resp) {
24             //获取数据
25             let brands = resp.data;
26             let tableData = " <tr>\n" +
27                 "         <th>序号</th>\n" +
28                 "         <th>品牌名称</th>\n" +
29                 "         <th>企业名称</th>\n" +
30                 "         <th>排序</th>\n" +
31                 "         <th>品牌介绍</th>\n" +
32                 "         <th>状态</th>\n" +
33                 "         <th>操作</th>\n" +
34             "     </tr>";
35
36             for (let i = 0; i < brands.length ; i++) {
37                 let brand = brands[i];
38
39                 tableData += "\n" +
40                     "     <tr align=\"center\">\n" +
41                     "         <td>" + (i+1) + "</td>\n" +
42                     "         <td>" + brand.brandName + "</td>\n" +
43                     "         <td>" + brand.companyName + "</td>\n" +
44                     "         <td>" + brand.ordered + "</td>\n" +
45                     "         <td>" + brand.description + "</td>\n" +
46                     "         <td>" + brand.status + "</td>\n" +

```

```

47             "\n" +
48             "      <td><a href="#">修改</a> <a href="#">删除</a></td>\n" +
49             "    </tr>";
50     }
51     // 设置表格数据
52     document.getElementById("brandTable").innerHTML = tabledata;
53   }
54 }
55 </script>
56 </body>
57 </html>

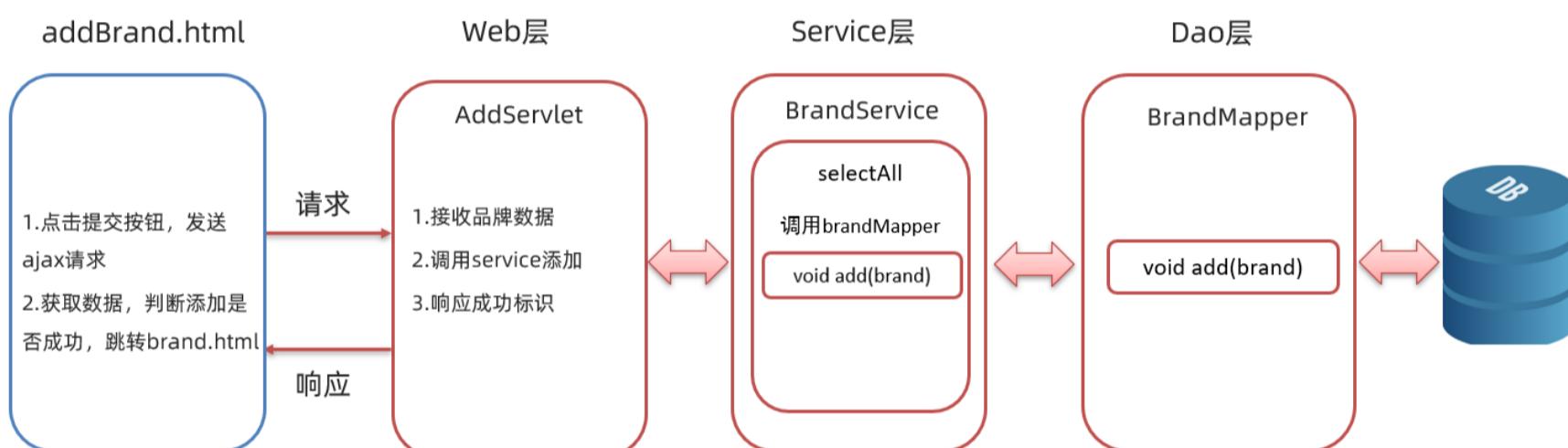
```

6.3 添加品牌功能



如上所示，当我们点击 `新增` 按钮，会跳转到 `addBrand.html` 页面。在 `addBrand.html` 页面输入数据后点击 `提交` 按钮，就会将数据提交到后端，而后端将数据保存到数据库中。

具体的前后端交互的流程如下：



说明：

前端需要将用户输入的数据提交到后端，这部分数据需要以 json 格式进行提交，数据格式如下：

```
{"brandName": "鸿星尔克", "companyName": "鸿星尔克", "ordered": "200", "description": "to be no.1", "status": "1"}
```

6.3.1 后端实现

在 `com.itheima.web` 包下创建名为 `AddServlet` 的 `servlet`，具体的逻辑如下：

- 获取请求参数

由于前端提交的是 json 格式的数据，所以我们不能使用 `request.getParameter()` 方法获取请求参数

- 如果提交的数据格式是 `username=zhangsan&age=23`，后端就可以使用 `request.getParameter()` 方法获取
- 如果提交的数据格式是 json，后端就需要通过 `request` 对象获取输入流，再通过输入流读取数据
- 将获取到的请求参数 (json格式的数据) 转换为 `Brand` 对象
- 调用 `service` 的 `add()` 方法进行添加数据的逻辑处理
- 将 json 数据响应回给浏览器。

`AddServlet` 代码如下：

```

1 @WebServlet("/addServlet")
2 public class AddServlet extends HttpServlet {
3
4     private BrandService brandService = new BrandService();
5
6     @Override
7     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
8         ServletException, IOException {

```

```

8
9     //1. 接收数据, request.getParameter 不能接收json的数据
10    /* String brandName = request.getParameter("brandName");
11       System.out.println(brandName); */
12
13     // 获取请求体数据
14     BufferedReader br = request.getReader();
15     String params = br.readLine();
16     // 将JSON字符串转为Java对象
17     Brand brand = JSON.parseObject(params, Brand.class);
18     //2. 调用service 添加
19     brandService.add(brand);
20     //3. 响应成功标识
21     response.getWriter().write("success");
22 }
23
24 @Override
25 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
26     this doGet(request, response);
27 }
28 }
```

6.3.2 前端实现

在 `addBrand.html` 页面给 `提交` 按钮绑定点击事件，并在绑定的匿名函数中发送异步请求，代码如下：

```

1 //1. 给按钮绑定单击事件
2 document.getElementById("btn").onclick = function () {
3     //2. 发送ajax请求
4     axios({
5         method:"post",
6         url:"http://localhost:8080/brand-demo/addServlet",
7         data:???
8     }).then(function (resp) {
9         // 判断响应数据是否为 success
10        if(resp.data == "success"){
11            location.href = "http://localhost:8080/brand-demo/brand.html";
12        }
13    })
14 }
```

现在我们只需要考虑如何获取页面上用户输入的数据即可。

首先我们先定义如下的一个 js 对象，该对象是用来封装页面上输入的数据，并将该对象作为上面发送异步请求时 `data` 属性的值。

```

1 // 将表单数据转为json
2 var formData = {
3     brandName:"",
4     companyName:"",
5     ordered:"",
6     description:"",
7     status:"",
8 };
```

接下来获取输入框输入的数据，并将获取到的数据赋值给 `formData` 对象指定的属性。比如获取用户名的输入框数据，并把该数据赋值给 `formData` 对象的 `brandName` 属性

```

1 // 获取表单数据
2 let brandName = document.getElementById("brandName").value;
3 // 设置数据
4 formData.brandName = brandName;
```

说明：其他的输入框都用同样的方式获取并赋值。但是有一个比较特殊，就是状态数据，如下图是页面内容

状态:

```
<input type="radio" name="status" value="0">禁用  
<input type="radio" name="status" value="1">启用<br>
```

我们需要判断哪几个被选中，再将选中的单选框数据赋值给 `formData` 对象的 `status` 属性，代码实现如下：

```
1 let status = document.getElementsByName("status");  
2 for (let i = 0; i < status.length; i++) {  
3     if(status[i].checked){  
4         //  
5         formData.status = status[i].value ;  
6     }  
7 }
```

整体页面代码如下：

```
1 <!DOCTYPE html>  
2 <html lang="en">  
3  
4 <head>  
5     <meta charset="UTF-8">  
6     <title>添加品牌</title>  
7 </head>  
8 <body>  
9 <h3>添加品牌</h3>  
10 <form action="" method="post">  
11     品牌名称: <input id="brandName" name="brandName"><br>  
12     企业名称: <input id="companyName" name="companyName"><br>  
13     排序: <input id="ordered" name="ordered"><br>  
14     描述信息: <textarea rows="5" cols="20" id="description" name="description"></textarea><br>  
15     状态:  
16     <input type="radio" name="status" value="0">禁用  
17     <input type="radio" name="status" value="1">启用<br>  
18  
19     <input type="button" id="btn" value="提交">  
20 </form>  
21  
22 <script src="js/axios-0.18.0.js"></script>  
23  
24 <script>  
25     //1. 给按钮绑定单击事件  
26     document.getElementById("btn").onclick = function () {  
27         // 将表单数据转为json  
28         var formData = {  
29             brandName:"",  
30             companyName:"",  
31             ordered:"",  
32             description:"",  
33             status:"",  
34         };  
35         // 获取表单数据  
36         let brandName = document.getElementById("brandName").value;  
37         // 设置数据  
38         formData.brandName = brandName;  
39  
40         // 获取表单数据  
41         let companyName = document.getElementById("companyName").value;  
42         // 设置数据  
43         formData.companyName = companyName;  
44  
45         // 获取表单数据  
46         let ordered = document.getElementById("ordered").value;  
47         // 设置数据  
48         formData.ordered = ordered;  
49     }
```

```
50     // 获取表单数据
51     let description = document.getElementById("description").value;
52     // 设置数据
53     formData.description = description;
54
55     let status = document.getElementsByName("status");
56     for (let i = 0; i < status.length; i++) {
57         if(status[i].checked){
58             //
59             formData.status = status[i].value ;
60         }
61     }
62     //console.log(formData);
63     //2. 发送ajax请求
64     axios({
65         method:"post",
66         url:"http://localhost:8080/brand-demo/addServlet",
67         data:formData
68     }).then(function (resp) {
69         // 判断响应数据是否为 success
70         if(resp.data == "success"){
71             location.href = "http://localhost:8080/brand-demo/brand.html";
72         }
73     })
74 }
75 </script>
76 </body>
77 </html>
```

说明:

查询所有 功能和 添加品牌 功能就全部实现，大家肯定会感觉前端的代码很复杂；而这只是暂时的，后面学习了 vue 前端框架后，这部分前端代码就可以进行很大程度的简化。

VUE&Element

今日目标：

- 能够使用VUE中常用指令和插值表达式
- 能够使用VUE生命周期函数 mounted
- 能够进行简单的 Element 页面修改
- 能够完成查询所有功能
- 能够完成添加功能

1, VUE

1.1 概述

接下来我们学习一款前端的框架，就是 VUE。

Vue 是一套前端框架，免除原生JavaScript中的DOM操作，简化书写。

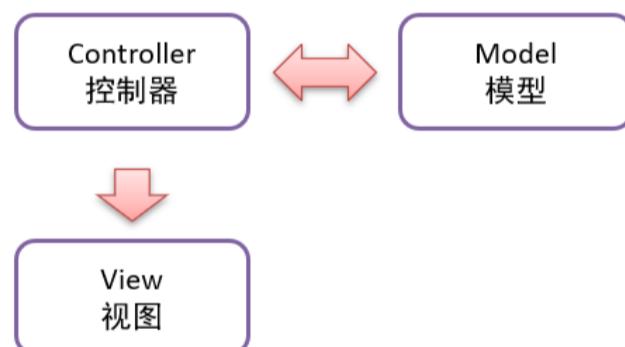
我们之前也学习过后端的框架 Mybatis，Mybatis 是用来简化 jdbc 代码编写的；而 VUE 是前端的框架，是用来简化 JavaScript 代码编写的。前一天我们做了一个综合性的案例，里面进行了大量的DOM操作，如下

```
// 获取表单数据
let brandName = document.getElementById("brandName").value;
// 设置数据
formData.brandName = brandName;

// 获取表单数据
let companyName = document.getElementById("companyName").value;
// 设置数据
formData.companyName = companyName;
```

学习了 VUE 后，这部分代码我们就不需要再写了。那么 VUE 是如何简化 DOM 书写呢？

基于MVVM(Model-View-ViewModel)思想，实现数据的双向绑定，将编程的关注点放在数据上。之前我们是将关注点放在了 DOM 操作上；而要了解 MVVM 思想，必须先聊聊 MVC 思想，如下图就是 MVC 思想图解



MVC：只能实现模型到视图的单向展示

C 就是咱们 js 代码，M 就是数据，而 V 是页面上展示的内容，如下图是我们之前写的代码

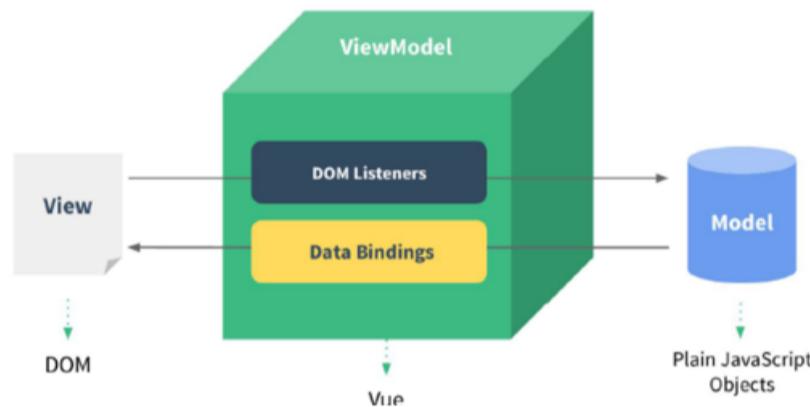
```

for (let i = 0; i < brands.length ; i++) {
    let brand = brands[i];
    箭头从brand指向“数据模型”

    tableData += "\n" +
        "    <tr align=\"center\"\>\n" +
        "        <td>" + (i+1) + "</td>\n" +
        "        <td>" + brand.brandName + "</td>\n" +
        "        <td>" + brand.companyName + "</td>\n" +
        "        <td>" + brand.ordered + "</td>\n" +
        "        <td>" + brand.description + "</td>\n" +
        "        <td>" + brand.status + "</td>\n" +
        "\n" +
        "        <td><a href="#">修改</a> <a href="#">删除</a></td>\n" +
    "    </tr>";
}

```

MVC 思想是没法进行双向绑定的。双向绑定是指当数据模型数据发生变化时，页面展示的会随之发生变化，而如果表单数据发生变化，绑定的数据也随之发生变化。接下来我们聊聊 MVVM 思想，如下图是三个组件图解



图中的 **Model** 就是我们的数据，**View** 是视图，也就是页面标签，用户可以通过浏览器看到的内容；**Model** 和 **View** 是通过 **ViewModel** 对象进行双向绑定的，而 **viewModel** 对象是 **Vue** 提供的。接下来让大家看一下双向绑定的效果，下图是提前准备的代码，输入框绑定了 **username** 模型数据，而在页面上也使用 **{}{}** 绑定了 **username** 模型数据

```

<div id="app">
    <input v-model="username">
    <!-- 插值表达式--&gt;
    {{username}}
&lt;/div&gt;
&lt;script src="js/vue.js"&gt;&lt;/script&gt;
&lt;script&gt;
    //1. 创建Vue核心对象
    new Vue({
        el:"#app",
        data(){
            return {
                username:""
            }
        }
    })
</pre>

```

通过浏览器打开该页面可以看到如下页面



当我们在输入框中输入内容，而输入框后面随之实时的展示我们输入的内容，这就是双向绑定的效果。

1.2 快速入门

Vue 使用起来是比较简单的，总共分为如下三步：

1. 新建 HTML 页面，引入 **Vue.js** 文件

```
1 | <script src="js/vue.js"></script>
```

2. 在JS代码区域，创建Vue核心对象，进行数据绑定

```
1 new Vue({
2   el: "#app",
3   data() {
4     return {
5       username: ""
6     }
7   }
8 });

});
```

创建 Vue 对象时，需要传递一个 js 对象，而该对象中需要如下属性：

- `el`：用来指定哪些标签受 Vue 管理。该属性取值 `#app` 中的 `app` 需要是受管理的标签的 `id` 属性值
- `data`：用来定义数据模型
- `methods`：用来定义函数。这个我们在后面就会用到

3. 编写视图

```
1 <div id="app">
2   <input name="username" v-model="username" >
3   {{username}}
4 </div>
```

`{{}}` 是 Vue 中定义的 插值表达式，在里面写数据模型，到时候会将该模型的数据值展示在这个位置。

整体代码如下：

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>Title</title>
6 </head>
7 <body>
8 <div id="app">
9   <input v-model="username">
10  <!--插值表达式-->
11  {{username}}
12 </div>
13 <script src="js/vue.js"></script>
14 <script>
15   //1. 创建Vue核心对象
16   new Vue({
17     el:"#app",
18     data(){ // data() 是 ECMAScript 6 版本的新的写法
19       return {
20         username:""
21       }
22     }
23
24     /*data: function () {
25       return {
26         username:""
27       }
28     }*/
29   });
30
31 </script>
32 </body>
33 </html>
```

1.3 Vue 指令

指令：HTML 标签上带有 v- 前缀的特殊属性，不同指令具有不同含义。例如：v-if, v-for...

常用的指令有：

指令	作用
v-bind	为HTML标签绑定属性值，如设置 href , css样式等
v-model	在表单元素上创建双向数据绑定
v-on	为HTML标签绑定事件
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	
v-else-if	
v-show	根据条件展示某元素，区别在于切换的是display属性的值
v-for	列表渲染，遍历容器的元素或者对象的属性

接下来我们挨个学习这些指令

1.3.1 v-bind & v-model 指令

指令	作用
v-bind	为HTML标签绑定属性值，如设置 href , css样式等
v-model	在表单元素上创建双向数据绑定

• v-bind

该指令可以给标签原有属性绑定模型数据。这样模型数据发生变化，标签属性值也随之发生变化

例如：

```
1 | <a v-bind:href="url">百度一下</a>
```

上面的 v-bind:" " 可以简化写成 : ，如下：

```
1 | <!--
2 |   v-bind 可以省略
3 | -->
4 | <a :href="url">百度一下</a>
```

• v-model

该指令可以给表单项标签绑定模型数据。这样就能实现双向绑定效果。例如：

```
1 | <input name="username" v-model="username">
```

代码演示：

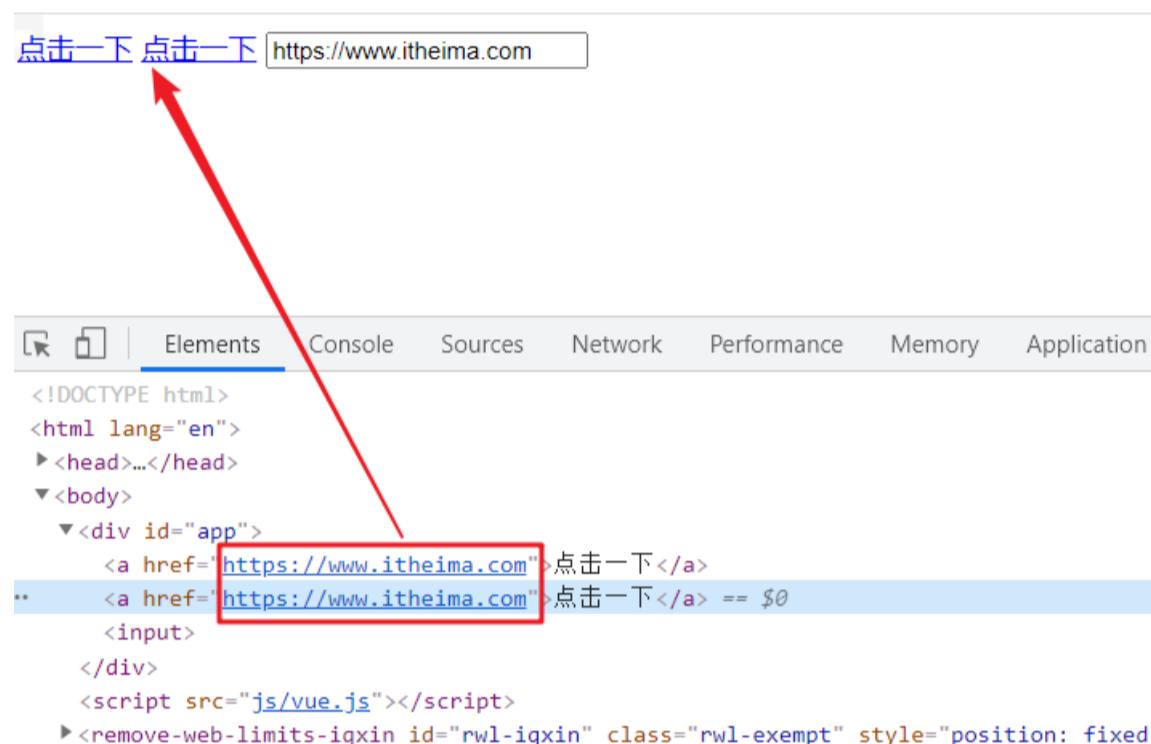
```
1 | <!DOCTYPE html>
2 | <html lang="en">
3 | <head>
4 |   <meta charset="UTF-8">
5 |   <title>Title</title>
6 | </head>
7 | <body>
8 |   <div id="app">
9 |     <a v-bind:href="url">点击一下</a>
10 |    <a :href="url">点击一下</a>
11 |    <input v-model="url">
12 |   </div>
13 | 
```

```

14 <script src="js/vue.js"></script>
15 <script>
16     //1. 创建Vue核心对象
17     new Vue({
18         el:"#app",
19         data(){
20             return {
21                 username:"",
22                 url:"https://www.baidu.com"
23             }
24         }
25     });
26 </script>
27 </body>
28 </html>

```

通过浏览器打开上面页面，并且使用检查查看超链接的路径，该路径会根据输入框输入的路径变化而变化，这是因为超链接和输入框绑定的是同一个模型数据



1.3.2 v-on 指令

指令	作用
v-on	为HTML标签绑定事件

我们在页面定义一个按钮，并给该按钮使用 v-on 指令绑定单击事件，html代码如下

```
1 | <input type="button" value="一个按钮" v-on:click="show()">
```

而使用 v-on 时还可以使用简化的写法，将 v-on: 替换成 @，html代码如下

```
1 | <input type="button" value="一个按钮" @click="show()">
```

上面代码绑定的 show() 需要在 Vue 对象中的 methods 属性中定义出来

```

1 new Vue({
2     el: "#app",
3     methods: {
4         show(){
5             alert("我被点了");
6         }
7     }
8 });

```

注意：v-on: 后面的事件名称是之前原生事件属性名去掉on。

例如：

- 单击事件：事件属性名是 onclick，而在vue中使用是 v-on:click

- 失去焦点事件：事件属性名是 `onblur`，而在Vue中使用时 `v-on:blur`

整体页面代码如下：

```

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>Title</title>
6  </head>
7  <body>
8  <div id="app">
9      <input type="button" value="一个按钮" v-on:click="show()"><br>
10     <input type="button" value="一个按钮" @click="show()">
11 </div>
12 <script src="js/vue.js"></script>
13 <script>
14     //1. 创建Vue核心对象
15     new Vue({
16         el:"#app",
17         data(){
18             return {
19                 username:"",
20             }
21         },
22         methods:{
23             show(){
24                 alert("我被点了...");}
25         }
26     });
27 </script>
28 </body>
29 </html>

```

1.3.3 条件判断指令

指令	作用
<code>v-if</code>	
<code>v-else</code>	条件性的渲染某元素，判定为true时渲染，否则不渲染
<code>v-else-if</code>	
<code>v-show</code>	根据条件展示某元素，区别在于切换的是display属性的值

接下来通过代码演示一下。在Vue中定义一个 `count` 的数据模型，如下

```

1 //1. 创建Vue核心对象
2 new Vue({
3     el:"#app",
4     data(){
5         return {
6             count:3
7         }
8     }
9 });

```

现在要实现，当 `count` 模型的数据是3时，在页面上展示 `div1` 内容；当 `count` 模型的数据是4时，在页面上展示 `div2` 内容；`count` 模型数据是其他值时，在页面上展示 `div3`。这里为了动态改变模型数据 `count` 的值，再定义一个输入框绑定 `count` 模型数据。html 代码如下：

```

1 <div id="app">
2   <div v-if="count == 3">div1</div>
3   <div v-else-if="count == 4">div2</div>
4   <div v-else>div3</div>
5   <hr>
6   <input v-model="count">
7 </div>

```

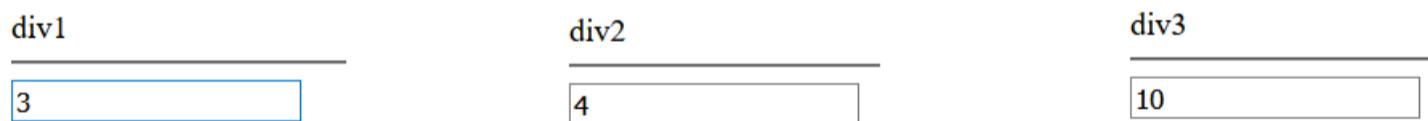
整体页面代码如下：

```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>Title</title>
6 </head>
7 <body>
8 <div id="app">
9   <div v-if="count == 3">div1</div>
10  <div v-else-if="count == 4">div2</div>
11  <div v-else>div3</div>
12  <hr>
13  <input v-model="count">
14 </div>
15
16 <script src="js/vue.js"></script>
17 <script>
18   //1. 创建Vue核心对象
19   new Vue({
20     el:"#app",
21     data(){
22       return {
23         count:3
24       }
25     }
26   });
27 </script>
28 </body>
29 </html>

```

通过浏览器打开页面并在输入框输入不同的值，效果如下



然后我们在看看 `v-show` 指令的效果，如果模型数据 `count` 的值是3时，展示 `div v-show` 内容，否则不展示，html页面代码如下

```

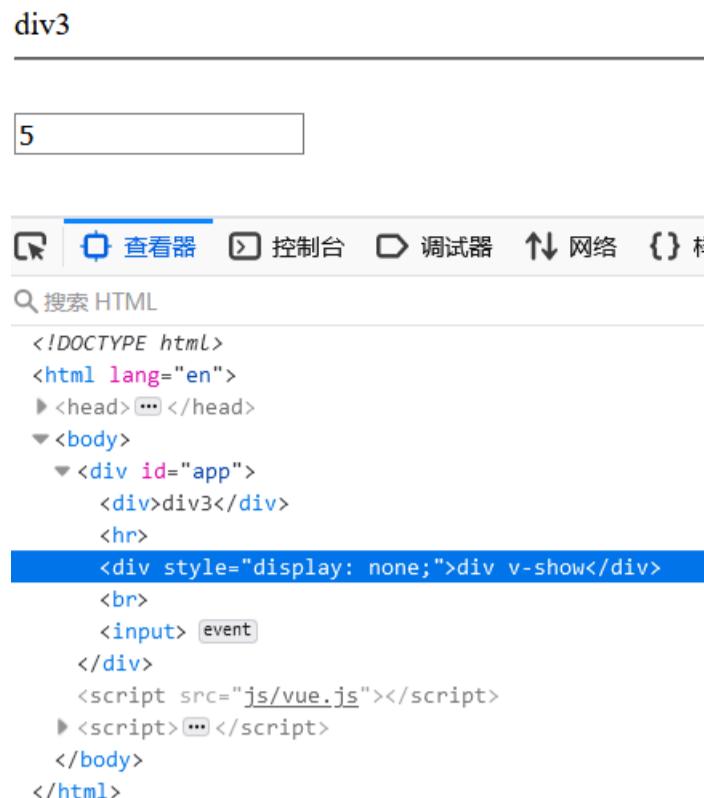
1 <div v-show="count == 3">div v-show</div>
2 <br>
3 <input v-model="count">

```

浏览器打开效果如下：



通过上面的演示，发现 `v-show` 和 `v-if` 效果一样，那它们到底有什么区别呢？我们根据浏览器的检查功能查看源代码



通过上图可以看出 `v-show` 不展示的原理是给对应的标签添加 `display` css属性，并将该属性值设置为 `none`，这样就达到了隐藏的效果。而 `v-if` 指令是条件不满足时根本就不会渲染。

1.3.4 v-for 指令

指令	作用
<code>v-for</code>	列表渲染，遍历容器的元素或者对象的属性

这个指令看到名字就知道是用来遍历的，该指令使用的格式如下：

```

1 <标签 v-for="变量名 in 集合模型数据">
2   {{变量名}}
3 </标签>
```

注意：需要循环那个标签，`v-for` 指令就写在那个标签上。

如果在页面需要使用到集合模型数据的索引，就需要使用如下格式：

```

1 <标签 v-for="(变量名,索引变量) in 集合模型数据">
2   <!--索引变量是从0开始，所以要表示序号的话，需要手动的加1-->
3   {{索引变量 + 1}} {{变量名}}
4 </标签>
```

代码演示：

```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>Title</title>
6 </head>
7 <body>
8 <div id="app">
9   <div v-for="addr in addrs">
10     {{addr}} <br>
11   </div>
12
13   <hr>
14   <div v-for="(addr,i) in addrs">
15     {{i+1}}--{{addr}} <br>
16   </div>
17 </div>
18
19 <script src="js/vue.js"></script>
20 <script>
```

```

21
22 //1. 创建Vue核心对象
23 new Vue({
24   el:"#app",
25   data(){
26     return {
27       addrs:["北京", "上海", "西安"]
28     }
29   }
30 });
31 </script>
32 </body>
33 </html>

```

通过浏览器打开效果如下

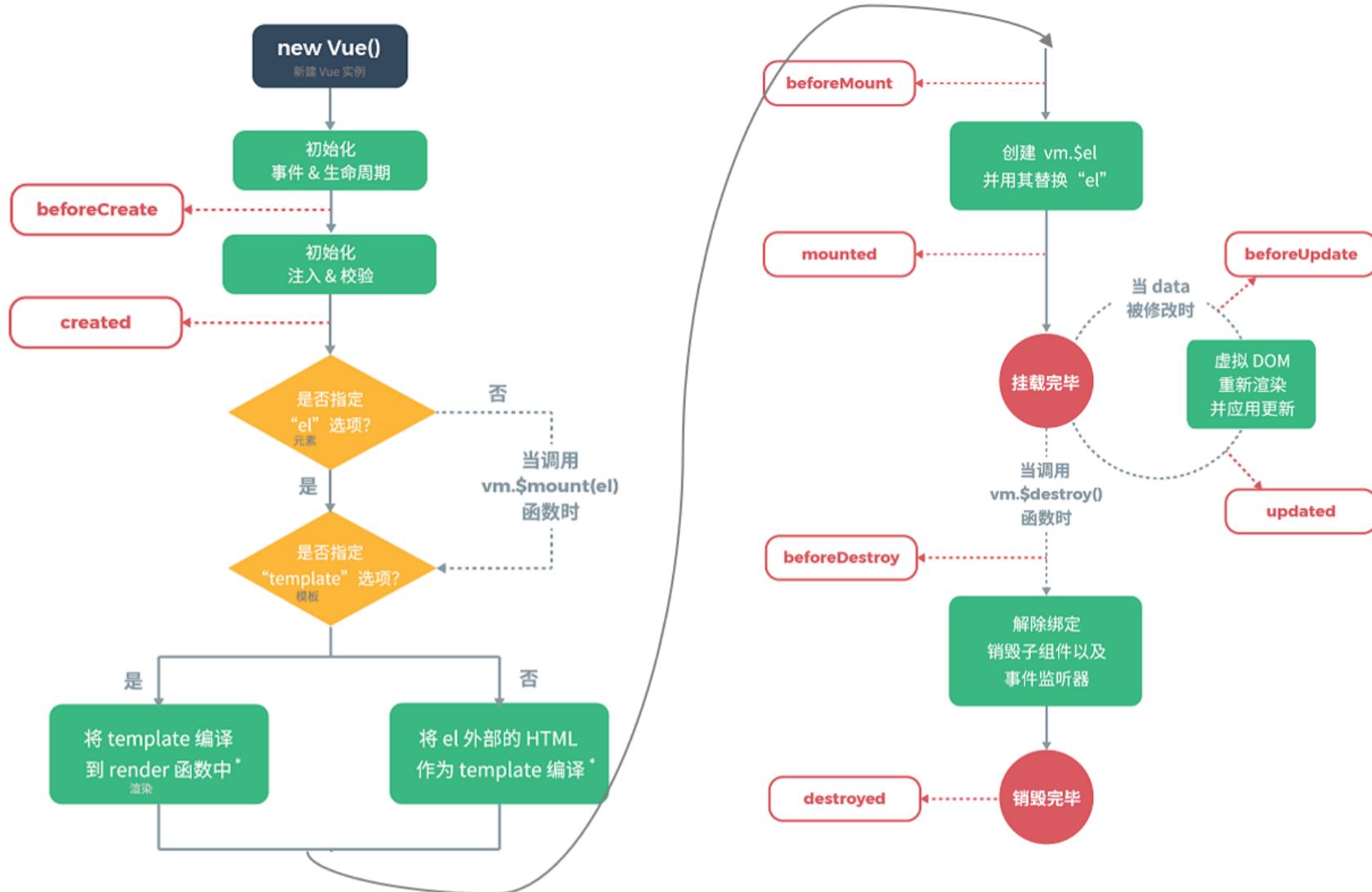


1.4 生命周期

生命周期的八个阶段：每触发一个生命周期事件，会自动执行一个生命周期方法，这些生命周期方法也被称为钩子方法。

状态	阶段周期
beforeCreate	创建前
created	创建后
beforeMount	载入前
mounted	挂载完成
beforeUpdate	更新前
updated	更新后
beforeDestroy	销毁前
destroyed	销毁后

下图是 Vue 官网提供的从创建 Vue 到效果 Vue 对象的整个过程及各个阶段对应的钩子函数



看到上面的图，大家无需过多的关注这张图。这些钩子方法我们只关注 `mounted` 就行了。

`mounted`: 挂载完成，Vue 初始化成功，HTML 页面渲染成功。而以后我们会在该方法中发送异步请求，加载数据。

1.5 案例

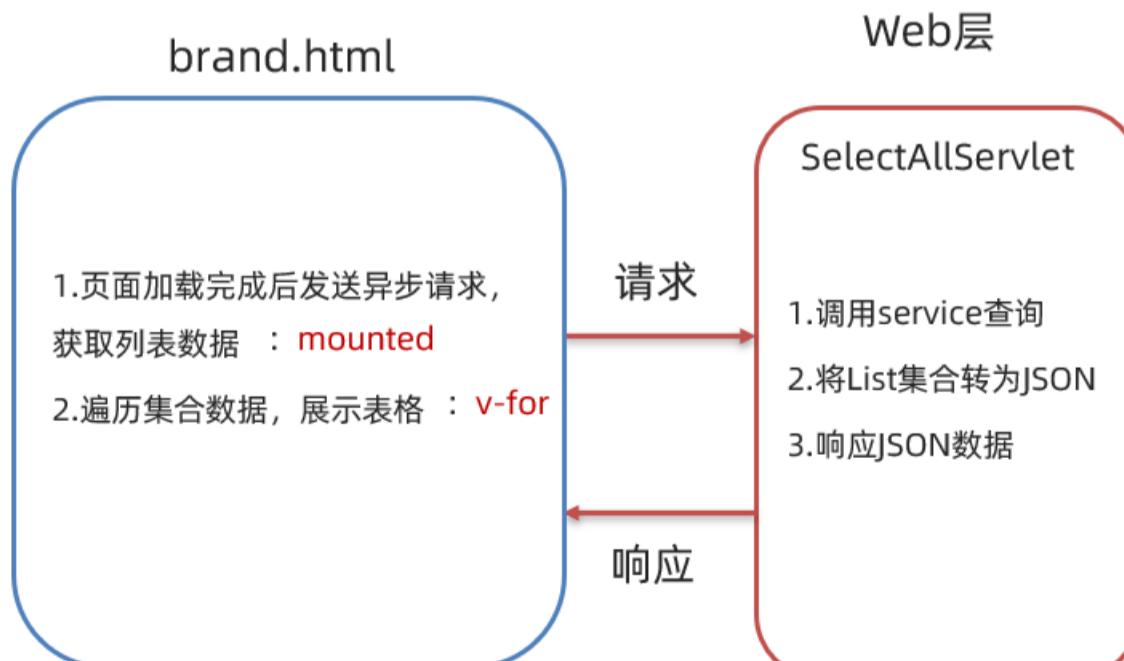
1.5.1 需求

使用 Vue 简化我们在前一天 ajax 学完后做的品牌列表数据查询和添加功能

新增						
序号	品牌名称	企业名称	排序	品牌介绍	状态	操作
1	三只松鼠	三只松鼠	100	三只松鼠，好吃不上火	启用	修改 删除
2	优衣库	优衣库	10	优衣库，服适人生	禁用	修改 删除
3	小米	小米科技有限公司	1000	为发烧而生	启用	修改 删除

此案例只是使用 Vue 对前端代码进行优化，后端代码无需修改。

1.5.2 查询所有功能



1. 在 brand.html 页面引入 vue 的js文件

```
1 | <script src="js/vue.js"></script>
```

2. 创建 Vue 对象

- 在 Vue 对象中定义模型数据
- 在钩子函数中发送异步请求，并将响应的数据赋值给数据模型

```

1 new Vue({
2   el: "#app",
3   data(){
4     return{
5       brands:[]
6     }
7   },
8   mounted(){
9     // 页面加载完成后，发送异步请求，查询数据
10    var _this = this;
11    axios({
12      method:"get",
13      url:"http://localhost:8080/brand-demo/selectAllServlet"
14    }).then(function (resp) {
15      _this.brands = resp.data;
16    })
17  }
18})

```

3. 修改视图

- 定义 `<div id="app"></div>`，指定该 `div` 标签受 Vue 管理
- 将 `body` 标签中所有的内容拷贝作为上面 `div` 标签中
- 删除表格的多余数据行，只留下一个
- 在表格中的数据行上使用 `v-for` 指令遍历

```

1 <tr v-for="(brand,i) in brands" align="center">
2   <td>{{i + 1}}</td>
3   <td>{{brand.brandName}}</td>
4   <td>{{brand.companyName}}</td>
5   <td>{{brand.ordered}}</td>
6   <td>{{brand.description}}</td>
7   <td>{{brand.statusStr}}</td>
8   <td><a href="#">修改</a> <a href="#">删除</a></td>
9 </tr>

```

整体页面代码如下：

```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>Title</title>
6 </head>
7 <body>
8 <div id="app">
9   <a href="addBrand.html"><input type="button" value="新增"></a><br>
10  <hr>
11  <table id="brandTable" border="1" cellspacing="0" width="100%">
12    <tr>
13      <th>序号</th>
14      <th>品牌名称</th>
15      <th>企业名称</th>
16      <th>排序</th>
17      <th>品牌介绍</th>
18      <th>状态</th>
19      <th>操作</th>
20    </tr>
21    <!--
22      使用v-for遍历tr
23    -->

```

```

24         <tr v-for="(brand,i) in brands" align="center">
25             <td>{{i + 1}}</td>
26             <td>{{brand.brandName}}</td>
27             <td>{{brand.companyName}}</td>
28             <td>{{brand.ordered}}</td>
29             <td>{{brand.description}}</td>
30             <td>{{brand.statusStr}}</td>
31             <td><a href="#">修改</a> <a href="#">删除</a></td>
32         </tr>
33     </table>
34 </div>
35 <script src="js/axios-0.18.0.js"></script>
36 <script src="js/vue.js"></script>
37
38 <script>
39     new Vue({
40         el: "#app",
41         data(){
42             return{
43                 brands:[]
44             }
45         },
46         mounted(){
47             // 页面加载完成后，发送异步请求，查询数据
48             var _this = this;
49             axios({
50                 method:"get",
51                 url:"http://localhost:8080/brand-demo/selectAllServlet"
52             }).then(function (resp) {
53                 _this.brands = resp.data;
54             })
55         }
56     })
57 </script>
58 </body>
59 </html>

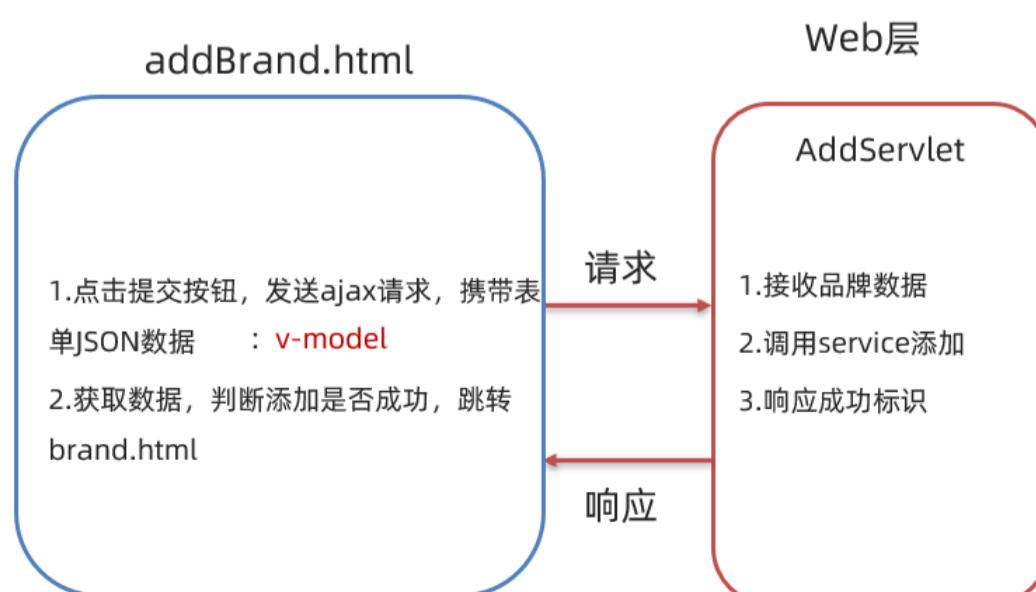
```

1.5.3 添加功能

页面操作效果如下：



整体流程如下



注意：前端代码的关键点在于使用 `v-model` 指令给标签项绑定模型数据，利用双向绑定特性，在发送异步请求时提交数据。

1. 在 `addBrand.html` 页面引入 vue 的js文件

```
1 | <script src="js/vue.js"></script>
```

2. 创建 Vue 对象

- 在 Vue 对象中定义模型数据 `brand`
- 定义一个 `submitForm()` 函数，用于给 `提交` 按钮提供绑定的函数
- 在 `submitForm()` 函数中发送 ajax 请求，并将模型数据 `brand` 作为参数进行传递

```
1 | new Vue({
2 |   el: "#app",
3 |   data(){
4 |     return {
5 |       brand:{}
6 |     }
7 |   },
8 |   methods:{
9 |     submitForm(){
10 |       // 发送ajax请求，添加
11 |       var _this = this;
12 |       axios({
13 |         method:"post",
14 |         url:"http://localhost:8080/brand-demo/addservlet",
15 |         data:_this.brand
16 |       }).then(function (resp) {
17 |         // 判断响应数据是否为 success
18 |         if(resp.data == "success"){
19 |           location.href = "http://localhost:8080/brand-demo/brand.html";
20 |         }
21 |       })
22 |
23 |     }
24 |   }
25 | })
```

3. 修改视图

- 定义 `<div id="app"></div>`，指定该 `div` 标签受 Vue 管理
- 将 `body` 标签中所有的内容拷贝作为上面 `div` 标签中
- 给每一个表单项标签绑定模型数据。最后这些数据要被封装到 `brand` 对象中

```
1 | <div id="app">
2 |   <h3>添加品牌</h3>
3 |   <form action="" method="post">
4 |     品牌名称: <input id="brandName" v-model="brand.brandName" name="brandName"><br>
5 |     企业名称: <input id="companyName" v-model="brand.companyName" name="companyName">
<br>
6 |     排序: <input id="ordered" v-model="brand.ordered" name="ordered"><br>
7 |     描述信息: <textarea rows="5" cols="20" id="description" v-
model="brand.description" name="description"></textarea><br>
8 |     状态:
9 |     <input type="radio" name="status" v-model="brand.status" value="0">禁用
10 |    <input type="radio" name="status" v-model="brand.status" value="1">启用<br>
11 |
12 |    <input type="button" id="btn" @click="submitForm" value="提交">
13 |  </form>
14 | </div>
```

整体页面代码如下：

```

1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="UTF-8">
6   <title>添加品牌</title>
7 </head>
8 <body>
9 <div id="app">
10   <h3>添加品牌</h3>
11   <form action="" method="post">
12     品牌名称: <input id="brandName" v-model="brand.brandName" name="brandName"><br>
13     企业名称: <input id="companyName" v-model="brand.companyName" name="companyName"><br>
14     排序: <input id="ordered" v-model="brand.ordered" name="ordered"><br>
15     描述信息: <textarea rows="5" cols="20" id="description" v-model="brand.description"
16       name="description"></textarea><br>
17     状态:
18       <input type="radio" name="status" v-model="brand.status" value="0">禁用
19       <input type="radio" name="status" v-model="brand.status" value="1">启用<br>
20
21     <input type="button" id="btn" @click="submitForm" value="提交">
22   </form>
23 </div>
24 <script src="js/axios-0.18.0.js"></script>
25 <script src="js/vue.js"></script>
26 <script>
27   new Vue({
28     el: "#app",
29     data(){
30       return {
31         brand:{}
32       }
33     },
34     methods:{
35       submitForm(){
36         // 发送ajax请求，添加
37         var _this = this;
38         axios({
39           method:"post",
40           url:"http://localhost:8080/brand-demo/addServlet",
41           data:_this.brand
42         }).then(function (resp) {
43           // 判断响应数据是否为 success
44           if(resp.data == "success"){
45             location.href = "http://localhost:8080/brand-demo/brand.html";
46           }
47         })
48       }
49     }
50   </script>
51 </body>
52 </html>

```

通过上面的优化，前端代码确实简化了不少。但是页面依旧是不怎么好看，那么接下来我们学习 Element，它可以美化页面。

2, Element

Element: 是饿了么公司前端开发团队提供的一套基于 Vue 的网站组件库，用于快速构建网页。

Element 提供了很多组件（组成网页的部件）供我们使用。例如 超链接、按钮、图片、表格等等~

如下图左边的是我们编写页面看到的按钮，上图右边的是 Element 提供的页面效果，效果一目了然。



我们学习 Element 其实就是学习怎么从官网拷贝组件到我们自己的页面并进行修改，官网网址是

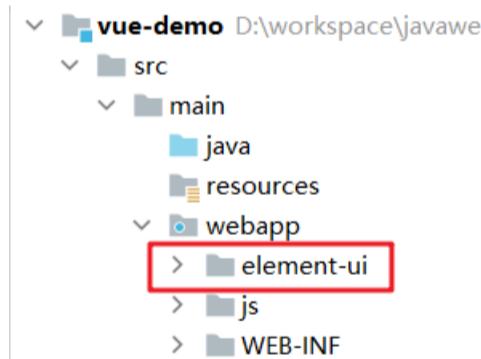
```
1 | https://element.eleme.cn/#/zh-CN
```

进入官网能看到如下页面

接下来直接点击 `组件`，页面如下

2.1 快速入门

1. 将资源 `04-资料\02-element` 下的 `element-ui` 文件夹直接拷贝到项目的 `webapp` 下。目录结构如下



2. 创建页面，并在页面引入Element 的css、js文件和 Vue.js

```
1 <script src="vue.js"></script>
2 <script src="element-ui/lib/index.js"></script>
3 <link rel="stylesheet" href="element-ui/lib/theme-chalk/index.css">
```

3. 创建Vue核心对象

Element 是基于 Vue 的，所以使用Element时必须要创建 Vue 对象

```
1 <script>
2   new Vue({
3     el: "#app"
4   })
5 </script>
```

4. 官网复制Element组件代码

Color 色彩

Typography 字体

Border 边框

Icon 图标

Button 按钮

Link 文字链接

Form

Radio 单选框

Checkbox 多选框

Input 输入框

基础的按钮用法。



在左菜单栏找到 `Button` 按钮，然后找到自己喜欢的按钮样式，点击 `显示代码`，在下面就会展示出对应的代码，将这些代码拷贝到我们自己的页面即可。

整体页面代码如下：

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>Title</title>
6  </head>
7  <body>
8  <div id="app">
9
10
11      <el-row>
12          <el-button>默认按钮</el-button>
13          <el-button type="primary">主要按钮</el-button>
14          <el-button type="success">成功按钮</el-button>
15          <el-button type="info">信息按钮</el-button>
16          <el-button type="warning">警告按钮</el-button>
17          <el-button type="danger">删除</el-button>
18      </el-row>
19      <el-row>
20          <el-button plain>朴素按钮</el-button>
21          <el-button type="primary" plain>主要按钮</el-button>
22          <el-button type="success" plain>成功按钮</el-button>
23          <el-button type="info" plain>信息按钮</el-button>
24          <el-button type="warning" plain>警告按钮</el-button>
25          <el-button type="danger" plain>危险按钮</el-button>
26      </el-row>
27
28      <el-row>
29          <el-button round>圆角按钮</el-button>
30          <el-button type="primary" round>主要按钮</el-button>
31          <el-button type="success" round>成功按钮</el-button>
32          <el-button type="info" round>信息按钮</el-button>
33          <el-button type="warning" round>警告按钮</el-button>
34          <el-button type="danger" round>危险按钮</el-button>
35      </el-row>
36
37      <el-row>
38          <el-button icon="el-icon-search" circle></el-button>
39          <el-button type="primary" icon="el-icon-edit" circle></el-button>
40          <el-button type="success" icon="el-icon-check" circle></el-button>
41          <el-button type="info" icon="el-icon-message" circle></el-button>
42          <el-button type="warning" icon="el-icon-star-off" circle></el-button>
43          <el-button type="danger" icon="el-icon-delete" circle></el-button>
44      </el-row>
45  </div>
```

```

47 <script src="js/vue.js"></script>
48 <script src="element-ui/lib/index.js"></script>
49 <link rel="stylesheet" href="element-ui/lib/theme-chalk/index.css">
50
51 <script>
52     new Vue({
53         el:"#app"
54     })
55 </script>
56
57 </body>
58 </html>

```

2.2 Element 布局

Element 提供了两种布局方式，分别是：

- Layout 布局
- Container 布局容器

2.2.1 Layout 局部

通过基础的 24 分栏，迅速简便地创建布局。也就是默认将一行分为 24 栏，根据页面要求给每一列设置所占的栏数。

在左菜单栏找到 `Layout 布局`，然后找到自己喜欢的按钮样式，点击 `显示代码`，在下面就会展示出对应的代码，显示出的代码中有样式，有html标签。将样式拷贝我们自己页面的 `head` 标签内，将html标签拷贝到 `<div id="app"></div>` 标签内。

整体页面代码如下：

```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <title>Title</title>
6
7     <style>
8         .el-row {
9             margin-bottom: 20px;
10        }
11        .el-col {
12            border-radius: 4px;
13        }
14        .bg-purple-dark {
15            background: #99a9bf;
16        }

```

```

17     .bg-purple {
18         background: #d3dce6;
19     }
20     .bg-purple-light {
21         background: #e5e9f2;
22     }
23     .grid-content {
24         border-radius: 4px;
25         min-height: 36px;
26     }
27     .row-bg {
28         padding: 10px 0;
29         background-color: #f9fafc;
30     }
31 </style>
32 </head>
33 <body>
34 <div id="app">
35     <el-row>
36         <el-col :span="24"><div class="grid-content bg-purple-dark"></div></el-col>
37     </el-row>
38     <el-row>
39         <el-col :span="12"><div class="grid-content bg-purple"></div></el-col>
40         <el-col :span="12"><div class="grid-content bg-purple-light"></div></el-col>
41     </el-row>
42     <el-row>
43         <el-col :span="8"><div class="grid-content bg-purple"></div></el-col>
44         <el-col :span="8"><div class="grid-content bg-purple-light"></div></el-col>
45         <el-col :span="8"><div class="grid-content bg-purple"></div></el-col>
46     </el-row>
47     <el-row>
48         <el-col :span="6"><div class="grid-content bg-purple"></div></el-col>
49         <el-col :span="6"><div class="grid-content bg-purple-light"></div></el-col>
50         <el-col :span="6"><div class="grid-content bg-purple"></div></el-col>
51         <el-col :span="6"><div class="grid-content bg-purple-light"></div></el-col>
52     </el-row>
53     <el-row>
54         <el-col :span="4"><div class="grid-content bg-purple"></div></el-col>
55         <el-col :span="4"><div class="grid-content bg-purple-light"></div></el-col>
56         <el-col :span="4"><div class="grid-content bg-purple"></div></el-col>
57         <el-col :span="4"><div class="grid-content bg-purple-light"></div></el-col>
58         <el-col :span="4"><div class="grid-content bg-purple"></div></el-col>
59         <el-col :span="4"><div class="grid-content bg-purple-light"></div></el-col>
60     </el-row>
61 </div>
62 <script src="js/vue.js"></script>
63 <script src="element-ui/lib/index.js"></script>
64 <link rel="stylesheet" href="element-ui/lib/theme-chalk/index.css">
65
66 <script>
67     new Vue({
68         el:"#app"
69     })
70 </script>
71 </body>
72 </html>

```

现在需要添加一行，要求该行显示8个格子，通过计算每个格子占 3 栏，具体的html 代码如下

```

1 <!--
2 添加一行，8个格子 24/8 = 3
3 -->
4 <el-row>
5   <el-col :span="3"><div class="grid-content bg-purple"></div></el-col>
6   <el-col :span="3"><div class="grid-content bg-purple-light"></div></el-col>
7   <el-col :span="3"><div class="grid-content bg-purple"></div></el-col>
8   <el-col :span="3"><div class="grid-content bg-purple-light"></div></el-col>
9   <el-col :span="3"><div class="grid-content bg-purple"></div></el-col>
10  <el-col :span="3"><div class="grid-content bg-purple-light"></div></el-col>
11  <el-col :span="3"><div class="grid-content bg-purple"></div></el-col>
12  <el-col :span="3"><div class="grid-content bg-purple-light"></div></el-col>
13 </el-row>

```

2.2.2 Container 布局容器

用于布局的容器组件，方便快速搭建页面的基本结构。如下图就是布局容器效果。

如下图是官网提供的 Container 布局容器实例：

日期	姓名	地址
2016-05-02	王小虎	上海市普陀区金沙江路 1518 弄
2016-05-02	王小虎	上海市普陀区金沙江路 1518 弄
2016-05-02	王小虎	上海市普陀区金沙江路 1518 弄
2016-05-02	王小虎	上海市普陀区金沙江路 1518 弄
2016-05-02	王小虎	上海市普陀区金沙江路 1518 弄
2016-05-02	王小虎	上海市普陀区金沙江路 1518 弄
2016-05-02	王小虎	上海市普陀区金沙江路 1518 弄

该效果代码中包含了样式、页面标签、模型数据。将里面的样式 `<style>` 拷贝到我们自己页面的 `head` 标签中；将 html 标签拷贝到 `<div id="app"></div>` 标签中，再将数据模型拷贝到 `vue` 对象的 `data()` 中。

整体页面代码如下：

```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>Title</title>
6
7   <style>
8     .el-header {
9       background-color: #B3C0D1;
10      color: #333;
11      line-height: 60px;
12    }
13
14   .el-aside {
15     color: #333;
16   }
17   </style>
18 </head>

```

```
19 <body>
20 <div id="app">
21   <el-container style="height: 500px; border: 1px solid #eee">
22     <el-aside width="200px" style="background-color: rgb(238, 241, 246)">
23       <el-menu :default-openeds="['1', '3']">
24         <el-submenu index="1">
25           <template slot="title"><i class="el-icon-message"></i>导航一</template>
26           <el-menu-item-group>
27             <template slot="title">分组一</template>
28             <el-menu-item index="1-1">选项1</el-menu-item>
29             <el-menu-item index="1-2">选项2</el-menu-item>
30           </el-menu-item-group>
31           <el-menu-item-group title="分组2">
32             <el-menu-item index="1-3">选项3</el-menu-item>
33           </el-menu-item-group>
34           <el-submenu index="1-4">
35             <template slot="title">选项4</template>
36             <el-menu-item index="1-4-1">选项4-1</el-menu-item>
37           </el-submenu>
38         </el-submenu>
39         <el-submenu index="2">
40           <template slot="title"><i class="el-icon-menu"></i>导航二</template>
41           <el-submenu index="2-1">
42             <template slot="title">选项1</template>
43             <el-menu-item index="2-1-1">选项1-1</el-menu-item>
44           </el-submenu>
45         </el-submenu>
46         <el-submenu index="3">
47           <template slot="title"><i class="el-icon-setting"></i>导航三</template>
48           <el-menu-item-group>
49             <template slot="title">分组一</template>
50             <el-menu-item index="3-1">选项1</el-menu-item>
51             <el-menu-item index="3-2">选项2</el-menu-item>
52           </el-menu-item-group>
53           <el-menu-item-group title="分组2">
54             <el-menu-item index="3-3">选项3</el-menu-item>
55           </el-menu-item-group>
56           <el-submenu index="3-4">
57             <template slot="title">选项4</template>
58             <el-menu-item index="3-4-1">选项4-1</el-menu-item>
59           </el-submenu>
60         </el-submenu>
61       </el-menu>
62     </el-aside>
63
64   <el-container>
65     <el-header style="text-align: right; font-size: 12px">
66       <el-dropdown>
67         <i class="el-icon-setting" style="margin-right: 15px"></i>
68         <el-dropdown-menu slot="dropdown">
69           <el-dropdown-item>查看</el-dropdown-item>
70           <el-dropdown-item>新增</el-dropdown-item>
71           <el-dropdown-item>删除</el-dropdown-item>
72         </el-dropdown-menu>
73       </el-dropdown>
74       <span>王小虎</span>
75     </el-header>
76
77   <el-main>
78     <el-table :data="tableData">
79       <el-table-column prop="date" label="日期" width="140">
80         </el-table-column>
81       <el-table-column prop="name" label="姓名" width="120">
82         </el-table-column>
83       <el-table-column prop="address" label="地址">
```

```
84         </el-table-column>
85     </el-table>
86     </el-main>
87   </el-container>
88 </el-container>
89 </div>
90 <script src="js/vue.js"></script>
91 <script src="element-ui/lib/index.js"></script>
92 <link rel="stylesheet" href="element-ui/lib/theme-chalk/index.css">
93
94 <script>
95   new Vue({
96     el: "#app",
97     data() {
98       const item = {
99         date: '2016-05-02',
100        name: '王小虎',
101        address: '上海市普陀区金沙江路 1518 弄'
102      };
103      return {
104        tableData: Array(20).fill(item)
105      }
106    }
107  })
108 </script>
109 </body>
110 </html>
```

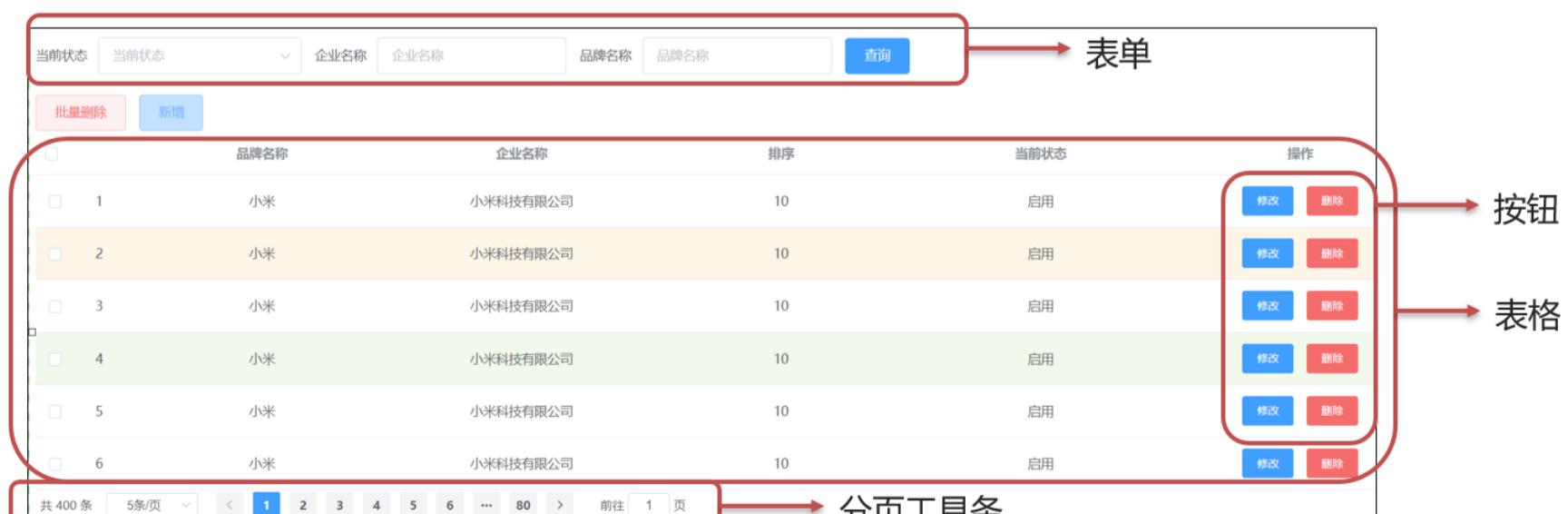
2.3 案例

其他的组件我们通过完成一个页面来学习。

我们要完成如下页面效果

当前状态	当前状态	企业名称	企业名称	品牌名称	品牌名称	查询	
批量删除		新增					
		品牌名称	企业名称	排序	当前状态	操作	
<input type="checkbox"/>	1	小米	小米科技有限公司	10	启用	修改	删除
<input type="checkbox"/>	2	小米	小米科技有限公司	10	启用	修改	删除
<input type="checkbox"/>	3	小米	小米科技有限公司	10	启用	修改	删除
<input type="checkbox"/>	4	小米	小米科技有限公司	10	启用	修改	删除
<input type="checkbox"/>	5	小米	小米科技有限公司	10	启用	修改	删除
<input type="checkbox"/>	6	小米	小米科技有限公司	10	启用	修改	删除

要完成该页面，我们需要先对这个页面进行分析，看页面由哪儿几部分组成，然后到官网进行拷贝并修改。页面总共有如下组成部分



还有一个是当我们点击 **新增** 按钮，会在页面正中间弹出一个对话框，如下：



2.3.1 准备基本页面

```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>Title</title>
6 </head>
7 <body>
8 <div id="app">
9
10 </div>
11
12 <script src="js/vue.js"></script>
13 <script src="element-ui/lib/index.js"></script>
14 <link rel="stylesheet" href="element-ui/lib/theme-chalk/index.css">
15
16 <script>
17   new Vue({
18     el: "#app"
19   })
20 </script>
21 </body>
22 </html>

```

2.3.2 完成表格展示

使用 Element 整体的思路就是 **拷贝 + 修改**。

2.3.2.1 拷贝

Element 官方网站展示了如何使用 Table 组件。在左侧菜单栏，**Table 表格** 被选中并高亮显示。右侧主体区域展示了带有不同背景色（正常、警告、成功、危险）的表格示例，并提供了 **显示代码** 和 **在线运行** 的按钮。

日期	姓名	地址
2016-05-02	王小虎	上海市普陀区金沙江路 1518 弄
2016-05-04	王小虎	上海市普陀区金沙江路 1518 弄
2016-05-01	王小虎	上海市普陀区金沙江路 1518 弄
2016-05-03	王小虎	上海市普陀区金沙江路 1518 弄

在左菜单栏找到 **Table 表格** 并点击，右边主体就会定位到表格这一块，找到我们需要的表格效果（如上图），点击 **显示代码** 就可以看到这个表格的代码了。

将html标签拷贝到 `<div id="app"></div>` 中，如下：

```
<template>
  <el-table
    :data="tableData"
    style="width: 100%"
    :row-class-name="tableRowClassName">
    <el-table-column
      prop="date"
      label="日期"
      width="180">
    </el-table-column>
    <el-table-column
      prop="name"
      label="姓名"
      width="180">
    </el-table-column>
    <el-table-column
      prop="address"
      label="地址">
    </el-table-column>
  </el-table>
</template>
```

将css样式拷贝到我们页面的 `head` 标签中，如下

```
<style>
  .el-table .warning-row {
    background: oldlace;
  }

  .el-table .success-row {
    background: #f0f9eb;
  }
</style>
```

将方法和模型数据拷贝到 Vue 对象指定的位置

```
<script>
export default {
  methods: {
    tableRowClassName({row, rowIndex}) {
      if (rowIndex === 1) {
        return 'warning-row';
      } else if (rowIndex === 3) {
        return 'success-row';
      }
      return '';
    }
  },
  data() {
    return {
      tableData: [          模型数据
        {
          date: '2016-05-02',
          name: '王小虎',
          address: '上海市普陀区金沙江路 1518 弄',
        }, {
          date: '2016-05-04',
          name: '王小虎',
          address: '上海市普陀区金沙江路 1518 弄'
        }, {
          date: '2016-05-01',
          name: '王小虎',
          address: '上海市普陀区金沙江路 1518 弄',
        }, {
      ]
    }
  }
}
```

拷贝完成后通过浏览器打开可以看到表格的效果

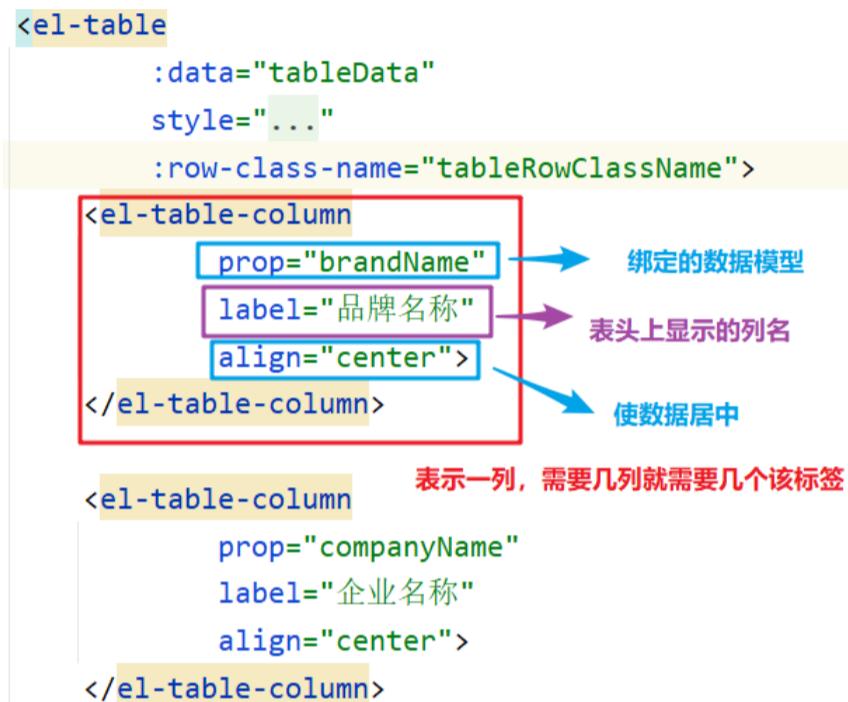
日期	姓名	地址
2016-05-02	王小虎	上海市普陀区金沙江路 1518 弄
2016-05-04	王小虎	上海市普陀区金沙江路 1518 弄
2016-05-01	王小虎	上海市普陀区金沙江路 1518 弄
2016-05-03	王小虎	上海市普陀区金沙江路 1518 弄

表格效果出来了，但是显示的表头和数据并不是我们想要的，所以接下来就需要对页面代码进行修改了。

2.3.2.2 修改

1. 修改表头和数据

下面是对表格代码进行分析的图解。根据下图说明修改自己的列数和列名



修改完页面后，还需要对绑定的数据模型进行修改，下图是对模型数据进行分析的图解



2. 给表格添加操作列

从之前的表格拷贝一列出来并对其进行修改。按钮是从官网的 button 按钮组件中拷贝并修改的



3. 给表格添加复选框列和标号列

给表格添加复选框和标号列，效果如下

	品牌名称	企业名称	排序	当前状态	操作
1	华为	华为科技有限公司	100	1	<button>修改</button> <button>删除</button>
2	华为	华为科技有限公司	100	1	<button>修改</button> <button>删除</button>
3	华为	华为科技有限公司	100	1	<button>修改</button> <button>删除</button>
4	华为	华为科技有限公司	100	1	<button>修改</button> <button>删除</button>

此效果也是从 Element 官网进行拷贝，先找到对应的表格效果，然后将其对应代码拷贝到我们的代码中，如下是复选框列官网效果图和代码

	日期	姓名	地址
<input type="checkbox"/>	2016-05-03	王小虎	上海市普陀区金沙江路 1518 弄
<input type="checkbox"/>	2016-05-02	王小虎	上海市普陀区金沙江路 1518 弄
<input type="checkbox"/>	2016-05-04	王小虎	上海市普陀区金沙江路 1518 弄

```
<template>
<el-table
  ref="multipleTable"
  :data="tableData"
  tooltip-effect="dark"
  style="width: 100%"
  @selection-change="handleSelectionChange">
  <el-table-column
    type="selection"
    width="55">
  </el-table-column>
  <el-table-column
    label="日期"
    width="120">
```

这里需要注意在 `<el-table>` 标签上有一个事件 `@selection-change="handleSelectionChange"`，这里绑定的函数也需要从官网拷贝到我们自己的页面代码中，函数代码如下：

```
handleSelectionChange(val) {
  this.multipleSelection = val;
}
```

从该函数中又发现还需要一个模型数据 `multipleSelection`，所以还需要定义出该模型数据
标号列也用同样的方式进行拷贝并修改。

2.3.3 完成搜索表单展示

在 Element 官网找到横排的表单效果，然后拷贝代码并进行修改

The screenshot shows the Element UI documentation page. The 'Form 表单' section is highlighted with a red box. Below it, a '显示代码' (Show Code) button is also highlighted with a red box.

点击上面的 `显示代码` 后，就会展示出对应的代码，下面是对这部分代码进行分析的图解

```

<el-form :inline="true" :model="formInline" class="demo-form-inline">
  <el-form-item label="审批人">
    <el-input v-model="formInline.user" placeholder="审批人"></el-input>
  </el-form-item>
  <el-form-item label="活动区域">
    <el-select v-model="formInline.region" placeholder="活动区域">
      <el-option label="区域一" value="shanghai"></el-option>
      <el-option label="区域二" value="beijing"></el-option>
    </el-select>
  </el-form-item>
  <el-form-item>
    <el-button type="primary" @click="onSubmit">查询</el-button>
  </el-form-item>
</el-form>
<script>
  export default {
    data() {
      return {
        formInline: {
          user: '',
          region: ''
        }
      }
    },
    methods: {
      onSubmit() {
        console.log('submit!');
      }
    }
  }
</script>

```

模型数据

绑定的函数

然后根据我们要的效果修改代码。

2.3.4 完成批量删除和新增按钮展示

从 Element 官网找具有着色效果的按钮，并将代码拷贝到我们自己的页面上

The screenshot shows the Element UI documentation for buttons. On the left, there's a sidebar with categories like Typography, Border, Icon, Button (which is highlighted with a red box), Link, Form, Radio, Checkbox, Input, InputNumber, Select, Cascader, Switch, Slider, TimePicker, and DatePicker. The main content area displays a grid of buttons with different styles: plain, primary, success, info, warning, and danger. Below the buttons, there's a note: "使用 type、plain、round 和 circle 属性来定义 Button 的样式。" (Use type, plain, round, and circle properties to define Button styles.) At the bottom, there are two rows of code examples:

```

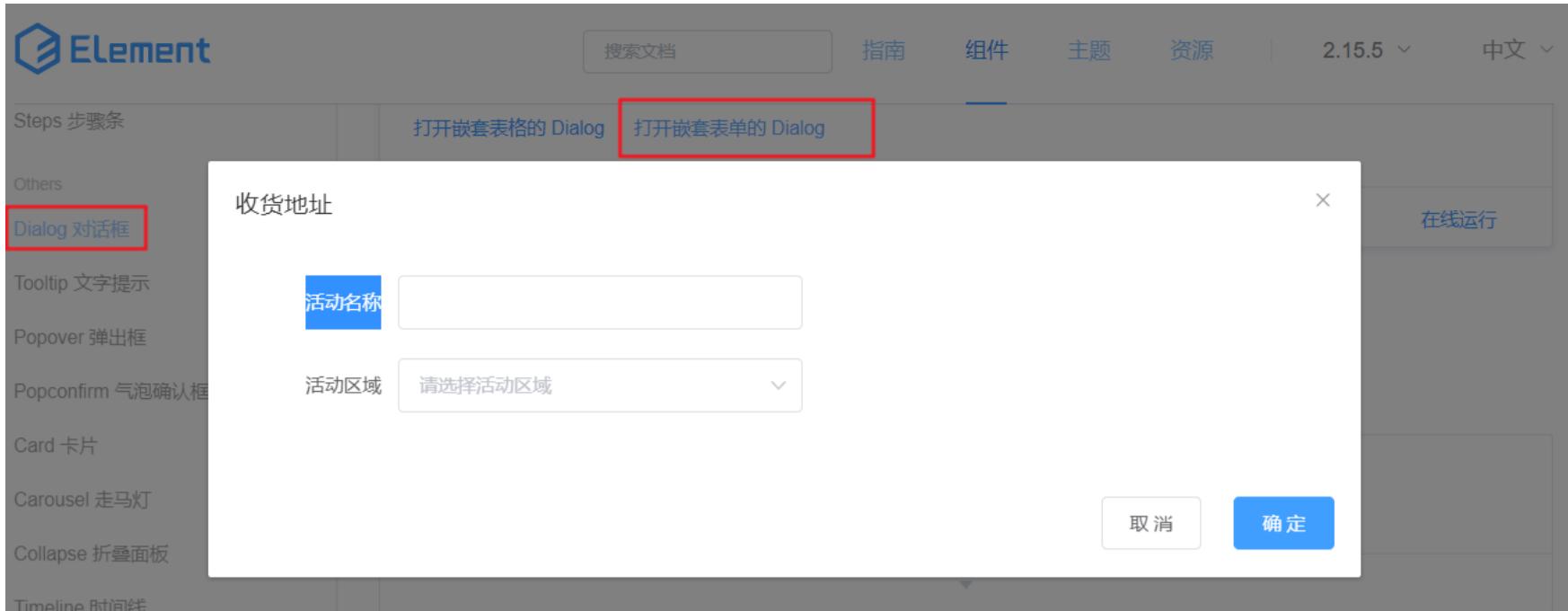
<el-row>
  <el-button>默认按钮</el-button>
  <el-button type="primary">主要按钮</el-button>
  <el-button type="success">成功按钮</el-button>
  <el-button type="info">信息按钮</el-button>
  <el-button type="warning">警告按钮</el-button>
  <el-button type="danger">危险按钮</el-button>
</el-row>

<el-row>
  <el-button plain>朴素按钮</el-button>
  <el-button type="primary" plain>主要按钮</el-button>
  <el-button type="success" plain>成功按钮</el-button>
  <el-button type="info" plain>信息按钮</el-button>
  <el-button type="warning" plain>警告按钮</el-button>
  <el-button type="danger" plain>危险按钮</el-button>
</el-row>

```

2.3.5 完成对话框展示

在 Element 官网找对话框，如下：



下面对官网提供的代码进行分析

```

<!-- Form -->
<el-button type="text" @click="dialogFormVisible = true">打开嵌套表单的 Dialog</el-button>

<el-dialog title="收货地址" :visible.sync="dialogFormVisible">
  <el-form :model="form">
    <el-form-item label="活动名称" :label-width="formLabelWidth">
      <el-input v-model="form.name" autocomplete="off"></el-input>
    </el-form-item>          绑定的模型数据
    <el-form-item label="活动区域" :label-width="formLabelWidth">
      <el-select v-model="form.region" placeholder="请选择活动区域">
        <el-option label="区域一" value="shanghai"></el-option>
        <el-option label="区域二" value="beijing"></el-option>
      </el-select>
    </el-form-item>
  </el-form>
  <div slot="footer" class="dialog-footer">
    <el-button @click="dialogFormVisible = false">取 消</el-button>
    <el-button type="primary" @click="dialogFormVisible = false">确 定</el-button>
  </div>                  嵌套标签的对话框
</el-dialog>

```

这个按钮绑定了单击事件，而该事件触发后，将 dialogFormVisible 模型数据的值置为 true，对话框就展示出来了。那这个模型数据的值如果是false，肯定是将对话框隐藏起来

表单效果根据要求自己进行修改

上图分析出来的模型数据需要在 Vue 对象中进行定义。

2.3.6 完成分页条展示

在 Element 官网找到 `Pagination 分页`，在页面主体部分找到我们需要的效果，如下

点击 `显示代码`，找到 `完整功能` 对应的代码，接下来对该代码进行分析

```

完整功能
<el-pagination
    @size-change="handleSizeChange"
    @current-change="handleCurrentChange"
    :current-page="currentPage4"
    :page-sizes="[100, 200, 300, 400]"
    :page-size="100"
    layout="total, sizes, prev, pager, next, jumper"
    :total="400">
</el-pagination>
</div>
</template>
<script>
export default {
  methods: {
    handleSizeChange(val) {
      console.log(`每页 ${val} 条`);
    },
    handleCurrentChange(val) {
      console.log(`当前页: ${val}`);
    }
  },
  data() {
    return {
      currentPage1: 5,
      currentPage2: 5,
      currentPage3: 5,
      currentPage4: 4
    };
  }
}

```

分页工具条

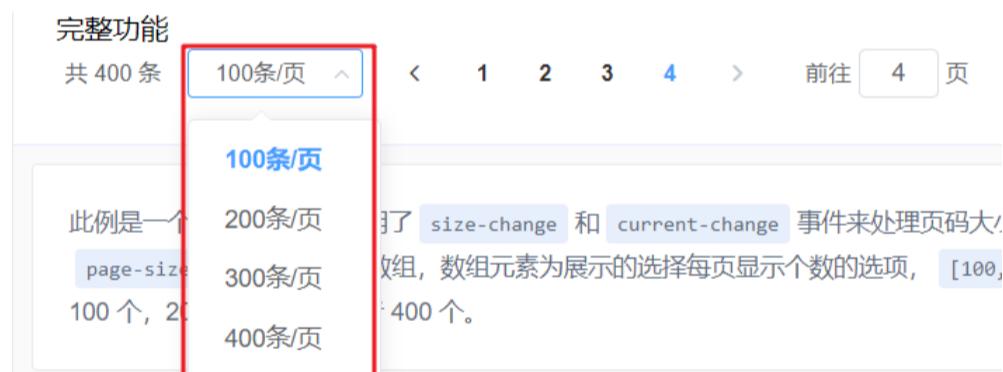
这个是 Element 定义的事件，绑定了两个函数，这两个函数需要在 Vue 中定义出来

这里绑定了 currentPage4 模型数据

上面代码属性说明：

- `page-size`：每页显示的条目数
- `page-sizes`：每页显示个数选择器的选项设置。

`:page-sizes="[100,200,300,400]"` 对应的页面效果如下：



- `currentPage`：当前页码。我们点击那个页码，此属性值就是几。
- `total`：总记录数。用来设置总的数据条目数，该属性设置后，Element 会自动计算出需分多少页并给我们展示对应的页码。

事件说明：

- `size-change`：`pageSize` 改变时会触发。也就是当我们改变了每页显示的条目数后，该事件会触发。
- `current-change`：`currentPage` 改变时会触发。也就是当我们点击了其他的页码后，该事件会触发。

2.3.7 完整页面代码

```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>Title</title>
6   <style>
7     .el-table .warning-row {
8       background: oldlace;
9     }
10    .el-table .success-row {
11       background: #f0f9eb;
12     }
13   </style>

```

```
14  </head>
15  <body>
16  <div id="app">
17      <!--搜索表单-->
18      <el-form :inline="true" :model="brand" class="demo-form-inline">
19          <el-form-item label="当前状态">
20              <el-select v-model="brand.status" placeholder="当前状态">
21                  <el-option label="启用" value="1"></el-option>
22                  <el-option label="禁用" value="0"></el-option>
23              </el-select>
24          </el-form-item>
25
26          <el-form-item label="企业名称">
27              <el-input v-model="brand.companyName" placeholder="企业名称"></el-input>
28          </el-form-item>
29
30          <el-form-item label="品牌名称">
31              <el-input v-model="brand.brandName" placeholder="品牌名称"></el-input>
32          </el-form-item>
33
34          <el-form-item>
35              <el-button type="primary" @click="onSubmit">查询</el-button>
36          </el-form-item>
37      </el-form>
38
39      <!--按钮-->
40      <el-row>
41          <el-button type="danger" plain>批量删除</el-button>
42          <el-button type="primary" plain @click="dialogVisible = true">新增</el-button>
43      </el-row>
44
45      <!--添加数据对话框表单-->
46      <el-dialog
47          title="编辑品牌"
48          :visible.sync="dialogVisible"
49          width="30%">
50          <el-form ref="form" :model="brand" label-width="80px">
51              <el-form-item label="品牌名称">
52                  <el-input v-model="brand.brandName"></el-input>
53              </el-form-item>
54
55              <el-form-item label="企业名称">
56                  <el-input v-model="brand.companyName"></el-input>
57              </el-form-item>
58
59              <el-form-item label="排序">
60                  <el-input v-model="brand.ordered"></el-input>
61              </el-form-item>
62
63              <el-form-item label="备注">
64                  <el-input type="textarea" v-model="brand.description"></el-input>
65              </el-form-item>
66
67              <el-form-item label="状态">
68                  <el-switch v-model="brand.status"
69                      active-value="1"
70                      inactive-value="0"
71                  ></el-switch>
72              </el-form-item>
73              <el-form-item>
74                  <el-button type="primary" @click="addBrand">提交</el-button>
75                  <el-button @click="dialogVisible = false">取消</el-button>
76              </el-form-item>
77          </el-form>
78      </el-dialog>
```

```
79      <!--表格-->
80      <template>
81          <el-table
82              :data="tableData"
83              style="width: 100%"
84              :row-class-name="tableRowClassName"
85              @selection-change="handleSelectionChange">
86              <el-table-column
87                  type="selection"
88                  width="55">
89              </el-table-column>
90              <el-table-column
91                  type="index"
92                  width="50">
93              </el-table-column>
94              <el-table-column
95                  prop="brandName"
96                  label="品牌名称"
97                  align="center">
98              </el-table-column>
99              <el-table-column
100                 prop="companyName"
101                 label="企业名称"
102                 align="center">
103             </el-table-column>
104             <el-table-column
105                 prop="ordered"
106                 align="center"
107                 label="排序">
108             </el-table-column>
109             <el-table-column
110                 prop="status"
111                 align="center"
112                 label="当前状态">
113             </el-table-column>
114             <el-table-column
115                 align="center"
116                 label="操作">
117                 <el-row>
118                     <el-button type="primary">修改</el-button>
119                     <el-button type="danger">删除</el-button>
120                 </el-row>
121             </el-table-column>
122         </el-table>
123     </template>
124
125     <!--分页工具条-->
126     <el-pagination
127         @size-change="handleSizeChange"
128         @current-change="handleCurrentChange"
129         :current-page="currentPage"
130         :page-sizes="[5, 10, 15, 20]"
131         :page-size="5"
132         layout="total, sizes, prev, pager, next, jumper"
133         :total="400">
134     </el-pagination>
135
136     </div>
137     <script src="js/vue.js"></script>
138     <script src="element-ui/lib/index.js"></script>
139     <link rel="stylesheet" href="element-ui/lib/theme-chalk/index.css">
140     <script>
141         new Vue({
142             el: '#app',
143             data: {
144                 tableData: [
145                     {
146                         id: 1,
147                         brandName: '华为',
148                         companyName: '华为技术有限公司',
149                         ordered: 1,
150                         status: '正常'
151                     },
152                     {
153                         id: 2,
154                         brandName: '小米',
155                         companyName: '小米集团',
156                         ordered: 2,
157                         status: '正常'
158                     },
159                     {
160                         id: 3,
161                         brandName: '荣耀',
162                         companyName: '荣耀有限公司',
163                         ordered: 3,
164                         status: '正常'
165                     },
166                     {
167                         id: 4,
168                         brandName: 'OPPO',
169                         companyName: 'OPPO 科技有限公司',
170                         ordered: 4,
171                         status: '正常'
172                     },
173                     {
174                         id: 5,
175                         brandName: 'vivo',
176                         companyName: 'Vivo 科技有限公司',
177                         ordered: 5,
178                         status: '正常'
179                     }
180                 ]
181             }
182         })
183     </script>
```

```
144     el: "#app",
145     methods: {
146       tableRowClassName({row, rowIndex}) {
147         if (rowIndex === 1) {
148           return 'warning-row';
149         } else if (rowIndex === 3) {
150           return 'success-row';
151         }
152         return '';
153       },
154       // 复选框选中后执行的方法
155       handleSelectionChange(val) {
156         this.multipleSelection = val;
157
158         console.log(this.multipleSelection)
159       },
160       // 查询方法
161       onSubmit() {
162         console.log(this.brand);
163       },
164       // 添加数据
165       addBrand(){
166         console.log(this.brand);
167       },
168       //分页
169       handleSizeChange(val) {
170         console.log(`每页 ${val} 条`);
171       },
172       handleCurrentChange(val) {
173         console.log(`当前页: ${val}`);
174       }
175     },
176     data() {
177       return {
178         // 当前页码
179         currentPage: 4,
180         // 添加数据对话框是否展示的标记
181         dialogVisible: false,
182
183         // 品牌模型数据
184         brand: {
185           status: '',
186           brandName: '',
187           companyName: '',
188           id:'',
189           ordered:'',
190           description:''
191         },
192         // 复选框选中数据集合
193         multipleSelection: [],
194         // 表格数据
195         tableData: [
196           {
197             brandName: '华为',
198             companyName: '华为科技有限公司',
199             ordered: '100',
200             status: "1"
201           },
202           {
203             brandName: '华为',
204             companyName: '华为科技有限公司',
205             ordered: '100',
206             status: "1"
207           },
208           {
209             brandName: '华为',
210             companyName: '华为科技有限公司',
211             ordered: '100',
212           }
213         ]
214       }
215     }
216   }
217 }
```

```

209         status: "1"
210     },
211     {
212         brandName: '华为',
213         companyName: '华为科技有限公司',
214         ordered: '100',
215         status: "1"
216     }
217 }
218 })
219 </script>
220 </body>
221 </html>

```

3. 综合案例

3.1 功能介绍

功能列表：

当前状态		当前状态	企业名称	企业名称	品牌名称	品牌名称	查询
批量删除		新增					
		品牌名称	企业名称	排序	当前状态	操作	
	<input type="checkbox"/>	1 华为	华为科技有限公司	100	1	修改	删除
	<input checked="" type="checkbox"/>	2 华为	华为科技有限公司	100	1	修改	删除
	<input type="checkbox"/>	3 华为	华为科技有限公司	100	1	修改	删除
	<input type="checkbox"/>	4 华为	华为科技有限公司	100	1	修改	删除
共 400 条		5条/页	< 1 2 3 4 5 6 ... 80 >	前往 4 页			

以上是我们在综合案例要实现的功能。对数据的除了对数据的增删改查功能外，还有一些复杂的功能，如 [批量删除](#)、[分页查询](#)、[条件查询](#) 等功能

- [批量删除](#) 功能：每条数据前都有复选框，当我选中多条数据并点击 [批量删除](#) 按钮后，会发送请求到后端并删除数据库中指定的多条数据。
- [分页查询](#) 功能：当数据库中有很多数据时，我们不可能将所有的数据展示在一页里，这个时候就需要分页展示数据。
- [条件查询](#) 功能：数据库量大的时候，我们就需要精确的查询一些想看到的数据，这个时候就需要通过条件查询。

这里的 [修改品牌](#) 和 [删除品牌](#) 功能在课程上不做讲解，留作同学来下的练习。

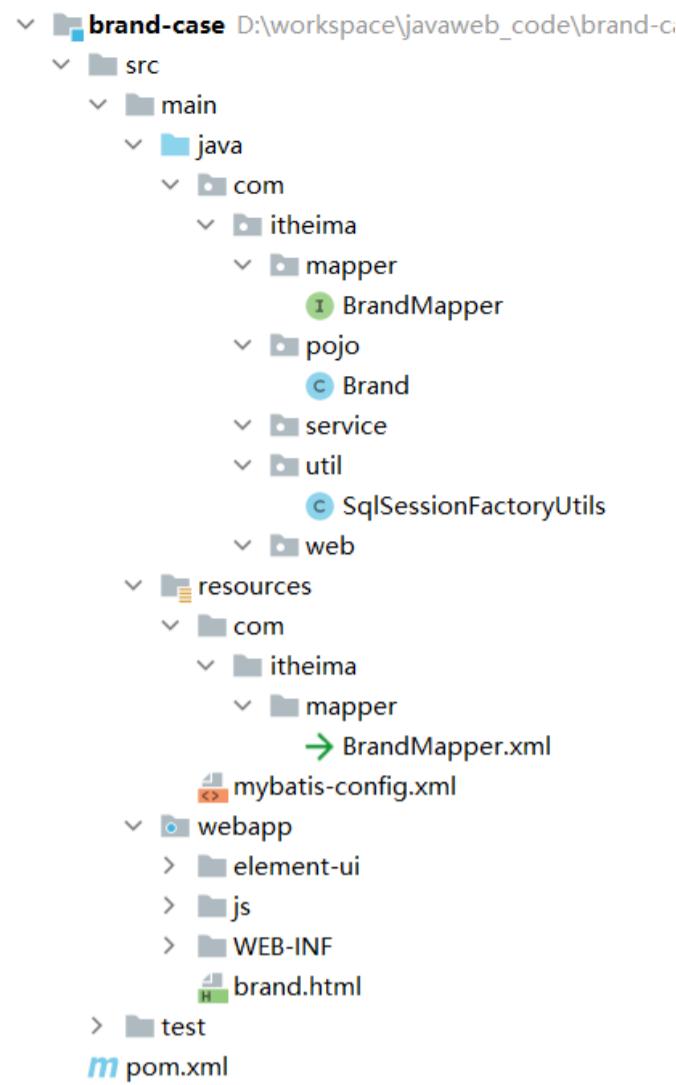
3.2 环境准备

环境准备我们主要完成以下两件事即可

- 将资料的 brand-case 模块导入到 idea 中
- 执行资料中提供的 tb_brand.sql 脚本

3.2.1 工程准备

将 [04-资料\01-初始工程](#) 中的 [brand-case](#) 工程导入到我们自己的 idea 中。工程结构如下：



3.2.2 创建表

下面是创建表的语句

```
1 -- 删除tb_brand表
2 drop table if exists tb_brand;
3 -- 创建tb_brand表
4 create table tb_brand (
5     -- id 主键
6     id          int primary key auto_increment,
7     -- 品牌名称
8     brand_name  varchar(20),
9     -- 企业名称
10    company_name varchar(20),
11    -- 排序字段
12    ordered      int,
13    -- 描述信息
14    description  varchar(100),
15    -- 状态: 0: 禁用 1: 启用
16    status       int
17 );
18 -- 添加数据
19 insert into tb_brand (brand_name, company_name, ordered, description, status)
20 values
21     ('华为', '华为技术有限公司', 100, '万物互联', 1),
22     ('小米', '小米科技有限公司', 50, 'are you ok', 1),
23     ('格力', '格力电器股份有限公司', 30, '让世界爱上中国造', 1),
24     ('阿里巴巴', '阿里巴巴集团控股有限公司', 10, '买买买', 1),
25     ('腾讯', '腾讯计算机系统有限公司', 50, '玩玩玩', 0),
26     ('百度', '百度在线网络技术公司', 5, '搜搜搜', 0),
27     ('京东', '北京京东世纪贸易有限公司', 40, '就是快', 1),
28     ('小米', '小米科技有限公司', 50, 'are you ok', 1),
29     ('三只松鼠', '三只松鼠股份有限公司', 5, '好吃不上火', 0),
30     ('华为', '华为技术有限公司', 100, '万物互联', 1),
31     ('小米', '小米科技有限公司', 50, 'are you ok', 1),
32     ('格力', '格力电器股份有限公司', 30, '让世界爱上中国造', 1),
33     ('阿里巴巴', '阿里巴巴集团控股有限公司', 10, '买买买', 1),
34     ('腾讯', '腾讯计算机系统有限公司', 50, '玩玩玩', 0),
35     ('百度', '百度在线网络技术公司', 5, '搜搜搜', 0),
36     ('京东', '北京京东世纪贸易有限公司', 40, '就是快', 1),
37     ('华为', '华为技术有限公司', 100, '万物互联', 1),
```

```

38      ('小米', '小米科技有限公司', 50, 'are you ok', 1),
39      ('格力', '格力电器股份有限公司', 30, '让世界爱上中国造', 1),
40      ('阿里巴巴', '阿里巴巴集团控股有限公司', 10, '买买买', 1),
41      ('腾讯', '腾讯计算机系统有限公司', 50, '玩玩玩', 0),
42      ('百度', '百度在线网络技术公司', 5, '搜搜搜', 0),
43      ('京东', '北京京东世纪贸易有限公司', 40, '就是快', 1),
44      ('小米', '小米科技有限公司', 50, 'are you ok', 1),
45      ('三只松鼠', '三只松鼠股份有限公司', 5, '好吃不上火', 0),
46      ('华为', '华为技术有限公司', 100, '万物互联', 1),
47      ('小米', '小米科技有限公司', 50, 'are you ok', 1),
48      ('格力', '格力电器股份有限公司', 30, '让世界爱上中国造', 1),
49      ('阿里巴巴', '阿里巴巴集团控股有限公司', 10, '买买买', 1),
50      ('腾讯', '腾讯计算机系统有限公司', 50, '玩玩玩', 0),
51      ('百度', '百度在线网络技术公司', 5, '搜搜搜', 0),
52      ('京东', '北京京东世纪贸易有限公司', 40, '就是快', 1),
53      ('华为', '华为技术有限公司', 100, '万物互联', 1),
54      ('小米', '小米科技有限公司', 50, 'are you ok', 1),
55      ('格力', '格力电器股份有限公司', 30, '让世界爱上中国造', 1),
56      ('阿里巴巴', '阿里巴巴集团控股有限公司', 10, '买买买', 1),
57      ('腾讯', '腾讯计算机系统有限公司', 50, '玩玩玩', 0),
58      ('百度', '百度在线网络技术公司', 5, '搜搜搜', 0),
59      ('京东', '北京京东世纪贸易有限公司', 40, '就是快', 1),
60      ('小米', '小米科技有限公司', 50, 'are you ok', 1),
61      ('三只松鼠', '三只松鼠股份有限公司', 5, '好吃不上火', 0),
62      ('华为', '华为技术有限公司', 100, '万物互联', 1),
63      ('小米', '小米科技有限公司', 50, 'are you ok', 1),
64      ('格力', '格力电器股份有限公司', 30, '让世界爱上中国造', 1),
65      ('阿里巴巴', '阿里巴巴集团控股有限公司', 10, '买买买', 1),
66      ('腾讯', '腾讯计算机系统有限公司', 50, '玩玩玩', 0),
67      ('百度', '百度在线网络技术公司', 5, '搜搜搜', 0),
68      ('京东', '北京京东世纪贸易有限公司', 40, '就是快', 1);

```

3.3 查询所有功能

当前状态	当前状态	企业名称	企业名称	品牌名称	品牌名称	查询	
		批量删除	新增				
		操作	操作	操作	操作	操作	操作
□	品牌名称	企业名称	排序	当前状态	操作	操作	操作
□	1 华为	华为科技有限公司	100	1	<button>修改</button>	<button>删除</button>	
□	2 华为	华为科技有限公司	100	1	<button>修改</button>	<button>删除</button>	
□	3 华为	华为科技有限公司	100	1	<button>修改</button>	<button>删除</button>	
□	4 华为	华为科技有限公司	100	1	<button>修改</button>	<button>删除</button>	

共 400 条 5条/页 < 1 2 3 4 5 6 ⋯ 80 > 前往 4 页

如上图所示是查询所有品牌数据在页面展示的效果。要实现这个功能，要先搞明白如下问题：

- 什么时候发送异步请求？

页面加载完毕后就需要在页面上看到所有的品牌数据。所以在 `mounted()` 这个构造函数中写发送异步请求的代码。

- 请求需要携带参数吗？

查询所有功能不需要携带什么参数。

- 响应的数据格式是什么样？

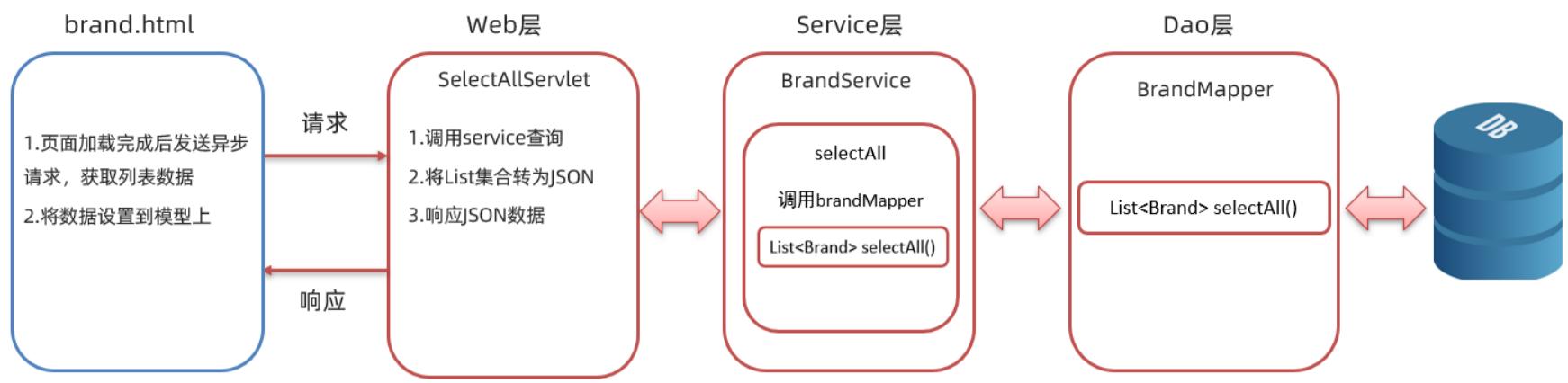
后端是需要将 `List<Brand>` 对象转换为 JSON 格式的数据并响应回给浏览器。响应数据格式如下：

```

[{"brandName": "华为", "companyName": "华为技术有限公司", "description": "万物互联", "id": 1, "ordered": 100, "status": 1, "statusStr": "启用"},  
 {"brandName": "小米", "companyName": "小米科技有限公司", "description": "are you ok", "id": 2, "ordered": 50, "status": 1, "statusStr": "启用"},  
 {"brandName": "格力", "companyName": "格力电器股份有限公司", "description": "让世界爱上中国造", "id": 3, "ordered": 30, "status": 1, "statusStr": "启用"}]

```

整体流程如下



我们先实现后端程序，然后再实现前端程序。

3.3.1 后端实现

3.3.1.1 dao方法实现

在 `com.itheima.mapper.BrandMapper` 接口中定义抽象方法，并使用 `@Select` 注解编写 sql 语句

```

1 /**
2  * 查询所有
3  * @return
4 */
5 @Select("select * from tb_brand")
6 List<Brand> selectAll();

```

由于表中有些字段名和实体类中的属性名没有对应，所以需要在 `com/itheima/mapper/BrandMapper.xml` 映射配置文件中定义结果映射，使用 `resultMap` 标签。映射配置文件内容如下：

```

1 <?xml version="1.0" encoding="UTF-8" ?>
2 <!DOCTYPE mapper
3   PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
4   "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
5 <mapper namespace="com.itheima.mapper.BrandMapper">
6
7   <resultMap id="brandResultMap" type="brand">
8     <result property="brandName" column="brand_name" />
9     <result property="companyName" column="company_name" />
10    </resultMap>
11 </mapper>

```

定义完结果映射关系后，在接口 `selectAll()` 方法上引用该结构映射。使用 `@ResultMap("brandResultMap")` 注解完整接口的 `selectAll()` 方法如下：

```

1 /**
2  * 查询所有
3  * @return
4 */
5 @Select("select * from tb_brand")
6 @ResultMap("brandResultMap")
7 List<Brand> selectAll();

```

3.3.1.2 service方法实现

在 `com.itheima.service` 包下创建 `BrandService` 接口，在该接口中定义查询所有的抽象方法

```

1 public interface BrandService {
2
3   /**
4    * 查询所有
5    * @return
6   */
7   List<Brand> selectAll();
8 }

```

并在 `com.itheima.service` 下再创建 `impl` 包; `impl` 表示是放 service 层接口的实现类的包。在该包下创建名为 `BrandServiceImpl` 类

```
1 public class BrandServiceImpl implements BrandService {  
2  
3     @Override  
4     public List<Brand> selectAll() {  
5         }  
6 }
```

此处为什么要给 service 定义接口呢? 因为 service 定义了接口后, 在 servlet 中就可以使用多态的形式创建 Service 实现类的对象, 如下:

```
@WebServlet("/selectAllServlet")  
public class SelectAllServlet extends HttpServlet {  
  
    private BrandService brandService = new BrandServiceImpl();  
}
```

这里使用多态是因为方便我们后期解除 `Servlet` 和 `service` 的耦合。从上面的代码我们可以看到 `SelectAllServlet` 类和 `BrandServiceImpl` 类之间是耦合在一起的, 如果后期 `BrandService` 有其它更好的实现类(例如叫 `BrandServiceImp1`), 那就需要修改 `SelectAllServlet` 类中的代码。后面我们学习了 `Spring` 框架后就可以解除 `SelectAllServlet` 类和红色框括起来的代码耦合。而现在咱们还做不到解除耦合, 在这里只需要理解为什么定义接口即可。

`BrandServiceImpl` 类代码如下:

```
1 public class BrandServiceImpl implements BrandService {  
2     //1. 创建SqlSessionFactory 工厂对象  
3     SqlSessionFactory factory = SqlSessionFactoryUtils.getSqlSessionFactory();  
4  
5     @Override  
6     public List<Brand> selectAll() {  
7         //2. 获取SqlSession对象  
8         SqlSession sqlSession = factory.openSession();  
9         //3. 获取BrandMapper  
10        BrandMapper mapper = sqlSession.getMapper(BrandMapper.class);  
11  
12        //4. 调用方法  
13        List<Brand> brands = mapper.selectAll();  
14  
15        //5. 释放资源  
16        sqlSession.close();  
17  
18        return brands;  
19    }  
20 }
```

3.3.1.3 servlet 实现

在 `com.itheima.web.servlet` 包下定义名为 `SelectAllServlet` 的查询所有的 `servlet`。该 `servlet` 逻辑如下:

- 调用 service 的 `selectAll()` 方法查询所有的品牌数据, 并接口返回结果
- 将返回的结果转换为 json 数据
- 响应 json 数据

代码如下:

```
1 @WebServlet("/selectAllServlet")  
2 public class SelectAllServlet extends HttpServlet {  
3  
4     private BrandService brandService = new BrandServiceImpl();  
5  
6     @Override
```

```

7     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
8         ServletException, IOException {
9             //1. 调用service查询
10            List<Brand> brands = brandService.selectAll();
11            //2. 转为JSON
12            String jsonString = JSON.toJSONString(brands);
13            //3. 写数据
14            response.setContentType("text/json; charset=utf-8"); //告知浏览器响应的数据是什么， 告知浏览器
使用什么字符集进行解码
15            response.getWriter().write(jsonString);
16        }
17
18    @Override
19    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
20        this.doGet(request, response);
21    }

```

3.3.1.4 测试后端程序

在浏览器输入访问 servlet 的资源路径 `http://localhost:8080/brand-case/selectAllServlet`，如果没有报错，并能看到如下信息表明后端程序没有问题



3.3.2 前端实现

前端需要在页面加载完毕后发送 ajax 请求，所以发送请求的逻辑应该放在 `mounted()` 钩子函数中。而响应回来的数据需要赋值给表格绑定的数据模型，从下图可以看出表格绑定的数据模型是 `tableData`

```

<el-table
    :data="tableData"
    style="..."
    :row-class-name="tableRowClassName"
    @selection-change="handleSelectionChange">
    <el-table-column

```

前端代码如下：

```

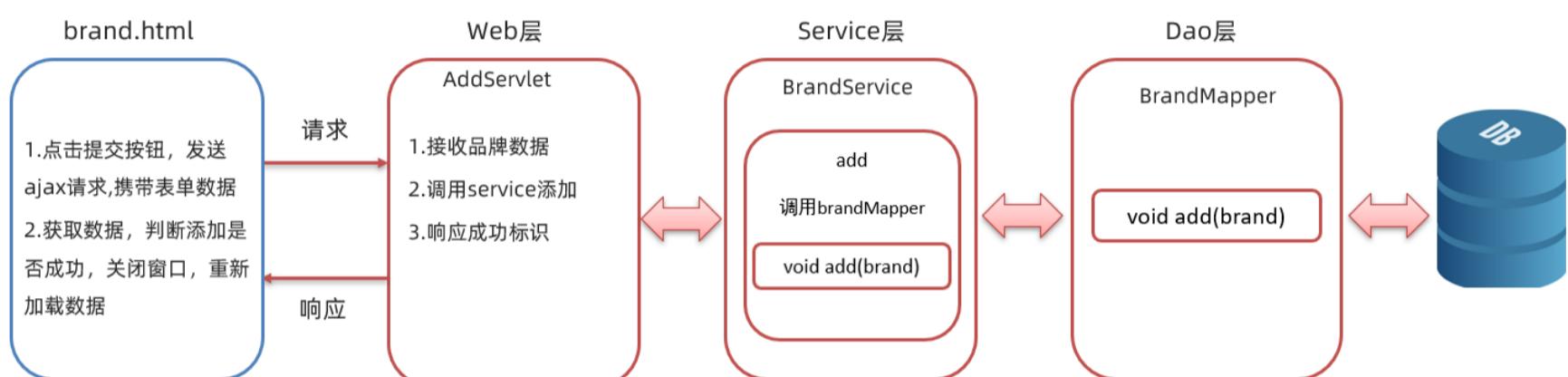
1 mounted() {
2     //当页面加载完成后，发送异步请求，获取数据
3     var _this = this;
4
5     axios({
6         method: "get",
7         url: "http://localhost:8080/brand-case/selectAllServlet"
8     }).then(function (resp) {
9         _this.tableData = resp.data;
10    })
11 }

```

3.4 添加功能



上图是添加数据的对话框，当点击 提交 按钮后就需要将数据提交到后端，并将数据保存到数据库中。下图是整体的流程：



页面发送请求时，需要将输入框输入的内容提交给后端程序，而这里是以 json 格式进行传递的。而具体的数据格式如下：

```
{"status": "1", "brandName": "鸿星尔克", "companyName": "鸿星尔克", "id": "", "ordered": "200", "description": "to be no.1"}
```

注意：由于是添加数据，所以上述json数据中id是没有值的。

3.4.1 后端实现

3.4.1.1 dao方法实现

在 `BrandMapper` 接口中定义 `add()` 添加方法，并使用 `@Insert` 注解编写sql语句

```

1 /**
2  * 添加数据
3  * @param brand
4 */
5 @Insert("insert into tb_brand values(null,#{brandName},#{companyName},#{ordered},#
6 void add(Brand brand);

```

3.4.1.2 service方法实现

在 `BrandService` 接口中定义 `add()` 添加数据的业务逻辑方法

```

1 /**
2  * 添加数据
3  * @param brand
4 */
5 void add(Brand brand);

```

在 `BrandServiceImpl` 类中重写 `add()` 方法，并进行业务逻辑实现

```

1 @Override
2 public void add(Brand brand) {
3     //2. 获取SqlSession对象
4     SqlSession sqlSession = factory.openSession();
5     //3. 获取BrandMapper
6     BrandMapper mapper = sqlSession.getMapper(BrandMapper.class);

```

```
7  
8     //4. 调用方法  
9     mapper.add(brand);  
10    sqlSession.commit(); //提交事务  
11  
12    //5. 释放资源  
13    sqlSession.close();  
14 }
```

注意：增删改操作一定要提交事务。

3.4.1.3 servlet实现

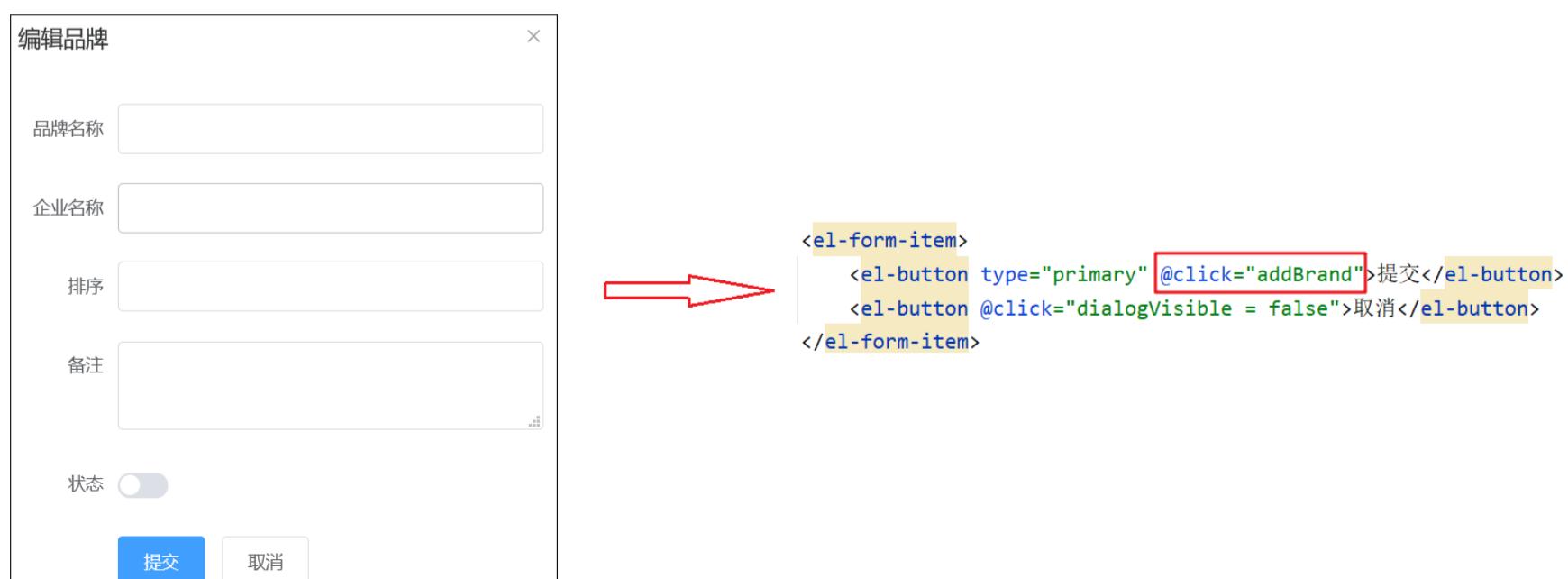
在 `com.itheima.web.servlet` 包写定义名为 `AddServlet` 的 Servlet。该 Servlet 的逻辑如下：

- 接收页面提交的数据。页面到时候提交的数据是 json 格式的数据，所以此处需要使用输入流读取数据
- 将接收到的数据转换为 `Brand` 对象
- 调用 service 的 `add()` 方法进行添加的业务逻辑处理
- 给浏览器响应添加成功的标识，这里直接给浏览器响应 `success` 字符串表示成功

servlet 代码实现如下：

```
1 @WebServlet("/addServlet")  
2 public class AddServlet extends HttpServlet {  
3  
4     private BrandService brandService = new BrandServiceImpl();  
5  
6     @Override  
7     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws  
ServletException, IOException {  
8  
9         //1. 接收品牌数据  
10        BufferedReader br = request.getReader();  
11        String params = br.readLine(); //json字符串  
12        //转为Brand对象  
13        Brand brand = JSON.parseObject(params, Brand.class);  
14        //2. 调用service添加  
15        brandService.add(brand);  
16        //3. 响应成功的标识  
17        response.getWriter().write("success");  
18    }  
19  
20    @Override  
21    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws  
ServletException, IOException {  
22        this.doGet(request, response);  
23    }  
24 }
```

3.4.2 前端实现



上图左边是页面效果，里面的 提交 按钮可以通过上图右边看出绑定了一个单击事件，而该事件绑定的是 addBrand 函数，所以添加数据功能的逻辑代码应该写在 addBrand() 函数中。在此方法中需要发送异步请求并将表单中输入的数据作为参数进行传递。如下

```
1 // 添加数据
2 addBrand() {
3     var _this = this;
4
5     // 发送ajax请求，添加数据
6     axios({
7         method:"post",
8         url:"http://localhost:8080/brand-case/addServlet",
9         data:_this.brand
10    }).then(function (resp) {
11        //响应数据的处理逻辑
12    })
13 }
```

在 then 函数中的匿名函数是成功后的回调函数，而 resp.data 就可以获取到响应回来的数据，如果值是 success 表示数据添加成功。成功后我们需要做一下逻辑处理：

1. 关闭新增对话框窗口

如下图所示是添加数据的对话框代码，从代码中可以看到此对话框绑定了 dialogVisible 数据模型，只需要将该数据模型的值设置为 false，就可以关闭新增对话框窗口了。

```
<!-- 添加数据对话框表单-->
<el-dialog
    title="编辑品牌"
    :visible.sync="dialogVisible"
    width="30%">
```

2. 重新查询数据

数据添加成功与否，用户只要能在页面上查看到数据说明添加成功。而此处需要重新发送异步请求获取所有的品牌数据，而这段代码在 查询所有 功能中已经实现，所以我们可以将此功能代码进行抽取，抽取到一个 selectAll() 函数中

```
1 // 查询所有数据
2 selectAll(){
3     var _this = this;
4
5     axios({
6         method:"get",
7         url:"http://localhost:8080/brand-case/selectAllServlet"
8     }).then(function (resp) {
9         _this.tableData = resp.data;
10    })
11 }
```

那么就需要将 mounted() 钩子函数中代码改进为

```
1 mounted(){
2     //当页面加载完成后，发送异步请求，获取数据
3     this.selectAll();
4 }
```

同时在新增响应的回调中调用 selectAll() 进行数据的重新查询。

3. 弹出消息给用户提示添加成功

```
this.$message({
    message: '恭喜你，这是一条成功消息',
    type: 'success'
});
```



恭喜你，这是一条成功消息

上图左边就是 elementUI 官网提供的成功提示代码，而上图右边是具体的效果。

注意：上面的this需要的是表示 VUE 对象的this。

综上所述，前端代码如下：

```
1 // 添加数据
2 addBrand() {
3     var _this = this;
4
5     // 发送ajax请求，添加数据
6     axios({
7         method:"post",
8         url:"http://localhost:8080/brand-case/addservlet",
9         data:_this.brand
10    }).then(function (resp) {
11        if(resp.data == "success"){
12            //添加成功
13            //关闭窗口
14            _this.dialogVisible = false;
15            // 重新查询数据
16            _this.selectAll();
17            // 弹出消息提示
18            _this.$message({
19                message: '恭喜你，添加成功',
20                type: 'success'
21            });
22        }
23    })
24 }
```

综合案例

今日目标：

- 能够完成查询所有功能
- 能够完成添加功能
- 能够理解 BaseServlet 思想
- 能够完成批量删除功能
- 能够完成分页查询功能
- 能够完成条件查询功能

1. 功能介绍

功能列表：

1. 查询所有	批量删除	新增
2. 新增品牌	<input type="checkbox"/>	品牌名称
3. 修改品牌	<input type="checkbox"/> 1	华为
4. 删除品牌	<input type="checkbox"/> 2	华为
5. 批量删除	<input type="checkbox"/> 3	华为
6. 分页查询	<input type="checkbox"/> 4	华为
7. 条件查询		

以上是我们在综合案例要实现的功能。对数据的除了对数据的增删改查功能外，还有一些复杂的功能，如 [批量删除](#)、[分页查询](#)、[条件查询](#) 等功能

- [批量删除](#) 功能：每条数据前都有复选框，当我选中多条数据并点击 [批量删除](#) 按钮后，会发送请求到后端并删除数据库中指定的多条数据。
- [分页查询](#) 功能：当数据库中有很多数据时，我们不可能将所有的数据展示在一页里，这个时候就需要分页展示数据。
- [条件查询](#) 功能：数据库量大的时候，我们就需要精确的查询一些想看到的数据，这个时候就需要通过条件查询。

这里的 [修改品牌](#) 和 [删除品牌](#) 功能在课程上不做讲解，留作同学来下的练习。

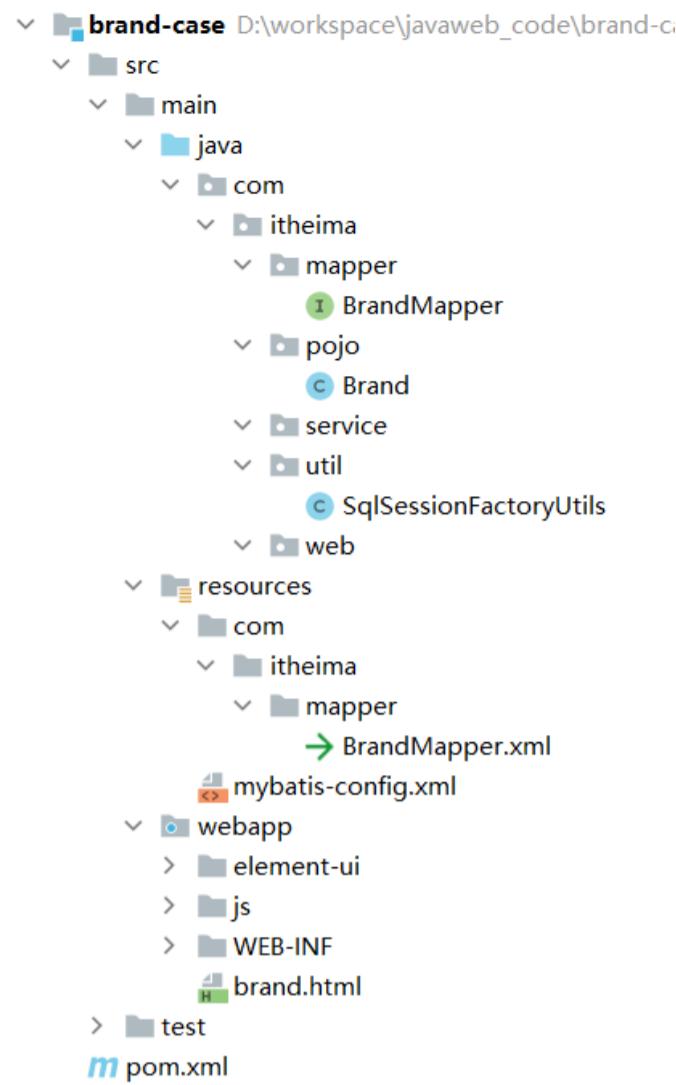
2. 环境准备

环境准备我们主要完成以下两件事即可

- 将资料的 brand-case 模块导入到 idea 中
- 执行资料中提供的 tb_brand.sql 脚本

2.1 工程准备

将 [04-资料\01-初始工程](#) 中的 [brand-case](#) 工程导入到我们自己的 idea 中。工程结构如下：



2.2 创建表

下面是创建表的语句

```
1 -- 删除tb_brand表
2 drop table if exists tb_brand;
3 -- 创建tb_brand表
4 create table tb_brand (
5     -- id 主键
6     id      int primary key auto_increment,
7     -- 品牌名称
8     brand_name  varchar(20),
9     -- 企业名称
10    company_name varchar(20),
11    -- 排序字段
12    ordered      int,
13    -- 描述信息
14    description  varchar(100),
15    -- 状态: 0: 禁用 1: 启用
16    status       int
17 );
18 -- 添加数据
19 insert into tb_brand (brand_name, company_name, ordered, description, status)
20 values
21     ('华为', '华为技术有限公司', 100, '万物互联', 1),
22     ('小米', '小米科技有限公司', 50, 'are you ok', 1),
23     ('格力', '格力电器股份有限公司', 30, '让世界爱上中国造', 1),
24     ('阿里巴巴', '阿里巴巴集团控股有限公司', 10, '买买买', 1),
25     ('腾讯', '腾讯计算机系统有限公司', 50, '玩玩玩', 0),
26     ('百度', '百度在线网络技术公司', 5, '搜搜搜', 0),
27     ('京东', '北京京东世纪贸易有限公司', 40, '就是快', 1),
28     ('小米', '小米科技有限公司', 50, 'are you ok', 1),
29     ('三只松鼠', '三只松鼠股份有限公司', 5, '好吃不上火', 0),
30     ('华为', '华为技术有限公司', 100, '万物互联', 1),
31     ('小米', '小米科技有限公司', 50, 'are you ok', 1),
32     ('格力', '格力电器股份有限公司', 30, '让世界爱上中国造', 1),
33     ('阿里巴巴', '阿里巴巴集团控股有限公司', 10, '买买买', 1),
34     ('腾讯', '腾讯计算机系统有限公司', 50, '玩玩玩', 0),
35     ('百度', '百度在线网络技术公司', 5, '搜搜搜', 0),
36     ('京东', '北京京东世纪贸易有限公司', 40, '就是快', 1),
37     ('华为', '华为技术有限公司', 100, '万物互联', 1),
```

```

38 ('小米', '小米科技有限公司', 50, 'are you ok', 1),
39 ('格力', '格力电器股份有限公司', 30, '让世界爱上中国造', 1),
40 ('阿里巴巴', '阿里巴巴集团控股有限公司', 10, '买买买', 1),
41 ('腾讯', '腾讯计算机系统有限公司', 50, '玩玩玩', 0),
42 ('百度', '百度在线网络技术公司', 5, '搜搜搜', 0),
43 ('京东', '北京京东世纪贸易有限公司', 40, '就是快', 1),
44 ('小米', '小米科技有限公司', 50, 'are you ok', 1),
45 ('三只松鼠', '三只松鼠股份有限公司', 5, '好吃不上火', 0),
46 ('华为', '华为技术有限公司', 100, '万物互联', 1),
47 ('小米', '小米科技有限公司', 50, 'are you ok', 1),
48 ('格力', '格力电器股份有限公司', 30, '让世界爱上中国造', 1),
49 ('阿里巴巴', '阿里巴巴集团控股有限公司', 10, '买买买', 1),
50 ('腾讯', '腾讯计算机系统有限公司', 50, '玩玩玩', 0),
51 ('百度', '百度在线网络技术公司', 5, '搜搜搜', 0),
52 ('京东', '北京京东世纪贸易有限公司', 40, '就是快', 1),
53 ('华为', '华为技术有限公司', 100, '万物互联', 1),
54 ('小米', '小米科技有限公司', 50, 'are you ok', 1),
55 ('格力', '格力电器股份有限公司', 30, '让世界爱上中国造', 1),
56 ('阿里巴巴', '阿里巴巴集团控股有限公司', 10, '买买买', 1),
57 ('腾讯', '腾讯计算机系统有限公司', 50, '玩玩玩', 0),
58 ('百度', '百度在线网络技术公司', 5, '搜搜搜', 0),
59 ('京东', '北京京东世纪贸易有限公司', 40, '就是快', 1),
60 ('小米', '小米科技有限公司', 50, 'are you ok', 1),
61 ('三只松鼠', '三只松鼠股份有限公司', 5, '好吃不上火', 0),
62 ('华为', '华为技术有限公司', 100, '万物互联', 1),
63 ('小米', '小米科技有限公司', 50, 'are you ok', 1),
64 ('格力', '格力电器股份有限公司', 30, '让世界爱上中国造', 1),
65 ('阿里巴巴', '阿里巴巴集团控股有限公司', 10, '买买买', 1),
66 ('腾讯', '腾讯计算机系统有限公司', 50, '玩玩玩', 0),
67 ('百度', '百度在线网络技术公司', 5, '搜搜搜', 0),
68 ('京东', '北京京东世纪贸易有限公司', 40, '就是快', 1);

```

3. 查询所有功能

当前状态	当前状态	企业名称	品牌名称	品牌名称	查询																														
<input type="button" value="批量删除"/> <input type="button" value="新增"/> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th></th><th>品牌名称</th><th>企业名称</th><th>排序</th><th>当前状态</th><th>操作</th></tr> </thead> <tbody> <tr> <td><input type="checkbox"/></td><td>1 华为</td><td>华为科技有限公司</td><td>100</td><td>1</td><td><input type="button" value="修改"/> <input type="button" value="删除"/></td></tr> <tr> <td><input checked="" type="checkbox"/></td><td>2 华为</td><td>华为科技有限公司</td><td>100</td><td>1</td><td><input type="button" value="修改"/> <input type="button" value="删除"/></td></tr> <tr> <td><input type="checkbox"/></td><td>3 华为</td><td>华为科技有限公司</td><td>100</td><td>1</td><td><input type="button" value="修改"/> <input type="button" value="删除"/></td></tr> <tr> <td><input type="checkbox"/></td><td>4 华为</td><td>华为科技有限公司</td><td>100</td><td>1</td><td><input type="button" value="修改"/> <input type="button" value="删除"/></td></tr> </tbody> </table>							品牌名称	企业名称	排序	当前状态	操作	<input type="checkbox"/>	1 华为	华为科技有限公司	100	1	<input type="button" value="修改"/> <input type="button" value="删除"/>	<input checked="" type="checkbox"/>	2 华为	华为科技有限公司	100	1	<input type="button" value="修改"/> <input type="button" value="删除"/>	<input type="checkbox"/>	3 华为	华为科技有限公司	100	1	<input type="button" value="修改"/> <input type="button" value="删除"/>	<input type="checkbox"/>	4 华为	华为科技有限公司	100	1	<input type="button" value="修改"/> <input type="button" value="删除"/>
	品牌名称	企业名称	排序	当前状态	操作																														
<input type="checkbox"/>	1 华为	华为科技有限公司	100	1	<input type="button" value="修改"/> <input type="button" value="删除"/>																														
<input checked="" type="checkbox"/>	2 华为	华为科技有限公司	100	1	<input type="button" value="修改"/> <input type="button" value="删除"/>																														
<input type="checkbox"/>	3 华为	华为科技有限公司	100	1	<input type="button" value="修改"/> <input type="button" value="删除"/>																														
<input type="checkbox"/>	4 华为	华为科技有限公司	100	1	<input type="button" value="修改"/> <input type="button" value="删除"/>																														

如上图所示是查询所有品牌数据在页面展示的效果。要实现这个功能，要先搞明白如下问题：

- 什么时候发送异步请求？

页面加载完毕后就需要在页面上看到所有的品牌数据。所以在 `mounted()` 这个构造函数中写发送异步请求的代码。

- 请求需要携带参数吗？

查询所有功能不需要携带什么参数。

- 响应的数据格式是什么样？

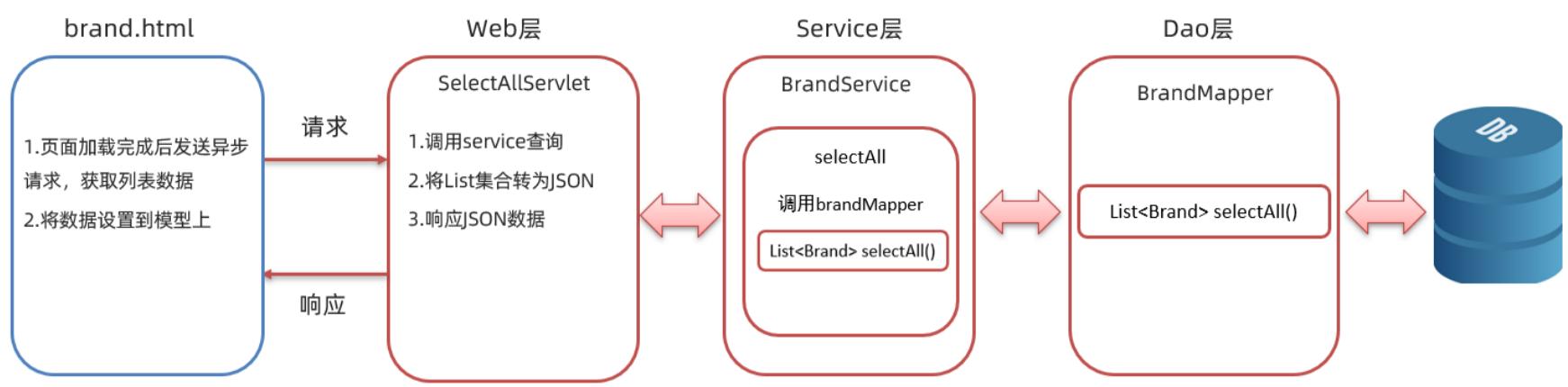
后端是需要将 `List<Brand>` 对象转换为 JSON 格式的数据并响应回给浏览器。响应数据格式如下：

```

[{"brandName": "华为", "companyName": "华为技术有限公司", "description": "万物互联", "id": 1, "ordered": 100, "status": 1, "statusStr": "启用"}, 
 {"brandName": "小米", "companyName": "小米科技有限公司", "description": "are you ok", "id": 2, "ordered": 50, "status": 1, "statusStr": "启用"}, 
 {"brandName": "格力", "companyName": "格力电器股份有限公司", "description": "让世界爱上中国造", "id": 3, "ordered": 30, "status": 1, "statusStr": "启用"}]

```

整体流程如下



我们先实现后端程序，然后再实现前端程序。

3.1 后端实现

3.1.1 dao方法实现

在 `com.itheima.mapper.BrandMapper` 接口中定义抽象方法，并使用 `@Select` 注解编写 sql 语句

```

1 /**
2  * @return
3  */
4 @Select("select * from tb_brand")
5 List<Brand> selectAll();

```

由于表中有些字段名和实体类中的属性名没有对应，所以需要在 `com/itheima/mapper/BrandMapper.xml` 映射配置文件中定义结果映射，使用 `resultMap` 标签。映射配置文件内容如下：

```

1 <?xml version="1.0" encoding="UTF-8" ?>
2 <!DOCTYPE mapper
3   PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
4   "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
5 <mapper namespace="com.itheima.mapper.BrandMapper">
6
7   <resultMap id="brandResultMap" type="brand">
8     <result property="brandName" column="brand_name" />
9     <result property="companyName" column="company_name" />
10    </resultMap>
11 </mapper>

```

定义完结果映射关系后，在接口 `selectAll()` 方法上引用该结构映射。使用 `@ResultMap("brandResultMap")` 注解完整接口的 `selectAll()` 方法如下：

```

1 /**
2  * @return
3  */
4 @Select("select * from tb_brand")
5 @ResultMap("brandResultMap")
6 List<Brand> selectAll();

```

3.1.2 service方法实现

在 `com.itheima.service` 包下创建 `BrandService` 接口，在该接口中定义查询所有的抽象方法

```

1 public interface BrandService {
2
3   /**
4    * @return
5    */
6   List<Brand> selectAll();
7 }

```

并在 `com.itheima.service` 下再创建 `impl` 包; `impl` 表示是放 service 层接口的实现类的包。在该包下创建名为 `BrandServiceImpl` 类

```
1 public class BrandServiceImpl implements BrandService {  
2  
3     @Override  
4     public List<Brand> selectAll() {  
5         }  
6 }
```

此处为什么要给 service 定义接口呢? 因为 service 定义了接口后, 在 servlet 中就可以使用多态的形式创建 Service 实现类的对象, 如下:

```
@WebServlet("/selectAllServlet")  
public class SelectAllServlet extends HttpServlet {  
  
    private BrandService brandService = new BrandServiceImpl();  
}
```

这里使用多态是因为方便我们后期解除 `Servlet` 和 `service` 的耦合。从上面的代码我们可以看到 `SelectAllServlet` 类和 `BrandServiceImpl` 类之间是耦合在一起的, 如果后期 `BrandService` 有其它更好的实现类(例如叫 `BrandServiceImp1`), 那就需要修改 `SelectAllServlet` 类中的代码。后面我们学习了 `Spring` 框架后就可以解除 `SelectAllServlet` 类和红色框括起来的代码耦合。而现在咱们还做不到解除耦合, 在这里只需要理解为什么定义接口即可。

`BrandServiceImpl` 类代码如下:

```
1 public class BrandServiceImpl implements BrandService {  
2     //1. 创建SqlSessionFactory 工厂对象  
3     SqlSessionFactory factory = SqlSessionFactoryUtils.getSqlSessionFactory();  
4  
5     @Override  
6     public List<Brand> selectAll() {  
7         //2. 获取SqlSession对象  
8         SqlSession sqlSession = factory.openSession();  
9         //3. 获取BrandMapper  
10        BrandMapper mapper = sqlSession.getMapper(BrandMapper.class);  
11  
12        //4. 调用方法  
13        List<Brand> brands = mapper.selectAll();  
14  
15        //5. 释放资源  
16        sqlSession.close();  
17  
18        return brands;  
19    }  
20 }
```

3.1.3 servlet 实现

在 `com.itheima.web.servlet` 包下定义名为 `SelectAllServlet` 的查询所有的 `servlet`。该 `servlet` 逻辑如下:

- 调用 service 的 `selectAll()` 方法查询所有的品牌数据, 并接口返回结果
- 将返回的结果转换为 json 数据
- 响应 json 数据

代码如下:

```
1 @WebServlet("/selectAllServlet")  
2 public class SelectAllServlet extends HttpServlet {  
3  
4     private BrandService brandService = new BrandServiceImpl();  
5  
6     @Override
```

```

7     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
8         ServletException, IOException {
9             //1. 调用service查询
10            List<Brand> brands = brandService.selectAll();
11            //2. 转为JSON
12            String jsonString = JSON.toJSONString(brands);
13            //3. 写数据
14            response.setContentType("text/json; charset=utf-8"); //告知浏览器响应的数据是什么， 告知浏览器
使用什么字符集进行解码
15            response.getWriter().write(jsonString);
16        }
17
18    @Override
19    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
20        this.doGet(request, response);
21    }

```

3.1.4 测试后端程序

在浏览器输入访问 servlet 的资源路径 `http://localhost:8080/brand-case/selectAllServlet`，如果没有报错，并能看到如下信息表明后端程序没有问题



3.2 前端实现

前端需要在页面加载完毕后发送 ajax 请求，所以发送请求的逻辑应该放在 `mounted()` 钩子函数中。而响应回来的数据需要赋值给表格绑定的数据模型，从下图可以看出表格绑定的数据模型是 `tableData`

```

<el-table
    :data="tableData"
    style="...">
    :row-class-name="tableRowClassName"
    @selection-change="handleSelectionChange">
        <el-table-column>

```

前端代码如下：

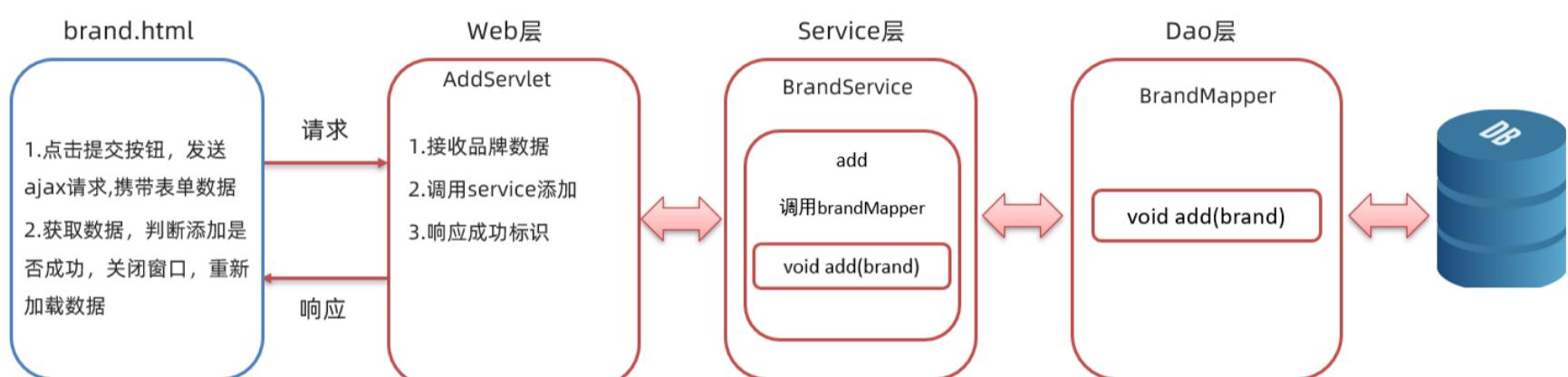
```

1 mounted() {
2     //当页面加载完成后，发送异步请求，获取数据
3     var _this = this;
4
5     axios({
6         method:"get",
7         url:"http://localhost:8080/brand-case/selectAllServlet"
8     }).then(function (resp) {
9         _this.tableData = resp.data;
10    })
11 }

```

4. 添加功能

上图是添加数据的对话框，当点击 提交 按钮后就需要将数据提交到后端，并将数据保存到数据库中。下图是整体的流程：



页面发送请求时，需要将输入框输入的内容提交给后端程序，而这里是以 json 格式进行传递的。而具体的数据格式如下：

```
{"status": "1", "brandName": "鸿星尔克", "companyName": "鸿星尔克", "id": "", "ordered": "200", "description": "to be no.1"}
```

注意：由于是添加数据，所以上述json数据中id是没有值的。

4.1 后端实现

4.1.1 dao方法实现

在 `BrandMapper` 接口中定义 `add()` 添加方法，并使用 `@Insert` 注解编写sql语句

```

1 /**
2  * 添加数据
3  * @param brand
4 */
5 @Insert("insert into tb_brand values(null,#{brandName},#{companyName},#{ordered},#
6 void add(Brand brand);

```

4.1.2 service方法实现

在 `BrandService` 接口中定义 `add()` 添加数据的业务逻辑方法

```

1 /**
2  * 添加数据
3  * @param brand
4 */
5 void add(Brand brand);

```

在 `BrandServiceImpl` 类中重写 `add()` 方法，并进行业务逻辑实现

```

1 @Override
2 public void add(Brand brand) {
3     //2. 获取sqlSession对象
4     SqlSession sqlSession = factory.openSession();
5     //3. 获取BrandMapper
6     BrandMapper mapper = sqlSession.getMapper(brandMapper.class);

```

```
7  
8     //4. 调用方法  
9     mapper.add(brand);  
10    sqlSession.commit(); //提交事务  
11  
12    //5. 释放资源  
13    sqlSession.close();  
14 }
```

注意：增删改操作一定要提交事务。

4.1.3 servlet实现

在 `com.itheima.web.servlet` 包写定义名为 `AddServlet` 的 Servlet。该 Servlet 的逻辑如下：

- 接收页面提交的数据。页面到时候提交的数据是 json 格式的数据，所以此处需要使用输入流读取数据
- 将接收到的数据转换为 `Brand` 对象
- 调用 service 的 `add()` 方法进行添加的业务逻辑处理
- 给浏览器响应添加成功的标识，这里直接给浏览器响应 `success` 字符串表示成功

servlet 代码实现如下：

```
1 @WebServlet("/addServlet")  
2 public class AddServlet extends HttpServlet {  
3  
4     private BrandService brandService = new BrandServiceImpl();  
5  
6     @Override  
7     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws  
8         ServletException, IOException {  
9  
10        //1. 接收品牌数据  
11        BufferedReader br = request.getReader();  
12        String params = br.readLine(); //json字符串  
13        //转为Brand对象  
14        Brand brand = JSON.parseObject(params, Brand.class);  
15        //2. 调用service添加  
16        brandService.add(brand);  
17        //3. 响应成功的标识  
18        response.getWriter().write("success");  
19    }  
20  
21    @Override  
22    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws  
23        ServletException, IOException {  
24        this.doGet(request, response);  
25    }  
26 }
```

4.2 前端实现

```

<el-form-item>
  <el-button type="primary" @click="addBrand">提交</el-button>
  <el-button @click="dialogVisible = false">取消</el-button>
</el-form-item>

```

上图左边是页面效果，里面的 提交 按钮可以通过上图右边看出绑定了一个单击事件，而该事件绑定的是 addBrand 函数，所以添加数据功能的逻辑代码应该写在 addBrand() 函数中。在此方法中需要发送异步请求并将表单中输入的数据作为参数进行传递。如下

```

1 // 添加数据
2 addBrand() {
3   var _this = this;
4
5   // 发送ajax请求，添加数据
6   axios({
7     method:"post",
8     url:"http://localhost:8080/brand-case/addServlet",
9     data:_this.brand
10 }).then(function (resp) {
11   //响应数据的处理逻辑
12 })
13 }

```

在 then 函数中的匿名函数是成功后的回调函数，而 resp.data 就可以获取到响应回来的数据，如果值是 success 表示数据添加成功。成功后我们需要做一下逻辑处理：

1. 关闭新增对话框窗口

如下图所示是添加数据的对话框代码，从代码中可以看到此对话框绑定了 dialogVisible 数据模型，只需要将该数据模型的值设置为 false，就可以关闭新增对话框窗口了。

```

<!-- 添加数据对话框表单-->
<el-dialog
  title="编辑品牌"
  :visible.sync="dialogVisible"
  width="30%">

```

2. 重新查询数据

数据添加成功与否，用户只要能在页面上查看到数据说明添加成功。而此处需要重新发送异步请求获取所有的品牌数据，而这段代码在 查询所有 功能中已经实现，所以我们可以将此功能代码进行抽取，抽取到一个 selectAll() 函数中

```

1 // 查询所有数据
2 selectAll(){
3   var _this = this;
4
5   axios({
6     method:"get",
7     url:"http://localhost:8080/brand-case/selectAllServlet"
8   }).then(function (resp) {
9     _this.tableData = resp.data;
10  })
11 }

```

那么就需要将 mounted() 钩子函数中代码改进为

```
1 mounted(){
2     //当页面加载完成后，发送异步请求，获取数据
3     this.selectAll();
4 }
```

同时在新增响应的回调中调用 `selectAll()` 进行数据的重新查询。

3. 弹出消息给用户提示添加成功



```
this.$message({
  message: '恭喜你，这是一条成功消息',
  type: 'success'
});
```

上图左边就是 elementUI 官网提供的成功提示代码，而上图右边是具体的效果。

注意：上面的this需要的是表示 VUE 对象的this。

综上所述，前端代码如下：

```
1 // 添加数据
2 addBrand() {
3     var _this = this;
4
5     // 发送ajax请求，添加数据
6     axios({
7         method:"post",
8         url:"http://localhost:8080/brand-case/addservlet",
9         data:_this.brand
10    }).then(function (resp) {
11        if(resp.data == "success"){
12            //添加成功
13            //关闭窗口
14            _this.dialogVisible = false;
15            // 重新查询数据
16            _this.selectAll();
17            // 弹出消息提示
18            _this.$message({
19                message: '恭喜你，添加成功',
20                type: 'success'
21            });
22        }
23    })
24 }
```

5, servlet优化

5.1 问题导入

Web 层的 Servlet 个数太多了，不利于管理和编写

通过之前的两个功能，我们发现每一个功能都需要定义一个 `servlet`，一个模块需要实现增删改查功能，就需要4个 `servlet`，模块一多就会造成 `servlet` 泛滥。此时我们就想 `servlet` 能不能像 `service` 一样，一个模块只定义一个 `servlet`，而每一个功能只需要在该 `servlet` 中定义对应的方法。例如下面代码：

```
1 @webServlet("/brand/*")
2 public class BrandServlet {
3     //查询所有
4     public void selectAll(...) {}
5
6     //添加数据
7     public void add(...) {}
8
9     //修改数据
10    public void update(...) {}
```

```
11 //删除删除  
12     public void delete(...) {}  
13 }  
14 }
```

而我们知道发送请求 `servlet`, `tomcat` 会自动的调用 `service()` 方法, 之前我们在自定义的 `servlet` 中重写 `doGet()` 方法和 `doPost()` 方法, 当我们访问该 `servlet` 时会根据请求方式将请求分发给 `doGet()` 或者 `doPost()` 方法, 如下图

```
protected void service(HttpServletRequest req, HttpServletResponse resp) {  
    String method = req.getMethod();  
    if (method.equals(METHOD_GET)) {  
        doGet(req, resp); 说明是get请求, 调用 doGet()方法  
    } else if (method.equals(METHOD_POST)) {  
        doPost(req, resp); 说明是 post 请求, 调用doPost()方法  
    }  
}
```

那么我们也可以仿照这样请求分发的思想, 在 `service()` 方法中根据具体的操作调用对应的方法, 如: 查询所有就调用 `selectAll()` 方法, 添加企业信息就调用 `add()` 方法。

为了做到通用, 我们定义一个通用的 `servlet` 类, 在定义其他的 `servlet` 是不需要继承 `HttpServlet`, 而继承我们定义的 `BaseServlet`, 在 `BaseServlet` 中调用具体 `servlet` (如 `BrandServlet`) 中的对应方法。

```
1 public class BaseServlet extends HttpServlet {  
2     @Override  
3     protected void service(HttpServletRequest req, HttpServletResponse resp) throws  
ServletException, IOException {  
4         //进行请求的分发  
5     }  
6 }
```

`BrandServlet` 定义就需要修改为如下:

```
1 @WebServlet("/brand/*")  
2 public class BrandServlet extends BaseServlet {  
3     //用户实现分页查询  
4     public void selectAll(...) {}  
5  
6     //添加企业信息  
7     public void add(...) {}  
8  
9     //修改企业信息  
10    public void update(...) {}  
11  
12    //删除企业信息  
13    public void delete(...) {}  
14 }
```

那么如何在 `BaseServlet` 中调用对应的方法呢? 比如查询所有就调用 `selectAll()` 方法。

可以**规定在发送请求时, 请求资源的二级路径 (`/brandServlet/selectAll`) 和需要调用的方法名相同**, 如:

查询所有数据的路径以后就需要写成: `http://localhost:8080/brand-case/brandServlet/selectAll`

添加数据的路径以后就需要写成: `http://localhost:8080/brand-case/brandServlet/add`

修改数据的路径以后就需要写成: `http://localhost:8080/brand-case/brandServlet/update`

删除数据的路径以后就需要写成: `http://localhost:8080/brand-case/brandServlet/delete`

这样的话, 在 `BaseServlet` 中就需要获取到资源的二级路径作为方法名, 然后调用该方法

```
1 public class BaseServlet extends HttpServlet {  
2     @Override
```

```

3     protected void service(HttpServletRequest req, HttpServletResponse resp) throws
4         ServletException, IOException {
5             //1. 获取请求路径
6             String uri = req.getRequestURI(); // 例如路径为: /brand-case/brand/selectAll
7             //2. 获取最后一段路径, 方法名
8             int index = uri.lastIndexOf('/');
9             String methodName = uri.substring(index + 1); // 获取到资源的二级路径 selectAll
10
11             //2. 执行方法
12             //2.1 获取BrandServlet /userservlet 字节码对象 Class
13             //System.out.println(this);
14
15             Class<? extends BaseServlet> cls = this.getClass();
16             //2.2 获取方法 Method对象
17             try {
18                 Method method = cls.getMethod(methodName, ? ? ? );
19                 //4, 调用该方法
20                 method.invoke(this, ? ? ? );
21             } catch (NoSuchMethodException e) {
22                 e.printStackTrace();
23             } catch (IllegalAccessException e) {
24                 e.printStackTrace();
25             } catch (InvocationTargetException e) {
26                 e.printStackTrace();
27             }
28         }

```

通过上面代码发现根据方法名获取对应方法的 `Method` 对象时需要指定方法参数的字节码对象。解决这个问题，可以将方法的参数类型规定死，而方法中可能需要用到 `request` 对象和 `response` 对象，所以指定方法的参数为 `HttpServletRequest` 和 `HttpServletResponse`，那么 `BrandServlet` 代码就可以改进为：

```

1 @WebServlet("/brand/*")
2 public class BrandServlet extends BaseServlet {
3     //用户实现分页查询
4     public void selectAll(HttpServletRequest req, HttpServletResponse resp) {}
5
6     //添加企业信息
7     public void add(HttpServletRequest req, HttpServletResponse resp) {}
8
9     //修改企业信息
10    public void update(HttpServletRequest req, HttpServletResponse resp) {}
11
12    //删除企业信息
13    public void delete(HttpServletRequest req, HttpServletResponse resp) {}
14 }

```

`BaseServlet` 代码可以改进为：

```

1 public class BaseServlet extends HttpServlet {
2
3     //根据请求的最后一段路径来进行方法分发
4     @Override
5     protected void service(HttpServletRequest req, HttpServletResponse resp) throws
6         ServletException, IOException {
7         //1. 获取请求路径
8         String uri = req.getRequestURI(); // 例如路径为: /brand-case/brand/selectAll
9         //2. 获取最后一段路径, 方法名
10        int index = uri.lastIndexOf('/');
11        String methodName = uri.substring(index + 1); // 获取到资源的二级路径 selectAll
12
13        //2. 执行方法
14        //2.1 获取BrandServlet /userservlet 字节码对象 Class
15        //System.out.println(this);

```

```

16     Class<? extends BaseServlet> cls = this.getClass();
17     //2.2 获取方法 Method对象
18     try {
19         Method method = cls.getMethod(methodName, HttpServletRequest.class,
HttpServletResponse.class);
20             //2.3 执行方法
21             method.invoke(this,req,resp);
22     } catch (NoSuchMethodException e) {
23         e.printStackTrace();
24     } catch (IllegalAccessException e) {
25         e.printStackTrace();
26     } catch (InvocationTargetException e) {
27         e.printStackTrace();
28     }
29 }
30 }
```

5.2 代码优化

5.2.1 后端优化

定义了 `BaseServlet` 后，针对品牌模块我们定义一个 `BrandServlet` 的 Servlet，并使其继承 `BaseServlet`。在 `BrandServlet` 中定义以下功能的方法：

- `查询所有` 功能：方法名声明为 `selectAll`，并将之前的 `SelectAllServlet` 中的逻辑代码拷贝到该方法中
- `添加数据` 功能：方法名声明为 `add`，并将之前的 `AddServlet` 中的逻辑代码拷贝到该方法中

具体代码如下：

```

1 @webServlet("/brand/*")
2 public class BrandServlet extends BaseServlet{
3     private BrandService brandService = new BrandServiceImpl();
4
5     public void selectAll(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
6         //1. 调用service查询
7         List<Brand> brands = brandService.selectAll();
8
9         //2. 转为JSON
10        String jsonString = JSON.toJSONString(brands);
11        //3. 写数据
12        response.setContentType("text/json;charset=utf-8");
13        response.getWriter().write(jsonString);
14    }
15
16    public void add(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
17
18        //1. 接收品牌数据
19        BufferedReader br = request.getReader();
20        String params = br.readLine(); //json字符串
21
22        //转为Brand对象
23        Brand brand = JSON.parseObject(params, Brand.class);
24
25        //2. 调用service添加
26        brandService.add(brand);
27
28        //3. 响应成功的标识
29        response.getWriter().write("success");
30    }
31 }
```

5.2.2 前端优化

页面中之前发送的请求的路径都需要进行修改，`selectAll()` 函数中发送异步请求的 `url` 应该改为 `http://localhost:8080/brand-case/brand/selectAll`。具体代码如下：

```
1 // 查询分页数据
2 selectAll(){
3     var _this = this;
4
5     axios({
6         method:"get",
7         url:"http://localhost:8080/brand-case/brand/selectAll"
8     }).then(function (resp) {
9         _this.tableData = resp.data;
10    })
11 }
```

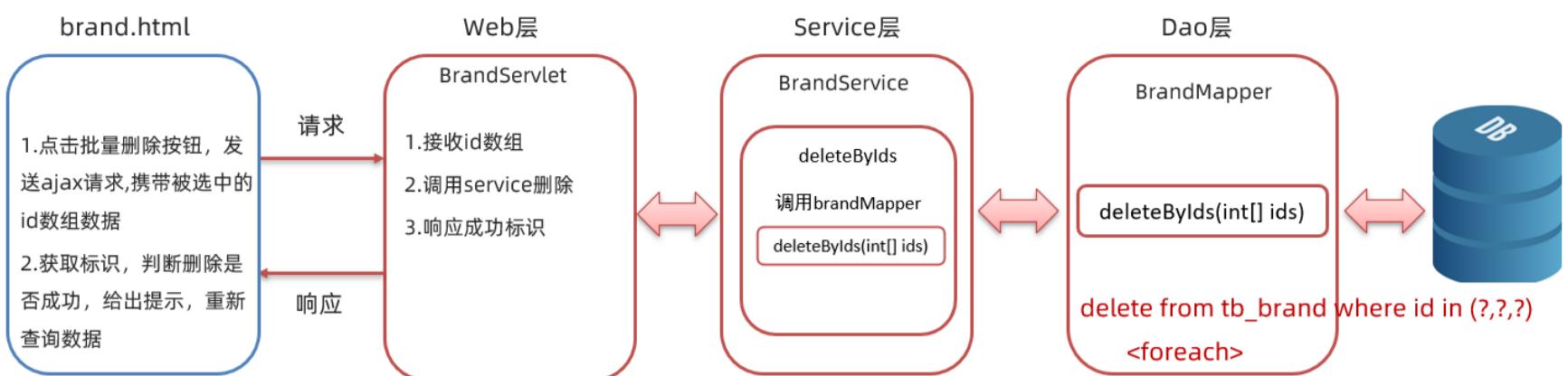
`addBrand()` 函数中发送异步请求的 `url` 应该改为 `http://localhost:8080/brand-case/brand/add`。具体代码如下：

```
1 // 添加数据
2 addBrand() {
3     //console.log(this.brand);
4     var _this = this;
5
6     // 发送ajax请求，添加数据
7     axios({
8         method:"post",
9         url:"http://localhost:8080/brand-case/brand/add",
10        data:_this.brand
11    }).then(function (resp) {
12        if(resp.data == "success"){
13            //添加成功
14            //关闭窗口
15            _this.dialogVisible = false;
16            // 重新查询数据
17            _this.selectAll();
18            // 弹出消息提示
19            _this.$message({
20                message: '恭喜你，添加成功',
21                type: 'success'
22            });
23        }
24    })
25 }
```

6. 批量删除

批量删除				新增
	品牌名称	操作	操作	
<input checked="" type="checkbox"/>	1 华为			
<input checked="" type="checkbox"/>	2 小米			
<input checked="" type="checkbox"/>	3 格力			
<input type="checkbox"/>	4 阿里巴巴			

如上图所示点击多条数据前的复选框就意味着要删除这些数据，而点击了 `批量删除` 按钮后，需要让用户确认一下，因为有可能是用户误操作的，当用户确定后需要给后端发送请求并携带者需要删除数据的多个id值，后端程序删除数据库中的数据。具体的流程如下：



注意：

前端发送请求时需要将要删除的多个id值以json格式提交给后端，而该json格式数据如下：

```
1 [1,2,3,4]
```

6.1 后端实现

6.1.1 dao方法实现

在 `BrandMapper` 接口中定义 `deleteByIds()` 添加方法，由于这里面要用到动态 sql，属于复杂的sql操作，建议使用映射配置文件。

接口方法声明如下：

```
1 /**
2  * 批量删除
3  * @param ids
4  */
5 void deleteByIds(@Param("ids") int[] ids);
```

在 `BrandMapper.xml` 映射配置文件中添加 statement

```
1 <delete id="deleteByIds">
2   delete from tb_brand where id in
3     <foreach collection="ids" item="id" separator="," open="(" close=")">
4       #{id}
5     </foreach>
6 </delete>
```

6.1.2 service方法实现

在 `BrandService` 接口中定义 `deleteByIds()` 批量删除的业务逻辑方法

```
1 /**
2  * 批量删除
3  * @param ids
4  */
5 void deleteByIds( int[] ids);
```

在 `BrandServiceImpl` 类中重写 `deleteByIds()` 方法，并进行业务逻辑实现

```
1 @Override
2 public void deleteByIds(int[] ids) {
3   //2. 获取SqlSession对象
4   SqlSession sqlSession = factory.openSession();
5   //3. 获取BrandMapper
6   BrandMapper mapper = sqlSession.getMapper(BrandMapper.class);
7
8   //4. 调用方法
9   mapper.deleteByIds(ids);
10
11  sqlSession.commit(); //提交事务
12}
```

```

13     //5. 释放资源
14     sqlSession.close();
15 }

```

6.1.3 servlet实现

在 `BrandServlet` 类中定义 `deleteByIds()` 方法。而该方法的逻辑如下：

- 接收页面提交的数据。页面到时候提交的数据是 json 格式的数据，所以此处需要使用输入流读取数据
- 接收页面提交的数据。页面到时候提交的数据是 json 格式的数据，所以此处需要使用输入流读取数据
- 将接收到的数据转换为 `int[]` 数组
- 调用 service 的 `deleteByIds()` 方法进行批量删除的业务逻辑处理
- 给浏览器响应添加成功的标识，这里直接给浏览器响应 `success` 字符串表示成功

servlet 中 `deleteByIds()` 方法代码实现如下：

```

1 public void deleteByIds(HttpServletRequest request, HttpServletResponse response) throws
2   ServletException, IOException {
3     //1. 接收数据 json [1,2,3]
4     BufferedReader br = request.getReader();
5     String params = br.readLine(); //json字符串
6     //转为 int[]
7     int[] ids = JSON.parseObject(params, int[].class);
8     //2. 调用service添加
9     brandService.deleteByIds(ids);
10    //3. 响应成功的标识
11    response.getWriter().write("success");
12 }

```

6.2 前端实现

此功能的前端代码实现稍微有点麻烦，分为以下几步实现

6.2.1 获取选中的id值

```

<el-table
  :data="tableData"
  style="..." ...
  :row-class-name="tableRowClassName"
  @selection-change="handleSelectionChange">
  <el-table-column
    type="selection"
    width="55">
  </el-table-column>

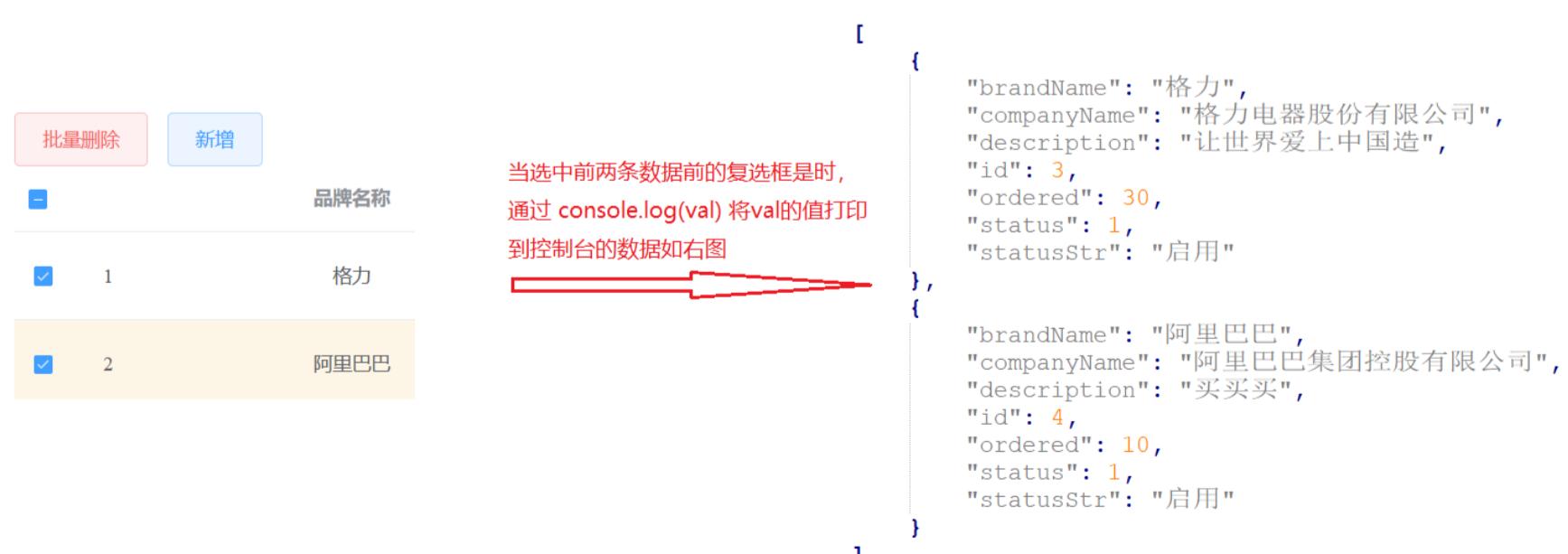
```

```

// 复选框选中后执行的方法
handleSelectionChange(val) {
  this.multipleSelection = val;
}

```

从上图可以看出表格复选框绑定了一个 `selection-change` 事件，该事件是当选择项发生变化时会触发。该事件绑定了 `handleSelectionChange` 函数，而该函数有一个参数 `val`，该参数是获取选中行的数据，如下



而我们只需要将所有选中数据的 id 值提交给服务端即可，获取 id 的逻辑我们书写的 `批量删除` 按钮绑定的函数中。

在 `批量删除` 按钮绑定单击事件，并给绑定触发时调用的函数，如下

```

<!-- 按钮-->
<el-row>
    <el-button type="danger" plain @click="deleteByIds">批量删除</el-button>
    <el-button type="primary" plain @click="dialogVisible = true">新增</el-button>
</el-row>

```

并在Vue对象中的 methods 中定义 `deleteByIds()` 函数，在该函数中从 `multipleSelection` 数据模型中获取所选数据的 id 值。要完成这个功能需要在 Vue 对象中定义一个数据模型 `selectedIds: []`，在 `deleteByIds()` 函数中遍历 `multipleSelection` 数组，并获取到每一个所选数据的 id 值存储到 `selectedIds` 数组中，代码实现如下：

```

1 //1. 创建id数组 [1,2,3], 从 this.multipleSelection 获取即可
2 for (let i = 0; i < this.multipleSelection.length; i++) {
3     let selectionElement = this.multipleSelection[i];
4     this.selectedIds[i] = selectionElement.id;
5 }

```

6.2.2 发送异步请求

使用 axios 发送异步请求并经上一步获取到的存储所有的 id 数组作为请求参数

```

1 //2. 发送AJAX请求
2 var _this = this;
3
4 // 发送ajax请求, 添加数据
5 axios({
6     method:"post",
7     url:"http://localhost:8080/brand-case/brand/deleteByIds",
8     data:_this.selectedIds
9 }).then(function (resp) {
10     if(resp.data == "success"){
11         //删除成功
12         // 重新查询数据
13         _this.selectAll();
14         // 弹出消息提示
15         _this.$message({
16             message: '恭喜你, 删除成功',
17             type: 'success'
18         });
19     }
20 })

```

6.2.3 确定框实现

由于删除操作是比较危险的；有时候可能是由于用户的误操作点击了 `批量删除` 按钮，所以在点击了按钮后需要先给用户确认提示。而确认框在 `elementUI` 中也提供了，如下图



而在点击 `确定` 按钮后需要执行之前删除的逻辑。因此前端代码实现如下：

```

1 // 批量删除
2 deleteByIds(){
3     // 弹出确认提示框
4     this.$confirm('此操作将删除该数据, 是否继续?', '提示', {

```

```

5     confirmButtonText: '确定',
6     cancelButtonText: '取消',
7     type: 'warning'
8 }).then(() => {
9     //用户点击确认按钮
10    //1. 创建id数组 [1,2,3], 从 this.multipleSelection 获取即可
11    for (let i = 0; i < this.multipleSelection.length; i++) {
12        let selectionElement = this.multipleSelection[i];
13        this.selectedIds[i] = selectionElement.id;
14    }
15    //2. 发送AJAX请求
16    var _this = this;
17    // 发送ajax请求, 添加数据
18    axios({
19        method:"post",
20        url:"http://localhost:8080/brand-case/brand/deleteByIds",
21        data:_this.selectedIds
22    }).then(function (resp) {
23        if(resp.data == "success"){
24            //删除成功
25            // 重新查询数据
26            _this.selectAll();
27            // 弹出消息提示
28            _this.$message({
29                message: '恭喜你, 删除成功',
30                type: 'success'
31            });
32        }
33    })
34 }).catch(() => {
35     //用户点击取消按钮
36     this.$message({
37         type: 'info',
38         message: '已取消删除'
39     });
40 });
41 }

```

7. 分页查询

我们之前做的 `查询所有` 功能中将数据库中所有的数据查询出来并展示到页面上，试想如果数据库中的数据有很多（假设有十几万条）的时候，将数据全部展示出来肯定不现实，那如何解决这个问题呢？几乎所有的网站都会使用分页解决这个问题。每次只展示一页的数据，比如一页展示10条数据，如果还想看其他的数据，可以通过点击页码进行查询

共 51 条 5条/页 < 1 2 3 4 5 6 ... 11 > 前往 页

7.1 分析

7.1.1 分页查询sql

分页查询也是从数据库进行查询的，所以我们要分页对应的SQL语句应该怎么写。分页查询使用 `LIMIT` 关键字，格式为：`LIMIT 开始索引 每页显示的条数`。以前前端页面在发送请求携带参数时，它并不明确开始索引是什么，但是它知道查询第几页。所以 `开始索引` 需要在后端进行计算，计算的公式是：`开始索引 = (当前页码 - 1) * 每页显示条数`

比如查询第一页的数据的 SQL 语句是：

```
1 | select * from tb_brand limit 0,5;
```

查询第二页的数据的 SQL 语句是：

```
1 | select * from tb_brand limit 5,5;
```

查询第三页的数据的 SQL 语句是：

```
1 | select * from tb_brand limit 10,5;
```

7.1.2 前后端数据分析

分页查询功能时候比较复杂的，所以我们要先分析清楚以下两个问题：

- **前端需要传递什么参数给后端**

根据上一步对分页查询 SQL 语句分析得出，前端需要给后端两个参数

- 当前页码： currentPage
- 每页显示条数： pageSize

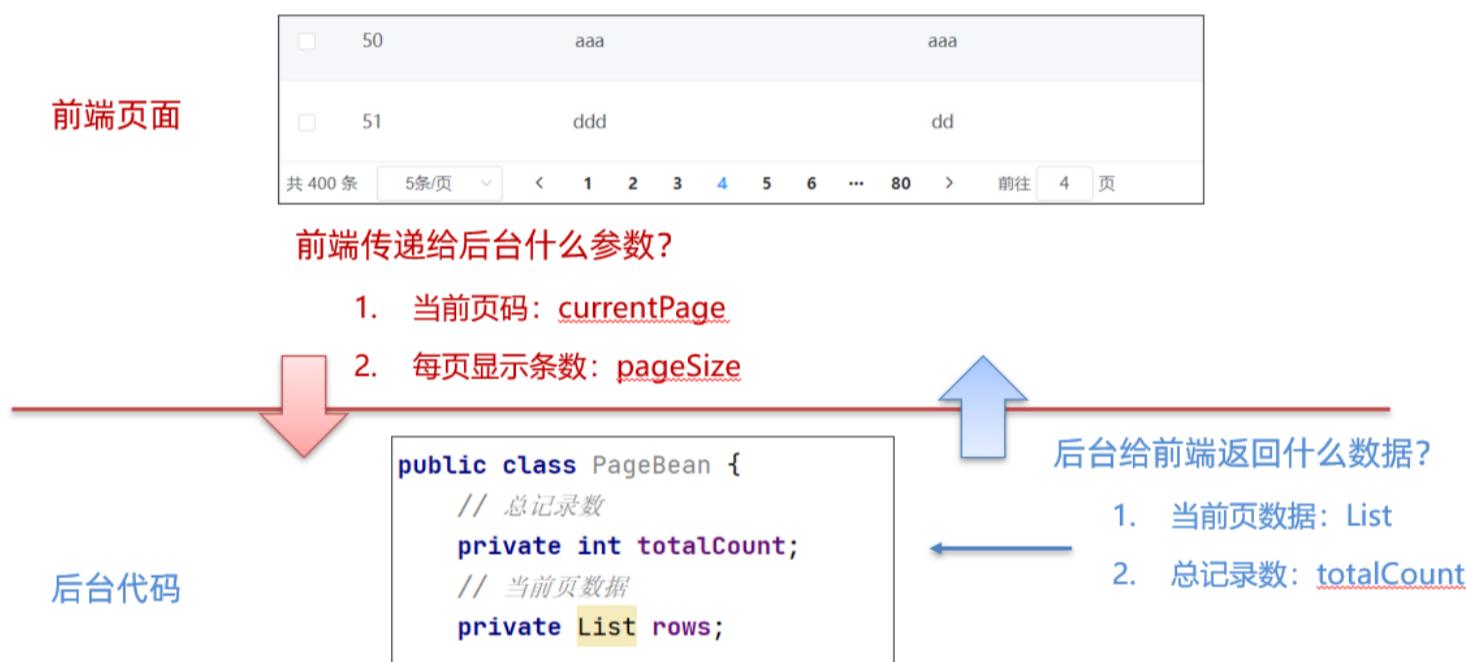
- **后端需要响应什么数据给前端**

<input type="checkbox"/> 50	aaa	aaa
<input type="checkbox"/> 51	ddd	dd
共 400 条		
5条/页		
< 1 2 3 4 5 6 ... 80 > 前往 4 页		

上图是分页查询页面展示的效果，从上面我们可以看出需要响应以下联股份数据

- 当前页需要展示的数据。我们在后端一般会存储到 List 集合中
- 总共记录数。在上图页面中需要展示总的记录数，所以这部分数据也需要。总的页面 elementUI 的分页组件会自动计算，我们不需要关心

而这两部分需要封装到 PageBean 对象中，并将该对象转换为 json 格式的数据响应回给浏览器



通过上面的分析我们需要先在 `pojo` 包下创建 `PageBean` 类，为了做到通过会将其定义成泛型类，代码如下：

```
1 //分页查询的JavaBean  
2 public class PageBean<T> {  
3     // 总记录数  
4     private int totalCount;  
5     // 当前页数据  
6     private List<T> rows;  
7  
8     public int getTotalCount() {  
9         return totalCount;  
10    }  
11  
12    public void setTotalCount(int totalCount) {  
13        this.totalCount = totalCount;  
14    }  
15  
16    public List<T> getRows() {  
17        return rows;  
18    }  
19  
20    public void setRows(List<T> rows) {  
21        this.rows = rows;  
22    }
```

```
23 }
24 }
```

7.1.3 流程分析

后端需要响应 `总记录数` 和 `当前页的数据` 两部分数据给前端，所以在 `BrandMapper` 接口中需要定义两个方法：

- `selectByPage()`：查询当前页的数据的方法
- `selectTotalCount()`：查询总记录数的方法

整体流程如下：



7.2 后端实现

7.2.1 dao方法实现

在 `BrandMapper` 接口中定义 `selectByPage()` 方法进行分页查询，代码如下：

```
1 /**
2  * 分页查询
3  * @param begin
4  * @param size
5  * @return
6 */
7 @Select("select * from tb_brand limit #{begin} , #{size}")
8 @ResultMap("brandResultMap")
9 List<Brand> selectByPage(@Param("begin") int begin,@Param("size") int size);
```

在 `BrandMapper` 接口中定义 `selectTotalCount()` 方法进行统计记录数，代码如下：

```
1 /**
2  * 查询总记录数
3  * @return
4  */
5 @Select("select count(*) from tb_brand ")
6 int selectTotalCount();
```

7.2.2 service方法实现

在 `BrandService` 接口中定义 `selectByPage()` 分页查询数据的业务逻辑方法

```
1 /**
2  * 分页查询
3  * @param currentPage 当前页码
4  * @param pageSize 每页展示条数
5  * @return
6  */
7 PageBean<Brand> selectByPage(int currentPage,int pageSize);
```

在 `BrandServiceImpl` 类中重写 `selectByPage()` 方法，并进行业务逻辑实现

```
1 @Override
2 public PageBean<Brand> selectByPage(int currentPage, int pageSize) {
3     //2. 获取SqlSession对象
4     SqlSession sqlSession = factory.openSession();
5     //3. 获取BrandMapper
```

```

6     BrandMapper mapper = sqlSession.getMapper(BrandMapper.class);
7     //4. 计算开始索引
8     int begin = (currentPage - 1) * pageSize;
9     // 计算查询条目数
10    int size = pageSize;
11    //5. 查询当前页数据
12    List<Brand> rows = mapper.selectByPage(begin, size);
13    //6. 查询总记录数
14    int totalCount = mapper.selectTotalCount();
15    //7. 封装PageBean对象
16    PageBean<Brand> pageBean = new PageBean<>();
17    pageBean.setRows(rows);
18    pageBean.setTotalCount(totalCount);
19
20    //8. 释放资源
21    sqlSession.close();
22    return pageBean;
23 }

```

7.2.3 servlet实现

在 `BrandServlet` 类中定义 `selectByPage()` 方法。而该方法的逻辑如下：

- 获取页面提交的 `当前页码` 和 `每页显示条目数` 两个数据。这两个参数是在url后进行拼接的，格式是 `url?currentPage=1&pageSize=5`。获取这样的参数需要使用 `request.getParameter()` 方法获取。
- 调用 service 的 `selectByPage()` 方法进行分页查询的业务逻辑处理
- 将查询到的数据转换为 json 格式的数据
- 响应 json 数据

servlet 中 `selectByPage()` 方法代码实现如下：

```

1 public void selectByPage(HttpServletRequest request, HttpServletResponse response) throws
2     ServletException, IOException {
3     //1. 接收 当前页码 和 每页展示条数 url?currentPage=1&pageSize=5
4     String _currentPage = request.getParameter("currentPage");
5     String _pageSize = request.getParameter("pageSize");
6
7     int currentPage = Integer.parseInt(_currentPage);
8     int pageSize = Integer.parseInt(_pageSize);
9
10    //2. 调用service查询
11    PageBean<Brand> pageBean = brandService.selectByPage(currentPage, pageSize);
12
13    //2. 转为JSON
14    String jsonString = JSON.toJSONString(pageBean);
15    //3. 写数据
16    response.setContentType("text/json;charset=utf-8");
17    response.getWriter().write(jsonString);
18 }

```

7.2.4 测试

在浏览器上地址栏输入 `http://localhost:8080/brand-case/brand/selectByPage?currentPage=1&pageSize=5`，查询到以下数据

```
{"rows": [{"brandName": "格力", "companyName": "格力电器股份有限公司", "description": "让世界爱上中国造", "id": 3, "ordered": 30, "status": 1, "statusStr": "启用"}, {"brandName": "阿里巴巴", "companyName": "阿里巴巴集团控股有限公司", "description": "买买买", "id": 4, "ordered": 10, "status": 1, "statusStr": "启用"}, {"brandName": "腾讯", "companyName": "腾讯计算机系统有限公司", "description": "玩玩玩", "id": 5, "ordered": 50, "status": 0, "statusStr": "禁用"}, {"brandName": "百度", "companyName": "百度在线网络技术公司", "description": "搜搜搜", "id": 6, "ordered": 5, "status": 0, "statusStr": "禁用"}, {"brandName": "京东", "companyName": "北京京东世纪贸易有限公司", "description": "就是快", "id": 7, "ordered": 40, "status": 1, "statusStr": "启用"}], "totalCount": 51}
```

7.3 前端实现

7.3.1 selectAll 代码改进

`selectAll()` 函数之前是查询所有数据，现需要改成分页查询。请求路径应改为 `http://localhost:8080/brand-case/brand/selectByPage?currentPage=1&pageSize=5`，而 `currentPage` 和 `pageSize` 是需要携带的参数，分别是当前页码和每页显示的条目数。

刚才我们对后端代码进行测试可以看出响应回来的数据，所以在异步请求的成功回调函数（`then` 中的匿名函数）中给页面表格的数据模型赋值 `_this.tableData = resp.data.rows;`。整体代码如下

```
1 var _this = this;
2 axios({
3     method:"post",
4     url:"http://localhost:8080/brand-case/brand/selectByPage? currentPage=1&pageSize=5"
5 }).then(resp =>{
6     //设置表格数据
7     _this.tableData = resp.data.rows; // {rows:[],totalCount:100}
8 })
```

响应的数据中还有总记录数，要进行总记录数展示需要在页面绑定数据模型

```
<!-- 分页工具条-->
<el-pagination
    @size-change="handleSizeChange"
    @current-change="handleCurrentChange"
    :current-page="currentPage"
    :page-sizes="[5, 10, 15, 20]"
    :page-size="5"
    layout="total, sizes, prev, pager, next, jumper"
    :total="totalCount">
</el-pagination>
```

注意：该数据模型需要在Vue对象中声明出来。

那异步请求的代码就可以优化为

```
1 var _this = this;
2 axios({
3     method:"post",
4     url:"http://localhost:8080/brand-case/brand/selectByPage?currentPage=1&pageSize=5"
5 }).then(resp =>{
6     //设置表格数据
7     _this.tableData = resp.data.rows; // {rows:[],totalCount:100}
8     //设置总记录数
9     _this.totalCount = resp.data.totalCount;
10 })
```

而页面中分页组件给 `当前页码` 和 `每页显示的条目数` 都绑定了数据模型

```
<!-- 分页工具条-->
<el-pagination
    @size-change="handleSizeChange"
    @current-change="handleCurrentChange"
    :current-page="currentPage"
    :page-sizes="[5, 10, 15, 20]"
    :page-size="pageSize"
    layout="total, sizes, prev, pager, next, jumper"
    :total="totalCount">
</el-pagination>
```

所以 `selectAll()` 函数中发送异步请求的资源路径中不能将当前页码和 每页显示条目数写死，代码就可以优化为

```

1 var _this = this;
2 axios({
3     method:"post",
4     url:"http://localhost:8080/brand-case/brand/selectByPage?
5 currentPage="+this.currentPage+"&pageSize=" + this.pageSize
6 }).then(resp =>{
7     //设置表格数据
8     _this.tableData = resp.data.rows; // {rows:[],totalCount:100}
9     //设置总记录数
10    _this.totalCount = resp.data.totalCount;
11 })

```

7.3.2 改变每页条目数



当我们改变每页显示的条目数后，需要重新发送异步请求。而下图是分页组件代码，`@size-change` 就是每页显示的条目数发生变化时会触发的事件

```

<!-- 分页工具条-->
<el-pagination
    @size-change="handleSizeChange"
    @current-change="handleCurrentChange"
    :current-page="currentPage"
    :page-sizes="[5, 10, 15, 20]"
    :page-size="pageSize"
    layout="total, sizes, prev, pager, next, jumper"
    :total="totalCount">
</el-pagination>

```

而该事件绑定了一个 `handlesizeChange` 函数，整个逻辑如下：

```

1 handlesizeChange(val) { //我们选择的是‘5条/页’此值就是 5.而我们选择了‘10条/页’此值就是 10
2     // 重新设置每页显示的条数
3     this.pageSize = val;
4     //调用 selectAll 函数重新分页查询数据
5     this.selectAll();
6 }

```

7.3.3 改变当前页码

当我们改变页码时，需要重新发送异步请求。而下图是分页组件代码，`@current-change` 就是页码发生变化时会触发的事件

```

<!-- 分页工具条-->
<el-pagination
    @size-change="handleSizeChange"
    @current-change="handleCurrentChange"
    :current-page="currentPage"
    :page-sizes="[5, 10, 15, 20]"
    :page-size="pageSize"
    layout="total, sizes, prev, pager, next, jumper"
    :total="totalCount">
</el-pagination>

```

而该事件绑定了一个 `handlesizeChange` 函数，整个逻辑如下：

```

1 handleCurrentChange(val) { //val 就是改变后的页码
2     // 重新设置当前页码
3     this.currentPage = val;
4     //调用 selectAll 函数重新分页查询数据
5     this.selectAll();
6 }

```

8. 条件查询

当前状态	当前状态	企业名称	企业名称	品牌名称	品牌名称	查询
------	------	------	------	------	------	-----------

上图就是用来输入条件查询的条件数据的。要做条件查询功能，先明确以下三个问题

- 3个条件之间什么关系？

同时满足，所用 SQL 中多个条件需要使用 and 关键字连接

- 3个条件必须全部填写吗？

不需要。想根据哪几个条件查询就写那个，所以这里需要使用动态 sql 语句

- 条件查询需要分页吗？

需要

根据上面三个问题的明确，我们就可以确定sql语句了：

```

select *
from tb_brand
<where>
    <if test="brand.brandName != null and brand.brandName != ''">
        brand_name like #{brand.brandName}
    </if>
    <if test="brand.companyName != null and brand.companyName != ''">
        and company_name like #{brand.companyName}
    </if>
    <if test="brand.status != null">
        and status = #{brand.status}
    </if>
</where>
limit #{begin} , #{size}

```

整个条件分页查询流程如下



8.1 后端实现

8.1.1 dao实现

在 `BrandMapper` 接口中定义 `selectByPageAndCondition()` 方法和 `selectTotalCountByCondition` 方法，来进行条件分页查询功能，方法如下：

```

1 /**
2  * 分页条件查询
3  * @param begin
4  * @param size
5  * @return
6 */
7 List<Brand> selectByPageAndCondition(@Param("begin") int begin,@Param("size") int
size,@Param("brand") Brand brand);
8
9 /**
10  * 根据条件查询总记录数
11  * @return
12  */
13 int selectTotalCountByCondition(Brand brand);

```

参数:

- `begin` 分页查询的起始索引
- `size` 分页查询的每页条目数
- `brand` 用来封装条件的对象

由于这是一个复杂的查询语句，需要使用动态sql；所以我们在映射配置文件中书写 sql 语句。`brand_name` 字段和 `company_name` 字段需要进行模糊查询，所以需要使用 `%` 占位符。映射配置文件中 statement 书写如下：

```

1 <!--查询满足条件的数据并进行分页-->
2 <select id="selectByPageAndCondition" resultMap="brandResultMap">
3   select *
4   from tb_brand
5   <where>
6     <if test="brand.brandName != null and brand.brandName != '' ">
7       and brand_name like #{brand.brandName}
8     </if>
9
10    <if test="brand.companyName != null and brand.companyName != '' ">
11      and company_name like #{brand.companyName}
12    </if>
13
14    <if test="brand.status != null">
15      and status = #{brand.status}
16    </if>
17   </where>
18   limit #{begin} , #{size}
19 </select>
20
21 <!--查询满足条件的数据条目数-->
22 <select id="selectTotalCountByCondition" resultType="java.lang.Integer">
23   select count(*)
24   from tb_brand
25   <where>
26     <if test="brandName != null and brandName != '' ">
27       and brand_name like #{brandName}
28     </if>
29
30     <if test="companyName != null and companyName != '' ">
31       and company_name like #{companyName}
32     </if>
33
34     <if test="status != null">
35       and status = #{status}
36     </if>
37   </where>
38 </select>

```

8.1.2 service实现

在 `BrandService` 接口中定义 `selectByPageAndCondition()` 分页查询数据的业务逻辑方法

```
1  /**
2   * 分页条件查询
3   * @param currentPage
4   * @param pageSize
5   * @param brand
6   * @return
7   */
8  PageBean<Brand> selectByPageAndCondition(int currentPage, int pageSize, Brand brand);
```

在 `BrandServiceImpl` 类中重写 `selectByPageAndCondition()` 方法，并进行业务逻辑实现

```
1  @Override
2  public PageBean<Brand> selectByPageAndCondition(int currentPage, int pageSize, Brand
brand) {
3      //2. 获取SqlSession对象
4      SqlSession sqlSession = factory.openSession();
5      //3. 获取BrandMapper
6      BrandMapper mapper = sqlSession.getMapper(BrandMapper.class);
7
8
9      //4. 计算开始索引
10     int begin = (currentPage - 1) * pageSize;
11     // 计算查询条目数
12     int size = pageSize;
13
14     // 处理brand条件，模糊表达式
15     String brandName = brand.getBrandName();
16     if (brandName != null && brandName.length() > 0) {
17         brand.setBrandName("%" + brandName + "%");
18     }
19
20     String companyName = brand.getCompanyName();
21     if (companyName != null && companyName.length() > 0) {
22         brand.setCompanyName("%" + companyName + "%");
23     }
24
25     //5. 查询当前页数据
26     List<Brand> rows = mapper.selectByPageAndCondition(begin, size, brand);
27
28     //6. 查询总记录数
29     int totalCount = mapper.selectTotalCountByCondition(brand);
30
31     //7. 封装PageBean对象
32     PageBean<Brand> pageBean = new PageBean<>();
33     pageBean.setRows(rows);
34     pageBean.setTotalCount(totalCount);
35
36     //8. 释放资源
37     sqlSession.close();
38
39     return pageBean;
40 }
```

注意：`brandName` 和 `companyName` 属性值到时候需要进行模糊查询，所以前后需要拼接上 %。

8.1.3 servlet实现

在 `BrandServlet` 类中定义 `selectByPageAndCondition()` 方法。而该方法的逻辑如下：

- 获取页面提交的 `currentPage` 和 `每页显示条目数` 两个数据。这两个参数是在url后进行拼接的，格式是 `url?currentPage=1&pageSize=5`。获取这样的参数需要使用 `request.getParameter()` 方法获取。
- 获取页面提交的 `条件数据`，并将数据封装到一个Brand对象中。由于这部分数据到时候是需要以 json 格式进行提交的，所以我们需要通过流获取数据，具体代码如下：

```

1 // 获取查询条件对象
2 BufferedReader br = request.getReader();
3 String params = br.readLine(); //json字符串
4
5 //转为 Brand
6 Brand brand = JSON.parseObject(params, Brand.class);

```

- 调用 service 的 `selectByPageAndCondition()` 方法进行分页查询的业务逻辑处理
- 将查询到的数据转换为 json 格式的数据
- 响应 json 数据

servlet 中 `selectByPageAndCondition()` 方法代码实现如下：

```

1 /**
2  * 分页条件查询
3  * @param request
4  * @param response
5  * @throws ServletException
6  * @throws IOException
7 */
8
9 public void selectByPageAndCondition(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
10    //1. 接收 当前页码 和 每页展示条数    url?currentPage=1&pageSize=5
11    String _currentPage = request.getParameter("currentPage");
12    String _pageSize = request.getParameter("pageSize");
13
14    int currentPage = Integer.parseInt(_currentPage);
15    int pageSize = Integer.parseInt(_pageSize);
16
17    // 获取查询条件对象
18    BufferedReader br = request.getReader();
19    String params = br.readLine(); //json字符串
20
21    //转为 Brand
22    Brand brand = JSON.parseObject(params, Brand.class);
23
24
25    //2. 调用service查询
26    PageBean<Brand> pageBean =
27    brandService.selectByPageAndCondition(currentPage, pageSize, brand);
28
29    //2. 转为JSON
30    String jsonString = JSON.toJSONString(pageBean);
31    //3. 写数据
32    response.setContentType("text/json;charset=utf-8");
33    response.getWriter().write(jsonString);
34 }

```

8.2 前端实现

前端代码我们从以下几方面实现：

1. 查询表单绑定查询条件对象模型

这一步在页面上已经实现了，页面代码如下：

```

<!-- 搜索表单-->
<el-form :inline="true" :model="brand" class="demo-form-inline">

    <el-form-item label="当前状态">
        <el-select v-model="brand.status" placeholder="当前状态">
            <el-option label="启用" value="1"></el-option>
            <el-option label="禁用" value="0"></el-option>
        </el-select>
    </el-form-item>

    <el-form-item label="企业名称">
        <el-input v-model="brand.companyName" placeholder="企业名称"></el-input>
    </el-form-item>

    <el-form-item label="品牌名称">
        <el-input v-model="brand.brandName" placeholder="品牌名称"></el-input>
    </el-form-item>

```

2. 点击查询按钮查询数据

```

<el-form-item>
    <el-button type="primary" @click="onSubmit">查询</el-button>
</el-form-item>

```

从上面页面可以看到给 `查询` 按钮绑定了 `onSubmit()` 函数，而在 `onSubmit()` 函数中只需要调用 `selectAll()` 函数进行条件分页查询。

3. 改进 `selectAll()` 函数

子页面加载完成后发送异步请求，需要携带当前页码、每页显示条数、查询条件对象。接下来先对携带的数据进行说明：

- `当前页码` 和 `每页显示条数` 这两个参数我们会拼接到 URL 的后面
- `查询条件对象` 这个参数需要以 json 格式提交给后端程序

修改 `selectAll()` 函数逻辑为

```

1 var _this = this;
2
3 axios({
4     method:"post",
5     url:"http://localhost:8080/brand-case/brand/selectByPageAndCondition?
currentPage="+this.currentPage+"&pageSize="+this.pageSize,
6     data:this.brand
7 }).then(function (resp) {
8
9     //设置表格数据
10    _this.tableData = resp.data.rows; // {rows:[],totalCount:100}
11    //设置总记录数
12    _this.totalCount = resp.data.totalCount;
13 })

```

9. 前端代码优化

咱们已经将所有的功能实现完毕。而针对前端代码中的发送异步请求的代码，如下

```
1 var _this = this;
2
3 axios({
4     method:"post",
5     url:"http://localhost:8080/brand-case/brand/selectByPageAndCondition?
6     currentPage="+this.currentPage+"&pageSize="+this.pageSize,
7     data:this.brand
8 }).then(function (resp) {
9
10    //设置表格数据
11    _this.tableData = resp.data.rows; // {rows:[],totalCount:100}
12    //设置总记录数
13    _this.totalCount = resp.data.totalCount;
14 })
```

需要在成功的回调函数（也就是`then`函数中的匿名函数）中使用`this`，都需要在外边使用`_this`记录一下`this`所指向的对象；因为在外边的`this`表示的是Vue对象，而回调函数中的`this`表示的不是vue对象。这里我们可以使用`ECMAScript6`中的新语法（箭头函数）来简化这部分代码，如上面的代码可以简化为：

```
1 axios({
2     method:"post",
3     url:"http://localhost:8080/brand-case/brand/selectByPageAndCondition?
4     currentPage="+this.currentPage+"&pageSize="+this.pageSize,
5     data:this.brand
6 }).then((resp) => {
7
8    //设置表格数据
9    this.tableData = resp.data.rows; // {rows:[],totalCount:100}
10   //设置总记录数
11   this.totalCount = resp.data.totalCount;
12 })
```

箭头函数语法：

```
1 (参数) => {
2     逻辑代码
3 }
```

箭头函数的作用：

替换（简化）匿名函数。