Web 前端开发职业技能标准

串讲教案

（中级）

工业和信息化部教育与考试中心

二〇一九年制

第一篇 MySQL 数据库基础与应用

第二篇 PHP 技术与应用

第三篇 web 前后端数据交互技术

第四篇 响应式开发技术

第一篇 MySQL 数据库基础与应用

第一章 MySQL 介绍

第一节 MySQL 简介

Michael Widenius "Monty"是一位编程天才。19 岁的时候，他从赫尔辛基理工大学辍学开始全职工作，因为大学已经没有什么东西可以教他了。33 岁时，他发布了 MySQL，成为了全世界最流行的开源数据库。并以 10 亿美元的价格，将自己创建的公司 MySQL AB 卖给了 SUN，此后，Oracle 收购了 SUN。

Widenius 离开了 Sun 之后，觉得依靠 Sun/Oracle 来发展 MySQL，实在很不靠谱，于是决定另开分支，这个分支的名字叫做 MariaDB。MariaDB 名称来自他的女儿 Maria 的名字。

MariaDB 跟 MySQL 在绝大多数方面是兼容的，对于开发者来说，几乎感觉不到任何不同。目前 MariaDB 是发展最快的 MySQL 分支版本，新版本发布速度已经超过了 Oracle 官方的 MySQL 版本。

第二节 关系型数据库

关系型数据库以行和列的形式存储数据，这一系列的行和列被称为表，一组表组成了数据库。

表与表之间的数据记录有关系。

数据保存在表内，行（记录）用于记录数据，列（字段）用于规定数据格式

第二章 MySQL 管理

第一节 MySQL 数据库管理

查看数据库

show databases;

创建数据库

```
# 如果没有修改 my.ini 配置文件的默认字符集，在创建数据库时，指定字符集
create database db_name character set 'utf8';

# 特殊字符(关键字)用反引号
create database `create`;
```

MySQL\data 目录下将自动生成一个对应名称的目录，目录内部有一个 **db.opt** 文件

显示数据库创建信息

show create database db_name;

删除数据库

drop database db_name;

进入（使用）数据库

use database;

显示当前打开的数据库

select database();

第二节 MySQL 表结构管理

创建数据表

create table 表名(字段 字段类型,....)

MySQL\data 目录下的数据库目录中将自动生成一个对应名称的.frm 文件

删除数据表

drop table 表名;

查看数据表

```
show tables;

# 查看字母是'abc'开头的表
show tables like 'abc%';  # %是通配符
```

查看表创建信息

show create table 表名;

查看数据表结构

desc 表名;

第三节 MySQL 用户管理

登录

Mysql 是基于 C/S 架构，必须在客户端通过终端窗口，连接 MySQL 服务器，进行操作。

```
mysql -h host -u user -p
Enter password: ********
```

用户管理

- 超级用户 root
- 修改账号密码，例：

```
## DOS 命令下修改，将 root 账号密码修改为 1234
mysqladmin -u root password 1234   ##语句最后不要加分号，否则密码就是 “1234；”

##mysql 命令
set password for 'root'@'localhost'= password('1234');
```

- 创建用户

使用 create 语句进行创建用户，语句格式如下：

create user 'username'@'host' identified by 'password';

其中：username 表示要创建的用户名；host 表示指定该用户在哪个主机上可以登陆，如果是本地用户可用 localhost，如果想让该用户可以从任意远程主机登陆，可以使用通配符%；password 表示该用户的登陆密码，密码可以为空，如果为空则该用户可以不需要密码登陆服务器。例：

```
create user 'zhangsan'@'localhost' identified by '123456';
```

- 删除用户

删除用户使用 drop 语句，语句格式如下：

drop user 'username'@'host';

修改配置文件 my.ini

字符集

MySQL 默认字符集是 latin(拉丁)，改变为 utf8 才能正确显示中文

[mysqld] 下添加

character-set-server=utf8

init-connect='\set NAMES utf8'

修改默认引擎

InnoDB 优于 MYISAM

default-storage-engine=MYISAM 修改为 default-storage-engine=InnoDB

个性化 mysql 提示符

MySQL 默认提示符是" mysql>" ，可以个性化定制，例如："mysql(数据库)>"

[mysql]下添加

prompt="mysql(\d)>"

第四节 默认数据库

- information_schema

  提供了访问数据库元数据的方式。什么是元数据呢？元数据是关于数据的数据，如数据库名或表名，列的数据类型，或访问权限等。有些时候用于表述该信息的其他术语包括"数据词典"和"系统目录"。

- performance_schema

  mysql 5.5 版本 新增了一个性能优化的引擎

- mysql

  这个是 MySQL 的核心数据库，主要负责存储数据库的用户、权限设置、关键字等 MySQL 自己需要使用的控制和管理信息。不可以删除，也不要轻易修改这个数据库里面的表信息。

- test

  安装时候创建的一个测试用数据库，空数据库，没有任何表，可以删除（新版 mysql 已取消）。

第三章 SQL 基本语法

第一节 SQL 语句简介

SQL 语言

SQL(Structured Query Language)是用于访问和处理数据库的**标准计算机语言**。使用 SQL 访问和处理数据系统中的数据，这类数据库包括：Oracle,mysql,Sybase, SQL Server, DB2, Access 等等。

基本规范

- SQL 对大小写不敏感，一般数据库名称、表名称、字段名称全部小写
- MySQL 要求在每条 SQL 命令的末端使用分号（MS Access 和 SQL Server 2000，则不必在每条 SQL 语句之后使用分号）。

注释

```
mysql> select 1+1;     # 这个注释直到该行结束
mysql> select 1+1;     -- 这个注释直到该行结束
mysql> select 1  /* 这是一个在行中间的注释 */ + 1;
mysql> select 1+
/*
这是一个
多行注释的形式
*/
1;
```

第二节 MySQL 基本数据类型

字段类型

数据类型是指列、存储过程参数、表达式和局部变量的数据特征，它决定了数据的存储方式，代表了不同的信息类型。不同的数据库，数据类型有所不同，MySQL 数据库有以下几种数据类型：

- 字符串型

类型	字节	大小	说明

char	1	0-255 字符 (2^8)	定长字符串
varchar	2	0-65 535 字符 (2^16)	变长字符串
tinytext	1	0-255 字符 (2^8)	短文本（与 char 存储形式不同）
text	2	0-65 535 字符(2^16)	文本
mediumtext	3	0-16 777 215 字符 (2^24)	中等长度文本
longtext	4	0-4 294 967 295 字符 (2^32)	极大文本

注意：**char 和 varchar 需要指定长度**，例如：char（10）

整数型

类型	字节	范围（有符号）	范围（无符号）	用途
tinyint	1	(-128，127)	(0，255)	很小整数值
smallint	2	(-32 768，32 767)	(0，65 535)	小整数值
mediumint	3	(-8 388 608，8 388 607)	(0，16 777 215)	中整数值
int 或 integer	4	(-2 147 483 648，2 147 483 647)	(0，4 294 967 295)	整数值
bigint	8	(-9 233 372 036 854 775 808，9 223 372 036 854 775 807)	(0，18 446 744 073 709 551 615)	很大的整数值

很多人喜欢在定义数据时，这样写：

```
create table tbl_name(
age int(10)
);
```

int 后面()中的数字，不代表占用空间容量。而代表最小显示位数。这个东西基本没有意义，除非你对字段指定 zerofill。mysql 会自动分配长度：int(11)、tinyint(4)、smallint(6)、mediumint(9)、bigint(20)。所以，建议在使用时，就用这些默认的显示长度就可以了。不用再去自己填长度（比如：int(10)、tinyint(1)之类的基本没用）。

浮点型

类型	字节	范围	用途
float(M,D)	4	23bit（约 6~7 位 10 进制数字）	单精度浮点数 值绝对能保证精度为 6~7 位有效数字
double(M,D)	8	52bit（约 15~16 位 10 进制数字）	双精度浮点数值 精度为 15~16 位有效数字
decimal(M,D)	M+2	依赖于 M 和 D 的值	定点型

M（精度），代表总长度（整数位和小数位）限制

D（标度），代表小数位的长度限制。

M 必需大于等于 D

数字的修饰符	功能	说明
unsigned	无符号	非负数
zerofill	前导 0	整形前加 0（自动添加 unsigned）

日期型

类型	字节	范围	格式	用途
date	3	1000-01-01---9999-12-31	YYYY-MM-DD	日期值
time	3	-838:59:59---838:59:59	HH:MM:SS	时间值或持续时间
year	1	1901---2155	YYYY	年份值
datetime	8	1000-01-01 00:00:00---9999-12-31 23:59:59	YYYY-MM-DD HH:MM:SS	混合日期和时间值
timestamp	4	1970-01-01 00:00:00/2038 结束时间是第 2147483647 秒，北京时间 2038-1-19 11:14:07，格林尼治时间 2038 年 1 月 19 日 凌晨 03:14:07	YYYYMMDD HHMMSS	时间戳，混合日期和时间值

列举与枚举

名称	字节	说明
set	1、2、3、4 或 8	列举：可以取 SET 列表中的一个或多个元素（多选）
enum	1 或 2	枚举：可以取 ENUM 列表中的一个元素（单选）

```
create table students(
 id tinyint, # 微小整型
 name varchar(10), #变长字符
 sex enum('m','w'), #单选
 birthday date, # 日期型
 tel char(11), # 定长字符
 city char(1), # 城市
 hobby set('1','2','3','4'), #多选
 introduce      text # 个人介绍
);
```

字段属性

属性	功能	说明
not null	非空	必须有值，不允许为 null
default	默认值	当插入记录时没有赋值，自动赋予默认值（允许为 null）
primary key	主键	惟一标识一行数据的字段（主键自动为 not null）
auto_increment	自动增量	不能单独使用，必须与 primary key 一起定义
unique(unique key)	唯一	记录不能重复（一张表可以有多个 unique，允许为 null）

第三节 数据的 增、删、改

增删改查（简称：CURD）

增

```
# 方法 1：指定字段
insert into students(name,age) values('张三','20');

# 方法 2： 省略字段名，字段位要一一对应，不能跳过（auto_increment 字段，可以使用 null 或 default）
```

```
insert into students values(null,'张三','20');

# 方法 3：批量增加数据
insert into students(name,age) values('张三','20')，('李四','21')，('王五','22') ……
```

删

用 delete 删除记录，一定要加 where 条件，否则表数据全部删除！！
delete from 表名 where xx=xxx;

用 truncate 删除记录，不能加 where 条件，直接删除全部记录，id 索引重新从 1 开始
truncate table 表名;

改

#单条修改
update 表名 set xx=xx,xxx=xx where xxx=xxx and xxx=xxx;

```
#多条修改
update students
    set name = case id # id 字段
        when 1 then 'zhangsan'
        when 2 then 'lisi'
        when 3 then 'wangwu'
        when 4 then 'zhaoliu'
    end,
    city = case id
        when 1 then '2'
        when 2 then '4'
        when 3 then '1'
        when 4 then '2'
    end
where id in (1,2,3,4);
```

第四章 数据的 查

SELECT 语句用于从表中选取数据。结果被存储在一个结果表中（称为结果集）。

第一节 查询表达式

```
# 当前使用的数据库
select database();

# 查看当前 MySQL 版本
select version();

# 查看当前用户
select user();

# 查看运算结果
select 1+2;
```

第二节 条件表达式

from 子句

```
# 字段用','隔开，至少有一个字段，最终结果集按照这个这个顺序显示
select 字段 1,字段 2... from 表名;

# *代表所有字段
select * from 表名;
```

- distinct(去重)

```
# 去重后的结果，distinct 必须紧接着 select 后面
select distinct 字段 from 表名;

# 统计不重复的个数
select count(distinct 字段) from 表名;
```

where 子句

- where 子句适用于对记录的 **删、改、查** 操作
- 对记录进行过滤，如果没有指定 where 子句，则显示所有记录

- 在 where 表达式中，可以使用函数或运算符，运算符包括：

类型	运算符
算术	+ - * / %
比较	> < >= <= != =
逻辑	or || and && not !
提升优先级	( )

```
# 搜索 id<20 的所以数据
select * from students where id<20;

# 搜索 id 编号为偶数的数据
select * from students where id%2=0;
```

- where 条件关键字

  in：查询一个集合的数据

```
# 搜索 id 在（1,3,7）之中的数据
select * from students where id=1 || id=3 || id=7;
select * from students where id in(1,3,7);

# 一次删除多条记录
delete from students where id=3 || id=15 || id=23;
delete from students where id in(3,15,23);
```

between...and... ：查询一个区间的数据

```
# 搜索 id 在 20-40 之间的数据
select * from students where id>20 && id<40;
select * from students where id between 20 and 40;

# 删除 id 在 20-40 之间的数据
delete from students where id between 20 and 40;
```

not：排除

```
# 搜索 id 除了 20-40 之间的数据
select * from students where id not between 30 and 40;
```

like 子句

用于模糊查询 %：任意字符长度 _ ：一个字符长度

```
# 搜索 name 名字以 5 结尾的数据
select * from students where name like '张%';

# 搜索 name 名字包含字母 s 的数据
select * from students where name like '%二%';

# 搜索 id 以 5 结尾的两位数 数据
select * from students where id like '_5';
```

limit 子句

控制查询记录条数，数据表中的记录，**索引从 0 开始**

```
select * from students limit 2 # 返回两条记录
select * from students limit 3,4 # 从索引为 3 的记录开始，返回 4 条记录

# php 中的分页功能，偏移值的计算：（当前页-1） * 每页记录数

select name from student limit 4 offset 3
# 还可以使用 offset（偏移）：从索引为 3 的记录开始，返回 4 条
```

group by(结果分组)

根据**给定数据列的每个成员**对查询结果进行分组统计，最终得到一个分组汇总表

利用 group by 分组信息进行统计，常见的是配合 max 等聚合函数筛选数据后分析。

select 指定的字段要么作为分组的依据（Group By 语句的后面），要么就要被包含在聚合函数中。

```
#  简单分组
select sex from students group by sex;
+-----+
| sex |
+-----+
| m   |
| w   |
+-----+

# 聚合函数分组
select count(*),city from students group by city;
+----------+----------+
| count(*) | livecity |
+----------+----------+
```

```
|      7 |      1 |
|      8 |      2 |
|      1 |      3 |
|      3 |      4 |
|      2 |      5 |
+----------+----------+
```

order by(结果排序)

按照给定的字段进行排序，asc：升序(默认) desc：降序

如果同时选择多个字段，先按第一个字段排序，如果第一个字段值相等，再尝试第二个字段，以此类推

```
# 默认升序
select * from students order by birthday;

# 降序
select * from students order by birthday desc;
```

查询语句的书写顺序

select →字段→ from→表名→where→group by→order by→limit

别名

第三节 多表查询

a

students学生表

Id	name
6	张三
7	李四

b

score成绩表

Id	user_id	course_id	score
1	1	2	85
2	6	4	98

c

course科目表

Id	name
3	数学
4	语文

a表 人名 select a.name from students as a where a.id=6;

b表 分数 select b.score from score as b where b.user_id=6 and b.course_id=4;

C表 科目 select c.name from course as c where c.id=4;

合并语句 并添加 **字段别名**

```
select a.name,c.name as course ,b.score from
students as a, score as b, course as c
where a.id=6 and b.user_id=6 and b.course_id=4 and c.id=4;
```

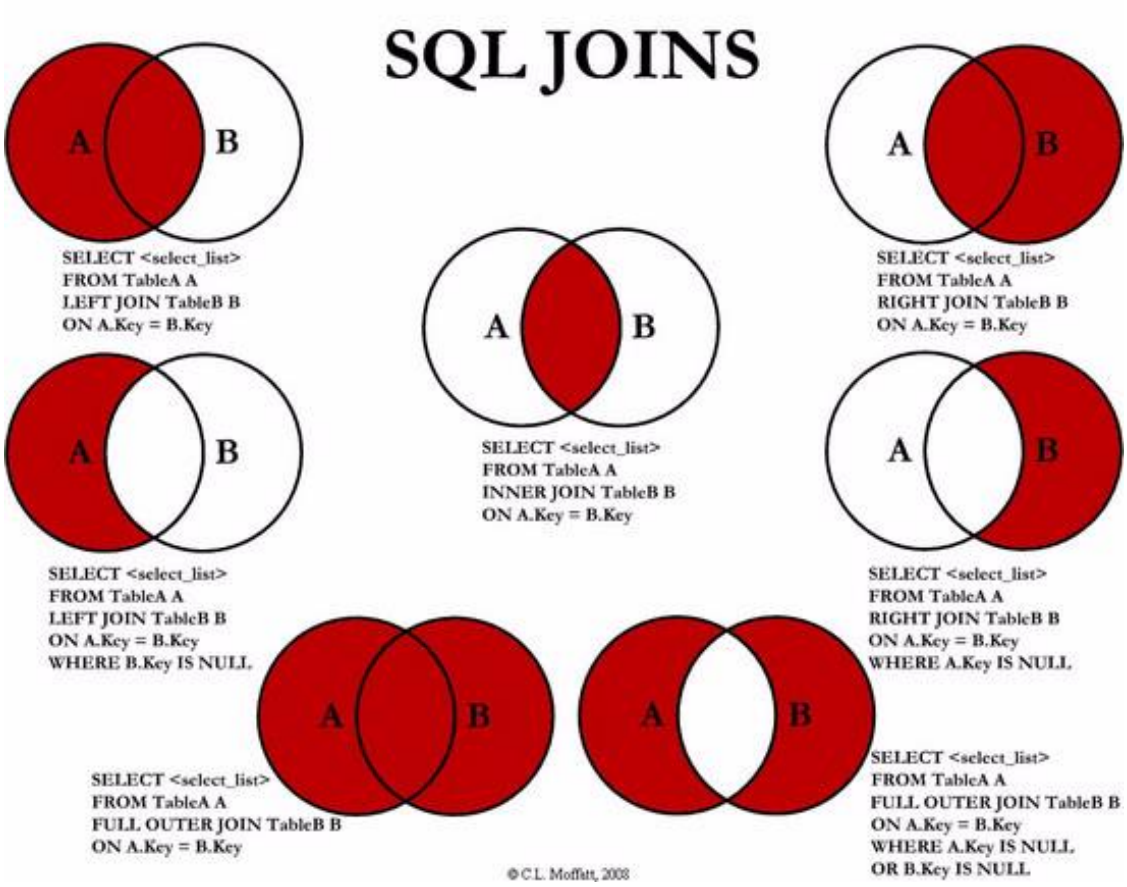
最终结果

name	course	score
赵六	物理	78.50

```
# 上图的语句中的 where 语句还可以修改
select a.name,c.name,b.score from students as a,score as b,course as c where b.user_id=6 and b.course_id=4 and a.id=b.user_id and c.id=b.course_id;
```

表连接

有时为了得到完整的结果，我们需要从两个或更多的表中获取结果。我们就需要执行 join



内连接

- JOIN: 如果表中有至少一个匹配，则返回行

```
select a.*,b.name from students as a [inner] join city as b on a.livecity=b.id;
```

- LEFT JOIN: 即使右表中没有匹配，也从左表返回所有的行

```
select a.*,b.name from students as a left join city as b on a.livecity=b.id;
```

- RIGHT JOIN: 即使左表中没有匹配，也从右表返回所有的行

```
select a.*,b.name from students as a right join city as b on a.livecity=b.id;
```

- FULL JOIN: 只要其中一个表中存在匹配，就返回行（mysql 不支持）

```
# 带有连接的分组
select city.name, count(*) from students,city where students.livecity=city.id group by st
udents.livecity;
+--------+----------+
| name   | count(*) |
+--------+----------+
| 北京   |        7 |
```

```
| 上海  |     8 |
| 杭州  |     1 |
| 深圳  |     3 |
+--------+----------+
```

第四节 子查询

子查询（subquery）是指出现在其他 SQL 语句内的 select 子句（嵌套在查询内部，且必须始终出现在圆括号内）

```
# 查找城市名称是北京的
# 普通方式查询
select * from students where livecity=1;

# 子查询方式
select * from students where livecity=(select id from city where name='北京');
```

- 子查询可以包含多个关键字或条件，如：distinct、group by、order by、limit、函数等
- 子查询的外层可以是：select，insert，update

第五章 视图与事务

第一节 视图

视图是从一个或几个基本表（或视图）中导出的虚拟的表。在系统的数据字典中仅存放了视图的定义，不存放视图对应的数据。视图是原始数据库数据的一种变换，是查看表中数据的另外一种方式。可以将视图看成是一个移动的窗口，通过它可以看到感兴趣的数据。 视图是从一个或多个实际表中获得的，这些表的数据存放在数据库中。那些用于产生视图的表叫做该视图的基表。一个视图也可以从另一个视图中产生。

数据库中视图是一个重要的概念，其优势在于：

- 安全：有的数据是需要保密的，如果直接把表给出来进行操作会造成泄密，那么可以通过创建视图把相应视图的权限给出来即可保证数据的安全。
- 高效：复杂的连接查询，每次执行时效率比较低，建立视图，每次从视图中获取，将会提高效率。
- 定制数据：将常用的字段放置在视图中。

创建视图

CREATE VIEW view_name AS SELECT column_name(s) FROM table_name WHERE condition;

其中，view*name 为欲创建的视图名，column*name(s)为查询的列名，table_name 为查询的表名，condition 为查询条件。

修改视图

```
# ALTER 语句：
ALTER VIEW view_name AS SELECT column_name(s) FROM table_name WHERE condition;
# CREATE OR REPLACE 语句：
ALTER VIEW view_name AS SELECT column_name(s) FROM table_name WHERE condition;
```

删除视图

DROP view_name;

查询视图

SHOW TABLES;
SHOW TABLE STATUS;

这两个命令不仅可以显示表名及表信息，而且会显示出所有视图名称及视图信息。除此之外，使用 SHOW CREATE VIEW 命令可以查看某个视图的定义，格式如下：

SHOW CREATE VIEW view_name;

关系数据库表是用于存储和组织信息的数据结构,数据结构的不同，直接影响操作数据的效率和功能，对于 MySQL 来说，它提供了很多种类型的存储引擎，可以根据对数据处理的需求，选择不同的存储引擎，从而最大限度的利用 MySQL 强大的功能。

第二节 事务

MyISAM 引擎

MyISAM 表是独立于操作系统的，这说明可以轻松地将其从 Windows 服务器移植到 Linux 服务器，建立一个 MyISAM 引擎的 tb_Demo 表，就会生成以下三个文件：

tb*demo.frm 存储表定义 tb*demo.MYD 存储数据 tb_demo.MYI 存储索引。

MyISAM 无法处理事务，特别适合以下几种情况下使用：

1. 选择密集型的表。MyISAM 存储引擎在筛选大量数据时非常迅速，这是它最突出的优点。
2. 插入密集型的表。MyISAM 的并发插入特性允许同时选择和插入数据。例如：MyISAM 存储引擎很适合管理邮件或 Web 服务器日志数据。

InnoDB 引擎

InnoDB 是一个健壮的事务型存储引擎，InnoDB 还引入了外键约束，在以下场合下，使用 InnoDB 是最理想的选择：

1. 更新密集的表。InnoDB 存储引擎特别适合处理多重并发的更新请求。
2. 事务。InnoDB 存储引擎是支持事务的标准 MySQL 存储引擎。
3. 外键约束。MySQL 支持外键的存储引擎只有 InnoDB。

4. 自动灾难恢复。与其它存储引擎不同，InnoDB 表能够自动从灾难中恢复。

事务处理

以银行转账业务为例，张三→李四转账 100 元，这是一个完整事务，需要两步操作：

1. 张三数据表减去 100 元

2. 李四数据表增加 100 元

如果在 1 步完成后，操作出现错误（断电、操作异常等），使 2 步没有完成，此时，张三减去了 100 元，而张三却没有收到 100 元

为了避免这种情况的发生，就将整个操作定义为一个事务，任何操作步骤出现错误，都会回滚到上一次断点位置，避免出现其他错误。

```
# 开始
begin;
update tbl_a set money=money-100 where name='zhangsan';
update tbl_b set money=money+100 where name='lisi';

# 提交
commit;

# 回滚
rollback;
```

第六章 索引约束分区

第一节 索引

索引是帮助 MySQL 高效获取数据的**数据结构**

数据库在保存数据之外，还维护着满足特定查找算法的数据结构，这些数据结构以某种方式引用（指向）数据，这样就可以在这些数据结构上实现高级查找算法。这种数据结构，就是索引。索引可以大大提高 MySQL 的检索速度。

在 MySQL 中，对于一个 Primary Key 的列，MySQL 已经自动对其建立了 Unique 和 Index。

```
# 创建索引
create table 表名(
id int not null,
username varchar(16) not null,
index(username(length))   ### 用 username 字段作为索引
);

# 显示索引
show index from 表名;

# 删除索引
alter table 表名 drop index name;
```

第二节 约束(constraint)

约束保证数据的完整性和一致性，根据约束的字段数目的多少，约束又分为表级约束和列级约束

- 列级约束：针对某一字段来使用
- 表级约束：针对两个或两个以上的字段使用

约束类型包括：

- not null(非空约束)
- primary key (主键约束)
- unique key (唯一约束)
- default (默认约束)
- foreign key(外键约束)

唯一(unique)约束

unique 约束唯一标识数据库表中的每条记录。

unique 和 primary key 约束均为列提供了唯一性的保证。

primary key 被自动定义为 unique 约束。

注意: 每个表可以有多个 unique 约束，但是每个表只能有一个 primary key 约束。

```
# 第一种方式
create table persons(
id_p int not null,
address varchar(255),
city varchar(255),
phone varchar(11) unique # 定义字段的同时，定义约束
);

# 第二种方式
create table persons(
id_p int not null,
address varchar(255),
city varchar(255),
phone varchar(11),
unique phone(phone) # 单独一行命令，定义约束

# 第三种方式
alter table persons add unique city(city); #修改表
```

默认(default)约束

用于约束对应列中的值的默认值（除非默认为空值，否则不可插入空值）

```
create table persons(
id tinyint primary key auto_increment,
name varchar(30),
sex enum('m','w') default 'm', # 定义 sex 默认值为:'m'
)
```

主键(primary key)约束

每张数据表只能存在一个主键，主键保证记录的唯一性，主键自动为 not null（同时作为表的索引）。

```
# 为没有主键的表添加主键
alter table 表名 add primary key (字段名)
```

外键(foreign key)约束

外键约束是为了保持数据一致性，完整性，实现一对一或一对多关系

- 子表（具有外键列的表）和 父表（子表所参照的表），存储引擎只能为 innoDB。
- 外键列和参照列必须具有相似的数据类型。
  - 如果是数字类型，数字的长度、是否有符号位 必须相同
  - 字符类型的长度则可以不同
- 外键列和参照列必须创建索引（如果外键列不存在索引的话，MySQL 将自动创建索引）。

```
# 先建父表 子表才能建外键  父表和子表必须都是 innodb 引擎

# city 父表
create table city(
id tinyint primary key,
name varchar(10) not null
)engine=INNODB;

# students 子表
create table students(
id tinyint primary key auto_increment, #id
# 定义字段时同时定义
city tinyint,  # 外键字段类型要与主表相同
foreign key(city) references city(id),  # city 字段作为外键 ， 引用 city 表中的 id
)engine=INNODB;
--------------------------------------------------------------------------
# 主表的数据可以修改，但不能删除
# 删除 city 中的记录
delete from city where id=1;

# 创建外键以后，再删除 city 记录，就会报错：
ERROR 1451 (23000): Cannot delete or update a parent row: a foreign key constraint fail
s (`hxsd`.`students`, CONSTRAINT `students_ibfk_1` FOREIGN KEY (`city`) REFERE
NCES `city` (`id`))
```

第三节 删除约束

- 删除 primary key

alter table 表名 drop primary key;

- 删除 index

alter table 表名 drop index index_name;

- 删除外键约束

alter table 表名 drop foreign key FK_ID;

第四节 索引与约束的关系

索引是面向数据库本身的，用于查询优化等操作。约束则更多的是业务上的关系。

通常，创建唯一约束就自动获取唯一索引，是因为数据库认为对数据库进行唯一检查时，如果该字段上有索引会很快，所以创建唯一约束就默认创建唯一索引。同样，常见的主键即是唯一性的约束，也是个索引。但对于 not null 这样的约束，数据库是不会创建索引的。

第五节 分区

如果一张表的数据量太大，不仅查找数据的效率低下，而且难以找到一块集中的存储来存放。为了解决这个问题，数据库推出了分区的功能。MySQL 表分区主要有以下四种类型：

- RANGE 分区：

  RANGE 即范围分区，根据区间来判断位于哪个分区。这些区间要连续且不能相互重叠，使用 VALUES LESS THAN 操作符来进行定义。

```
create table test(
id int DEFAULT null,
name char(30),
datedata date
)
PARTITION BY RANGE (year(datedata)) (
PARTITION part1 VALUES LESS THAN (1990) ,
PARTITION part2 VALUES LESS THAN (1995) ,
PARTITION part3 VALUES LESS THAN (2000) ,
PARTITION part4 VALUES LESS THAN MAXVALUE );
```

- LIST 分区

LIST 分区类似于按 RANGE 分区，区别在于 LIST 分区是基于列值匹配一个离散值集合中的某个值来进行选择。

```
create table test1(
id int not null,
name char(30),
career VARCHAR(30)
)
PARTITION BY LIST (id) (
PARTITION part0 VALUES IN (1,5) ,
PARTITION part1 VALUES IN (11,15) ,
PARTITION part2 VALUES IN (6,10) ,
PARTITION part3 VALUES IN (16,20)
);
```

- HASH 分区

  HASH 分区基于用户定义的表达式返回值来选择分区，该表达式对要插入到表的行中列值进行 Hash 计算。

```
CREATE TABLE employees (
 id INT NOT NULL,
firstname VARCHAR(30),
lastname VARCHAR(30),
hired DATE NOT NULL DEFAULT '1970-01-01',
separated DATE NOT NULL DEFAULT '9999-12-31',
job_code INT,
store_id INT
)
PARTITION BY HASH(store_id) PARTITIONS 4;
```

- KEY 分区

  KEY 分区类似 HASH，但是 HASH 允许用户使用自定义表达式，而 KEY 分区不允许，它需要使用 MySQL 服务器提供的 HASH 函数，同时 HASH 分区只支持整数分区，而 KEY 分区支持除 BLOB 和 TEXT 类型外其他列。

```
CREATE TABLE tk (
col1 INT NOT NULL,
col2 CHAR(5),
col3 DATE,
PRIMARY KEY(col1)
)
PARTITION BY KEY (col1) PARTITIONS 3;
```

第七章 存储过程、触发器

第一节 存储过程

存储过程（Stored Procedure）是一种在数据库中存储复杂程序，以便外部程序调用的一种数据库对象。

存储过程是为了完成特定功能的 SQL 语句集，经编译创建并保存在数据库中，用户可通过指定存储过程的名字并给定参数(需要时)来调用执行。

存储过程思想上很简单，就是数据库 SQL 语言层面的代码封装与重用。

- 优点
  - 存储过程可封装，并隐藏复杂的商业逻辑。
  - 存储过程可以回传值，并可以接受参数。
  - 存储过程无法使用 SELECT 指令来运行，因为它是子程序，与查看表，数据表或用户定义函数不同。
  - 存储过程可以用在数据检验，强制实行商业逻辑等。
- 缺点
  - 存储过程，往往定制化于特定的数据库上，因为支持的编程语言不同。当切换到其他厂商的数据库系统时，需要重写原有的存储过程。
  - 存储过程的性能调校与撰写，受限于各种数据库系统。

存储过程的创建和调用

- 存储过程就是具有名字的一段代码，用来完成一个特定的功能。
- 创建的存储过程保存在数据库的数据字典中。

create procedure 存储过程名称(in|out|inout 参数名称 参数类型,.....)
begin
过程体;
end

实例

```
create procedure getStudentCount()
begin
```

```
        select count(*) as num from student where classid=8;
end
```

查询、修改与删除

- 查询

查看所有存储过程状态：

```
SHOW PROCEDURE STATUS;
```

查看对应数据库下所有存储过程状态：

```
SHOW PROCEDURE STATUS WHERE DB='数据库名';
```

查看名称包含 Student 的存储过程状态：

```
SHOW PROCEDURE STATUS WHERE name LIKE '%Student%';
```

查询存储过程详细代码：

SHOW CREATE PROCEDURE 过程名;

- 修改

ALTER PROCEDURE 过程名 ([过程参数[,...]]) 过程体;

- 删除

DROP PROCEDURE 过程名;

注意：不能在一个存储过程中删除另一个存储过程，只能调用另一个存储过程。

调用存储过程

MySQL 存储过程用 call 和过程名以及一个括号，括号里面根据需要，加入参数，参数包括输入参数、输出参数、输入输出参数调用，格式如下：

CALL 存储过程名 ([过程参数[,...]])

第二节 触发器

触发器（trigger），也叫触发程序，是与表有关的命名数据库对象，是 MySQL 中提供给程序员来保证数据完整性的一种方法，它是与表事件 INSET、UPDATE、DELETE 相关的一种特殊的存储过程，它的执行是由事件来触发，比如当对一个表进行 INSET、UPDATE、DELETE 事件时就会激活它执行。因此，删除、新增或者修改操作可能都会激活触发器，所以不要编写过于复杂的触发器，也不要增加过

多的触发器，这样会对数据的插入、修改或者删除带来比较严重的影响，同时也会带来可移植性差的后果，所以在设计触发器的时候一定要有所考虑。

创建触发器

CREATE TRIGGER trigger_name trigger_time trigger_event ON tb_name FOR EACH ROW trigger_stmt

其创建触发器的参数说明如下表：

参数	说明
trigger_name	标识触发器名称，由用户自行指定；
trigger_time：	标识触发时机，取值为 BEFORE 或 AFTER；
trigger_event：	标识触发事件，取值为 INSERT、UPDATE 或 DELETE；
tb_name：	标识建立触发器的表名，即在哪张表上建立触发器；
trigger_stmt：	触发器程序体，可以是一句 SQL 语句， 或者用 BEGIN 和 END 包含的多条语句，与存储过程类似。
FOR EACH ROW	表示任何一条记录上的操作满足触发事件都会触发该触发器。

触发事件类型介绍如下表所示：

事件	描述
INSERT 型触发器	插入某一行时激活触发器，可能通过 INSERT、LOAD DATA、REPLACE 语句触发
UPDATE 型触发器	更改某一行时激活触发器，可能通过 UPDATE 语句触发
DELETE 型触发器	删除某一行时激活触发器，可能通过 DELETE、REPLACE 语句触发

我们想要使班级表中的班内学生数随着学生的添加自动更新，则采用触发器来实现最为合适，创建触发器如下：

```
CREATE trigger tri_stuInsert after insert
on student for each row
BEGIN
        declare c int;
        set c = (select stuCount from class where classID=new.classID);
        update class set stuCount = c + 1 where classID = new.classID;
END
```

从需求我们可以得知，班内学生数的变化是在插入学生记录之后发生的，所以创建的触发器类型为 after insert 类型。

查看触发器

SHOW TRIGGERS;

删除触发器

DROP TRIGGER [IF EXISTS] trigger_name

触发器执行顺序

日常开发中创建的数据库通常都是 InnoDB 数据库，在数据库上建立的表大都是事务性表，也就是事务安全的，这时触发器的执行顺序主要是：

1. 如果 BEFORE 类型的触发器执行失败，SQL 无法正确执行。
2. 如果 SQL 执行失败时，AFTER 类型的触发器不会触发。
3. 如果 AFTER 类型的触发器执行失败，数据会回滚。

如果是对数据库的非事务表进行操作，当触发器执行顺序中的任何一步执行出错，那么就无法回滚了，数据可能会出错。

第八章 MYSQL 函数

第一节 运算函数

- abs(x) : 返回 x 的绝对值
- floor(x) : 返回小于 x 的最大整数值
- round(x,y) : 返回参数 x 的四舍五入的有 y 位小数的值
- mod(x,y) : 返回 x/y 的模（余数）
- greatest(x1,x2,...,xn) : 返回集合中最大的值
- least(x1,x2,...,xn) : 返回集合中最小的值

第二节 字符串函数

- trim(str) :去除字符串首尾两端的空格
- upper(str) : 字符串转大写
- concat(s1,s2...,sn) : 将 s1,s2...,sn 连接成字符串

```
# concat
insert into tbl_name values( concat('abc','def') );
```

第三节 日期函数

- year(date) : 返回日期 date 的年份(1000~9999)
- month(date) : 返回 date 的月份值(1~12)
- day(date)：返回 date 的日(1~31)
- curdate() : 返回当前的日期
- week(date) : 返回日期 date 为一年中第几周(0~53)
- now() : 返回当前的日期和时间
- curtime() : 返回当前的时间
- hour(time) : 返回 time 的小时值(0~23)

- minute(time) : 返回 time 的分钟值(0~59)

第四节 聚合函数

- count(col) : 统计记录的条数
- sum(col) : 求和
- avg(col) : 求平均值
- max(col): 求最大值
- min(col) : 求最小值

```
# count 统计总记录数
select count(*) from tbl_name;

# sum 年龄总和
select sum(age) from tbl_name;

# avg 平均年龄
select avg(age) from tbl_name;

# 最大年龄
select min(birthday) from tbl_name;  # 日期最小的
```

FOUND_ROWS()

```
# FOUND_ROWS 函数配合 SQL_CALC_FOUND_ROWS 用于获取总记录数（忽略 limit）

select SQL_CALC_FOUND_ROWS * from qa_list limit 3;
select FOUND_ROWS();
```

第一个 sql 里面的 SQL_CALC_FOUND_ROWS 不可省略，它表示需要取得结果数，也是后面使用 FOUND_ROWS()函数的铺垫。
FOUND_ROWS()返回的结果是临时的。如果程序往后会用到这个数字，必须提前它保存在一个变量中待用。

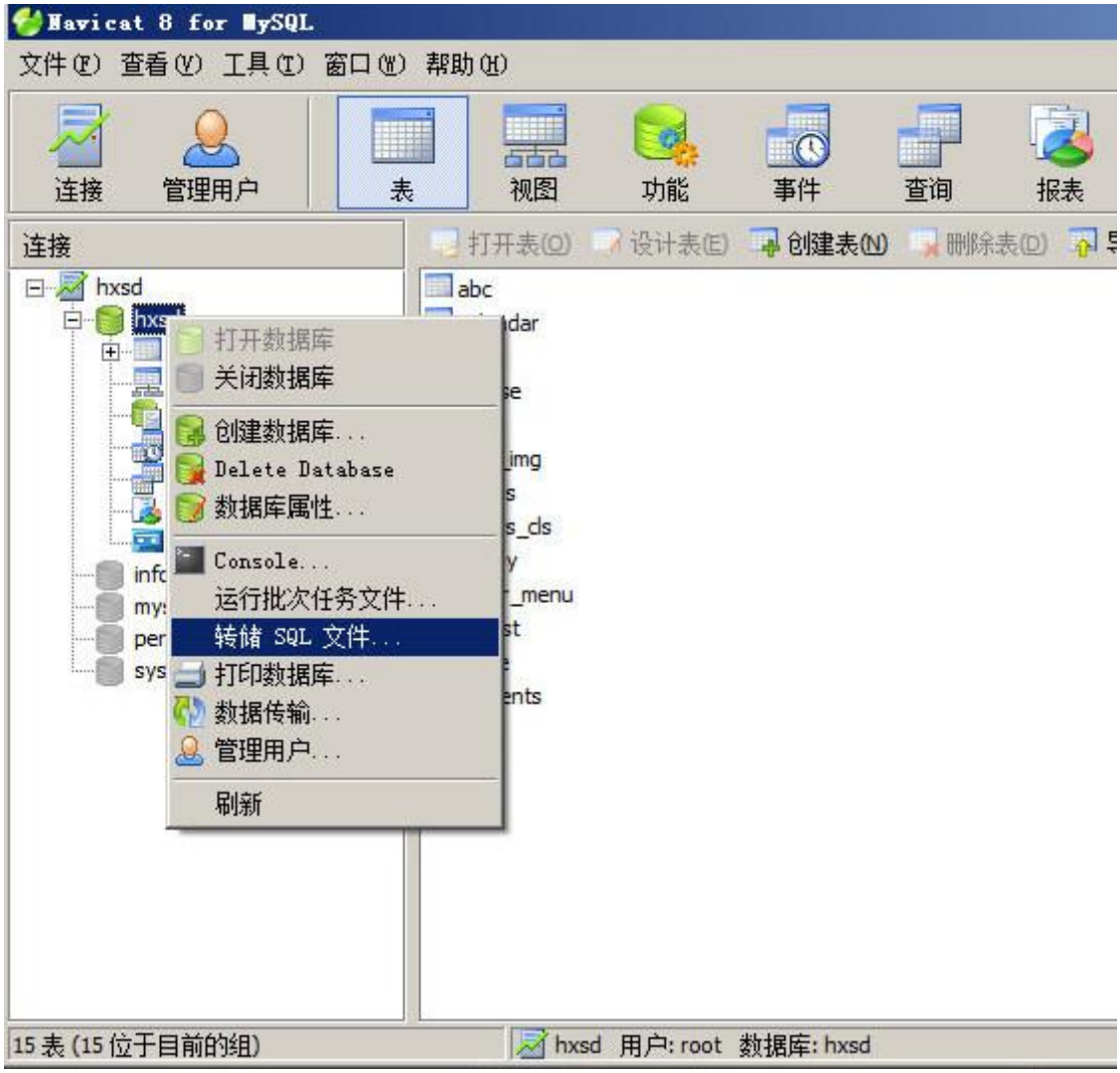
FOUND_ROWS()与 count()的区别：
1、当 SQL 限制条件太多时， count()的执行效率不是很高，最好使用 FOUND_ROWS()
2、当 SQL 查询语句没有 where 等条件限制时，使用 count()函数的执行效率较高。

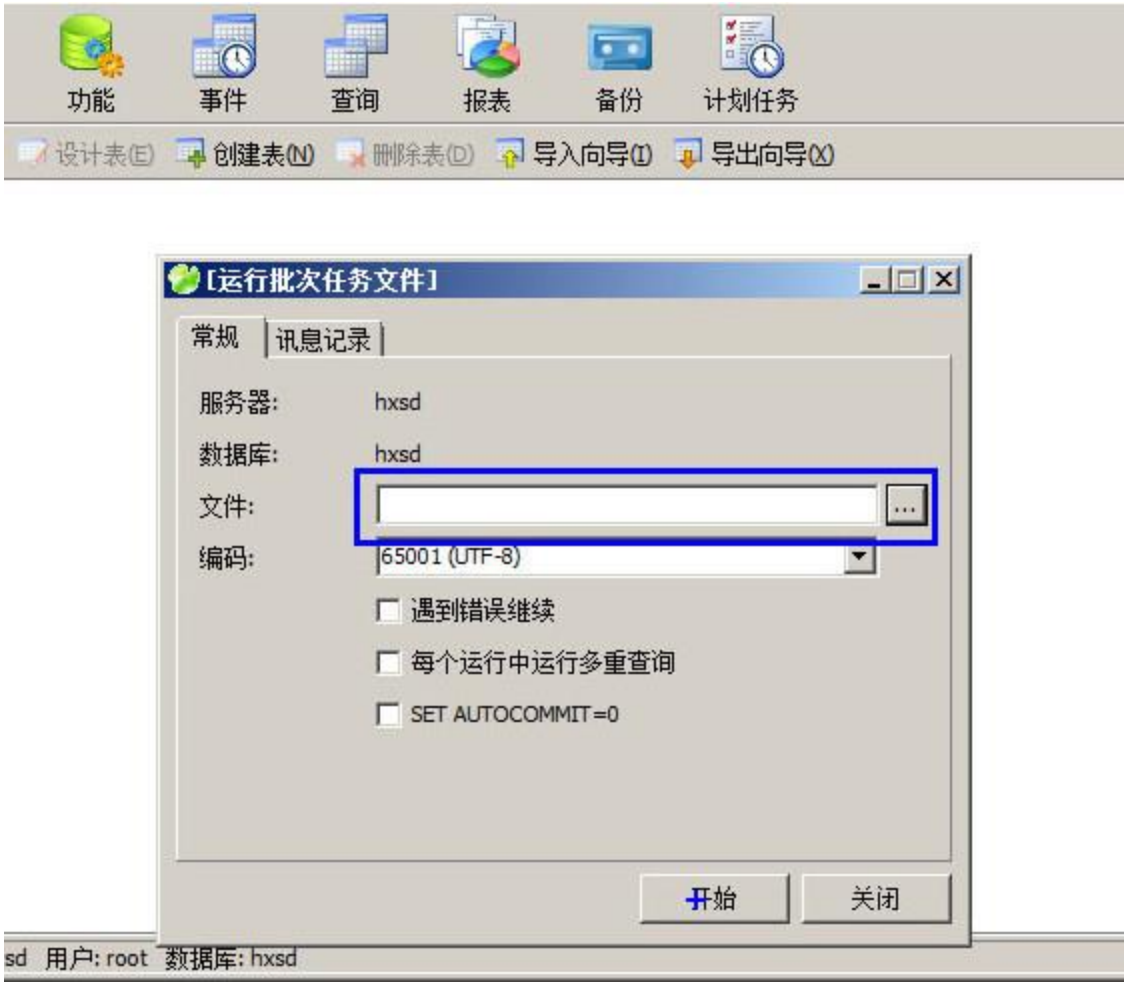
第九章 数据库备份

使用 Navicat 可视化工具导入 导出数据，备份 恢复数据

第一节 备份数据

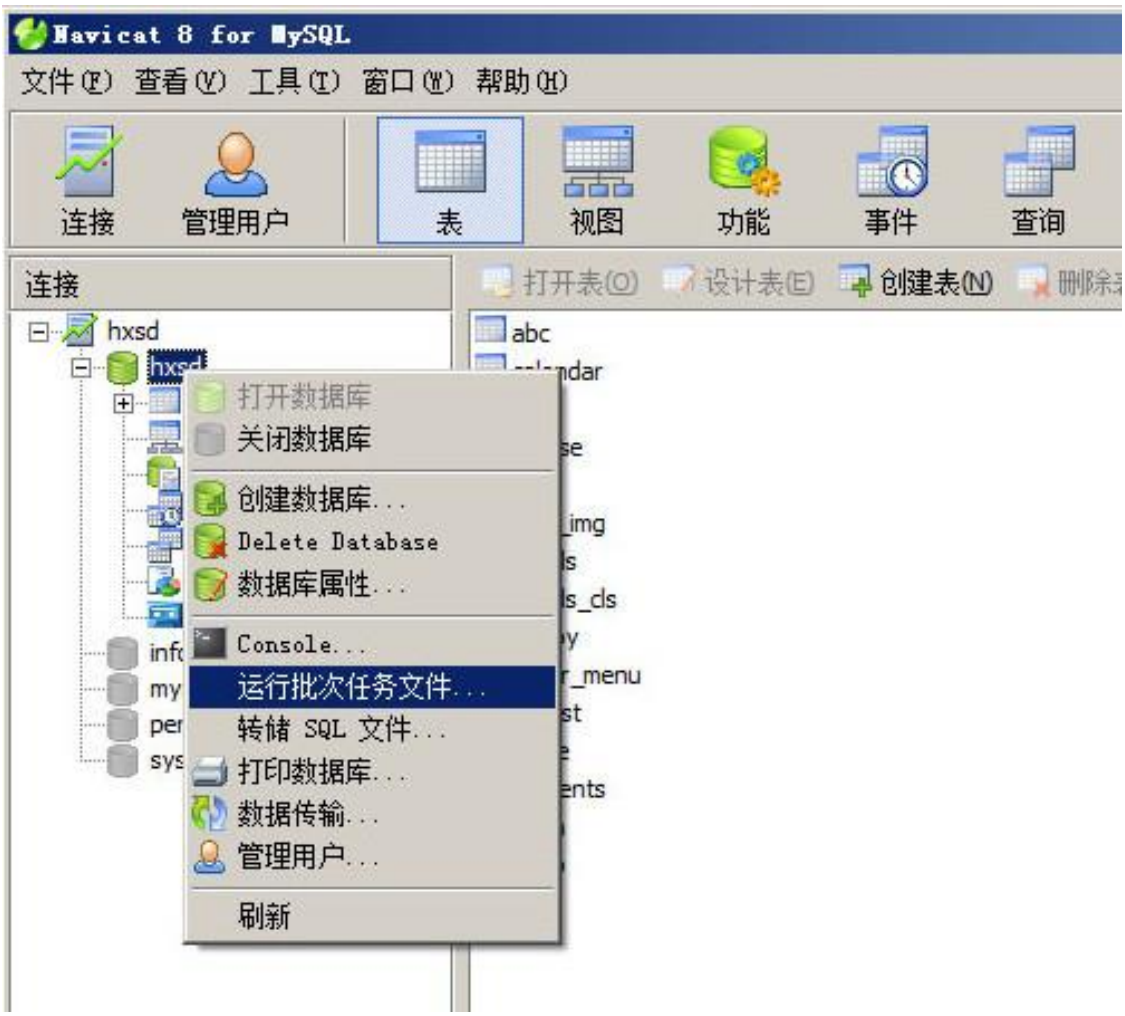
mysqldump -u [用户名] -p [密码] [数据库名] > [path]/[名称].sql





第二节 恢复数据

mysql>source c:\system.sql


Navicat 8 for MySQL
文件(F) 查看(V) 工具(T) 窗口(W) 帮助(H)
连接
管理用户
表
视图
功能
事件
查询
连接
打开表(O)
设计表(E)
创建表(N)
hxsd
abc
打开数据库
关闭数据库
创建数据库...
Delete Database
数据库属性...
Console...
运行批次任务文件...
转储 SQL 文件...
打印数据库...
数据传输...
管理用户...
刷新

第二篇 PHP 技术与应用

第一章 网站介绍

第一节 动态网站

概念

- 误区：不是指网站当中包含动态图片、滚动图等动态效果
- 正解：采用数据库技术开发的网站，网页上的内容都是通过数据库提取出来动态更新的

B/S 软件

- 通过浏览器和服务器(Browser / Server)进行通信的软件，客户端给服务器发请求，服务器处理客户请求返回结果
- 优势：不用下载，不用更新，打开网页，直接使用
- 劣势：功能受限（受浏览器环境限制）

使用技术

- 前端：html+css+javascript
- 后端：php、asp、java
- 数据库：MySQL、SQLServer、ORACLE、DB2

第二节 动态网站开发所需构件

web 前端开发

- HTML
- CSS
- javascript

- 浏览器

Web 后端开发（服务器端）

- web 服务器：**Apache**、IIS、Tomcat…
- 数据库：**MySQL**、MariaDB、Oracle、SQL Server…
- 服务器端编程语言：**PHP**、ASP、JSP…

第三节 php 语言简介

PHP（原名 Personal Home Page 的缩写，已经正式更名为 "PHP: Hypertext Preprocessor"，中文名：“超文本预处理器”）是一种通用开源脚本语言。语法吸收了 C 语言、Java 和 Perl 的特点，利于学习，使用广泛，主要适用于 Web 开发领域。PHP 独特的语法混合了 C、Java、Perl 以及 PHP 自创的语法。用 PHP 做出的动态页面与其他的编程语言相比，PHP 是将程序嵌入到 HTML 文档中，执行效率很高。

php 的诞生

1994 年 Rasmus Lerdorf 设计了 PHP 的第一个版本 PHP1.0，并于 1995 年将其通过社区发布。1996 年又设计了 PHP2，1998 年，由于 Zeev Suraski 和 Andi Gutmans 当时正在做毕业设计，需要一个用于开发 Web 程序的语言，他们也考虑了 asp 和 jsp，但由于 ASP 只能运行在 Windows 平台，而 JSP 又过于复杂，因此，他们最后选择了 PHP，但他们发现，PHP 的功能当时还十分有限，因此，他们决定为 PHP 重新编写底层的解析程序，这就是 PHP 的第一个被广泛使用的版本----PHP3.0。

php 的优点

PHP 是最好的语言，php 和最好的语言几乎没有关系。一般用来讽刺一些没见过世面，把自己归属于某一种编程语言的语言教徒。

事实上每一门语言都有其设计的灵魂，有它的取舍。很难说什么是最好的。比起做个脑残粉总是鼓吹自己擅长的语言，不如多学几门语言，开阔视野。

PHP 可能不是世界上最好的语言，但他一定是最适合新手学习编程的入门语言，也是进入 Web 开发领域的绝佳语言。

- **简单易学**

  既能面向过程，又能面向对象，安装后各种扩展集一身，包括但不限于 mysql、json、mbstr 等，方便至极。html 和 PHP 混写，执行效率很高，最新版本都内置了小型 webserver（连 apache 都不用）。

- **日臻完善**

  语言创建者积极上进优化。各大 php 论坛非常活跃，语言 bug 全世界的程序员和你一起修复。

- **框架成熟**

  框架层出不穷，国外 Ci、laravel、kohana、zf，国内 thinkphp 功能完善，敏捷开发就靠它！！

- **使用广泛**

  最重要的是大小公司都用，全球前 100 万的站点中，有 70%左右的站点用 PHP 开发，找工作不愁啊！！！！

php 的缺点

- **标准库的函数名、参数顺序实在是难以预测**

  例如：字符串操作系列函数，strpos、strcspn 里两个词素没有任何分隔符，到了 str*replace*、*str*repeat 却又冒出了下划线

- **协调性和可靠性**

  例如：变量名是大小写敏感的，函数名和类名却是大小写不敏感的

- **奇异的写法**

  例如：数组末尾添加一个元素, $arr[] = 1

- **PHP 的异常捕获系统非常异常**

  很难说清楚到底哪些情况会抛出异常哪些并不会

第四节 环境配置

集成环境

要想使用这门语言，需要搭配相应的开发环境，主要包括：

- Apache web 服务器
- MySQL 数据库
- PHP 语言引擎

以上三个模块可以分别安装，再进行相关的配置（手动安装配置相对复杂）。通常使用以上三者的**集成环境**进行快速部署：

WAMP：Window 操作系统 **LAMP**：Linux 操作系统 **MAMP**：苹果 MAC 操作系统

以上三个集成环境针对不同操作系统，下载安装后，开发环境就已经自动搭建好了。

以 WAMP 为例，首先，需要在网站上下载相应的软件到本地电脑。 安装成功，右下角 W 的图标为绿色

运行原理

- get 请求 ：从服务器上获取数据
- post 请求：向服务器提交数据

配置文件

配置文件是每一项服务的核心，配置文件缺失或修改错误，将导致服务无法正常运行

- Apache 服务配置文件：**D:\wamp64\bin\apache\apache2.4.23\conf\httpd.conf**
- MySQL 服务配置文件：**D:\wamp64\bin\mysql\mysql5.7.14\my.ini**
- PHP 模块配置文件：**D:\wamp64\bin\apache\apache2.4.23\bin\php.ini**

注意：配置文件修改后，必须重启服务

www 目录

www 目录在 wamp 目录下，是网站的根目录，开发的所有文件，都要放在 www 目录中，通过浏览器输入 localhost 访问本地服务器

localhost

本地服务器，安装了 wamp 软件的你的电脑，既是客户机，又是服务器，通过浏览器访问的是你本地的服务器，故名称为 localhost

第二章 PHP 语言基础

第一节 hello world

文件命名

- 文件后缀名为 php
- 文件名中不可包含中文、空格、特殊符号
- 建议使用有意义的英文单词命名

语言标记

```
//标准风格
<?php
        .....
?>
```

注意：纯 php 脚本文件要求：，1. 开始标签要在第一行顶头写，2. 删除结束标签。

php 标签之外是 html 语言环境，在纯 php 代码环境下，这些 html 字符（包括看不见的空格或者回车，制表符号）也会一同输出，引发意外错误。因此，在编码规范中规定：库文件、class 类文件等只要是纯 php 代码的文件，要删除结尾的 ?> 结束标签。

```
<!DOCTYPE html>
<html>
        <body>
                <button>ajax 异步请求</button>
        </body>
</html>
<script src="jquery-1.11.3.min.js"></script>
<script>
        $("button").click(function(){
                $.get("doAction.php",function(data){
                        alert(data? "ok":"no");
                })
        })
</script>

<?php
        echo false; //输出:""
?>
```

```
//结束标签后有空格，输出：" "
//所以要删除 php 的结束标签
```

注释符与结束符

```
//单行注释
/*多行注释*/

//结束符使用英文分号 “;”
$hello="hello world";
```

常用命令和系统函数

- **echo**

  echo 输出：只能输出字符串、数字、布尔（true:1 false:空）类型的数据

```
<?php
   $hello="hello world";
        echo "<h1>{$hello}</h1>";
?>

//快速输出----------------------------------------------
<h1><?= $hello ?></h1> //相当于 <?php echo $hello ?>
```

- **var_dump()**

此函数显示一个或多个表达式的结构信息，包括表达式的类型与值。数组将递归展开值，通过缩进显示其结构

注意： var_dump 中的变量必须是存在的，如果变量存在但值是空，则返回 false 没有变量时，则返回 NULL 该函数有输出的功能，因此不必加其它的输出函数

- **require 与 include**

  1. require

     常用于引入重要文件，若引入失败会直接影响到当前整个脚本，引入失败报 Error 错误

  2. include

     常用于引入普通文件，若引入失败不会对当前脚本有较大的影响，引入失败报 Warning 错误

  3. require_once

避免重复引入，其他规则同 require

4. include_once

避免重复引入，其他规则同 include

- **header**

用于定义 HTTP 协议头信息。

第二节 变量与常量

变量

- 声明变量：$
- 命名规则：字母数字下划线、首字母不能为数字、严格区分大小写、且不能使用关键字！推荐驼峰命名法
- 变量销毁：unset(变量名)，被销毁的变量在内存中被释放
- 引用变量

```
//变量引用：用不同的名字访问同一个变量内容
$man="zhangsan";
$student=&$man; //用&符号引用
var_dump($man===$student);//true
```

常量

使用 define 关键字定义常量，常量命名要全部大写，常量的数据类型不能是 资源、对象

```
//定义常量
define("SCHOOL","清华大学");

//判断常量
var_dump( defined("SCHOOL") ); //true
```

变量与常量的差异

差异	变量	常量
定义	$声明	define()函数定义
命名	大小写敏感	必须大写（行业规范）
赋值	可以重新赋值	不能再赋值
数据类型	8 种数据类型	只能是标量
销毁	unset() 销毁	不能销毁
判断方法	isset() 判断是否定义	defined() 判断是否定义
作用域	局部作用域	全局作用域

第三节 数据类型

八种数据类型

- 四种标量类型
  - 布尔型（boolean）
  - 整 型（integer）范围：2^32 或 2^64（超出自动转换为浮点型）
  - 浮点型（float）范围：双精度
  - 字符串（string）
- 两种复合类型
  - 数组型（array）
  - 对象型（object）
- 两种特殊类型
  - 资源型（resource）
  - 空 型（null）

不同进制

```
$number1 = 0b10;      //0b 开头 二进制 结果:2
$number2 = 0123;      //0 开头 八进制 结果:83
$number3 = 0x123;     //0X 开头 十六进制 结果:291
```

对象型

具有一定功能和特征的单个事物，对象是属性和方法的集合

资源型

变量可以是文件夹、一个文件、从数据库中得到的结果集等，操作这个变量，相当于操作这些资源。

NULL 型

null 也是数据，通常表示一种状态，变量没有任何值，就用 null 表示。以下情况会得到 null：

- 直接将一个变量赋值为 null
- 将一个变量销毁后再次使用该变量
- 直接使用一个不存在的变量

第四节 数据类型转换

自动类型转换

在特殊运算时，会有自动类型转换的情况

自动转换为： false	自动转换为：true	自动转换为：int	自动转换为： string
整型： 0	非 0 整数	true : 1	true : "1"
浮点： 0.0 或 0.00	非 0 浮点数	false : 0	false : " "
字符串： " " 或 "0"	非空字符串	null: 0	null: " "
数组：空数组	非空数组		
null	对象		

未知变量（或被销毁变量）			

强制类型转换

- 改变原变量类型

  使用 **settype()** 函数可以将一个指定变量强制转换为另一种指定类型

- 不改变原变量类型

  格式：新变量 = (指定变量类型) 原变量;

```
//强制类型转换：不改变原变量类型
//定义一个存储变量
$old = 1;
$new;  //用于接收转换之后的变量

//开始转换
$new = (Boolean) $old; //true
$new = (Integer) $old;  //1
$new = (Float) $old;    //float 1
$new = (String) $old;   //"1"
$new = (Array) $old;    //Array
$new = (Object) $sum; //Object
```

常用变量检测函数

函数	说明
gettype()	判断变量类型： 返回 boolean、integer、double（ float 返回“double”）、string、array、object、resource、null、unknown type
isset()	检测变量是否存在
empty()	判断一个变量是否为空（字符串、数组、0、null）
其他	判断变量类型的系列函数 is*array()*、*is*bool()、is*float()*、*is*integer()、is*null()*、*is*numeric()、is*object()*、*is*resource()、 is*string()* 和 *is*scalar()是否为标量

常用数学运算函数

函数名	语法	说明
rand	rand([int, int])	返回一个随机整数
mt_rand	mt_rand ([int, int])	使用 Mersenne Twister 算法返回随机整数

abs	abs (mixe)	返回数字的绝对值
min	min(array) min(mixed, mixed [, …])	返回最小值
max	max (array) max(mixed, mixed [, …])	返回最大值
round	round (float [, int [, int]])	四舍五入
floor	floor (float)	舍去小数部分取整
ceil	ceil (float)	小数部分非零返回整数部分就加 1
pow	pow(mixed, mixed)	返回 base 的 exp 次方的值
sqrt	sqrt (float)	返回平方根
sin	sin (float)	返回正弦值
cos	cos (float)	返回余弦值
tan	tan (float)	返回正切值
deg2rad	deg2rad (float)	将角度转换为弧度
bindec	bindec (string)	二进制转十进制
decbin	decbin (int)	十进制转二进制
octdec	octdec (string)	八进制转十进制
decoct	decoct (int)	十进制转八进制
hexdec	hexdec (string)	十六进制转十进制
dechex	dechex (int)	十进制转十六进制
exp	float exp(float arg)	返回 **e** 的 arg 幂次值，**e** 为自然对数的底数，值为 2.718282。
pi	float pi (void)	返回圆周率的值

第五节 字符串

三种定义方式

单引号	双引号	定界符
不支持解析变量	支持解析变量，不支持表达式	同双引号

不能插入单引号	不能插入双引号	可以插入双引号和单引号
只能转义 反斜杠 和 单引号	支持解析转义字符(\n、\t、\s 等）	同双引号

```php
$num=10;

//单引号
$title='hxsd'.$num.'beijing';

//双引号 支持解析变量，不能解析表达式,例如：{$hxsd+1}
$school ="<div id='box'>
        <h1>{$hxsd}</h1>
</div>";

//定界符（可以任意取名）
$str2 = <<<mark
<div id='box'>
        <h1 class="main">{$hxsd}</h1>
</div>
mark;

//注意：-----------------------------------------------------
//定界符中可以使用单、双引号，可以解析变量
//定界符中的字符串，会保留所有格式，包括各种空格（尽量顶格写，避免开始的空格）
//结束标记一定要顶格写
//开始、结束标记（上面代码中的‘mark’）后不能再有空格等字符（包括注释语句也不能有）

//字符串支持折行，注意:折行位置会多一个空格
$txt="abcdef
ghijk"; //显示结果：abcd efghijk
```

常用字符串操作函数

函数	功能
trim（）	去除字符串首尾空白等特殊符号或指定字符序列
strtolower()	转换为小写
strtoupper()	转换为大写
htmlspecialchars()	格式化字串中的 html 标签
strip_tags()	函数剥去 HTML、XML 以及 PHP 的标签
md5()	计算字符串的 MD5 散列值
strlen()	字节长度 utf8 编码 西文：1 字节 汉字：3 字节

mb_strlen()	字符个数
substr()	字符串截取函数（中文乱码）
mb_substr()	字符串截取函数（中文不乱码）
strstr()	查找字符串的首次出现，开始向后截取全部，并返回
strpos()、stripos()	查找字符串在另一字符串中第一次、最后一次出现的位置 （**不区分大小写**）
strrpos()、strripos()	同上 （**区分大小写**）
	翻转字符串
	按字节来比较字符串
str_replace()	字符串替换函数
str_repeat（）	重复一个字符串
ltrim() 、 trim() 、 rtrim()	去除左侧 、 两侧 、 右侧多余字符(默认删除空格)
explode(符号 , 字符串)	字符串 → 数组
implode(符号 , 数组)	数组 → 字符串
md5（）、sha1（）	字符串加密

第六节 数组

数组类型

- 关联式数组：下标为有意义的字符串
- 索引式数组：下标为默认从 0 开始的数值

定义数组

- 直接赋值方式定义

```
$a[] = 10;
$a[] = 20;
$a['name'] = '张三';
$a['sex'] = '男';

//结果
```

```
/*
 * 0 => int 10
 * 1 => int 20
 * 'name' => string '张三' (length=9)
 * 'sex' => string '男' (length=3)
 */
```

- 使用 array()函数

```
$b = array(10,20,30);
$b = array('name'=>'张三','sex'=>'男','age'=>28);
```

- 快捷赋值

```
$c = [10,20,30];
$c = ['name'=>'张三'，'sex'=>'男','age'=>28];
```

- 二维数组与多维数组

```
//二维数组
$group = array(
        'one'=>array('张三','李四','王五'),
        'two'=>array('赵六','孙七'),
);

//定义一个三维数组
$class = array(
        'class01'=>array(
                'one'=>array('张三','李四','王五'),
                'two'=>array('赵六','孙七'),
    ),
        'class02'=>array(
                'one'=>array('张三','李四','王五'),
                'two'=>array('赵六','孙七'),
    );
);

//获取李四
echo $class['class02']['one'][1]; //李四
```

数组的遍历

```
//for 遍历索引数组----------------------------------------------------
$arr=[11,22,33,44，55];
for($i=0; $i<count($arr); $i++){
    var_dump($arr[$i]);
};
```

```php
//foreach-------------------------------------------------------------
$f_arr=["name"=>"zhangsan","age"=>18,"sex"=>"m"];
foreach ($f_arr as $key=>$value) {
   echo $key.":".$value."<br>";
};
//list（只用于索引数组）-----------------------------------------------
list($a,$b,$c,$d,$e) = ["张三","李四","王五","小明","小红"];
echo $a,$b,$c,$d,$e;
```

数组的输出

```php
$b = 3.1; $c = TRUE; var_dump($b,$c);

/* 输出： float(3.1) bool(true)*/
```

复制数组

```php
//复制数组
$arr1=[111,222,333];
$arr2=$arr1; //$arr2 是个新数组
var_dump($arr2===$arr1);//false

//用&引用数组
$arr3=&$arr1;
var_dump($arr3===$arr1);//true
```

数组常用函数

函数	功能
array_values()	数组中所有的值放入一个新的索引数组，并返回
array_keys()	数组中所有的键放入一个新的索引数组，并返回
array_reverse()	颠倒数组顺序
in_array(元素，数组)	检查数组中是否存在某个值
count()	计算数组中的单元数目 或 对象中的属性个数
array_unshift()	在数组开头插入一个或多个元素
array_push()	在数组结尾插入一个或多个元素
array_unique()	移除数组中重复的值
array_pop()	删除数组最后一个元素
array_shift()	删除数组开头的元素

array_splice(数组，索引，数量)	删除指定元素（ 删除元素后，会重排数组索引，用 unset 删除不会重排数组索引）
sort()	排序 (升序)
rsort()	排序（降序）
array_merge()	合并一个或多个数组

数组转 JSON

函数	说明
json_encode(数组)	JSON 格式编码 （参数必须是 utf-8 编码，否则会得到空字符或者 null）
json_decode(字符串，参数)	对 JSON 格式的字符串进行解码，参数： true : 返回 数组 false : 返回 对象

第七节 运算符

类型	运算符
算术运算符	+ - * / %
赋值运算符	= += -= *= /= %= .=
比较运算符	> >= < <= != !== == ===
逻辑运算符	&& || not and or
字符串运算符	.
其他运算符	三元运算符 ==？ ：
错误抑制符	@
提升优先级符号	( )

```php
$a=10;
$b=20;
$c="hxsd";

//数字与数字相加：数学运算
echo $a+$b; //30

//数字与字符串相加
echo $a+'2018 你好';//2028  数字开头为 2018
echo "2018 你好"+$a;//2028  同上
```

```
echo $a+'你好 2018';//10  以字母开头字符串，转换成 0
echo "你好 2018"+$a;//10  同上

//数字与 bool 值
echo $a+true;//11  true:1   false：0
```

第三章 流程控制

第一节 分支结构

单一分支结构

```php
//判断你是否长得帅
$face = '帅';

//开始判断
if($face=='帅'){
  echo '你真是帅呆了，酷毙了~~';
}
```

双向分支结构

```php
//开始判断
if(true){
  echo "ok";
}else{
  echo "no";
}
```

多向分支结构一

```php
//学生成绩管理系统
$score = 100;

//判断学生成绩在那个分数段
if($score>=90 && $score<=100){
  echo "优+";
}else if($score>=80 && $score<=89){
  echo "优";
}else if($score>=70 && $score<=79){
  echo "良";
}else if($score>=60 && $score<=69){
  echo "中";
}else($score>=0 && $score<=59){
  echo "差";
}
```

多向分支结构二

```
$action = 'save';

//分支
switch($action){
  case 'save':
    echo "输入数据！ ";
    break;
  case 'del':
    echo "删除数据！ ";
    break;
  default:
    echo "请输入正确数据";
}
```

第二节 循环结构

while 循环

```
While(条件表达式){
        //若条件成立，则执行这里的循环体
        //改变初值的条件;
}
```

do……while 循环

```
do{
        //循环体
        //改变初值的条件;
}while(条件表达式);
```

for 循环

```
//语法格式
for(初值;范围;递增){
        //循环体
}
```

特殊的流程控制语句

语句	说明
break	在 switch 分支当中和循环当中使用，代表结束当前分支或循环

continue	在循环当中使用，代表跳过当前层循环，若指定数字，则跳过指定层循环

第四章 函数

第一节 函数的分类

系统函数

php 提供了丰富的系统函数，可直接使用。这些函数涵盖了软件开发的大部分功能，具体的使用方法，请查看 php 开发手册。除了各种功能分类清晰的函数，还有一些常用的杂项函数：

杂项函数	描述
define()	定义一个常量
defined()	判断常量是否存在
die()	输出一条消息，并退出当前脚本
eval()	把字符串按照 PHP 代码来运行
sleep()	延迟代码执行若干秒

自定义函数

自定义函数命名口诀：字母数字下划线，首字母不能为数字，不会区分大小写，且不能使用关键字，不能重复来定义

第二节 函数的参数

形参

- 形式上的参数
- 在函数定义时声明

实参

- 实际上的参数，在函数使用时声明
- 实参和形参类型需一致
- 实参和形参数量要一一对应（定义的参数必须传值，除非有默认值）

形参的默认值

- 若某个形参的值总是固定的某一个值，可以使用默认值指定
- 具有默认值的形参，放到参数列表后面

```
function fun3($a,$b=20){
        return $a+$b;
}
//有默认值的参数可以省略
fun3(10)
```

查看实参函数

函数	说明
func*get*args()	返回一个包含函数参数列表的数组
func*get*arg(索引)	返回参数数组的某一个
func*num*args()	返回实参的个数

第三节 函数的返回值

- 函数当中若遇到 return，则会将 return 后方的内容返回到函数调用处进行保存
- return 后面的语句将不再执行
- 若函数没有任何返回，则默认返回 null 类型

第四节 变量的作用域

局部变量

在函数内部定义，只作用于函数内部

```
//默认情况下，函数参数通过值传递（因而即使在函数内部改变参数的值，它并不会改变函数外部的值）
$a=1;//外部变量
function test($arg){
        $arg+=100;
}
test($a);//只将外部变量的值传进去
var_dump($a) ;// 1  外部变量$a 的值并没有改变
```

全局变量

在函数外部定义，作用于当前整个脚本，在函数内部使用需要使用 global 关键字声明

```
$a=10;//全局变量
function test(){
        global $a; //用 global 引入全局变量，在函数内部使用，子函数引入父函数变量，不能使用 global
        $a++;
}
test();
var_dump($a); //结果：11
```

注意：global 只能用于引入全局变量，子函数引入父函数变量，不能使用 global。

引用变量传参

使用 global 命令，将全局变量引入到函数内部，但这种方式不够灵活，可以使用**引用变量**的方法：

```
//定义变量时，用&修饰参数
$a=10;
function run(&$arg){
  $arg++;
}
run($a);
var_dump($a) //结果：11
```

静态变量

在函数内部定义，作用于函数内部，使用 static 关键字声明

```
function test(){
        static $a=1; //静态变量
        $a++;
        echo $a;
}
test(); //2
test(); //3
test(); //4
//echo $a; //报错 静态变量只作用于函数内部，外部无法读取
```

第五节 超全局变量

超全局变量

在全部作用域中始终可用的内置变量

超全局变量	说明
$_GET	通过 URL 参数传递 (get 请求) 给当前脚本的变量的数组
$_POST	通过 'post' 方式给当前脚本的变量的数组
$_COOKIE	保存 Cookies 信息的数组
$_SESSION	保存 SESSION 信息的数组
$_FILES	通过 HTTP POST 方式上传到当前脚本的项目的数组
$_ENV	包含服务器端环境变量（版本，目录，用户名等等）的数组
$_SERVER	服务器和执行环境信息
$_REQUEST	默认包含了 *_GET，*_POST 和 $_COOKIE 的数组
$GLOBALS	以上变量的集合

获取 application/json 的 post 数据

php://input 是一个流，可以读取没有处理过的 POST 数据（即原始数据）

```
$postjson = file_get_contents("php://input");
```

第六节 其他应用

变量函数

```
//定义测试函数
function test(){
        echo 'web 前端';
}

//将函数的名称以字符串形式存储到指定变量
$ceshi = 'test';

//此时该变量可以作为函数来使用，使用规则和函数一致
$ceshi();

//应用于 回调函数----------------------------------------------------
function play(){
```

```
    echo "play";
}
function run($fn){
    $fn();
};
run("play");//将回调函数名作为字符串传入
```

递归

```
//回文数递归(recursion)函数
function recursion($num){
  echo $num;
  //判断
  if($num>1){
    recursion($num-1);
  }
  echo $num;
}
//调用函数
recursion(3);  //结果:3 2 1 1 2 3
```

第五章 正则表达式

正则表达式（regular expression）描述了一种字符串匹配的模式，正则表达式的主要作用：

可以用来检查一个串是否含有某种子串

将匹配的子串做替换或者从某个串中提取符合某个条件的子串等

第一节 基本概念

- 他是一个特殊的公式，由原子、元字符、模式修正符组成；它也是一个字符串！只不过是一个**特殊的字符串**
- 主要实现对字符串的**匹配、分割、查找、替换**等操作

第二节 正则函数

用单引号定义正则

php 的正则是以文本方式声明，文本当中用"/ /"包裹正则内容，如果使用双引号，正则中的“$”会被解析为变量（要加\转义符），因此，用单引号定义正则更安全。

PCRE 规则与函数

PCRE（Perl Compatible Regular Expressions ）是一个 Perl（一种功能丰富的计算机程序语言）兼容的正则表达式库，是以 preg 开头的一套函数：

函数	功能
preg_match(正则 , 字符串 , 结果数组)	单一匹配 结果数组: 第一个匹配到的结果 返回：匹配数量 （错误:false）
preg*match*all(正则 , 字符串 , 结果数组)	匹配所有 结果数组: 全部匹配到的结果 返回：匹配数量（错误:false）
preg_replace(正则 , 替换 , 被替换字符串)	替换 返回：替换后字符串
preg_split(正则 , 字符串)	用正则分割字符串 返回：数组

第三节 元字符

正则模式当中的元字符，通常都具有特殊的含义

元字符	说明
a-z	英文小写字母
A-Z	英文大写字母
0-9	数字
\r \n \t	非打印字符
\d	数字，相当于 0-9
\D	\d 取反
\w	字母 数字 下划线
\W	\w 取反
\s	空白字符
\S	非空白字符
[ ]	任意匹配[ ]中单个字符
.	匹配任意字符（换行符除外）
{n}	匹配 n 次
{n,}	匹配至少 n 次
{n,m}	至少 n 次，最多 m 次
*	匹配 0 个或多个，相当于{0,}
+	匹配 1 个或多个，相当于{1,}
?	匹配 0 个或 1 个，相当于{0,1}
^	1、匹配正则开头 2、放在[^]，内容取反
$	匹配正则结尾
|	匹配 | 两侧任选其一
( )	1. 分组 2.子存储

注意：

正则需要转义字符： ( ) { } ? * + . [ ] \ / ^ $ |

第四节 模式修正符

对一套正则模式进行调整的符号

符号	说明
i	不区分大小写
s	忽略换行符
U	反转匹配模式（贪婪/非贪婪）

```
$reg='/a(bc){1,3}?/U';
```

第五节 子模式与后向引用

子模式

子模式（子表达式 或 子匹配），在正则表达式中，可以使用 (和) 将模式中的子字符串括起来，以形成一个子模式。

```
//定义正则模式
$str = "110105198012186123"; //身份证号码
$reg = '/\d{6}(\d{4})(\d{2})(\d{2})/';
$n=preg_match_all($reg,$str,$res);
var_dump($n,$res);

//将小括号()内的值，单独作为数组保存起来
/*
array (size=4)
  0 =>
    array (size=1)
      0 => string '11010519800618' (length=14)
  1 =>
    array (size=1)
      0 => string '1980' (length=4)
  2 =>
    array (size=1)
      0 => string '12' (length=2)
  3 =>
    array (size=1)
      0 => string '18' (length=2)
 */
```

后向引用

正则表达式一个最重要的特性就是将匹配成功的模式的某部分进行存储供以后使用这一能力。

- 对一个正则表达式模式或部分模式两边添加圆括号()可以把这部分表达式存储到一个临时缓冲区中。

- 所捕获的每个子匹配都按照在正则表达式模式中从左至右所遇到的内容按顺序存储。
- 存储子匹配的缓冲区编号从 1 开始，连续编号至最大 99 个子表达式。
- 每个缓冲区都可以使用'\n'(或用'$n')访问，其中 n 为 1 至 99 的阿拉伯数字，用来按顺序标识特定缓冲区(子表达式)。

```
$str="<script>alert('web 前端')</script>";

//$ 字符具有特定的含义。代表第 1 到第 99 个子表达式
$str=preg_replace('/<script>(.*?)<\/script>/', "$1", $str);

//结果：$str=alert('web 前端')
```

第六章 日期时间操作函数

第一节 时间戳（timestamp）

在著名的 unix 系统中，使用了一种简洁高效的时间表示方法，即： 将 1970 年 1 月 1 日 0 时 0 分 0 秒作为“unix 纪元”的原点，从 1970 年 1 月 1 日开始经过的秒数存储为一个 32 位整数，这个整数随着时间的变化不断增加，又被称为时间戳（timestamp）。

第二节 常用函数

函数	说明
time()	返回时间戳（timestamp）
mktime(时 , 分 , 秒 , 月 , 日 , 年)	创建一个指定时间戳
date(格式 , [时间戳])	格式化时间戳函数(无时间戳参数,返回当前时间)
date*default*timezone_get()	获取默认时区
date*default*timezone_set()	设置时区 中国时区：“PRC” 英国（格林尼治）：“UTC”
strtotime('yesterday')	将任何英文文本的日期时间描述解析为 Unix 时间戳
checkdate(月 , 日 , 年)	验证日期是否合法, 例: checkdate(2,31,2018) 结果: false

第三节 date()主要参数

- 年

参数	说明	返回值例子
Y	年 4 位数字	1999 或 2003
y	年 2 位数字	99 或 03
L	是否为闰年	闰年: 1，否则: 0

- 月

参数	说明	返回值例子
n	月 数字表示（无前导零）	1 到 12
m	月 数字表示（有前导零）	01 到 12
M	月 三个字母缩写	Jan 到 Dec
F	月 完整的文本格式	January 到 December
t	月总天数	28 到 31

- 日

参数	说明	返回值例子
d	日 （有前导零）	01 到 31
j	日 （无前导零）	1 到 31
z	一年中的第几天	0 到 365

- 星期

参数	说明	返回值例子
l (小写 L)	星期 完整文本格式	Sunday 到 Saturday
D	星期 3 个字母文本	Mon 到 Sun
w	星期 数字表示	0（星期天）到 6（星期六）
W	一年中的第几周	42（当年的第 42 周）

- 时间

参数	说明	返回值例子
a	小写的上午和下午值	am 或 pm
A	大写的上午和下午值	AM 或 PM
g	小时，12 小时格式，无前导零	1 到 12
h	小时，12 小时格式，有前导零	01 到 12
G	小时，24 小时格式，无前导零	0 到 23
H	小时，24 小时格式，有前导零	00 到 23
i	分（有前导零）	00 到 59
s	秒（有前导零）	00 到 59

- 完整的日期时间

参数	说明	返回值例子
c	ISO 8601 格式的日期	2004-02-12T15:19:21+00:00

r	RFC 822 格式的日期	Thu, 21 Dec 2000 16:01:07 +0200

第七章 错误日志处理

第一节 3 种错误类型

语法错误

语法错误最常见，并且最容易修复。这类错误会阻止脚本执行。通常发生在程序开发时，可以通过错误报告进行修复，再重新运行。

```
$name="hxsd"  //NOTICE 错误,结尾没有分号";"
$age=18;
```

运行时错误

只有程序运行到某行，或在某些特定的情形下运行才会发生错误。这种错误一般不会阻止 PHP 脚本的运行，但是会阻止脚本做希望它所做的任何事情。

```
require "abc.php";
```

逻辑错误

这种错误实际上是最麻烦的，不但不会阻止 PHP 脚本的执行，也不会显示出错误消息

```
//在 if 语句中判断两个变量的值是否相等
$a=10;
$b=20;
if($a=$b){  //如果错把比较运行符号“==”写成赋值运行符号“=”就是一种逻辑错误，很难会被发现
        echo "ok";
}
```

第二节 3 种错误级别

错误类型	说明
E_NOTICE	注意： 这些都是比较小而且不严重的错误，比如去访问一个未被定义的变量。通常，这类的错误是不提示给用户的，但有时这些错误会影响到运行的结果。

E_WARNING	警告： 这就是稍微严重一些的错误了，比如想要包含 include 一个不存在的文件。这样的错误信息会提示给用户，但不会导致程序终止运行。
Fatal error	致命错误： 这些就是严重的错误，比如你想要初始化一个根本不存在的类的对象，或调用一个不存在的函数，这些错误会导致程序停止运行，PHP 也会把这些错误展现给用户。

第三节 屏蔽 PHP 错误提示

在配置文件

修改 php.ini 配置文件

设置项	说明
error*reporting* = *E*ALL & ~E_NOTICE	显示所有的错误和提示 忽略 notice 错误
display_errors = Off	Off : 关闭错误提示 On： 显示错误提示

在当前脚本

error_reporting 函数

在 php 脚本顶部，error_reporting(常量)，见附录：php 报错级别

```
error_reporting(E_NOTICE); //只显示 NOTICE 错误
error_reporting(E_WARNING); //只显示 WARNING 错误
```

@错误抑制符

@ 可以屏蔽函数执行过程中遇到问题而产生的一些错误、警告信息（不显示出错信息）。 这样除了用户界面友好一些外，更重要的是安全性，因为屏蔽了出错文件的路径等信息。

```
//赋值一个不存在的变量，会报 NOTICE 错误
@$a=$b;

//在有可能出错的函数前加@, 然后 or die("")
@mysqli_connect(...) or die("数据库连接错误");
```

第四节 错误日志

保存位置：wamp64 /logs/php_error.log

php.ini 配置文件

设置项	说明
log_errors=On	开启错误日志记录
error*log='D:/wamp64/logs/php*error.log'	指定错误日志存储位置

第八章 文件处理系统

第一节 文件类型与路径

文件类型

- 定义：一般指存储在外部介质上具有名字（文件名）的一组相关数据集合
- windows 操作系统中常见文件类型：文件、目录、未知文件

路径

路径	说明
/	绝对路径：硬盘根目录
./	相对路径：当前目录
../	相对路径：上级目录
文件路径相关函数	说明
basename()	获取路径中的文件名部分
dirname()	获取路径中的目录部分
pathinfo($path，参数)	获取路径中的大部分信息，返回数组，参数： PATHINFO*DIRNAME PATHINFO*BASENAME PATHINFO*EXTENSION PATHINFO*FILENAME

第二节 目录操作

目录操作函数

函数	说明
mkdir(完整路径目录)	创建一个目录
rmdir(完整路径目录)	删除一个目录

opendir(完整路径目录)	打开目录
readdir(资源)	读取目录
closedir(资源)	关闭目录,释放内存
is_dir(完整路径目录)	判断是否为一个有效目录
scandir(完整路径目录)	扫描目录，返回文件数组 特例： count（数组）==2 是空目录

遍历目录

```php
//定义目录
$path = "目录";

//打开目录
$res = opendir($path);

//读取目录中的文件（每次调用，读取一个）
$file = readdir($res);

//关闭目录
closedir($res);
```

第三节 文件操作

文件属性函数

函数	说明
filesize(完整路径文件名)	取得一个文件的大小(字节)
filectime(完整路径文件名)	获取文件的创建时间(create)
filemtime(完整路径文件名)	获取文件的修改时间(modify)
fileatime(完整路径文件名)	获取文件的访问时间(access)

文件操作函数

函数	说明

fopen(完整路径文件名 , 参数)	打开文件(参数：r 只读 w 写入 x 创建写入)
fread(资源 , 长度)	读取文件 长度：字节
fwrite(资源 , 内容)	写入的内容
fclose(资源)	关闭文件(资源)，释放内存
file*get*contents(完整路径文件名)	将整个文件读入一个字符串 相当于 fopen fread fclose 组合动作
file*put*contents(完整路径文件名，要写入的数据，[FILE_APPEND])	将字符串写入文件 相当于 fopen fwrite fclose 组合动作 FILE_APPEND：追加写入
readfile()	将内容读入内存缓冲区
copy(源文件 , 目标文件)	拷贝文件(如果目标文件已存在，将会被覆盖)
unlink(完整路径文件名)	删除文件

中文名称

window 中文版操作系统的编码是 GBK，所以，需要将文件名进行转码才能在正常显示，使用 iconv() 函数，进行字符串编码转换：

```php
$path="./中文目录/经典案例";

$path=rtrim($path,"/")."/";

//utf-8 → gbk
$path=iconv("UTF-8","GBK",$path);

//打开目录
$res=opendir($path);//现在是 GBK 编码

//读取文件
while($file=readdir($res)){
   //中文 windows 是 gbk 编码,需要转为 utf-8,才能显示正常
   $file=iconv("GBK","UTF-8",$file);
   var_dump($file);
}

//保存中文名的文件
file_put_contents($path.iconv("utf-8", "GBk","中文名字.txt"), "文件内容");
```

文件判断用函数

函数	说明
is_file(完整路径文件名)	判断是否为一个有效文件
is_readable(完整路径文件名)	判断是否可读
is_writable(完整路径文件名)	判断是否可写
is_executable(完整路径文件名)	判断是否可执行
file_exists(完整路径文件名)	判断一个**文件**或**目录**是否存在

文件/目录重命名

函数	说明
rename(源，目标)	相同文件夹为 改名，不同文件夹为 移动 移动 文件，如果目标文件已存在，将会被覆盖 移动 目录，如果目标目录已存在，将会报错，返回 false

第九章 文件上传和下载

第一节 上传设置

客户端设置

form 的 method=post form 的 enctype="multipart/form-data" input 的 type=file

```
<form action="upload.php" method="post" enctype = "multipart/form-data">
        <input type="file" name="fname">
   <!--MAX_FILE_SIZE 有兼容问题-->
   <input type="hidden" name="MAX_FILE_SIZE" value="30000" />
        <button type="submit">提交</button>
</form>
```

服务端设置

在 php.ini 中进行设置：

配置项	功能
file_uploads = On	是否允许上传文件
upload*tmp*dir ="G:/wamp/tmp"	临时存储上传文件的目录
upload*max*filesize = 2M	上传单个文件的大小
max*file*uploads = 20	单次上传文件的最大数量
post*max*size = 8M	表单 post 方式上传文件（总和）大小
max*execution*time = 120	表单上传的超时的时间设置(默认 120 秒)

默认 upload*max*filesize = 2M，如果你想上传超过 8M 的文件，比如：20M，你在设定 upload*max*filesize = 20M 的同时，还要修改 post*max*size 选项（默认为 8M）。否则 post 数据超出限制，$_FILES 将会为空（报错）。

$_FILES 数组

$_FILES 超全局数组中，包含着从客户端提交文件的全部信息：

```
array (size=1)
  'fname' =>    //上传文件表单项的 name
    array (size=5)
      'name' => string 'home.html' (length=21)  //上传文件的名称
      'type' => string 'text/html' (length=9)   //文件类型(常见文件都可以上传)
```

```
'tmp_name' => string 'C:\wamp\tmp\php57A2.tmp' (length=23)   //上传文件的临时文件名
'error' => int 0   //上传文件遇到的错误号
'size' => int 28131   //上传文件的大小
```

error 编号	说明
0	上传过程中没有发生任何错误
1	上传的文件超过了 php.ini 中 upload*max*filesize 选项限制的值
2	超出了 HTML 表单隐藏域属性的 MAX__FILE__SIZE 元素所指定的最大值（浏览器兼容）
3	文件只有部分上传
4	没有上传任何文件
5	上传文件大小为 0(已废弃)
6	找不到临时文件夹
7	文件写入失败
文件上传函数	说明
move*uploaded*file(临时文件名 , 新文件名)	将临时文件保存为正式文件

上传步骤

```php
var_dump($_FILES);

$file=$_FILES["fname"];

//获取$_FILES 文件  临时文件
$tmp_file=$file["tmp_name"];

//获取后缀名
$ext=pathinfo($file["name"],PATHINFO_EXTENSION);//后缀名

//创建新文件名
$new_file="file".rand(1,1000000).".".$ext;

//保存路径
$path="./files";

//格式化路径
$path=rtrim($path,'/')."/";
```

```php
//拼接上传目录
$new_file=$path.$new_file;

//临时文件转换成正式文件
move_uploaded_file($tmp_file,$new_file);
```

第二节 下载设置

浏览器对于可识别的文件格式(html、txt、png、jpg 等)，默认是直接打开的，对于无法识别的文件，才作为附件来下载。为了可以让可识别的文件也直接可以下载，需要进行如下设置：

```php
<a title="点击下载" href="down.php?name=01.jpg&type=image/jpeg"><img src="01.jpg"></a>

// 1. 读取文件
$flienname=$_GET["name"];
$filetype=$_GET["type"];

//2. 文件的描述信息 Content-Disposition（内容配置）指定为 attachment(附件)(必须)
header("Content-Disposition:attachment;filename={$filename}");

//3. 指定被下载文件的类型(非必须)
header("Content-Type:{$filetype}");

//4. 指定被下载文件大小（非必须）
header("Content-Length:".filesize($filename));

//5. 将内容读入内存缓冲区
readfile($filename);
```

注意：在 readfile($filename) 之前，不能有任何输出语句（错误信息、var_dump 调试语句，echo 输出等）,否则下载的文件会出错。

第十章 PHP 图形图像处理---GD 库

PHP 中的绘图工具

作用：绘制饼图、柱形图、折线图、验证码、水印、缩放等

支持：GIF、JPEG、PNG、WBMP 等格式

第一节 绘画步骤

```php
//示例
//1.创建画布
$img = imagecreatetruecolor(200,200);

//2.准备颜料
$bg = imagecolorallocate($img,200,200,200);
$red = imagecolorallocate($img,255,0,0);
$blue = imagecolorallocate($img,0,0,255);

//3.绘画...
imagefill($img,0,0,$bg);            //填充背景
imageellipse($img, 200, 200, 100, 100, $red); //画空心圆
imagesetpixel($img, 100, 100，$blue);//绘制像素点

//绘画其他.....

//4.输出图像
header('Content-Type:image/jpeg');       //设置 http 协议头信息
imagejpeg($img);         //输出图像流

//4.释放资源
imagedestroy($img);
```

第二节 常用函数

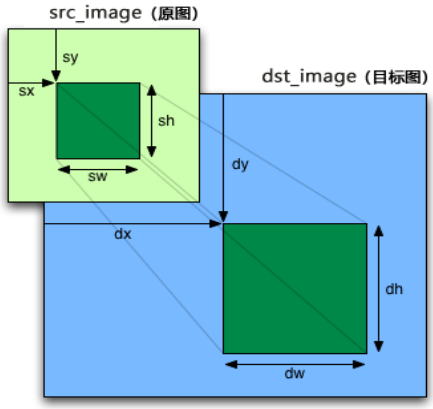
GD 库图片函数的参数很多，具体使用方法**参照 php 手册**

函数	说明
imagecreate(宽,高)	创建一个 256 色画布

imagecreatetruecolor(宽,高)	创建一个真彩画布
imagecreatefromgif(完整路径文件名)	将一张 gif 图像放入画布
imagecreatefrompng(完整路径文件名)	将一张 png 图像放入画布
imagecreatefromjpeg(完整路径文件名)	将一张 jpeg 图像放入画布
imagecreatefromstring(字符串)	从字符串中的图像流新建一图像
imagecolorallocate()	定义颜色
imagecolorallocatealpha()	定义带 alpha 通道（透明）的颜色
imagefill()	填充背景
imagesetpixel()	绘制像素点
imageline()	绘制线条
imagerectangle()、imagefilledrectangle()	矩形线框、实心矩形框
imagepolygon()、imagefilledpolygon()	多边形线框、实心多边形
imageellipse()、imagefilledellipse()	椭圆线框、实心椭圆
imagearc()、imagefilledarc()	圆弧、实心圆弧（扇形）
imagettftext()	绘制文本需引入字库(支持中文字库，输出中文)
getimagesize(完整路径文件名)	获取图像的宽、高、大小、类型等相关信息 成功返回：数组 失败返回：false
imagesx(资源)	取得图像宽度
imagesy(资源)	取得图像高度
imagecopy()	用于拷贝图像或图像的一部分 成功返回：true ， 失败返回：false
imagecopyresampled()	用于拷贝部分图像并调整大小（重新采样）
imagegif()	输出 gif 格式图像
imagejpeg()	输出 jpeg 格式图像
imagepng()	输出 png 格式图像

imagedestroy()	释放资源

imagecopyresample（dst_image,**src_image** , dx,**dy** , sx,**sy** , dw,**dh** , sw,**sh**）



第三节 绘制基本几何图形

最基本的集合图形主要有点、线、圆、多边形等，有了基本几何图形，就可以演变出更为复杂的图形。GD 库也提供了三个绘制基本图形的函数， imageline()函数、imagearc()函数、imagerectangle()函数分别用来绘制线段，弧形（包括圆弧），矩形。

- imageline()函数用来绘制线段

  bool imageline(resource image, int x1, int y1, int x2, int y2, int color)

  该函数参数表示使用 color 颜色在图像 image 上从坐标(x1,y1)到(x2,y2)画一条线段。在画布里，坐标原点为左上角，横坐标水平向右为正，纵坐标垂直向下为正。

- imagearc()函数用于绘制椭圆弧（包括圆弧）

  bool imagearc(resource image, int cx, int cy, int w, int h, int s, int e, int color)

  该函数参数表示用 color 颜色在图像 image 上以(cx,cy)为中心，画一个椭圆弧，w 和 h 分别是这个椭圆的宽和高，s 和 e 都是角度，分别表示圆弧起点角度和圆弧终点角度，0 度位置在中心的三点钟位置（x 轴正轴方向）。

- imagerectangle()函数用来绘制一个矩形

  bool imagerectangel(resource image, int x1, int y1, int x2, int y2, int color)

该函数参数表示用 color 颜色在图像 image 上画一个矩形，左上角坐标为(x1,y1)，右下角坐标(x2,y2)。

第四节 图像填充

- imagefill()函数用于图像区域填充

  bool imagefill(resource image, int x, int y, int color)

  该函数表示用 color 在 image 图像的坐标(x,y)位置执行区域填充，与(x,y)点颜色相同且相邻的点都会被填充。

- imagefilledarc()函数用于画一椭圆弧并填充

  bool imagefilledarc(resource image, int cx, int cy, int w, int h, int s, int e, int color, int style)

  它的参数说明同 imagearc()函数，与之相比，多了参数 style，它用来表示填充的方式，可选择有如下几种：

  IMG*ARC*PIE：普通填充，产生圆形边界

  IMG*ARC*CHORD：只是用直线连接了起始和结束点与 IMG*ARC*PIE 互斥

  IMG*ARC*NOFILL：指明弧或弦只有轮廓，不填充

  IMG*ARC*EDGE：指明用直线将起始和结束点与中心点相连

- imagefilledellipse()函数用于画一个椭圆并填充

  bool imagefilledellipse(resource image, int cx, int cy, int width, int height, int color)

  该函数表示用 color 颜色在 image 图像上画一个填充的椭圆形，这个椭圆形的中心点在(cx,xy)，宽为 windth，高为 height。

- imagefilledrectangle()函数用于画一个矩形并填充

  bool imagefilledrectangle(resource image, int x1, int y1, int x2, int y2, int color)

参数意义和 imagerectangle()函数一致，与之不同的是 imagefilledrectangle()函数会填充颜色。

- imagefilledpolygon()函数用来画一个多边形并填充

bool imagefilledpolygon(resource image, array points, int num*points, int color 该函数表示用 color 颜色在 image 图像上画一个填充的多边形，多边形的顶点按照顺序存储在 points 数组中，num*points 表示多变形的边数，num_points 的数值必须大于等于 3，且小于等于 points 数组的值个数的一半。

第五节 在图像中添加文字

前面在加载图像章节在加载失败则以图像方式输出错误信息的时候已经见过这个函数：imagestring()。imagestring()的主要功能就是在图像中添加函数，它的语法格式如下：

bool imagestring(resource image, int font, int x, int y, string string, int color)

该函数表示用 color 颜色在 image 图像上的(x,y)位置，使用 font 字体绘制字符串 string。绘制字符串的时候，是从(x,y)右下方开始绘制。通常情况下 font 使用 1，2，3，4，5 等内置字体。

第六节 拷贝图像

- getimagesize()函数用于获取图像的尺寸：

  array getimagesize(string filename)

- imagecopy()函数用于拷贝图像或图像的一部分：

  bool imagecopy(resource dst*im, resource src*im, int dst*x, int dst*y, int src*x, int src*y, int src*w, int src*h)

- imagecopyresized()函数用于拷贝图像或图像的一部分并调整大小，它的语法结构如下：

  bool imagecopyresized(resource dst*im, resource src*im, int dst*x, int dst*y, int src*x, int src*y, int dst*w, int dst*h, int src*w, int src*h)

- imagecopymerge()用于拷贝并合并成图像的一部分：

  bool imagecopymerge(resource dst*im, resource src*im, int dst*x, int dst*y, int src*x, int src*y, int src*w, int src*h, int pct)

第十一章 PHP 与 WEB 页面交互

第一节 PHP 与 Web 页面交互

PHP 与 Web 页面交互是实现 PHP 网站与用户交互的重要手段。

在 PHP 中提供了两种与 Web 页面交互的方法，一种是通过 Web 表单提交数据，另一种是直接通过 URL 参数传递数据。

Web 表单提交数据有两种方式：GET 方法和 POST 方法。

POST 方法不依赖于 URL，不会将传递的参数值显示在地址栏中，而是将参数值放置在是 HTTP 包的包体中，这样可以传输更多的内容，传输方法也更加安全，所以 POST 方法通常用于上传信息。

GET 方法完全依赖于 URL，参数值会附在 URL 之后，以?分割 URL 和传输数据，多个参数用&连接，这样传输安全性很低，而且受到 URL 长度的限制，传输内容很小，所以 GET 方法通常用于获取信息。

Web 表单用 get 方法提交数据，最终效果如同直接通过 URL 参数传递数据。

PHP 针对这两种请求方法，提供了两个全局变量*_POST*[]*和*_GET[ ]，分别用来获取 POST 请求和 GET 请求的参数值。

$_GET[ ]

建立一个 get 请求的表单页面名叫 form_get.html：

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>GET 方式的表单</title>
  </head>
  <body>
    <form action="form_get.php" method="get">
      账户：<input name="username" type="text" /><br />
      密码：<input name="password" type="password" /><br />
      邮箱：<input name="email" type="email" /><br />
      <input type="submit" />
    </form>
  </body>
</html>
```

```php
<?php
   echo "<pre>";
   var_dump ( $_GET );
   $username = $_GET ["username"];
   $password = $_GET ["password"];
   $email = $_GET ["email"];
   echo "<br />接收到的账户： " . $username;
   echo "<br />接收到的密码： " . $password;
   echo "<br />接收到的邮箱： " . $email;
?>
```

$_POST[ ]

同 form*get.html，我们建立一个名叫 form*post.html，内容仅仅把<form>元素的 action 和 method 修改一下：

```html
<!DOCTYPE html>
<html>
   <head>
      <meta charset="UTF-8">
      <title>POST 方式的表单</title>
   </head>
   <body>
      <form action="form_post.php" method="post">
         账户：<input name="username" type="text" /><br />
         密码：<input name="password" type="password" /><br />
         邮箱：<input name="email" type="email" /><br />
         <input type="submit" />
      </form>
   </body>
</html>
```

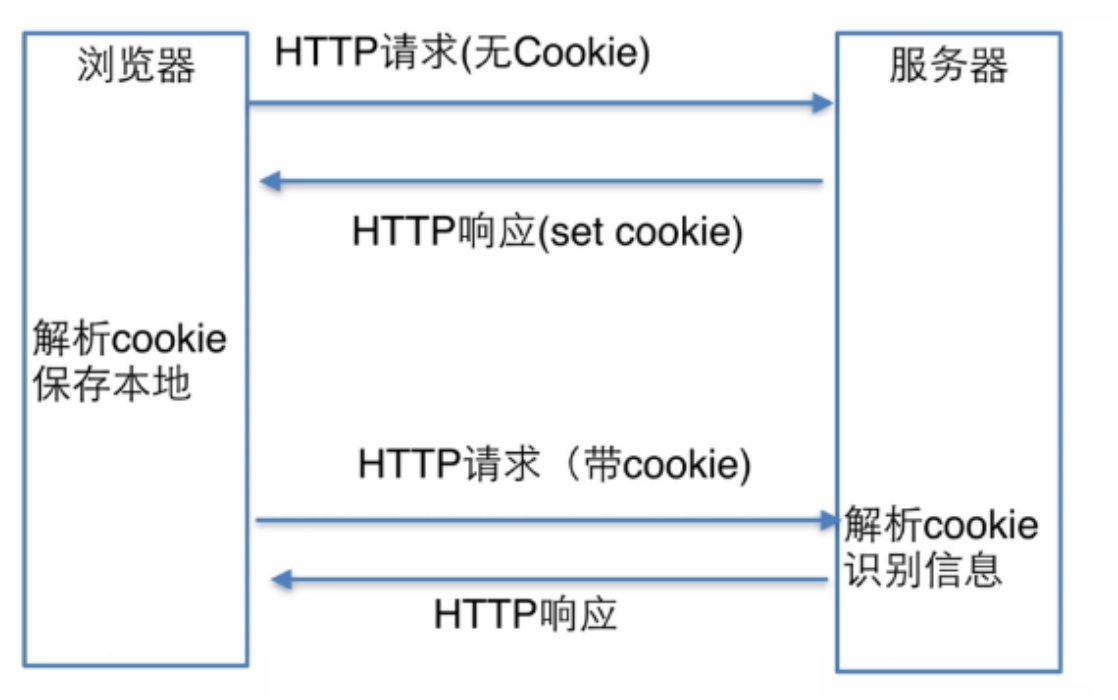
运行界面除了地址栏和标题栏，其他内容完全和 formget.html 一致。这时候点击跳转到同目录下的 formpost.php 文件，文件内容为：

```php
<?php
echo "<pre>";
var_dump ( $_POST );
$username = $_POST ["username"];
$password = $_POST ["password"];
$email = $_POST ["email"];
echo "<br />接收到的账户： " . $username;
echo "<br />接收到的密码： " . $password;
echo "<br />接收到的邮箱： " . $email;
?>
```

第二节 会话控制

会话控制是一种跟踪用户的通信方式

例如：当一个用户在请求一个页面后，再次请求这个页面，网站是无法知道这个用户刚才是否曾经来访问过。由此我们就会觉得奇怪，平时我们在电商网站购物时，只要我们在这个站点内，不论我们怎么跳转页面，网站总会记得我是谁，这是怎么做到的呢？这就是运用了 HTTP 会话控制。在网站中跟踪一个变量，通过对变量的跟踪，使多个请求事物之间建立联系，根据授权和用户身份显示不同的内容、不同页面。



cookie

cookie 是在服务器端创建，并写回到客户端浏览器

浏览器接到指令则在本地临时文件夹中创建了一个 cookie 文件，其中保存了你的 cookie 内容

客户端浏览器每次访问网站时，都会检测是否有该网站的 cookie 信息，如果有的话，也会同时发送过去。

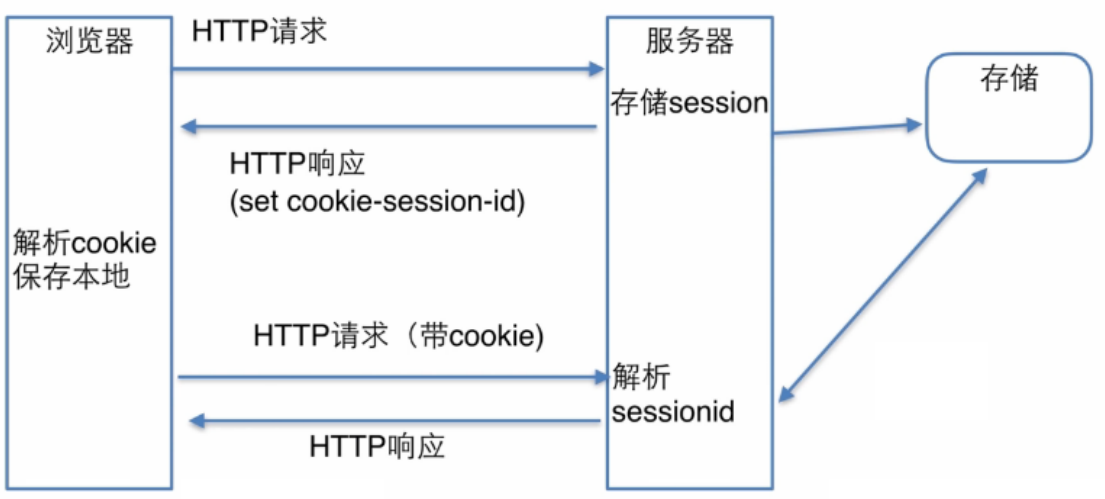
注意：

cookie 内容的存储是**键/值**对的方式，**键和值都只能是字符串**。

函数	功能
setcookie(key , value ,有效期)	设置会话 cookie 参数

```
//定义 cookie
//setcookie(键, 值, 有效期(秒))
setcookie("name","zhangsan",time()+1000); //如果不设置有效期，关闭浏览器就会消失
setcookie("pwd","123",time()+1000);

//删除 cookie（设定过期时间，使失效）
setcookie("name","",time()-1);
setcookie('age',null,time()-1);
setcookie('sex',"",time()-1);
```

session



session 与 cookie 相似，只是原来将信息存在客户端，现在保存到服务端

客户端第一次访问时将信息保存到服务器，同时分配给用户一个固定长度的字符串（sessionID），并以 cookie 方式保存在客户端

在 php.ini 配置文件中，可以指定这个 sessionID 的名称：

session.name = PHPSESSID

函数	说明
session_start()	开启 session
session_destroy()	销毁服务器端 session 文件

```php
//开启 session 要保证在它之前，没有向浏览器输出过任何内容,通常放在代码第一行
session_start();

//往 session 中存储信息
$_SESSION['name'] = '张三';
$_SESSION['sex'] = '男';
$_SESSION['age'] = 18;

//获取 session 信息
$name=$_SESSION['name'];

//销毁 session 中的信息
unset($_SESSION);

//销毁 session 文件
session_destroy();

//销毁客户端 cookie 信息
setcookie('PHPSESSID','',time()-1);
```

cookie 与 session 的区别

区别	cookie	session
存放位置	客户端	服务器端
安全性	不够安全	安全
资源占用	存放在客户端，不占服务器资源	占服务器资源
生命周期	固定时长	每次访问，重新计算时长
文件大小	4k	不限制

建议将登陆信息等重要信息存放为 session，其他信息如果需要保留，可以放在 cookie 中

第十二章 面向对象的编程

第一节 类/对象

世间万物皆对象，你自己本身也是一个对象；面向对象的核心就是，让对象帮我们实现功能

- 对象：具有一定功能和特征的单个事物，就是对象
- 类：具有相同功能和特征的对象的抽象，就是类
- 类与对象的关系：对象是类的实例化，类是对象的抽象化

定义类与对象实例化

```php
//使用 class 关键字定义一个类 类名的首字母要求大写
class School{
  //成员属性
  //成员方法
}

//使用 new 运算符，实例化一个类的对象
$hxsd = new School();

//使用 instanceof 判断$hxsd 是否是 School 的实例
$bool=$hxsd instanceof School;
var_dump($bool);
```

修饰符

访问修饰符（3p）	修饰对象	类外	子类	本类
public	属性 方法	√	√	√
protected	属性 方法	×	√	√
private	属性 方法	×	×	√
额外修饰符	修饰对象	说明		

static	属性 方法	静态：可以不 new 实例，直接通过 类::方法 类::属性 的方式调用。 静态方法不能调用非静态属性（需先实例化对象 ）
final	类 方法	最终：修饰的类不能被继承，修饰的方法不能被重写
abstract	类 方法	抽象：定义抽象类

类的成员

成员属性

- 必须使用 3P(public protected private)修饰符进行修饰
- 可以用 static 修饰为静态，静态属性不会随对象实例销毁
- 可以没有初始值
- 初始值不能为资源型和对象型
- 初始值不能为变量或函数调用
- 成员属性不能重复定义

成员方法

- 必须使用 3P 修饰符进行修饰
- 可以没有形参、返回值与程序体

类常量

- 在类中始终保持不变的值定义为常量，类中方法不可更改它的值
- 类常量可以使用类名或类的实例，用 :: 来访问
- 类的常量属于类本身，不能被子类继承 或 覆盖。
- 不能使用表示变量的符号 $，全部大写

```
//使用 class 关键字定义一个类 类名的首字母要求大写

class School{
        //成员属性----------------------
        public $students=100; //公共 学生人数
        public $anme; //公共 学校名称 未命名
        protected $tech=100; //受保护的 技术资料
   private $money=100000; //私有 资金
```

```
        //类常量-----------------------
        const ADDRESS="清华东路 100 号";

        //成员方法-----------------------
        //构造方法
        function __construct($shool_name){
                $this->students=1500;
                $this->name=$shool_name;
        }
  //自定义方法
        public function introduce(){
                echo "学校简介：名称：".$this->name.",本校有".$this->students."名学生，
本校有教师".$this->staff."人，地址：".self::ADDRESS."<br>";
        }
}

//-----------------------------------------------------------------------
//使用 new 运算符，实例化一个类的对象
$shool = new School("清华大学","1500");

//使用 var_dump 可以直接查看对象属性
var_dump（$shool）;

//使用 get_class_methods（）,可以查看类中定义的方法
var_dump（get_class_methods($shool)）;

//修改对象属性
$shool->students=2000;
$shool->staff=50;

//类常量可以直接访问
var_dump(School::ADDRESS);
var_dump($shool::ADDRESS);
```

注意：

类中只能包含 **成员属性、成员方法、类常量**

类中成员属性和成员方法的位置可以互换，建议成员属性在前，成员方法在后

魔术方法

PHP 中把以两个下划线 __ 开头的函数（方法）称为魔术方法（Magic methods），这些函数不需要显示的调用，而是由某种特定的条件触发。

魔术方法	功能
__construct()	类的构造函数，当一个类被实例化的时候自动调用
__destruct()	类的析构函数，脚本执行结束的时候、销毁对象的时候、重新定义对象的时候
__get()	读取不可访问或不存在属性时被调用
__set()	当给不可访问或不存在属性赋值时被调用
__isset()	当对不可访问属性调用 isset()或 empty()时调用
__unset()	当对不可访问属性调用 unset()时被调用
__sleep()	当使用 serialize 时被调用（不需要保存大对象的所有数据时很有用)
__wakeup()	当使用 unserialize 时被调用，可用于做些对象的初始化操作
__toString()	当一个类被转换成字符串时被调用
__clone()	进行对象 clone 时被调用，用来调整对象的克隆行为
__invoke()	当以函数方式调用对象时被调用
_*set*state()	调用 var_export()导出类时，此静态方法会被调用
__call()	调用非 public 或不存在的方法时被调用
__callStatic()	调用不可访问或不存在的静态方法时被调用
__debugInfo()	打印所需调试信息

指针

指针	指向	说明
$this	实例化对象	只能应用于成员方法中 可以调用本类 或从 父类中继承来的属性和方法 避免用$this 引用静态成员属性和类常量（使用 self）
self	类本身	可以访问 本类 或 父类 中的静态属性和静态方法
parent	父类	访问 父类 中的静态属性和方法

static（静态）

- 静态属性只能是标量类型，不能使用表达式、另一个变量、函数返回值 或 指向一个对象
- 静态属性/方法，可以不实例化对象而直接访问（类::属性 或 类::方法）
- 静态属性不能通过一个类的实例化对象来访问

- 静态方法依然可以通过一个类的实例化对象来访问
- 静态属性/方法默认为公有（public）。
- 静态方法使用 self::方法 self::属性 的方式访问其他静态属性/方法（不能访问非静态属性/方法，否则会报错）。

单例模式

单例模式确保某个类只有一个实例

例如：PHP 一个主要应用场合就是应用程序与数据库打交道的场景，在一个应用中会存在大量的数据库操作，针对数据库句柄连接数据库的行为，使用单例模式可以避免大量的 new 操作。因为每一次 new 操作都会消耗系统和内存的资源。

```
class Cls{
    //3.定义静态成员属性用于存储实例化对象（静态属性赋值后不会消失）
    static $obj = null;

    //1.构造方法私有化
    private function __construct(){}

    //2.定义静态成员方法
    static function getObj(){
        if(self::$obj==null){
            self::$obj = new Cls();
            return self::$obj;
        }else{
            die('已经实例化了，不能重复实例化！');
        }
    }
    //方法
    public function demo(){
        echo '我是单例模式中的一个方法';
    }
}

//调用静态成员方法
$obj = Cls::getObj();
var_dump($obj);

//调用方法
$obj->demo();
```

第二节 封装

为了最大限度的保护类内部信息的细节而准备，使用 **3P 修饰符** 和 **gusic 魔术方法** 来实现类的封装

gusic 魔术方法	说明
__get(属性名)	当在类外部**访问**非公有成员属性时，自动调用， 并且，参数传入要获取的成员属性的名称，返回获取的属性值
__unset(属性名)	当在类外部**销毁**非公有成员属性时，自动调用， 并且，将正在销毁的成员属性名传递进来
__set(属性名，属性值)	当在类外部**设置**非公有成员属性时，自动调用， 并且，将正在添加的成员属性的名字和值同时传进来
__isset(属性名)	当在类外部**判断**非公有成员属性是否存在时，自动调用， 并且，将正在判断的属性名传进来
__call(方法名和参数列表)	当在类外部**访问**非公有成员**方法**时，自动调用， 并且，将正在访问的方法名和参数列表传进来

gusic 魔术方法在实际开发中，并不经常使用，仅作为知识点掌握即可。

第三节 继承（extends）

自己没有，别人有，可以拿来，就是继承

```
//使用 extends 关键字 定义继承类
class B extends A{...}  //B 继承 A
```

单继承

```
//php 不支持多继承
class A{...}
class B{...}
class C{...}
class D extends A,B,C{....} //不能同时继承多个类

//若想继承多个类，可以采用如下方法：
class B extends A{...}  //B 继承 A
class C extends B{...}  //C 继承 B，同时继承了 A
class D extends C{...}  //D 继承 C，同时继承了 B 和 A
```

方法重写（重载）

- 子类包含与父类重名的方法，则子类的方法重写（覆盖）父类方法
- 子类重写父类方法，访问权限一定要高于或等于父类方法的访问权限：public > protected > private
- 子类重写父类方法，比父类方法多出的参数，必须指定默认值
- static（静态）方法必须还是 static
-

第四节 多态

按字面的意思就是“多种状态”。在面向对象语言中，多态就是把子类对象赋值给父类引用，然后调用父类的方法，去执行子类覆盖父类的那个方法。

```
class Man{
   //做工作
   public function work(){
          echo "这个方法需要在子类具体实现";
          }
   //从事何种工作
   public function doWork($shenfen_obj){//通过传入不同“身份”的对象，实现不同形态
          $shenfen_obj->work();
          }
}

//定义油漆工类 继承 Man 类
class Painter extends Man{
    public function work(){
        echo "油漆工正在刷漆！ ";
          }
}

//定义教师类 继承 Man 类
class Teacher extends Man{
          public function work(){
          echo "老师正在上课！ ";
          }
}

//实例化一个人
$man=new Man();
```

```php
//现在是油漆工
$man->doWork(new Painter());

//转换了身份，现在是老师啦
$man->doWork(new Teacher());
```

第五节 __autoload()

__autoload()方法并不是一个魔术方法，但是这个方法非常有用，当实例化或继承一个不存在的类，会自动调用__autoload()，同时将该类的类名作为参数

```php
//require "Model.class.php";

//当前页面需要调用 class A
function __autoload($arg){
   var_dump($arg); //Model 类名

        //引入 model.class.php
        require "{$arg}.class.php";
}

//实例化对象
$shool=new Model();
```

第六节 抽象类和接口

抽象类（abstract）

- 任何一个类，如果它里面至少有一个方法是抽象的（该方法用 abstract 修饰，没有函数体），那么这个类就必须被声明为抽象的。
- 被定义为抽象的方法只是声明了其调用方式（参数），不能定义其具体的功能实现。
- 定义为抽象的类不能被实例化。

- 继承一个抽象类的时候，子类必须定义父类中的所有抽象方法；这些方法的访问权限一定要高于或等于父类方法的访问权限：public > protected > private。此外，方法的调用方式必须匹配，即类型和所需参数的数量必须一致。

```
//定义抽象类
abstract class Dev_teamA {
        public $lesson_A="python";
        public $lesson_B="bigData";

        //实现 dev_python 方法
        public function dev_python(){
                echo $this->lesson_A."课程开发完成！";
        }
        //未实现的方法 使用 abstract 定义为抽象方法
        abstract function dev_bigData();
}

//子类必须定义父类中的所有抽象方法
class Dev_teamB extends Dev_teamA{

  //子类必须定义父类中的所有抽象方法
        public function dev_bigData(){
                echo $this->lesson_B."课程开发完成！";
        }

  //子类可以重新父类方法
        public function dev_python(){
                echo "重新完善".$this->lesson_A."课程";
        }
}

//实例化对象
$dev=new Dev_teamB();

//调用方法
$dev->dev_python();
$dev->dev_bigData();
```

接口（interface）

一个抽象类里面的所有方法都是抽象方法，并且都是 public 权限，那么这种特殊的抽象类就叫接口（interface）。

- 接口是通过 **interface** 关键字来定义的，就像定义一个标准的类一样，但其中定义所有的方法都是空的，不需要 abstract 修饰。
- 使用接口可以指定某个类必须实现哪些方法，但不需要定义这些方法的具体内容。
- 接口中定义的所有方法都必须是 public 的。
- 接口中可以定义常量，不可以定义成员属性。接口常量和类常量的使用完全相同。
- 要实现一个接口，使用 **implements** 操作符。类中必须实现接口中定义的所有方法，否则会报一个致命错误。
- 类可以实现多个接口，用逗号来分隔多个接口的名称。

```php
//定义 Ai 课程 interface
interface Ai{
   const LESSON_A="python";
   const LESSON_B="bigData";

   public function dev_python(); //研发 python 课程
   public function dev_bigData(); //研发 BigData 课程
}

//定义 Ui 课程 interface
interface Ui{
   //不要与其他接口内的常量重名
   const LESSON_C="moviePoster"; //电影海报
   public function dev_moviePoster(); //研发电影海报课程
}

/*dev_teamA 类使用 implements 关键字，可以同时实现 Ai，Ui 接口
*dev_teamA 只实现了 Ai 接口的 dev_python 方法，没有实现 dev_bigData
*因此，dev_teamA 类还是个抽象类，必须用 abstract 修饰
*/
abstract class dev_teamA implements Ai,Ui{
   //实现 dev_python 方法
   public function dev_python(){
      echo self::LESSON_A."课程开发完成！！";
   }
         //实现 dev_moviePoster()
   public function dev_moviePoster(){//研发电影海报课程
      echo self::LESSON_C."课程开发完成！！";
   }
```

```
    //未实现的方法 继续使用 abstract 定义为抽象方法
    abstract function dev_bigData();
}

//dev_teamB 类 extends 了 dev_teamA 类，继续进行课程研发
class dev_teamB extends dev_teamA{

    public function dev_bigData(){
        echo self::LESSON_B."课程开发完成！ ";
    }

    public function dev_python(){
        echo "重新完善".self::LESSON_A."课程";
    }
}
//-------------------------------------------------------------
//实例化对象
$dev=new dev_teamB();

//调用方法
$dev->dev_python();
echo "<br>";
$dev->dev_bigData();
echo "<br>";
$dev->dev_moviePoster();
```

抽象类与接口的差别

抽象类是对根源的抽象，比如，男人，女人这两个类，可以为这两个类设计一个更高级的抽象类---人。

接口是对动作的抽象，比如，定义一个吃饭接口，开车接口，踢球接口等等

一个类只能继承一个抽象类（因为，你不可能同时是“人”类 又是“猫”类），但是一个类可以同时实现多个接口。

第十三章 php 操作 MySQL

第一节 MySQL

MySQL 是一个关系型数据库数据系统， PHP+MySQL 是最流行使用方法，当然，PHP 也支持其他的数据库，如 Mango，MangoDB，Microsoft SQL Server，Oracle OCI8，Sybase，DB++，PostgreSQL，Access，SQLite 等。

第二节 配置 MySQL

在 PHP 中想要对 MySQL 数据库进行操作，需要使用 mysqli 扩展，这时候需要修改 PHP 的配置文件，打开 php.ini，找到“;extension=mysqli”，将前面的分号去掉。修改好以后保存，重启 Apache 服务。

```
<?php
   $mysqli = new mysqli("localhost", "root", "123456");
   if ($mysqli) {
      echo "env mysql success";
   } else {
      echo "env mysql error";
   }
   $mysqli->close();
?>
```

第三节 访问数据库

连接 MySQL 服务器

建立与 MySQL 数据库的连接，使用 mysqli_connect()函数，它的语法格式如下：

```
mysqli mysqli_connect ([string server[, string username[, string password[, string dbname[, int
 port[, string socket]]]]]])
```

它的参数含义如下：

- server：可选参数，MySQL 服务器地址。
- username：可选参数，MySQL 服务器用户名。
- password：可选参数，MySQL 服务器用户密码。

- dbname：可选参数，要连接的数据库名字。
- port：可选参数，MySQL 服务器的端口号，默认 3306。
- socket：可选参数，使用设置的 socket 或者 pipe。

前四个参数比较常用，后两个很少用到。

选择 MySQL 数据库

PHP 提供了 mysqli*elect*db()函数来选择 MySQL 数据库，它的语法格式如下：

```
<?php
   $server = "localhost";
   $username = "root";
   $password = "123456";
   $dbname = "php_db";

   // 快速写法
   $mysqli = new mysqli ( $server, $username, $password, $dbname );

   // 兼容写法
   $mysqli = new mysqli ( $server, $username, $password );
   mysqli_select_db ( $mysqli, $dbname );

   // 对象写法
   $mysqli = new mysqli ();
   $mysqli->connect ( $server, $username, $password );
   $mysqli->select_db ( $mysqli, $dbname );
?>
```

断开 MySQL 服务器

使用 mysqli_close()函数来关闭与 MySQL 服务器的连接。

执行 SQL 语句

数据库的“增删改查”说到底都是一句 SQL 语句，PHP 提供了 mysqli_query()函数来执行 SQL 语句，它的语法如下：

```
mixed mysqli_query (mysqli link, string query[, int resultmode])
```

link : 是调用 mysqli_connect()函数返回的 mysqli 对象，

query : 是要执行的 SQL 语句，

resultmode : 是可选参数，它的默认值是 MYSQLI*STORE*RESULT，如果需要查询的数据量很大，需要使用 MYSQLI*USE*RESULT。

```php
<?php
   $server = "localhost";
   $username = "root";
   $password = "123456";
   $dbname = "account_db";
   $mysqli = new mysqli ( $server, $username, $password, $dbname );

   // 增
   $result = mysqli_query ( $mysqli, "insert into account(username, password, email) values('张三', '123456', 'zhangsan@example.com')" );
   // 删
   $result = mysqli_query ( $mysqli, "delete from account where user='张三'" );
   // 改
   $result = mysqli_query ( $mysqli, "update account set password='abcdefg' where user='张三'" );
   // 查
   $result = mysqli_query ( $mysqli, "select * from account " );
   mysqli_close ( $mysqli );
?>
```

第四节 解析结果集

面向对象	面向过程	说明
free() close() free_result()	mysqli*free*result()	释放与结果集占用的内存
fetch_row()	mysqli*fetch*row()	以索引数组方式返回一行数据
fetch_assoc()	mysqli*fetch*assoc()	以关联数组方式返回一行数据
fetch_array()	mysqli*fetch*array()	以数组的方式返回一行数据
fetch_object()	mysqli*fetch*object()	以对象的方式返回一行数据
data_seek()	mysqli*data*seek()	移动结果集中的指针到任意行
num_rows	mysqli*num*rows()	获取结果集中行的数量

第五节 PDO（PHP Data Object）

PDO 是 PHP 数据对象的英文缩写，英文全称为 PHP Data Object，是又 MySQL 官方封装的、基于面向对象编程思想的、使用 C 语言开发的数据库抽象层。

第六节 配置 PDO

Windows 下启动 PDO 需要在“php.ini”文件中进行配置，添加扩展：

extension=php_pdo.dll

在最新版 PHP 中，PDO 已经默认开启，只需要启动其他数据库扩展即可。配置好这些后重启 Apache 服务。执行 phpinfo()函数，看到 PDO 配置项，说明开启成功。

第七节 访问数据库

与 mysqli 扩展类似，PDO 扩展也是实例一个 PDO 对象，然后可以调用相关方法和属性来执行数据库的操作。

连接服务器

使用 PDO 与服务器建立连接，需要先使用构造方法来创建 PDO 实例，PDO 的构造方法如下：

```
_construct(string data_source_name [,string user[,string pwd[,array driver_options]]])
```

data*source*name : 数据源，该参数包括了数据库名，主机名。MySQL 数据库的 DSN 为：“mysql:host=localhost;dbname=account_db; port=3306”

user：数据库服务器用户名

pwd：为数据库服务器密码

数据库连接成功后，将返回一个 PDO 的实例，连接失败将会抛出一个 PDOException 异常，通常会使用 try/catch 语句进行处理。

关闭数据库

要想关闭连接，需要销毁对象以确保所有到它的引用都被删除，可以给变量赋一个 NULL。

第八节 执行 SQL 语句

PDO 提供了三种执行 SQL 语句的方法，分别是 exec()，query()，预处理语句。

exec()

exec()方法可以执行一条语句，并返回受影响的行数，它的语法格式如下：

int PDO::exec(String sql)

exec()方法通常用于 insert into，delete，update 等语句。

```
<?php
   echo "<pre>";
   $dbms = "mysql";
   $server = "localhost";
   $username = "root";
   $password = "123456";
   $dsn = "$dbms:host=$server";
   try {
      $pdo = new PDO ( $dsn, $username, $password );
      echo "PDO 连接 MySQL 数据库服务器成功";
      print ($pdo->exec ( "create database account_pdo_db" )) ;
      print ($pdo->exec ( "use account_pdo_db" )) ;
      print ($pdo->exec ( "set names utf8" )) ;
      print ($pdo->exec ( "create table account(id int auto_increment primary key, username varc
har(50) not null, password varchar(50) not null, email  varchar(50) not null)" )) ;
      print ($pdo->exec ( "insert into account(username, password, email) values ('Jack', '123456
', 'jack@example.com')" )) ;
      print ($pdo->exec ( "insert into account(username, password, email) values ('Lucy', '123456
', 'lucy@example.com')" )) ;
      $pdo = null;
   } catch ( PDOException $e ) {
      echo "PDO 连接 MySQL 数据库服务器失败";
      die ();
   }
?>
```

query()

query()方法不同于 exec()，通常用于 select 语句中，它的返回值是 PDOStatement 的实例，是 POD 里的结果集。它的语法如下：

```
PDOStatement PDO::query(String sql)

<?php
echo "<pre>";
$dbms = "mysql";
$server = "localhost";
$username = "root";
$password = "123456";
$dbname = "account_pdo_db";
```

```
$dsn = "$dbms:host=$server;dbname=$dbname";
try {
   $pdo = new PDO ( $dsn, $username, $password );
   echo "<p>PDO 连接 MySQL 数据库服务器成功</p>";
   $result = $pdo->query ( "select * from account" );
   foreach ( $result as $row ) {
      var_dump ( $row );
   }
   $pdo = null;
} catch ( PDOException $e ) {
   echo "PDO 连接 MySQL 数据库服务器失败";
   die ();
}
?>
```

第九节 预处理语句

PDO 提供对预处理语句的支持。

预处理语句是预先将一个预处理的 sql 语句发送到数据库服务器，执行其他 sql 语句只是修改预处理语句里对应的参数。简单的说，就是将 sql 语句强制一分为二：第一部分为前面相同的命令和结构部分，第二部分为后面可变的数据部分。预处理语句，可以减轻数据库服务器压力。

定义预处理语句

使用 prepare()方法执行 sql 预处理语句，得到一个 PDOStatement 实例，sql 预处理语句通常有如下两种定义方式：

- 命名参数：自定义的有意义的字符串作为命名参数，前面加上冒号。

insert into table_name(name,password,email) values(:name,:password,:email)

- 问号数据占位符：使用“？”作为参数。

insert into table_name(name,password,email) values(?,?,?)

绑定参数

往预处理语句绑定参数有三种方法：

- bindParam()方法一个一个绑定，绑定完成执行 execute()方法使之生效。
- bindValue()方法一个一个绑定，绑定完成执行 execute()方法使之生效。

- 直接使用 execute()方法传递一个数组，命名参数使用关联数组，数据占位符使用索引数组。

```php
<?php
   echo "<pre>";
   $dbms = "mysql";
   $server = "localhost";
   $username = "root";
   $password = "123456";
   $dbname = "account_pdo_db";
   $dsn = "$dbms:host=$server;dbname=$dbname";
   try {
      $pdo = new PDO ( $dsn, $username, $password );
      echo "PDO 连接 MySQL 数据库服务器成功";

      // 数据占位符
      $pre = $pdo->prepare("insert into account(username, password, email) values (?,?,?)");
      $name = "Peter";
      $pwd = "333333";

      $pre->bindParam ( 1, $name );
      $pre->bindValue ( 2, $pwd );
      $pre->bindValue ( 3, "Peter@example.com" );
      $pre->execute ();

      $name = "张三";
      $pre->execute (array($name,"1245","zhangsan@example.com"));

      // 命名参数
      $pre = $pdo->prepare("insert into account(username, password, email) values (:name,:pwd,:
email)" );
      $name = "老王";
      $pwd = "00544abc";
      $email = "laowang@example.com";
      $pre->bindParam ( ":name", $name );
      $pre->bindParam ( ":pwd", $pwd );
      $pre->bindParam ( ":email", $email );
      $pre->execute();
      $name = "柴科夫斯基";
      $pre->execute(array(":name"=>$name,":pwd"=>"love",":email"=>"abc@abc.com"));
      $pdo = null;
   } catch ( PDOException $e ) {
      echo "PDO 连接 MySQL 数据库服务器失败";
      die();
   }
?>
```

第十节 解析结果集

PDO 提供了三种方法来解析结果集

fetch()方法

从结果集中获取下一行的数据，返回的数组依赖于提取的类型，它的语法格式为：

```
array PDOStatement::fetch([int fetch_style[, int cursor_orientation[, int cursor_offset])
```

```
<?php
$dbms = "mysql";
$server = "localhost";
$username = "root";
$password = "123456";
$dbname = "account_pdo_db";
$dsn = "$dbms:host=$server;dbname=$dbname";
try {
   $pdo = new PDO ( $dsn, $username, $password );
   echo "<p>PDO 连接 MySQL 数据库服务器成功</p>";
   $result = $pdo->query("select * from account");
   var_dump($result->fetch());
   var_dump($result->fetch(PDO::FETCH_ASSOC)); //返回索引为集列名的数组
   var_dump($result->fetch(PDO::FETCH_NUM)); //返回索引为数值的数组
   var_dump($result->fetch(PDO::FETCH_OBJ)); //返回属性名对应结果集的列表的匿名对象
   $pdo = null;
} catch ( PDOException $e ) {
   echo "<p>PDO 连接 MySQL 数据库服务器失败</p>";
   die ();
}
?>
```

fetchAll()方法

fetchAll()方法可以返回一个包含结果集中所有行的数组，它的语法格式如下：

```
array PDOStatement::fetchAll([int $fetch_style[, mixed fetch_arg[, array ctor_arg]]])
```

```
<?php
$dbms = "mysql";
$server = "localhost";
$username = "root";
$password = "123456";
$dbname = "account_pdo_db";
$dsn = "$dbms:host=$server;dbname=$dbname";
try {
```

```
    $pdo = new PDO ( $dsn, $username, $password );
    echo "<p>PDO 连接 MySQL 数据库服务器成功</p>";

    echo "<hr />";
    $result = $pdo->query ( "select * from account" );
    var_dump ( $result->fetchAll (PDO::FETCH_NUM) );

    echo "<hr />";
    $result = $pdo->query ( "select * from account" );
    var_dump ( $result->fetchAll ( PDO::FETCH_ASSOC ) );

    $pdo = null;
} catch ( PDOException $e ) {
    echo "<p>PDO 连接 MySQL 数据库服务器失败</p>";
    die ();
}
?>
```

fetchColumn()方法

fetchColumn()方法用来从结果集中获取下一行中指定列的值。它的基本语法如下：

```
string PDOStatement::fetchColumn([int column_num])
//column_num 是列的索引数字（从 0 开始），默认值为 0，表示第一列

<?php
$dbms = "mysql";
$server = "localhost";
$username = "root";
$password = "123456";
$dbname = "account_pdo_db";
$dsn = "$dbms:host=$server;dbname=$dbname";
try {
    $pdo = new PDO ( $dsn, $username, $password );
    echo "<p>PDO 连接 MySQL 数据库服务器成功</p>";

    echo "<hr />";
    $result = $pdo->query ( "select * from account" );
    var_dump ( $result->fetchColumn());
    echo "<hr />";
    var_dump ( $result->fetchColumn(1));
    echo "<hr />";
    var_dump ( $result->fetchColumn(2));
    echo "<hr />";
    var_dump ( $result->fetchColumn(3));

    $pdo = null;
} catch ( PDOException $e ) {
```

```
    echo "<p>PDO 连接 MySQL 数据库服务器失败</p>";
    die ();
}
?>
```

第十一节 SQL 注入

SQL 注入就是通过把 SQL 命令插入到 Web 表单提交或输入域名或页面请求的查询字符串，最终达到欺骗服务器执行恶意的 SQL 命令。SQL 注入通常由于执行 sql 语句时，没有对用户通过 Web 表单提交的参数或者查询的字符串没有进行特殊字符过滤等，导致欺骗服务器执行恶意 SQL 命令。

为了防止这种情况，写脚本文件的时候，通常会考虑防止 SQL 注入，最简单的办法就是使用预处理语句，因为预处理语句事先已经编译了语句，传递的参数是不参与解释的。

附录：

PHP 的报错级别

值	常量	描述
1	E_ERROR	致命的运行时错误。 错误无法恢复过来。 脚本的执行被暂停
2	E_WARNING	非致命的运行时错误。 脚本的执行不会停止
4	E_PARSE	编译时分析错误。解析应该只由分析器生成的错误
8	E_NOTICE	可能发现了一个错误
16	E*CORE*ERROR	在 PHP 启动时的致命错误
32	E*CORE*WARNING	在 PHP 启动时的非致命错误
64	E*COMPILE*ERROR	致命的编译时错误
128	E*COMPILE*WARNING	非致命编译时警告
256	E*USER*ERROR	用户生成的致命错误
512	E*USER*WARNING	用户生成的非致命警告
1024	E*USER*NOTICE	用户生成的通知
2048	E_STRICT	协同性或兼容性错误
4096	E*RECOVERABLE*ERROR	可捕获的致命错误

8191	E_ALL	所有的错误和警告

$_SERVER 参数

参数	说明
$*SERVER['REMOTE*PORT']	端口
$*SERVER['REMOTE*ADDR']	浏览当前页面的用户的 IP 地址
$*SERVER['SERVER*NAME']	服务器主机的名称
$*SERVER['SERVER*ADMIN']	管理员信息
$*SERVER['SERVER*PORT']	服务器所使用的端口
$*SERVER['SERVER*SIGNATURE']	包含服务器版本和虚拟主机名的字符串
$_SERVER['argv']	传递给该脚本的参数
$_SERVER['argc']	传递给程序的命令行参数的个数
$*SERVER['GATEWAY*INTERFACE']	CGI 规范的版本
$*SERVER['SERVER*SOFTWARE']	服务器标识的字串
$*SERVER['SERVER*PROTOCOL']	请求页面时通信协议的名称和版本 例如："HTTP/1.0"。
$*SERVER['REQUEST*METHOD']	访问页面时的请求方法 例如："GET", "POST"
$*SERVER['QUERY*STRING']	查询(query)的字符串
$*SERVER['DOCUMENT*ROOT']	当前运行脚本所在的文档根目录
$*SERVER['HTTP*ACCEPT']	当前请求的 Accept: 头部的内容
$*SERVER['HTTP*ACCEPT_CHARSET']	当前请求的 Accept-Charset: 头部的内容
$*SERVER['HTTP*ACCEPT_ENCODING']	当前请求的 Accept-Encoding: 头部的内容
$*SERVER['HTTP*CONNECTION']	当前请求的 Connection: 头部的内容。例如："Keep-Alive"
$*SERVER['HTTP*HOST']	当前请求的 Host: 头部的内容
$*SERVER['HTTP*REFERER']	链接到当前页面的前一页面的 URL 地址
$*SERVER['HTTP*USER_AGENT']	当前请求的 User_Agent: 头部的内容

$_SERVER['HTTPS']	如果通过 https 访问,则被设为一个非空的值(on)，否则返回 off
$*SERVER['SCRIPT*FILENAME']	当前执行脚本的绝对路径名
$*SERVER['SCRIPT*NAME']	包含当前脚本的路径。这在页面需指向自己时非常有用
$*SERVER['PHP*SELF']	正在执行脚本的文件名
$*SERVER['PHP*AUTH_USER']	当 PHP 运行在 Apache 模块方式下，并且正在使用 HTTP 认证功能，这个变量便是用户输入的用户名。
$*SERVER['PHP*AUTH_PW']	当 PHP 运行在 Apache 模块方式下，并且正在使用 HTTP 认证功能，这个变量便是用户输入的密码
$*SERVER['AUTH*TYPE']	当 PHP 运行在 Apache 模块方式下，并且正在使用 HTTP 认证功能，这个变量便是认证的类型
$*SERVER['PATH*TRANSLATED']	当前脚本所在文件系统（不是文档根目录）的基本路径

第十四章 简介

Laravel 是一款简洁、优雅的 PHP Web 开发框架，它可以让你从面条一样杂乱的代码中解脱出来；它可以帮你构建一个完美的网络 APP，而且每行代码都可以简洁、富于表达力。

第一节 安装

官方手册推荐我们使用 Laravel Homstead 虚拟机，我们可以着手去配置该虚拟机，若是初学者的话，建议先使用本地环境 wamp 即可。

1. 检查环境

新建 phpinfo.php，输入 phpinfo()函数，并进行访问查看。

- PHP >= 7.0.0
- PHP OpenSSL 扩展
- PHP PDO 扩展
- PHP Mbstring 扩展
- PHP Tokenizer 扩展
- PHP XML 扩展

2. 安装 Composer

// 下载地址(点击 Composer-Setup.exe 按钮)
https:*//getcomposer.org/download/*

3.下载 Laravel

composer global require "laravel/installer"

第二节 配置

在使用前，要确保以下几个内容的配置是正确的，配置有问题会影响我们的项目开发。

1. 虚拟域名

框架安装完毕后要给项目定义一个虚拟域名，具体步骤如下：

httpd.conf

打开 wamp/bin/apache/apache2.4.23/conf/httpd.conf 文件，查找 Include conf/extra/httpd-vhosts.conf，将其前方的#号去掉。

hosts

编辑 C:\Windows\System32\drivers\etc\hosts 文件，添加：127.0.0.1 www.test.com

httpd.vhost

编辑 wamp/bin/apache/ apache2.4.23/conf/extra/httpd-vhosts.conf 文件，参考如下代码：

```
// 复制已存在的 VirtualHost 标签包含的所有内容，并修改如下三步(含)
<VirtualHost *:80>
        ServerName www.test.com       //1. 修改这里与 hosts 文件中的域名一样
        ServerAlias localhost
        DocumentRoot c:/wamp/www/test/public        //2.修改这里为你的 Laravel 框架入口文件所在路径
        <Directory  "c:/wamp/www/test/public">        //3.这里的修改同上
                Options +Indexes +Includes +FollowSymLinks +MultiViews
                AllowOverride All
                Require local
        </Directory>
</VirtualHost>
```

2. 配置文件

Laravel 的所有配置文件都存储在 config 目录下

3. 目录权限

确保 storage 目录和 bootstrap/cache 目录应该允许 Web 服务器写入，否则 Laravel 将无法运行。

4. 应用秘钥

如果应用程序密钥没有被设置，就不能确保你的用户会话和其他加密数据的安全！

```
//在 git 命令行输入如下命令即可
php artisan key:generate
```

5. .env 配置

确保.env 配置文件中的数据库配置是正确的(如下)

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=myshop
DB_USERNAME=root
DB_PASSWORD=1234
```

6. 维护模式

若你的项目进入了维护状态，你需要执行以下命令来关闭你的应用；维护完毕记得开启！

```
//维护
php artisan down

//启动
php artisan up
```

第三节 目录结构

认识目录结构，有助于我们在后续的开发更迅速与便捷

```
test
  |----app/ *                  //包含应用程序的核心代码(Controller、Middleware、Model)
  |----bootstrap/          //包含引导框架并配置自动加载的文件
  |----config/ *           //包含应用程序所有的配置文件(app.php、database.php)
  |----database/ *         //包含数据填充和迁移文件
  |----public/ *           //包含了入口文件 index.php、资源文件(Js/Css/Img)
  |----resoure/ *          //包含了视图和未编译的资源文件(LESS/SASS)
  |----routes/ *           //包含了应用的所有路由定义(web.php/api.php)
  |----storage/                //包含编译的 Blade 模板
  |----tests/                  //包含自动化测试文件
  |----vendor/                 //包含了你的 Composer 依赖包
  |----.env*                   //环境配置文件
  |----artisan*                //Artisan 是 Laravel 中自带的命令行工具的名称
  |----composer.json       //Composer 用于安装依赖包的配置文件
  |----composer.lock       //优先读取的依赖版本文件，可确保使用者使用相同版本依赖包
  |----package.json            //依赖包详细描述文件
  |----phpunit.xml             //一个面向程序员的 PHP 测试框架
  |----readme.md           //使用说明书
  |----server.php          //模拟了 web server 的 rewrite 功能，主要用于测试
  |----webpack.mix.js      //资源文件打包配置文件
```

第十五章 路由

当我们请求 index.php 这个加载框架的起点，它就为我们加载 Composer 生成定义的自动加载器，紧接着就找到 bootstrap/app.php 脚本中检索 Laravel 应用程序的实例。

当程序完成上述几步以后，就会通过 HTTP 请求去访问 app/Http/Kernel.php 中的 HTTP 内核，从而使得内核把我们带到了 Routes/web.php 路由文件中。

执行完了上述的一系列过滤、加载、启动之后，Laravel 开始给我们分发路由了，路由的核心就是要让我们定位到某个闭包程序或某个控制器上去。

第一节 基本应用

简单路由

路由文件在 Routes/web.php

```
//这是一个测试路由，当我们访问 www.test.com/test 时，会访问到此路由，并在页面输出 Hello World
Route::get('/test', function () {
    return 'Hello World';
});
```

请求方式

路由指定了请求方式，必须由该方式进行访问，否则页面会提示未找到

```
Route::get($uri, $callback);          //GET 方式请求(常用)
Route::post($uri, $callback);         //POST 方式请求(常用)
Route::put($uri, $callback);          //伪造 PUT 方法请求
Route::patch($uri, $callback);        //伪造 PATCH 方法请求
Route::delete($uri, $callback);       //伪造 DELETE 方法请求
```

路由参数

说到请求，当然少不了参数，Laravel 的路由位置也可以传递参数

单个参数

路由中的参数，可以在闭包程序中直接使用，也可以在控制器中直接获取

```
//当我们请求 www.test.com/user/10 时，会将 10 的参数传递到当前路由
Route::get('user/{id}', function ($id) {
    return '当前用户的 id 是：'.$id;
});
```

多个参数

参数与参数之间的符号分割没有明确的限定，通常为/或-

```
//当我们请求 www.test.com/user/5-info-1 时，可以访问到如下路由的内容
Route::get('user/{id}-{service}-{page}', function ($id, $service, $page) {
   //输出参数信息
        return '您正在查看用户 id 为：'.$id.'的'.$service.'服务的第'.$page.'页的内容！';
});
```

可选参数

某些情况下，有的参数是可有可无的，我们可以使用?问好来标注改参数是可有可无的。

```
//当我们请求 www.test.com/user/zhangsan 或 www.test.com/user 时
Route::get('user/{name?}', function ($name = null) {
   if($name != null){
                return "这是 zhangsan 的信息";
        }else{
                return "这里是用户的列表信息";
        }
});
```

正则约束

约束一个

若不给用户所传递的参数限定格式，将导致一些不必要的麻烦，因此，我们可以使用正则对其进行约束。

```
//在路由后方追加 where 方法，第一个参数是要进行约束的参数名称，第二个参数则是正则模式的规则
Route::get('user/{id}', function ($id) {
   return '当前用户的 id 是：'.$id;
})->where('id','\d+');
```

约束多个

```
Route::get('user/{id}-{service}-{page}', function ($id, $service, $page) {
   //输出参数信息
        return '您正在查看用户 id 为：'.$id.'的'.$service.'服务的第'.$page.'页的内容！';
})->where(['id'=>'\d+', 'service'=>'\w+', 'page'=>'\d+']);
```

命名路由

某些路由的可能会非常非常长，例如 users/list/info/{id}-{service}-{page}...，若想要在跳转到或获取该路由时，可能不太方便，因此，Laravel 给我们准备了命名路由。

```
//直接在路由后方追加 name 方法，并定义当前路由的别名
Route::get('user/info', function () {
   //
})->name('uinfo');
```

生成链接

这样一来，若我们想要进行跳转，我们可以这样写

```
// 生成 URL...
$url = route('uinfo');     //www.test.com/user/info

// 生成重定向...
return redirect()->route('uinfo');
```

第二节 路由组

我们的项目程序都是分为前台后台模块的，有些项目甚至分了其他诸多模块，因此，若一直按之前的路由书写方式写下去，将导致路由文件中的路由毫无章法，因此，Laravel 为我们准备了路由组

```
//前台路由组（没有任何条件限定）
Route::group([这里存放中间件、路由前缀等信息，默认为空(老版本)],function(){
 //加载首页的路由
  Route::get('/',function(){
        return "这里是前台首页！";
  });
});
```

中间件

如果你想要让用户在进入当前路由组之前，给其添加一些要求，我们可以使用路由组中间件进行限定

```
//添加了一个 login 的中间件限定，用户必须登录后才能访问前台首页（知乎）
Route::middleware('login')->group(function(){
 //加载首页的路由
  Route::get('/',function(){
        return "这里是前台首页！";
  });
});
```

如果有多个中间件限定，我们也可以以数组方式进行使用

```
Route::middleware(['login','auth','status'])->group(function(){...});
```

命名空间

这里的命名空间主要是针对控制器的，因为项目划分前后台的原因，我们需要将前台的控制器放置到前台目录，后台控制器放置到后台目录，因此，这个时候就涉及到了前后台命名空间的问题。

```
//若你的控制器来自于后台，则可以直接使用如下方式指定
Route::namespace('Admin')->group(function () {
    // 在 "App\Http\Controllers\Admin" 命名空间下的控制器
});
```

路由前缀

当我们通过路由访问某个应用时，因为前后台的原因，路由肯定也应当有一些响应的区别，因此，我们可以使用路由前缀来规定访问前台和访问后台的路由。

```
//当我们请求 www.test.com/admin/users 时会访问到如下的方法
Route::prefix('admin')->group(function () {
    Route::get('users', function () {
        // 匹配包含 "/admin/users" 的 URL
    });
});
```

同时使用

若你想要再同一个路由组中同时使用中间件、路由前缀、命名空间等信息，你可以这样写

```
Route::middleware('login')->prefix('home')->namespace('home')->group(function(){
  //加载首页的控制器
  Route::get('/','IndexController@index');
});
```

第十五章 中间件

上一章，我们讲到了中间件，但是只是说了它怎么使用，还不知道怎么创建，因此，本节来带大家详细操作一下。

第一节 基本应用

中间件可以过滤用户进入应用的 HTTP 请求，它是在用户发送 HTTP 请求之后，执行路由内容操作之前来进行执行的一段内容【参考 Laravel 运行过程图示】

生成文件

生成的文件会被自动存储到 app/Http/Middleware 目录内，并同时生成了 Login.php 文件

```
//在 git 中执行下面指令
php artisan make:middleware Login
```

书写规则

这里我们写一个测试的登录验证，判断用户在进入后台框架时是否进行了登录

```
//这是中间件类中的 handle 方法，用于处理传入的请求
public function handle($request, Closure $next)
{
 //判断用户是否登录
 if(!$request->session()->has('uid')){
  //若没有登录，跳转到登录页
  return rediract('/login');
 }
 //若登录了，则执行下一步操作
 return $next($request);
}
```

添加中间件

给指定的路由添加中间件，在浏览器访问 www.test.com

```
Route::middleware('login')->prefix('home')->namespace('home')->group(function(){
 //加载首页的控制器
 Route::get('/','IndexController@index');
});
```

注册中间件

第四部中的访问是失败的，系统告诉我们没有当前中间件类，原因是我们还没有对其进行注册，在 app/Http/Kernel.php 中前去注册吧

```
//这里是路由注册
protected $routeMiddleware = [
  'auth' => \Illuminate\Auth\Middleware\Authenticate::class,
  'auth.basic' => \Illuminate\Auth\Middleware\AuthenticateWithBasicAuth::class,
  'bindings' => \Illuminate\Routing\Middleware\SubstituteBindings::class,
  'can' => \Illuminate\Auth\Middleware\Authorize::class,
  'guest' => \App\Http\Middleware\RedirectIfAuthenticated::class,
  'throttle' => \Illuminate\Routing\Middleware\ThrottleRequests::class,
  'login' => \App\Http\Middleware\Login::class, //指定命名空间与类名
];
```

以上五个步骤都完成了，我们的中间件即可生效！

第二节 CSRF 跨域请求伪造

跨站请求伪造是一种恶意的攻击，它凭借已通过身份验证的用户身份来运行未经过授权的命令。我们要做的就是防止有人恶意攻击我们的网站。因此需要借助 Laravel 帮我们生成的这张 CSRF「令牌」

1. 基本使用

```
//通过表单发送 post 请求时，需要添加令牌验证
<form method="POST" action="/profile">
    {{ csrf_field() }}
    ...
</form>
```

2. JS 的 ajax 请求

需要添加的内容有两块，一个在 head 头不，一个在 post 请求内部

```
//head 头添加 token 请求
<meta name="csrf-token" content="{{ csrf_token() }}">

//定义发送 post 请求的按钮
<button id="btn">单击发送 ajax 的 post 请求</button>

//定义发送 post 请求的 js 代码(使用 JQuery)
<script>
```

```
 $('#btn').click(function(){
  //单击事件中添加如下代码
  $.ajaxSetup({
    headers: {
      'X-CSRF-TOKEN': $('meta[name="csrf-token"]').attr('content')
    }
  });
  //发送 post 请求给指定路由
          $.post('/testAjax',{},function(data){
            alert(data);
          });
 });
</script>
```

第十六章 控制器

第四节所讲的路由，大多是直接在闭包函数中来执行操作的。这固然可以，但是，我们不推荐你在闭包函数中书写程序逻辑代码，因此，Laravel 还给我们提供了另一种处理逻辑程序的介质-控制器

第一节 基础控制器

生成控制器

下面命令可以直接在 app/Http/Controllers 当中直接直接生成控制器文件

```
//和 ThinkPHP 当中的控制器定义方式一致，类名也采用大写字母开头的驼峰命名法
php artisan make:controller UsersController
```

除了上面的直接生成，你还可以将控制器生成在一个目录中

```
//这样做的好处是，可以将前后台或其他模块的控制器划分开【注意命名空间】
php artisan make:controller Admin/UsersController
```

定义路由

这些路由，是指向控制指定方法的路由

```
Route::get('users/{id}', 'UsersController@show');
```

若是带命名空间的控制器，我们需要按照如下方式书写【代表要找 Admin/UsersController 】

```
Route::get('users/{id}', 'Admin\UsersController');
```

控制器结构

如下是一个控制器内部的结构

```
namespace App\Http\Controllers;        //当前控制器所在的空间

use App\User;
use App\Http\Controllers\Controller;    //控制器基类的导入

class UserController extends Controller
{
    /**
     * 展示给定用户的信息。
     *
     * @param  int  $id
     * @return Response
```

```
    */
    public function show($id)    //这里可以接受用户跳转时传递的 id 信息
    {
        //响应 `resource/view/users/show.blade.php` 页面，并传递参数
        return view('users/show', ['user' => User::findOrFail($id)]);
    }
}
```

第二节 控制器中间件

之前，我们只给路由组添加过中间件，除了可以给路由组来添加中间件，我们还可以给任何一个指定的路由指定中间件，包括控制器路由。

```
//我们可以这样写
Route::get('users/{id}', 'Admin\UsersController')->middleware('login');
```

或者，你也可以给控制器当中的指定方法添加中间件

```
class UsersController extends Controller
{
    /**
     * 实例化一个新的控制器实例。
     *
     * @return void
     */
    public function __construct()
    {
        $this->middleware('auth'); //给当前控制器添加 auth 中间件

        $this->middleware('log')->only('index'); //只给当前控制器的 index 方法添加 log 中间件

        $this->middleware('subscribed')->except('store');    //除了 store 方法都添加中间件
    }
}
```

第三节 资源控制器

资源路由可以将典型的「CRUD」路由指定到一个控制器上，仅需一行代码即可实现。

生成资源控制器

```
//直接在 Controllers 目录生成
php artisan make:controller GoodsController --resource

//生成到指定空间下
php artisan make:controller Admin/GoodsController --resource
```

定义路由

因为资源控制器的特性是，只要我们按照其要求请求指定的路由，我们就可以访问到资源控制的指定方法，因此我们在定义路由的时候不需要再去指定方法，这将极大地节省路由文件的空间。

```
Route::resource('goods', 'GoodsController');
```

资源控制器操作处理

这里罗列了我们请求资源控制器指定方法的动作、路由及能够请求到的方法。

动作	URI	操作	路由名称
GET	/photos	index	photos.index
GET	/photos/create	create	photos.create
POST	/photos	store	photos.store
GET	/photos/{photo}	show	photos.show
GET	/photos/{photo}/edit	edit	photos.edit
PUT/PATCH	/photos/{photo}	update	photos.update
DELETE	/photos/{photo}	destroy	photos.destroy

请求方法伪造

通过观察上面表格，我们发现，有两个动作比较特殊，一个是 PUT/PATFCH，另一个就是 DELETE，HTTP 请求没有给我们准备这两个请求方式，因此，我们需要伪造两种请求方式

```
//伪造 PUT/PATCH 请求
{{method_field('PUT')}}

//伪造 DELETE 请求
{{method_field('DELETE')}}
```

Ajax 删除实例

```
//删除按钮的单击事件
   $('.del').click(function(){

      //获取自己的节点
      var _this = $(this);

      //获取 id
      var id = $(this).attr('id');
```

```
    //添加 confirm 弹框
    if(confirm("确定要删除当前数据吗？")){

        //发送 ajax 请求
        $.post('/admin/users/'+id,{"_method":"DELETE","_token":"{{csrf_token()}}"},function
(data){

            //判断是否成功
            if(data==1){
                alert('恭喜，删除成功！');
                _this.parent().parent().parent().remove();

            }else{
                alert('抱歉，删除失败！');
            }

        });
    }
});
```

部分资源路由

当声明一个资源路由时，你可以指定控制器处理部分操作，而不必使用全部默认操作：

```
//UsersController 中只有 index 和 show 方法可以被访问
Route::resource('/users', 'UsersController', ['only' => [
    'index', 'show'
]]);

//UsersController 中除了列表中的方法，均可被访问
Route::resource('/users', 'UsersController', ['except' => [
    'create', 'store', 'update', 'destroy'
]]);
```

附加资源控制器

在某些情况下，资源控制器的方法不足以为我们提供足够的服务，此时，我们就需要为其追加一些方法了。

```
//这是追加的路由
Route::get('/users/test', 'UsersController@test');
//这里是资源控制器的路由
Route::resource('/users', 'UsersController');
```

路由缓存

如果你的应用只用到了基于控制器的路由，那么你应该充分利用 Laravel 的路由缓存。使用路由缓存将极大地减少注册全部应用路由的时间。某些情况下，路由注册甚至可以快一百倍。要生成路由缓存，只需在 Artisan 命令行中执行

php artisan route:cache

运行这个命令之后，每一次请求的时候都将会加载缓存的路由文件。记住，如果添加了新的路由，你需要刷新路由缓存。

php artisan route:clear

注意：基于闭包的路由并不能被缓存。如果要使用路由缓存，你必须将所有的闭包路由转换成控制器类路由。

第十七章 获取请求

路由和控制器我们均已经经过了系统的学习，本节的课程主要练习 HTTP 的请求，以及在控制器当中如何进行获取，很重要的一个操作对象是 $request 请求对象。

```php
<?php
//当前控制器的命名空间
namespace App\Http\Controllers;
//用于获取 HTTP 请求的处理类导入
use Illuminate\Http\Request;

class UsersController extends Controller
{
        //在指定的方法形参列表中，使用类型约束的方法，传入请求对象
    public function store(Request $request)
    {
        //使用 request 请求对象进行数据的获取，而非$_POST 的原生写法
        $name = $request->input('name');

        //你还可以使用另外一种方式获取
            $name = $request->name;
    }
}
```

第一节 基本应用

请求路径

```php
// 若请求的目标地址是：http://test.com/users/show，则 path 会获取 users/show
$uri = $request->path();
```

检测请求路径

```php
// 若请求的路径是：http://test.com/admin/users，则下面实例为真
if ($request->is('admin/*')) {
    //
}
```

获取完整 URL

```php
// 没有请求参数...
$url = $request->url();

// 有请求参数...
$url = $request->fullUrl();
```

获取请求方法

```
//若此时请求到该方法的方式为GET，则输出 GET
$method = $request->method();

//判断当前方法是通过哪种请求方式请求的
if ($request->isMethod('post')) {
   //
}
```

第二节 获取请求数据

本节内容是一个重点，所有操作均为数据获取服务

获取所有数据

```
//以数组形式存储
$res = $request->all();
```

获取指定数据

```
//会获取到form 表单中表单项name 的值
$name = $request->input('name');

//指定默认值(若表单提交信息中没有 name 相对应的值，则采用默认值)
$name = $request->input('name', 'Sally');
```

获取数组的值

```
//我们可以采用点的形式获取数组的相对应的值
$name = $request->input('products.0.name');

//获取所有二维数组中的name 值
$names = $request->input('products.*.name');
```

只获取查询字符串

这里查询字符串所指的是，通过 get 传递到当前方法的参数信息。

```
//获取 get 请求中 name 指向的值
$name = $request->query('name');

//查询字符串也可以定义默认值
$name = $request->query('name', 'Helen');
```

获取部分输入数据

```
//只从请求数组中获取下标为 username 和 password 的信息
$input = $request->only(['username', 'password']);
//这种写法也支持
$input = $request->only('username', 'password');
//获取除了 credit_card 的信息
$input = $request->except(['credit_card']);
//同上
$input = $request->except('credit_card');
```

判断请求信息是否包含某值

```
//判断请求信息中是否包含下标为 name 的值
if ($request->has('name')) {
    //
}

//判断请求信息是否包含 name 和 email 的值
if ($request->has(['name', 'email'])) {
    //
}
```

第三节 获取旧输入数据

表单提交时，通常会对输入数据做信息验证，若信息内容不通过，则会跳回到表单也，此时我们刚刚输入的信息会丢失，这不便于用户执行后续的操作，因此，Laravel 为我们准备了闪存这一概念

将数据闪存至 Session

```
//闪存所有信息
$request->flash();

//闪存指定信息
$request->flashOnly(['username', 'email']);

//去除法闪存信息
$request->flashExcept('password');
```

闪存后重定向

```
//闪存所有信息，并跳转至 form 路由
return redirect('form')->withInput();
```

```
//闪存除了 password 之外的信息，并跳转至 form 路由
return redirect('form')->withInput(
   $request->except('password')
);
```

视图页面的输出

```
<input type="text" name="username" value="{{ old('username') }}">
```

第四节 Cookies SESSION

Laravel 给我们准备的 Cookies 操作，会自动加密，因此更加安全，虽然我们也可以使用 PHP 的原生 Cookie 进行设置，但是推荐大家使用以下方法设置 Cookies

获取 Cookie 值

```
$value = $request->cookie('name');
```

设置 Cookie

```
//响应信息到页面，并将 cookie 写入用户浏览器
return response('Hello World')->cookie('name', 'value', $minutes);

//以下方法同上（response 信息可以省略）
$cookie = cookie('name', 'value', $minutes);
return response('Hello World')->cookie($cookie);
```

获取 SESSION 值

```
//获取指定的 session 信息
$value = $request->session()->get('key');
$value = $request->session()->get('key', 'default');        //默认值

//获取所有的 session 信息
$data = $request->session()->all();

//判断某个 session 信息是否存在
if ($request->session()->has('users')) {
   //
}
```

设置 SESSION 值

```
// 通过 HTTP 请求实例...
$request->session()->put('key', 'value');
```

```
// 通过全局辅助函数
session(['key' => 'value']);
```

删除 SESSION 信息

```
//删除指定值
$request->session()->forget('key');
//删除所有信息
$request->session()->flush();
```

全局辅助函数

```
// 获取 Session 中的一条数据...
$value = session('key');

// 指定一个默认值...
$value = session('key', 'default');

// 存储一条数据至 Session 中...
session(['key' => 'value']);
```

第五节 文件处理

获取上传文件信息

```
//放入文件表单项信息
$file = $request->file('photo');

//或者直接将表单项名称写成这样也行
$file = $request->photo;
```

判断上传文件是否存在

```
if ($request->hasFile('photo')) {
    //
}
```

判断上传文件是否有效

```
if ($request->file('photo')->isValid()) {
    //
}
```

获取文件路径 、拓展名

```
//获取上传文件完整路径
$path = $request->photo->path();

//获取拓展名信息
$extension = $request->photo->extension();
```

获取上传文件的其他信息

```
//通过该网址查看
http://api.symfony.com/3.0/Symfony/Component/HttpFoundation/File/UploadedFile.html
```

存储上传文件

这里是移动文件到本地目录的重点内容，默认情况下，文件会上传到 Storage 目录，若想要修改，需要进行如下设置

设置上传文件存储目录

在 config/filesystems.php 中进行设置即可。

```
'local' => [
        'driver' => 'local',
        'root' => storage_path('uploads'),
         ],
//修改为
'local' => [
        'driver' => 'local',
        'root' => public_path('uploads'),
         ],
```

执行上传

```
//文件会上传到配置文件设置的目录当中
$path = $request->photo->store();

//若要生成子目录，可以在 store 方法中指定，例如
$path = $request->photo->store('images');
```

第十八章 响应信息

响应信息，其实就是返回一些内容给客户端浏览器，例如：返回字符串、返回数组、返回对象、返回模板等都属于是响应信息。关键字：return

第一节 响应模板

```
return view('home/Index/index');
```

响应模板，并追加信息

```
//响应单个参数
return view('home/Index/index',['res'=>$res]);

//响应多个参数
return view('home/Index/index',['res'=>$res,'id'=>$id]);
```

响应 JSON 数据

```
//响应 JSON 数据
 return response()->json(['name' => 'Abigail','state' => 'CA']);
```

第二节 重定向

```
Route::get('dashboard', function () {
 //重定向至 home/dashboard 路由
 return redirect('home/dashboard');

 //重定向至上一页面并闪存表单信息
 return back()->withInput();

 //重定向至指定路由，同时往 session 中存储指定信息
 return redirect('/users/create')->with('error','添加时遇到了错误！');
 //模板中输出错误信息
 @if (session('error'))
    <div class="alert alert-success">
       {{ session('status') }}
    </div>
 @endif

 //重定向至命名路由
 return redirect()->route('login');
 return redirect()->route('profile', ['id' => 1]);   //还可以追加参数
```

```
 //重定向至控制器指定方法
 return redirect()->action('HomeController@index');
 return redirect()->action('UserController@profile', ['id' => 1]);
});
```

第三节 文件下载

```
//将指定目录下的指定文件信息进行下载
return response()->download($pathToFile, $name);
```

第四节 文件响应

若你想要将文件直接输出至网页，而不是下载【例如图像或是 PDF】，你可以使用如下方法

```
return response()->file($pathToFile);
```

第十九章 Blade 模板引擎

掌握了路由，学习了控制器，了解了数据的请求与响应，本节学习 Laravel 的 Blade 模板引擎的一些相关的使用技巧。

第一节 模板引擎

区块占位

首先需要定义一个公共模板，我们这里在 layouts 目录中生成了 index.blade.php 模板

```
<!-- 文件保存于 resources/views/layouts/index.blade.php -->

<html>
  <head>
                <!--这里是一个单行占位-->
    <title>应用程序名称 - @yield('title')</title>
  </head>
  <body>
                <!--这里是一个区块占位-->
    @section('sidebar')
      这是 master 的侧边栏。
    @show

    <div class="container">
      @yield('content')
    </div>
  </body>
</html>
```

第二节 继承布局

公共模板定义好了，那意味着你的框架已经打好了，接下来就是让其他模块来继承公共模块了，然后将需要进行内容替换的位置进行替换即可。

```
<!-- 文件保存于 resources/views/child.blade.php -->
//继承公共模板内容
@extends('layouts/index')

//替换单行占位的内容
@section('title', 'Page Title')

//替换区块占位的 sidebar 内容
@section('sidebar')
```

```
    @parent      //这里的 parent 代表了，依然要延续使用公共框架的内容

    <p>这将被添加到主侧边栏。</p>
@endsection

//替换单行占位的 content 内容
@section('content')
    <p>This is my body content.</p>
@endsection
```

第三节 控制结构

if 语句

```
@if (count($records) === 1)
    我有一条记录！
@elseif (count($records) > 1)
    我有多条记录！
@else
    我没有任何记录！
@endif
```

switch 语句

```
@switch($i)
    @case(1)
        First case...
        @break

    @case(2)
        Second case...
        @break

    @default
        Default case...
@endswitch
```

循环结构

```
@for ($i = 0; $i < 10; $i++)
    目前的值为 {{ $i }}
@endfor

@foreach ($users as $user)
    <p>此用户为 {{ $user->id }}</p>
@endforeach
```

```
@while (true)
   <p>死循环了。</p>
@endwhile
```

特殊的流程控制语句

```
@foreach ($users as $user)
   @if ($user->type == 1)
      @continue          //跳过当前层遍历
   @endif

   <li>{{ $user->name }}</li>

   @if ($user->number == 5)
      @break    //终止遍历
   @endif
@endforeach
```

或者，你可以使用另外一种语法结构

```
@foreach ($users as $user)
   @continue($user->type == 1)        //直接将条件放到括号

   <li>{{ $user->name }}</li>

   @break($user->number == 5)         //同上
@endforeach
```

第二十章 数据库

本节内容主要为 Laravel 操作数据库的方法

第一节 数据库配置

在一开始的配置阶段，我们就已经配置过一次了，为了让我们有更加深刻的印象，这里再给大家普及一次。

```
// config/database.php 配置文件，我们会发现里面有 env 函数，代表若 env 已经配置好了数据库相关设置就不用再这里进行设置了。
'mysql' => [
  'driver' => 'mysql',
  'host' => env('DB_HOST', '127.0.0.1'),
  'port' => env('DB_PORT', '3306'),
  'database' => env('DB_DATABASE', 'forge'),
  'username' => env('DB_USERNAME', 'forge'),
  'password' => env('DB_PASSWORD', ''),
  'unix_socket' => env('DB_SOCKET', ''),
  'charset' => 'utf8mb4',
  'collation' => 'utf8mb4_unicode_ci',
  'prefix' => '',
  'strict' => true,
  'engine' => null,
],
```

我们推荐大家在.env 环境配置中配置数据库设置。

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=myshop
DB_USERNAME=root
DB_PASSWORD=123
```

第二节 原生 SQL

Laravel 为我们准备了一套操作数据库的原生方法，类似于 ThinkPHP 中的增删改查操作。

运行 SQL 语句前，先检查 use Illuminate\Support\Facades\DB 是不是已经进行了导入

Select 查询

```
//原生语句支持 PDO 预处理，采用如下方式即可
$users = DB::select('select * from users where active = ?', [1]);
```

Insert 添加

```
//仍然支持 PDO 预处理，采用如下方式即可
DB::insert('insert into users (id, name) values (?, ?)', [1, 'Dayle']);
```

Update 修改

```
$affected = DB::update('update users set votes = 100 where name = ?', ['John']);
```

Delete 删除

```
$deleted = DB::delete('delete from users');
```

其他语句

```
//对于表的相关操作，都可以使用 statement 来执行
DB::statement('drop table users');
```

第三节 SQL 监听

Laravel 框架操作数据库不太方便的一个地方就是查看执行之后的 SQL 语句结构，某些情况下我们书写的 SQL 结构可能是错误的，此时，我们需要通过 Laravel 为我们准备的 SQL 监听方法监听每一条 SQL

```
//在 app/Providers/AppServiceProvider.php 文件的 boot 方法下添加如下内容即可
file_put_contents('.sqls',"[".date('Y-m-d H:i:s')."]".$query->sql."\r\n",FILE_APPEND);
```

第四节 查询构造器

用原生的 SQL 语句感觉很麻烦？别担心，Laravel 给我们准备了另一种更加简便，流畅的数据库操作方式，那就是查询构造器。

查询

记得，查询构造器的使用仍然需要导入 use Illuminate\Support\Facades\DB 类。

查询所有

```
//指定要操作的数据表，并调用 get 方法获取所有 users 表中的信息
$users = DB::table('users')->get();
```

获取单条

```
//获取单行信息的时候一定要有一个唯一条件
$user = DB::table('users')->where('name', 'John')->first();
```

获取单个值

```
//只获取 name 值为 John 这一条信息中的 email 信息
$email = DB::table('users')->where('name', 'John')->value('email');
```

分块查询

这种查询方式适用于查询结果较多的情况，由于批量数据耗费资源过多，因此我们可以将数千条数据分批显示，例如每 10 条分一批，1000 条数据可以分 100 批。

```
//若使用分块查询，则必须给定 orderBy 排序条件
DB::table('users')->orderBy('id')->chunk(100, function ($users) {
    foreach ($users as $user) {
        //
    }
});
```

聚合函数

Larave 实现了与很多 MySQL 的原生函数的兼容，因此我们可以直接对其进行使用；例如：count、 max、 min、 avg 和 sum 等……

```
//统计 users 表共有多少条信息
$users = DB::table('users')->count();

//查询 orders 表的最大 price 信息
$price = DB::table('orders')->max('price');
```

若你想要在使用聚合函数的同时对其加入某些条件，你可以利用如下的方式。

```
//对 id 在 100 以内的数据进行 price 平均值的运算
$price = DB::table('orders')->where('id', '<', 100)->avg('price');
```

指定字段查询

以往我们的查询，通常是 select * from ... 这种语法，这是查询所有字段的语法，若我们不想要所有的字段，指向查询几个指定字段的信息，我们可以采用如下的方法

```
//查询 users 表中所有信息的 name、email 字段的信息，并对 email 字段定义了别名
$users = DB::table('users')->select('name', 'email as user_email')->get();
```

若你想要对查询结果去重【去除重复】，你可以采用如下方式

```
// distinct() 方法可以去除 users 表查询结果中重复的信息
$users = DB::table('users')->distinct()->get();
```

某些特殊情况下，直接使用 select 有可能导致报错，因此，Laravel 为我们准备了另一种方法，让我们能够像原生 SQL 的使用方法一样进行指定字段信息的查询。

```
//我们可以看到 select() 方法中，使用了 DB::raw() 方法，这样就可以按照原生 SQL 写法使用了
$users = DB::table('users')
    ->select(DB::raw('count(*) as user_count, status'))
    ->where('status', '<>', 1)
    ->groupBy('status')
    ->get();
```

多表联查 Join

若要进行多表联查， 你仅需要确定每个表之间有关联的字段即可使用 Join 方法进行联查

```
//以下语句进行了三表联查，分别为 users 表，contacts 表、orders 表，使用 join 方法时，指定表名，并将关联条件书写在第二个参数的位置即可实现多表联查，是不是很简单！
$users = DB::table('users')
    ->join('contacts', 'users.id', '=', 'contacts.user_id')
    ->join('orders', 'users.id', '=', 'orders.user_id')
    ->select('users.*', 'contacts.phone', 'orders.price')
    ->get();
```

Where 格式

其实前面的例子，我们已经看过不少 where 的使用方法了，这里我们要再系统的看一下 where 如何用

```
//判断是否相等
$users = DB::table('users')->where('votes', '=', 100)->get();        //等价于
$users = DB::table('users')->where('votes', 100)->get();  //等号可以省略

//算数运算符
$users = DB::table('users')->where('votes', '>=', 100)->get();       //大于等于
$users = DB::table('users')->where('votes', '<>', 100)->get();       //不等于
```

```php
//模糊查询
$users = DB::table('users')->where('name', 'like', '%T%')->get();

//多条件查询（结构太长，我们可以换行操作）
$users = DB::table('users')->where([
    ['status', '=', '1'],
    ['subscribed', '<>', '1'],
])->get();

//或者使用如下方法（或者的关系）
$users = DB::table('users')->where('votes', '>', 100)->orWhere('name', 'John')->get();

//查询创建时间是 2016 年 12 月 31 日的所有信息【前提 created_at 字段为 datetime 类型】
$users = DB::table('users')->whereDate('created_at', '2016-12-31')->get();

//查询 12 月新注册 users 的信息【前提 created_at 字段为 datetime 类型】
$users = DB::table('users')->whereMonth('created_at', '12')->get();

//查询 31 号这天新注册的 users 信息【前提 created_at 字段为 datetime 类型】
$users = DB::table('users')->whereDay('created_at', '31')->get();

//查询 2016 一整年的用户注册信息【前提 created_at 字段为 datetime 类型】
$users = DB::table('users')->whereYear('created_at', '2016')->get();
```

排序 orderBy

```php
//第一个参数为要进行排序的字段，第二个参数为排序方式
$users = DB::table('users')->orderBy('name', 'desc')->get();

//随机排序
$randomUser = DB::table('users')->inRandomOrder()->first();
```

分页 Limit

```php
//skip 的含义是跳过多少条，而 take 的含义是取走多少条，因此和 limit 10,5 的含义一致
$users = DB::table('users')->skip(10)->take(5)->get();

//你也可以采用下面的这种方式，结果同上
$users = DB::table('users')->offset(10)->limit(5)->get();
```

添加

```php
//类似于 ThinkPHP 的 insert 方法，将要添加的信息以一维数组的形式放入 insert 方法即可。
DB::table('users')->insert(['email' => 'john@example.com', 'votes' => 0]);
```

若你想要获取刚才添加成功的信息的 id，你可以直接用变量获取添加成功之后的结果

```
//直接变量获取
$id = DB::table('users')->insert(['email' => 'john@example.com', 'votes' => 0]);
```

修改

```
//同理，和 ThinkPHP 的 Update 方法类似，将要修改的数组直接放到 update 方法即可，记得指定 id 条件
DB::table('users')->where('id', 1)->update(['votes' => 1]);
```

删除

```
//删除所有信息【不推荐】
DB::table('users')->delete();

//附加条件的删除
DB::table('users')->where('votes', '>', 100)->delete();

//重置 id 的删除
DB::table('users')->truncate();
```

第五节 数据库迁移

数据库迁移，是 Laravel 的一个非常重要的特性，它能够让我们通过编写 php 代码而实现对数据库内数据表的创建、修改、重构等功能！

创建迁移文件

需要选择系统默认的结构 create_tableName_table ，中间 tableName 替换成你的数据表名称即可，迁移文件会在 database/migrations 目录生成。

php artisan make:migration create_users_table

书写迁移结构

这里书写的迁移结构，其实就是创建数据表的 SQL 语句，我们需要按照 Laravel 给我们准备的语法进行数据表的创建！

```
//迁移文件有两个方法，一个是 up，一个是 down；up 用于创建数据表结构，down 用于回滚数据库迁移
class CreateFlightsTable extends Migration
{
    /**
```

```
    * 运行数据库迁移
    *
    * @return void
    */
   public function up()
   {
         //这里是创建数据表的结构
       Schema::create('flights', function (Blueprint $table) {
           $table->increments('id');
           $table->string('name');
           $table->string('airline');
           $table->timestamps();
       });
   }

   /**
    * 回滚数据库迁移
    *
    * @return void
    */
   public function down()
   {
       Schema::drop('flights');
   }
}
```

书写创建结构时，请参考字段 部分的表格

运行迁移

```
//直接执行迁移
php artisan migrate
```

第六节数据填充

主要为你刚刚通过数据库迁移生成的空表进行测试数据的添加

创建填充文件

文件会在 database/seeds 目录中生成。

```
//需要采用 Laravel 要求的 TableNameTableSeeder 的命名格式
php artisan make:seeder UsersTableSeeder
```

书写填充内容

在生成的填充文件的 run 方法中，根据数据表的每一个字段，进行响应类型的随机数据创建即可

```
public function run()
    {
        DB::table('users')->insert([
            'name' => str_random(10),
            'email' => str_random(10).'@gmail.com',
            'password' => bcrypt('secret'),
        ]);
    }
```

调用填充文件

这一步如果不执行，你的迁移文件是不会生效的，因此，你需要在 database/seeds/DatabaseTableSeeder.php 文件对我们刚才书写完毕的填充文件进行调用。

```
public function run()
{
  //调用一下 users 表的填充类【将来若有其他填充文件，也在这里进行一一调用】
  $this->call(UsersTableSeeder::class);
}
```

运行填充

```
//运行该句会对所有填充文件进行执行
php artisan db:seed

//若你不想使用 6.3 步骤去对填充文件进行调用，你也可以使用这种指定填充文件的方式进行数据填充
php artisan db:seed --class=UsersTableSeeder
```

迁移和填充

如果你对数据表迁移文件进行了调整，修改，再运行迁移的时候，数据表的内容会被清空，此时，你可以采用如下方式在迁移的同时进行数据填充

```
php artisan migrate:refresh --seed
```

第三篇 web 前后端数据交互技术

第一章 HTTP 通讯协议

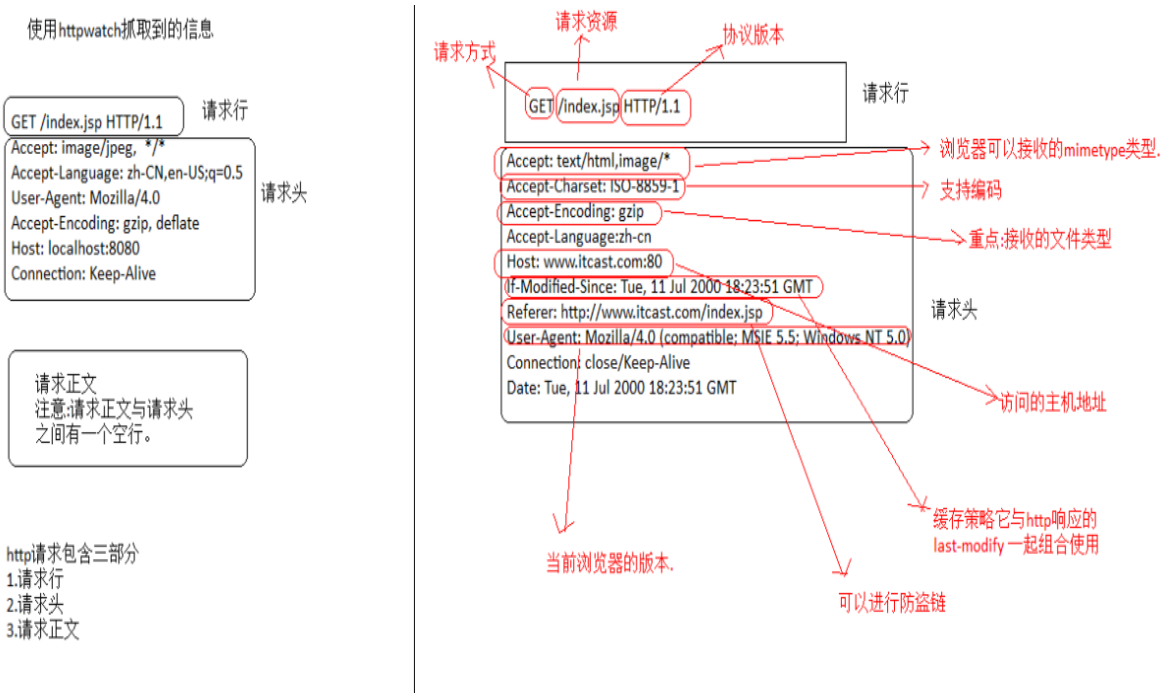
第一节 简介

协议是指计算机通信网络中两台计算机之间进行通信所必须共同遵守的规定或规则。

HTTP（超文本传输协议）是一种通信协议，它允许将超文本标记语言(HTML)文档从 Web 服务器传送到客户端的浏览器。

HTTP 消息结构

结构	说明
request line	请求行：第一行必须是请求行，用来说明请求类型、要访问的资源以及使用的 HTTP 版本。
header	请求头：用来说明服务器要使用的附加信息。
blank line	空白行：请求头部与请求体之间必须有一个空白行，必不可少
body	请求体：也叫请求正文，可以添加任意的其他数据



状态行	**说明**
Host	接受请求的服务器地址，可以是：IP：端口 或 域名
User-Agent	发送请求的应用程序名称（浏览器信息）
Connection	指定与连接相关的属性，如：Connection:Keep-Alive
Accept-Charset	客户端可以接受的编码格式
Accept-Encoding	客户端可以接受的数据压缩格式
Accept-Language	客户端可以接受的语言
referer	当前请求来自哪个链接（防盗连）
content-type	请求的文件类型
cookie	该网站相关的会话信息

第二节 特点

单向请求，HTTP 协议永远都是客户端发起请求，服务器回送响应。这样就限制了使用 HTTP 协议，无法实现在客户端没有发起请求的时候，服务器将消息推送给客户端。

第三节 工作流程

一次 HTTP 操作称为一个事务，其工作过程可分为四步：

- 首先客户机与服务器需要建立连接。
- 建立连接后，客户机发送一个请求给服务器，请求方式的格式为：统一资源标识符（URL）、协议版本号，客户机信息和可能的内容。
- 服务器接到请求后，给予相应的响应信息，其格式为一个状态行，包括信息的协议版本号、一个成功或错误的代码（status Code 状态码），后边服务器信息、实体信息和可能的内容。
- 客户端接收完服务器所返回的信息后，与服务器断开连接。

  如果在以上过程中的某一步出现错误，那么产生错误的信息将返回到客户端。

  对于用户来说，这些过程是由 HTTP 自己完成的，用户只要用鼠标点击，等待信息显示就可以了。

第四节 Status Code

Status Code 是 HTTP 协议状态码，当发起 http 请求后，服务器端会会返回不同的状态码，用以标示其工作状态：

1xx：信息，服务器收到请求，需要请求者继续执行操作

100 继续。客户端应继续其请求 101 切换协议。服务器根据客户端的请求切换协议。只能切换到更高级的协议，例如，切换到 HTTP 的新版本协议

2xx：成功，操作被成功接收并处理

200 请求成功。一般用于 GET 与 POST 请求 201 已创建。成功请求并创建了新的资源 202 已接受。已经接受请求，但未处理完成 203 非授权信息。请求成功。 204 无内容。服务器成功处理，但未返回内容。

3xx：重定向，需要进一步的操作以完成请求

300 请求的资源可在多处得到 301 删除请求数据 304 网页自请求者上次请求后再也没有更改过

4xx：客户端错误，请求包含语法错误或无法完成请求

400 错误请求，如语法错误 401 请求授权失败 403 请求不允许 404 没有发现文件、查询或 URl

5xx：服务器错误，服务器在处理请求的过程中发生了错误

500 服务器产生内部错误 502 服务器暂时不可用，有时是为了防止发生系统过载 503 服务器过载或暂停维修 505 服务器不支持或拒绝支请求头中指定的 HTTP 版本

第二章 AJAX 的实现

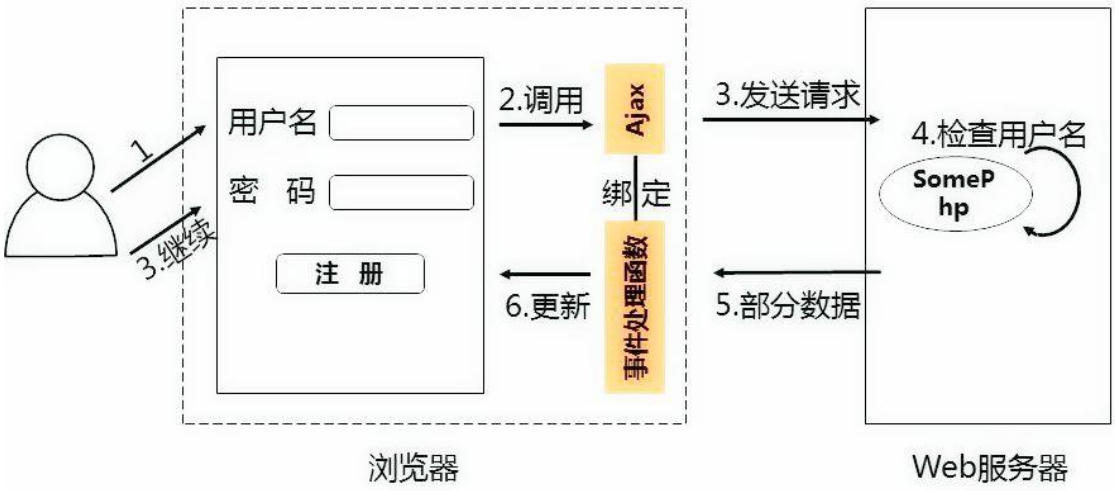
第一节 简介

synchronous：同步

Asynchronous：异步

AJAX（Asynchronous JavaScript and XML） 是一种在无需重新加载整个页面，仅更新部分网页的**技术**。

AJAX 通过**异步**方式与服务器进行少量数据交换。



优点

- 页面无需刷新，用户的体验非常好。
- 使用异步方式与服务器通信，具有更加迅速的响应能力。
- 可以把一些服务器负担的工作转嫁到客户端，减轻服务器和宽带的负担，节约空间和宽带租用成本。ajax 的原则是“按需取数据”，可以最大程度的减少冗余请求和响应对服务器造成的负担。
- 基于标准化、并被广泛支持的技术，不需要下载插件或者小程序。

缺点

- ajax 不支持浏览器 back 按钮
- ajax 暴露了与服务器交互的细节，存在安全隐患
- 对搜索引擎的支持比较弱
- 破坏了程序的异常机制
- 不容易调试

第二节 AJAX 工作原理

既然 Ajax 能够发送 HTTP 请求，整个请求的过程就必须要符合 HTTP 协议。Ajax 发送 HTTP 请求时需要指定以下参数：

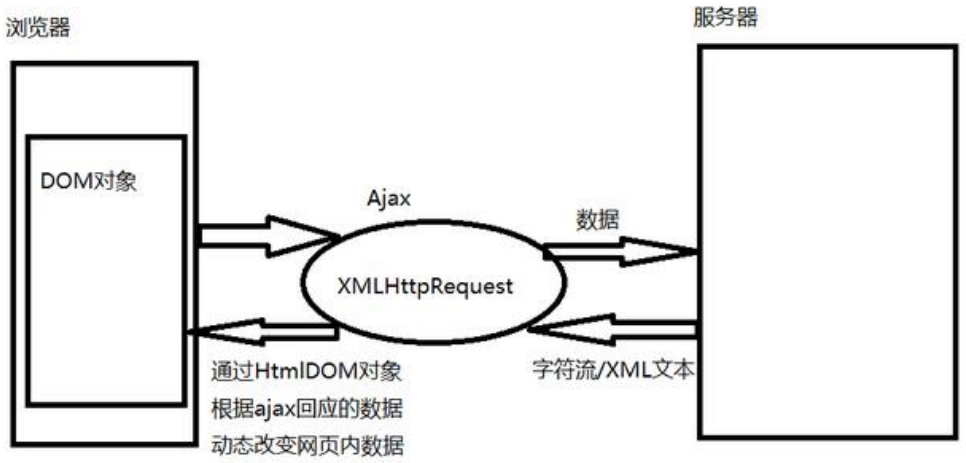
- 要请求的资源即 URL 地址。
- 指定请求的方式，常用的有 get 和 post。
- 指定需要发送给服务器的数据，以“名=值&名=值”的方式书写。
- 指定可以接收的数据类型，即告诉服务器可以回传的内容类型是什么，包括：HTML、JS 脚本、JSON、XML 等，常用的是文本 txt 和 JSON。

以上参数就包含了 Ajax 发送 HTTP 请求时请求头中的信息，但服务器响应后会回传结果，因此 Ajax 还必须有能够接收服务器回传的结果的参数。

匿名函数 success:function(re){ }。re 变量用于接收服务器回传的结果。

Ajax 的基本格式如下：

```
$.ajax({
        url:'请求的资源',
        type:'请求方式 get|post',
        data:'发送数据 名=值&名=值',
        dataType:'回传值的类型',
        success:function(re){ re 接收返回值

        }
})
```



XML

可扩展标记语言(eXtensible Markup Language)，是一种用于标记电子文件使其具有结构性的标记语言。

XML 和 HTML 的区别：

XML	HTML
所有的标记必须成对出现	不是所有的都需要成对出现
区分大小写	不区分大小写

由于 XML 结构过于复杂，逐渐被 JSON 格式所取代。

XMLHttpRequest 对象是 window 的一个方法（构造函数），用于在后台与服务器交换数据，所有现代的浏览器都支持该对象，它能够实现：

在后台（异步方式）向服务器发送数据

在页面已加载后从服务器 请求/接收 数据，在不重新加载页面的情况下更新网页

尽管名为 XMLHttpRequest，它并不限于和 XML 文档一起使用：它可以接收任何形式的文本文档

AJAX 通过操作 XMLHttpRequest 对象，与服务器异步交互数据，要完整实现一个 AJAX 异步调用和局部刷新,通常需要以下几个步骤:

- 创建 XMLHttpRequest 对象
- 创建一个新的 HTTP 请求,并指定该 HTTP 请求的方法、URL 及验证信息.
- 发送数据
- 接收数据

第三章 JSONP

Jsonp(JSON with Padding) 可以**异步、跨域**进行数据请求。

ajax 由于受到同源策略的限制，不能进行跨域请求，但在某些情况下，又需要进行跨域请求数据，就诞生了 jsonp 技术，严格说，jsonp 不是技术，是 json 的一种"使用模式"。

实现 jsonp 跨域请求，需要前后两端进行配合。

第一节 hello world

前端

html 页面中的，有三种标签，可以跨域（get）请求资源，分别是：link 、img、script，试着用 script 发送一个异步请求：

域：127.0.0.1:8888

```
<script>
   function play(obj){
      alert(obj.msg);
   }
</script>

<script src="http://127.0.0.1:8000/doAction.php?callback=play"></script>
```

- 声明了一个 play 函数
- 从 127.0.0.1:8888 域 向 127.0.0.1:8000 域的 doAction.php 接口发送了 get 请求，并带有 cb=play 的参数

后端

用 php 创建一个后端接口：doAction.php

域：127.0.0.1:8000

```
<?php
        $fn=$_GET["callback"];
        $json='{"msg":"hello world"}';//json 格式的数据
        echo "{$fn}({$json})";
?>
```

结果

play 函数被调用，弹出”hello world“。

第二节 原理

以上就是一个完整的 jsonp，在这个案例中：

- 前端将要调用的函数名 play，作为参数传递给后台
- 后台接收参数，返回一个字符串，字符串的的内容就是调用 play 函数的语句，并且带有 json 格式数据的参数
- 前端接收到后端返回的字符串，既然是 script 标签发出的请求，接收回来的字符串就会被当做命令执行，该命令调用已经定义好的 play 函数，同时接收到了传递过来的 json 格式数据。

以上步骤，是实现一个 jsonp 的标准流程和基本原理。在这个过程中，如果前后端约定好了回调函数的名称，那 cb 这一传参步骤可以省略。

既然可以通过 get 传参，那么，前端就可以将其他请求信息传递到后端，完成更复杂的异步请求数据的操作。

注意：

后端只能以 json 格式作为参数（jsonp 也是因此得名的）。

第四章 iframe 页面异步通信

第一节 简介

iframe 是 HTML 的一个标签，它的作用就是在页面中再开放出一个窗口，且这个窗口能够重新显示一个页面。

所以 iframe 称作页面内框架，可以理解为页面中的页面。

第二节 JS 操作 iframe 标签

Js 操作 iframe 标签过程分为 2 步：

- 找到 iframe 标签
- 操作 iframe 标签

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Document</title>
  </head>
  <body>
    <h1>发布聊天内容：</h1>
    <iframe id="fr1" style="height:50px;"></iframe><br>
    姓名：<input type="text" id="namer" name="namer"><br>
    内容：<textarea id="content" name="content" style="height:50px;width:300px;"></textare
a><br>
    <input type="button" value="发送" onclick="sendMessage();" />
  </body>
  <script type="text/javascript">
    function sendMessage() {
      //获取两个输入框中的值
      var n = document.getElementById('namer').value;
      var c = document.getElementById('content').value;
      //拼接 url 地址
      var urlStr = "do.php?namer=" + n + "&content=" + c;
      //把 urlStr 值赋给 iframe 的 src 属性
      document.getElementById('fr1').src = urlStr;
    }
  </script>
</html>
```

编写 Js 中调用的 php 文件，该 php 文件主要功能是将输入的内容打印打印出来，代码如下：

```php
<?php
$namer = $_GET['namer'];
$content = $_GET['content'];
var_dump($namer,$content);
```

在点击发送数据的时候并没有刷新整个页面，而是借助于 iframe 标签实现了页面的异步刷新。当然，我们也可以在点击发送的时候向服务器发送数据，并借助 iframe 实现页面异步刷新。

第三节 jQuery 操作 iframe 标签

同 Js 一样，jQuery 操作 iframe 标签过程也分为 2 步：

- 找到 iframe 标签
- 操作 iframe 标签

```html
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Document</title>
    <script src="https://code.jquery.com/jquery-3.4.0.min.js"></script>
  </head>
  <body>
    <h1>发布聊天内容：</h1>
    <iframe id="fr1" style="height:50px;"></iframe><br>
    姓名：<input type="text" id="namer" name="namer"><br>
    内容：<textarea id="content" name="content" style="height:200px;width:500px;"></textarea><br>
    <input type="button" value="发送" onclick="sendMessage();" />
  </body>
  <script type="text/javascript">
    function sendMessage () {
      //获取两个输入框中的值
      var n = $("[name='namer']").val();
      var c = $("[name='content']").val();
      //拼接 url 地址
      var urlStr = "do.php?namer=" + n + "&content=" + c;
      //把 urlStr 值赋给 iframe 的 src 属性
      $("#fr1").attr('src', urlStr)
    }
```

```
    </script>
</html>
```

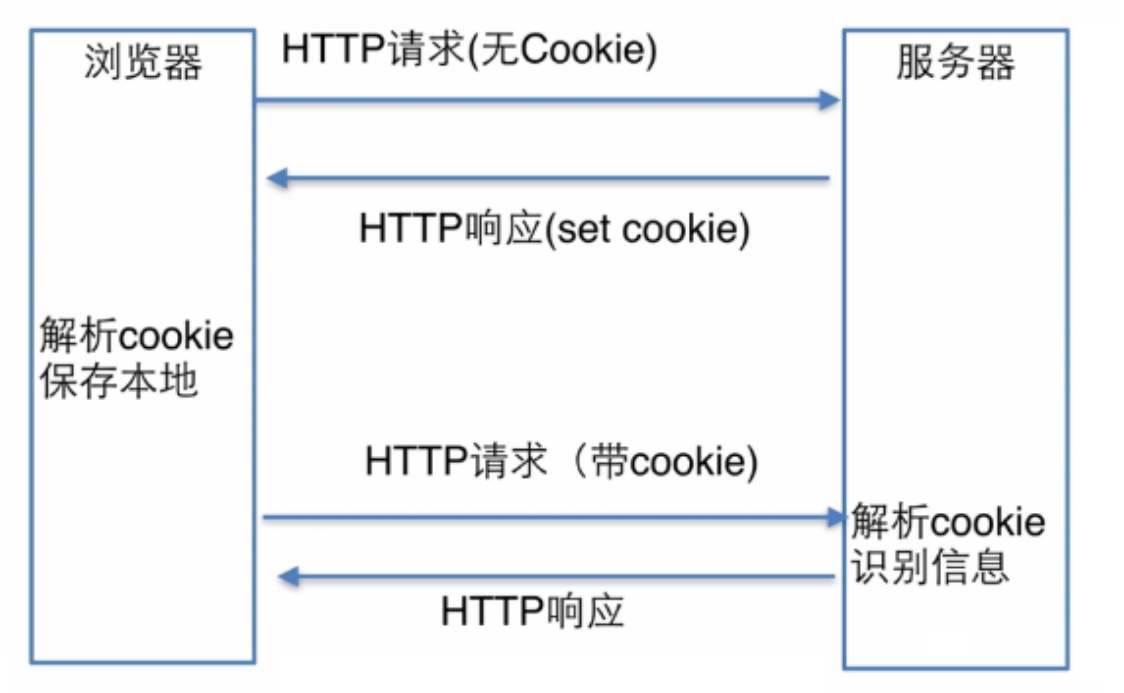
然后编写 Js 中调用的 php 文件，该 php 文件主要功能是将输入的内容插入到服务器的数据库中，并返会插入成功的条目数。

```
<?php
$namer = $_GET['namer'];
$content = $_GET['content'];
$pdo = new PDO('mysql:host=localhost;dbname=php','root','123456');
$sql = "insert into token(namer,content,pubtime) value('".$namer."','".$content."',".time().")";
$re = $pdo->exec($sql);
var_dump($re);
```

第五章 页面性能优化技术

会话控制是一种跟踪用户的通信方式

例如：当一个用户在请求一个页面后，再次请求这个页面，网站是无法知道这个用户刚才是否曾经来访问过。由此我们就会觉得奇怪，平时我们在电商网站购物时，只要我们在这个站点内，不论我们怎么跳转页面，网站总会记得我是谁，这是怎么做到的呢？这就是运用了 HTTP 会话控制。在网站中跟踪一个变量，通过对变量的跟踪，使多个请求事物之间建立联系，根据授权和用户身份显示不同的内容、不同页面。



Cookie 是一种会话变量。会话变量是用来存放数据的一个容器，这个容器对应的就是内存空间。

第一节 Cookie 的创建、使用和销毁

cookie 是在服务器端创建，并写回到客户端浏览器

浏览器接到指令则在本地临时文件夹中创建了一个 cookie 文件，其中保存了你的 cookie 内容

客户端浏览器每次访问网站时，都会检测是否有该网站的 cookie 信息，如果有的话，也会同时发送过去。

注意：

cookie 内容的存储是**键/值**对的方式，**键和值都只能是字符串**。

函数	功能
setcookie(key , value ,有效期)	设置会话 cookie 参数

```
//定义 cookie
//setcookie(键, 值, 有效期(秒))
setcookie("name","zhangsan",time()+1000); //如果不设置有效期，关闭浏览器就会消失
setcookie("pwd","123",time()+1000);

//删除 cookie（设定过期时间，使失效）
setcookie("name","",time()-1);
setcookie('age',null,time()-1);
setcookie('sex',"",time()-1);
```

创建

在 php 中使用 setcookie 方法来创建 cookie，例如：

```
<?php
$re = setcookie ( 'username', 'xiaozhang', time () + 30 * 60, '/' );
var_dump ( $re );
```

使用

PHP 中使用$_COOKIE 全局变量来获取 Cookie，例如：

```
<?php
$re = $_COOKIE ['username'];
var_dump($re);
```

销毁

PHP 中没有直接销毁的方法，通常会将 Cookie 的过期时间重新设置一下。

```
<?php
$re = setcookie ( 'username', null, time () + 1, '/' );
var_dump($re);
```

第二节 页面性能优化

为了减少服务器的压力，并使服务器尽快响应客户端请求的页面，需要对页面进行优化。通常会从如下两个方面入手对页面进行优化：

- 减少 HTTP 请求次数；
- 静态资源和动态程序分开，即通过不同的域名访问。
-

第三节 减少 HTTP 请求次数

减少 HTTP 请求次数的方法常见的有如下几种：

- 制作图片地图，页面中出现了网页中所有的背景图片，允许一个图片关联多个 url，即将多个图片合并为一个图片。
- 使用雪碧图，简单的说就是 CSS 中的 background-position 来定位图片中的某一具体部分，这样就可以把多张小图片合并为一张大图片。
- 合并 JS 和 CSS 文件，直接减少 HTTP 的请求次数。
- 充分利用浏览器缓存，如果图片或者脚本，样式文件内容比较固定，不经常被修改，那么，尽可能利用缓存技术，减少 HTTP 请求次数或文件下载次数。
-

第四节 静态资源与动态程序分开

动静分离是让动态网站里的动态网页根据一定规则把不变的资源和经常变的资源区分开来，动静资源做好了拆分以后，我们就可以根据静态资源的特点将其做缓存操作，这就是网站静态化处理的核心思路。

第六章 Socket 通信

第一节 简介

网络上的两个程序通过一个双向的通信连接实现数据的交换，这个连接的一端称为一个 socket。socket（套接字）既不是程序，也不是协议，其只是操作系统提供的通信层的一组抽象 API。

通信需要服务端和客户端组成：

服务端：

1. 使用 php 初始化 socket 然后绑定一个端口，对端口进行监听。
2. 调用 accept 阻塞，等待客户端连接。

客户端：

3. 初始化一个 socket，
4. 连接服务器，如果连接成功，客户端与服务器端建立连接。
5. 发送数据请求，服务器端接收请求并处理请求，把回应数据发送给客户端
6. 读取数据
7. 关闭连接，一次交互结束。

客户端------服务端是可以彼此交互的应用程序。客户端和服务端之间的交互需要连接。Socket 编程负责的就是为应用程序之间建立可进行交互的连接。

Socket 通信是双向的、长连接，它提供了网络通信的一些接口，我们基于这些接口可以控制数据的发送。

在一般情况下，服务器多用 PHP 来开发 Socket 通信，客户端多用 Javascript 或者 JQuery 来开发 Socket 通信。

服务器端函数	说明
socket_create(*net*,stream,$protocol)	创建 Socket，参数： *net：网络协议* <*br*/>stream：socket 流 $protocol：协议
socket_bind()	绑定 IP 和端口
socket_listen()	监听端口
socket_accept()	接受一个 socket 连接，与客户端建立连接

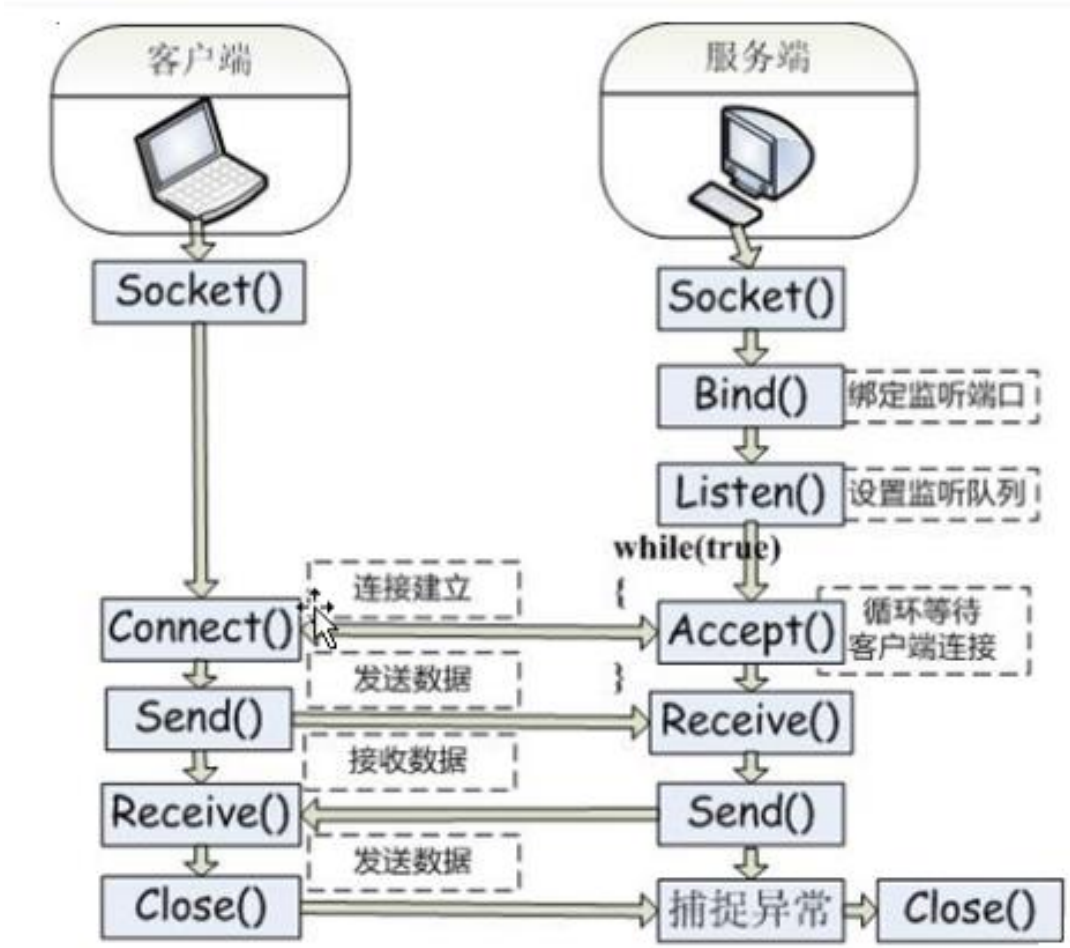
socket_read()	读取客户端发送过来的数据
socket_write()	将数据写到 socket 缓存向客户端发送
socket_close()	关闭 Socket 连接
客户端函数	说明
Websocket(wsurl)	向服务器发送连接
send(string)	向服务器发送数据
onmessage 事件	监听服务器发送过来的数据
onclose 事件	监听服务器开关状态

第二节 Socket 通信的工作原理

根据连接启动的方式以及本地套接字要连接的目标，套接字之间的连接过程可以分为三个步骤：

- 服务器监听：是服务器端套接字并不定位具体的客户端套接字，而是处于等待连接的状态，实时监控网络状态。
- 客户端请求：是指由客户端的套接字提出连接请求，要连接的目标是服务器端的套接字。为此，客户端的套接字必须首先描述它要连接的服务器的套接字，指出服务器端套接字的地址和端口号，然后就向服务器端套接字提出连接请求。
- 连接确认：是指当服务器端套接字监听到或者说接收到客户端套接字的连接请求，它就响应客户端套接字的请求，建立一个新的线程，把服务器端套接字的描述发给客户端，一旦客户端确认了此描述，连接就建立好了。而服务器端套接字继续处于监听状态，继续接收其他客户端套接字的连接请求。

socket 通讯的工作原理可以参考下面的流程图：



服务器端

```
$ip = '127.0.0.1';
$port = 8001;//端口

//创建 socket
$sock = socket_create(AF_INET,SOCK_STREAM,SOL_TCP）；
//参数：
//AF_INET： IPv4 网络协议。TCP 和 UDP 都可使用此协议。一般都用这个
//SOCK_STREAM:是基于 TCP 的，数据传输比较有保障
//SOL_TCP:TCP 协议

//绑定 ip 和端口
$ret = socket_bind($sock,$ip,$port);
```

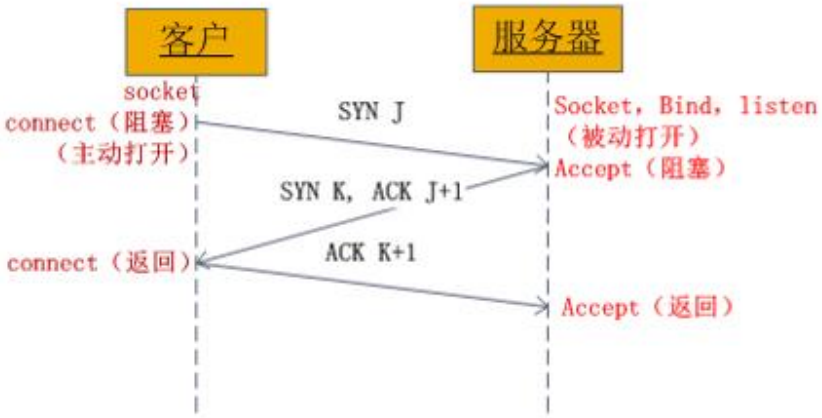
```
//监听
$ret = socket_listen($sock,4);  //最大监听套接字个数

$count = 0;
do {
   if (($msgsock = socket_accept($sock)) < 0) {
      echo "失败: ".socket_strerror($msgsock) ;
      break;
   } else {
      //发到客户端
      $msg ="测试成功！ ";
      socket_write($msgsock, $msg, strlen($msg));
      echo "测试成功了啊";

      $buf = socket_read($msgsock,8192);
      echo "收到的信息:{$buf}";

      if(++$count >= 5){
         break;
      }
   }
   socket_close($msgsock);
} while (true);

socket_close($sock);
```

客户端

```
$(function() {
        //连接 Socket 的 URL 地址
        var wsurl = "ws://127.0.0.1:8001/socket_server.php";
        var websocket; //用于存放客户端创建的 Socket 对象
        if (window.WebSocket) {
                websocket = new WebSocket(wsurl);
                websocket.onopen = function(event) {//onopen 事件，连接成功
                        $(body).append("<p>conneted success!</p>");
                }
                websocket.onmessage = function(event) {//onmessage 事件，接收消息，显示
在页面上
        var msg = JSON.parse(event.data);
        console.log(msg);
      }
         }
})
```

第三节 Socket 通信实现聊天室

PHP 实现 Socket 服务端



Js 实现 Socket 客户端

```
var ws = new WebSocket("ws://localhost:8080/msg");ws.onopen = function(evt)
{         console.log("Connection open ...");         ws.send("Hello WebSockets!");};ws.onmessage
= function(evt)
{         console.log("Received Message: " + evt.data);    ws.close();};ws.onclose = function(evt)
{
console.log("Connection closed.");
};
```

第四篇 响应式开发技术

第一章 Bootstrap 介绍

第一节 Bootstrap 概述

Bootstrap 是由 Twiter 公司(全球最大的微博)的两名技术工程师研发的一个基于 HTML、CSS、JavScript 的开源框架。该框架代码简洁、视觉优美，可用于快速、简单地构建基于 PC 及移动端设备的 Web 页面需求。 2010 年 6 月，Twiter 内部的工程师为了解决前端开发任务中的协作统一问题。经历各种方案后，Bootstrap 最终被确定下来，并于 2010 年 8 月发布。经过很长时间的迭代升级，由最初的 CSS 驱动项目发展成为内置很多 JavScript 插件和图标的多功能 Web 前端的开源框架。Bootstrap 最为重要的部分就是它的响应式布局，通过这种布局可以兼容 PC 端、PAD 以及手机移动端的页面访问。 Bootstrap 下载及演示： 国内文档翻译官网：htp:/w.botcs.com 瓢城 Web 俱乐部官网：htp:/w.ycku.com

第二节 Bootstrap 特点

Bootstrap 非常流行，得益于它非常实用的功能和特点。主要核心功能特点如下：

- 跨设备、跨浏览器 可以兼容所有现代浏览器，包括比较诟病的 IE7、8。当然，本课程不再考虑 IE9 以下浏览器。
- 响应式布局 不但可以支持 PC 端的各种分辨率的显示，还支持移动端 PAD、手机等屏幕的响应式切换显示。
- 提供的全面的组件 Bootstrap 提供了实用性很强的组件，包括：导航、标签、工具条、按钮等一系列组件，方便开发者调用。
- 内置 jQuery 插件 Bootstrap 提供了很多实用性的 jquery 插件，这些插件方便开发者实现 Web 中各种常规特效。
- 支持 HTML5、CSS3 HTML5 语义化标签和 CSS3 属性，都得到很好的支持。

- 支持 LESS 动态样式 LESS 使用变量、嵌套、操作混合编码，编写更快、更灵活的 CS。它和 Bootstrap 能很好的配合开发。

第三节 Bootstrap 结构

首先，想要了解 Bootstrap 的文档结构，需要在官网先把它下载到本地。Bootstrap 下载地址如下： Bootstrap 下载地址：htp:/v3.botcs.com/ (选择生产环境即可，v3.4)
解压后，目录呈现这样的结构：

```
Bootstrap/
├── css/
│   ├── Bootstrap .css
│   ├── Bootstrap .cs.map
│   ├── Bootstrap .min.css
│   ├── Bootstrap-them.css
│   ├── Bootstrap-them.cs.map
│   └── Bootstrap-them.min.css
├── js/
│   ├── Bootstrap.js
│   └── Bootstrap.min.js
└── fonts/
├── glyphicons-halfings-regular.eot
├── glyphicons-halfings-regular.svg
├── glyphicons-halfings-regular.tf
├── glyphicons-halfings-regular.wof
└── glyphicons-halfings-regular.wof2
```

主要分为三大核心目录：cs(样式)、js(脚本)、fonts(字体)。

- css 目录中有四个 css 后缀的文件，其中包含 min 字样的，是压缩版本，一般使用这个；不包含的属于没有压缩的，可以学习了解 css 代码的文件；而 map 后缀的文件则是 css 源码映射表，在一些特定的浏览器工具中使用。
- js 目录包含两个文件，是未压缩和压缩的 js 文件。
- fonts 目录包含了不同后缀的字体文件。

第四节 创建第一个页面

我们创建一个 HTML5 的页面，并且将 Bootstrap 的核心文件引入，然后测试是否正常显示。

第一个 Bootstrap

```
<!DOCTYPE html>
<htmlang="zh-cn"><head>
<meta charset="UTF-8">
<tile>Bootstrap 介绍</tile>
<link rel="styleshet" href="cs/Bootstrap.min.cs">
</head>
<body>
<buton class="btn btn-info">Bootstrap</buton>
<script src="js/jquery.min.js"></script>
<script src="js/Bootstrap.min.js"></script>
</body>
</html>
```

第二章 Bootstrap 栅格布局

第一节 栅格布局简介

Bootstrap 内置了一套响应式、移动设备优先的流式栅格系统，随着屏幕设备或视口（viewport）尺寸的增加，系统会自动分为最多 12 列。它包含了易于使用的预定义 classe，还有强大的 mixin 用于生成更具语义的布局。

容器：“行（row）”必须包含在 .container （固定宽度）或 .container-fluid （100%宽度）中，以便为其赋予合适的排列（aligment）和内补（padding）。

rem：实际上是设置列的高度的属性，rem 的值是整数，代表的高度是 rem 值*字体像素大小。

工作原理如下：

- “行（row）”必须包含在 .container （固定宽度）或 .container-fluid （100%宽度）中，以便为其赋予合适的排列（aligment）和内补（padding）。
- 通过“行（row）”在水平方向创建一组“列（column）”。
- 你的内容应当放置于“列（column）”内，并且，只有“列（column）”可以作为行（row）”的直接子元素。
- 类似 .row 和 .col-xs-4 这种预定义的类，可以用来快速创建栅格布局。Bootstrap 源码中定义的 mixin 也可以用来创建语义化的布局。
- 通过为“列（column）”设置 padding 属性，从而创建列与列之间的间隔（gutter）。通过为 .row 元素设置负值 margin 从而抵消掉为 .container 元素设置的 padding，也就间接为“行（row）”所包含的“列 column）”抵消掉了 padding。
- 负值的 margin 就是下面的示例为什么是向外突出的原因。在栅格列中的内容排成一行。
- 栅格系统中的列是通过指定 1 到 12 的值来表示其跨越的范围。例如，三个等宽的列可以使用三个 .col-xs-4 来创建。
- 如果一“行（row）”中包含了的“列（column）”大于 12，多余的“列（column）”所在的元素将被作为一个整体另起一行排列。
- 栅格类适用于与屏幕宽度大于或等于分界点大小的设备， 并且针对小屏幕设备覆盖栅格类。 因此，在元素上应用任何 .col-md-* 栅格类适用于与屏幕宽度

大于或等于分界点大小的设备，并且针对小屏幕设备覆盖栅格类。因此，在元素上应用任何 .col-lg-*不存在， 也影响大屏幕设备。

- 在前端开发中，使用框架会带来很舒畅的开发过程，但某些情况下，使用框架可能会导致性能问题，例如，开发一个网页，也用到了 BootStrap 框架的一些东西，但使用率仅仅有百分之几，这种情况下，浏览器在加载这个网页的时候，会将 BootStrap 剩下没有用到的百分之九十多也会加载，存在了一些性能浪费。这种情况下我们就需要对框架进行定制或优化，取出我们用到的代码，或者删除掉无用的代码，这需要对源码有相应的熟悉才能做到。

第二节 Bootstrap 相应设备类型

栅格系统最外层区分了四种宽度的浏览器：超小屏(<768px)、小屏(>=768px)、中屏(>=992px)和大屏>=1200px)。而内层.container 容器的自适应宽度为：自动、750px、970px 和 1170px。自动的意思为，如果你是手机屏幕，则全面独占一行显示。

```
<div class="container">
   <div class="row">
      <div class="col-lg-3 col-md-4 col-sm-6 col-xs-12 a">4</div>
      <div class="col-lg-3 col-md-4 col-sm-6 col-xs-12 a">4</div>
      <div class="col-lg-3 col-md-4 col-sm-6 col-xs-12 a">4</div>
      <div class="col-lg-3 col-md-4 col-sm-6 col-xs-12 a">4</div>
      <div class="col-lg-3 col-md-4 col-sm-6 col-xs-12 a">4</div>
      <div class="col-lg-3 col-md-4 col-sm-6 col-xs-12 a">4</div>
      <div class="col-lg-3 col-md-4 col-sm-6 col-xs-12 a">4</div>
      <div class="col-lg-3 col-md-4 col-sm-6 col-xs-12 a">4</div>
      <div class="col-lg-3 col-md-4 col-sm-6 col-xs-12 a">4</div>
      <div class="col-lg-3 col-md-4 col-sm-6 col-xs-12 a">4</div>
      <div class="col-lg-3 col-md-4 col-sm-6 col-xs-12 a">4</div>
      <div class="col-lg-3 col-md-4 col-sm-6 col-xs-12 a">4</div>
   </div>
</div>
```

为了显示明显的 CSS

```
.a {
height: 100px;
background-color: #eee;
border:1px solid #ccc;
}
```

有时我们可以设置列偏移，让中间保持空隙

```
<div class="container">
  <div class="row">
    <div class="col-md-8 a">8</div>
    <div class="col-md-3 col-md-offset-1 a">3</div>
  </div>
</div>
```

也可以嵌套，嵌满也是 12 列

```
<div class="container">
  <div class="row">
    <div class="col-md-9 a">
      <div class="col-md-8 a">1-8</div>
      <div class="col-md-4 a">9-12</div>
    </div>
    <div class="col-md-3 a">
      11-12
    </div>
  </div>
</div>
```

可以把两个列交换位置，push 向左移动，pull 向右移动

```
<div class="container">
  <div class="row">
    <div class="col-md-9 col-md-push-3 a">9</div>
    <div class="col-md-3 col-md-pull-9 a">3</div>
  </div>
</div>
```

移动设备优先

```
<meta name="viewport" content="width=device-width, initial-scale=1,
maximum-scale=1, user-scalable=no">
```

布局容器

Bootstrap 需要为页面内容和栅格系统包裹一个.container 容器。由于 padding 等属性的原因，这两种容器类不能相互嵌套。

```
//固定宽度
<div class="container">
...
</div>
//100%宽度
<div class="container-fluid">
...
</div>
```

第三节 栅格基本布局

Bootstrap 中的基本网格布局的代码如下：

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <link rel="stylesheet" href="css/bootstrap.css" type="text/css">
    <style>
      .row>div {
        background: pink;
        border: 1px solid rgba(86, 61, 124, 0.2);
      }

      .title {
        font-size: 30px;
        color: red;
        text-align: center;
      }
    </style>
  </head>
  <body>
    <p class="title">栅格最基本布局</p>
    <!--基本的方格布局-->
    <div class="row">
      <div class="col-2">col-2</div>
      <div class="col-2">col-2</div>
      <div class="col-2">col-2</div>
    </div>
```

栅栏布局中有一个特殊的类 col-auto，这个类会根据内容的大小自动的扩充列数，而普通的如果内容的长度超过列数就会加行。

第四节 栅格水平和垂直布局

水平布局是通过在行的 div 标签中利用 justify-content-类型来实现的，如下所示：

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <link rel="stylesheet" href="css/bootstrap.css" type="text/css">
```

```
    <style>
      .row>div {
        background: pink;
        border: 1px solid rgba(86, 61, 124, 0.2);
      }

      .title {
        font-size: 30px;
        color: red;
        text-align: center;
      }
    </style>
  </head>
  <body>
    <p class="title">水平布局</p>
    <!--水平的排列方式-->
    <div class="row justify-content-start">
      <div class="col-2">start</div>
      <div class="col-2">start</div>
      <div class="col-2">start</div>
    </div>
    <div class="row justify-content-center">
      <div class="col-2">center</div>
      <div class="col-2">center</div>
      <div class="col-2">center</div>
    </div>
    <div class="row justify-content-end">
      <div class="col-2">end</div>
      <div class="col-2">end</div>
      <div class="col-2">end</div>
    </div>
    <div class="row justify-content-around">
      <div class="col-2">around</div>
      <div class="col-2">around</div>
      <div class="col-2">around</div>
    </div>
    <div class="row justify-content-between">
      <div class="col-2">between</div>
      <div class="col-2">between</div>
      <div class="col-2">between</div>
    </div>
  </body>
</html>
```

垂直布局和水平布局类似，利用 align-items-类型来实现的垂直布局，但是垂直布局中只有 start、center、end 三种。

第五节 栅格排序和偏移

通过 order-数值这个类进行网格的位置排列，代码如下：

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <link rel="stylesheet" href="css/bootstrap.css" type="text/css">
    <style>
      .row>div {
        background: pink;
        border: 1px solid rgba(86, 61, 124, 0.2);
      }
    </style>
  </head>
  <body>
    <div class="row">
      <div class="col-3 order-0">我是老大</div>
      <div class="col-3 order-2">我是老二</div>
      <div class="col-3 order-1">我是老三</div>
    </div>
  </body>
</html>
```

列偏移可以通过 offset-数字这个类进行网格位置的偏移。

Bootstrap 还支持列嵌套，有了列嵌套，栅格布局显得非常的灵活，示例如下：

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <link rel="stylesheet" href="css/bootstrap.css" type="text/css">
    <style>
      .row>div {
        background: pink;
        border: 1px solid rgba(86, 61, 124, 0.2);
      }
    </style>
  </head>
  <body>
    <div class="row">
      <div class="col-11">外层
        <div class="row">
          <div class="col-6">内层 1</div>
          <div class="col-6">内层 2</div>
        </div>
```

```
            <div class="row">
                <div class="col-3">最内 1</div>
                <div class="col-4">最内 2</div>
            </div>
        </div>
    </div>
  </body>
</html>
```

第三章 排版样式

第一节 页面主体

Bootstrap 将全局 font-size 设置为 14px，line-height 行高设置为 1.428(即 20px)；<p>段落元素被设置等于 1/2 行高(即 10px)；颜色被设置为#333。 创建包含段落突出的文本

```
<p>Bootstrap 框架</p>
<p class="lead">Bootstrap 框架</p>
<p>Bootstrap 框架</p>
<p>Bootstrap 框架</p>
<p>Bootstrap 框架</p>
```

第二节 标题

从 h1 到 h6

```
<h1>Bootstrap 框架</h1> /36px
<h2>Bootstrap 框架</h2> /30px
<h3>Bootstrap 框架</h3> /24px
<h4>Bootstrap 框架</h4> /18px
<h5>Bootstrap 框架</h5> /14px
<h6>Bootstrap 框架</h6> /12px
```

Bootstrap 分别对 h1 ~ h6 进行了 CSS 样式的重构，并且还支持普通内联元素定义 class=(.h1 ~ h6)来实现相同的功能。 内联元素使用标题字体：

Bootstrap

在 h1 ~ h6 元素之间，还可以嵌入一个 smal 元素作为副标题，在标题元素内插入 small 元素

```
<h1>Bootstrap 框架 <small>Bootstrap 小标题</small></h1>
<h2>Bootstrap 框架 <small>Bootstrap 小标题</small></h2>
<h3>Bootstrap 框架 <small>Bootstrap 小标题</small></h3>
<h4>Bootstrap 框架 <small>Bootstrap 小标题</small></h4>
<h5>Bootstrap 框架 <small>Bootstrap 小标题</small></h5>
<h6>Bootstrap 框架 <small>Bootstrap 小标题</small></h6>
```

h1 ~ h3 下的 smal 为 23.4px、19.5px、15.6px；h4 ~ h6 下 smal 元素的大小只占父元素的 75% ,分别为：13.5px、10.5px、9px。在 h1 ~ h6 下的 smal 样式也进行了改变，颜色变成淡灰色：#7，行高为 1，粗度 为 40。

第三节 内联文本元素

添加标记，<mark>元素或.mark 类

```
<p>Bootstrap<mark>框架</mark></p>
```

各种加线条的文本

```
<del>Bootstrap 框架</del> /删除的文本
<s>Bootstrap 框架</s> /无用的文本
<ins>Bootstrap 框架</ins> /插入的文本
<u>Bootstrap 框架</u> /效果同上，下划线文本
```

各种强调的文本

```
<small>Bootstrap 框架</small> /标准字号的 85%
<strong>Bootstrap 框架</strong> /加粗 70
<em>Bootstrap 框架</em> /倾斜
```

第四节 对齐

设置文本对齐

```
<p class="text-left">Bootstrap 框架</p> /居左
<p class="text-center">Bootstrap 框架</p> /居中
<p class="text-right">Bootstrap 框架</p> /居右
<p class="text-justify">Bootstrap 框架</p> /两端对齐，支持度不佳
<p class="text-nowrap">Bootstrap 框架</p> /不换行
```

第五节 大小写

设置英文本大小写

```
<p class="text-lowercase">Bootstrap 框架</p> /小写
<p class="text-upercase">Bootstrap 框架</p> /大写
<p class="text-capitalize">Bootstrap 框架</p> /首字母大写
```

第六节 缩略语

缩略语

```
Bootstrap<abr tile="Bootstrap" class="intialism">框架</abr>
```

第七节 地址文本

设置地址，去掉了倾斜，设置了行高，底部 20px

```
<adres>
        <strong>Twiter, Inc.</strong><br>
        795 Folsom Ave, Suite 60<br>
        San Francisco, CA 94107<br>
        <abr tile="Phone">P:</abr>
        (123) 456-7890
</adres>
```

第八节 引用文本

默认样式引用，增加了做边线，设定了字体大小和内外边距

```
<blockquote>Bootstrap 框架</blockquote>
```

反向

```
<blockquote class="blockquote-revrse ">Bootstrap 框架</blockquote>
```

第九节 列表排版

移出默认样式

```
<ul class="list-unstyled">
   <li>Bootstrap 框架</li>
   <li>Bootstrap 框架</li>
   <li>Bootstrap 框架</li>
   <li>Bootstrap 框架</li>
   <li>Bootstrap 框架</li>
</ul>
```

设置成内联

```
<ul class="list-inline">
   <li>Bootstrap 框架</li>
   <li>Bootstrap 框架</li>
   <li>Bootstrap 框架</li>
   <li>Bootstrap 框架</li>
   <li>Bootstrap 框架</li>
</ul>
```

水平排列描述列表

```
<dl class="dl-horizontal">
        <dt>Bootstrap</dt>
        <dd>Bootstrap 提供了一些常规设计好的页面排版的样式供开发者使用。</dd>
</dl>
```

第十节 代码

内联代码

```
<code>&lt;section&gt;</code>
```

用户输入

```
pres <kbd>ctrl + ,</kbd>
```

代码块

```
<pre>&lt;p&gt;Please input.&lt;/p&gt;</pre>
```

Bootstrap 还列举了<var>表示标记变量，<samp>表示程序输出，只不过没有重新复写 CSS。

第三章 表格 按钮 表单 图片

第一节 表格

Bootstrap 提供了一些丰富的表格样式供开发者使用。

基本格式

```
<table class="table">
```

条纹状表格

让<tbody>里的行产生一行隔一行加单色背景效果

```
<table class="table table-striped">
```

带边框的表格

给表格增加边框

```
<table class="table table-borderd">
```

悬停鼠标

让<tbody>下的表格悬停鼠标实现背景效果

```
<table class="table table-hover">
```

状态类

可以单独设置每一行的背景样式

```
<tr class="success">
```

一共五种不同的样式可供选择。

样式	说明
active	鼠标悬停在行或单元格上
success	标识成功或积极的动作
info	标识普通的提示信息或动作
warning	标识警告或需要用户注意

danger	表示危险或潜在的带来负面影响的动作

隐藏某一行

```
<tr class="sr-only">
```

响应式表格

表格父元素设置响应式，小于 768px 出现边框

```
<body class="table-responsive">
```

第二节 按钮

基本

Bootstrap 提供了很多丰富按钮供开发者使用。可作为按钮使用的标签或元素转化成普通按钮

```
<a href="###" class="btn btn-default">Link</a>
<button class="btn btn-default">Button</button>
<input type="button" class="btn btn-default" value="input">
```

预定义样式

样式	说明
btn-default	默认样式
btn-success	成功样式
btn-info	一般信息样式
btn-warning	警告样式
btn-danger	危险样式
btn-primary	首选项样式
btn-link	链接样式

尺寸大小

从大到小的尺寸

```
<button class="btn btn-lg">Button</button>
<button class="btn">Button</button>
```

```
<button class="btn btn-sm">Button</button>
<button class="btn btn-xs">Button</button>
```

块级按钮

```
<button class="btn btn-block">Button</button>
<button class="btn btn-block">Button</button>
```

激活状态

```
<button class="btn active">Button</button>
```

禁用状态

```
<button class="btn active disabled">Button</button>
```

第三节 表单

基本格式

```
<form>
  <div class="form-group">
        <label>电子邮件</label>
        <input type="email" class="form-control" placeholder="请输入您的电子邮件">
  </div>
  <div class="form-group">
        <label>密码</label>
        <input type="password" class="form-control"  placeholder="请输入您的密码">
  </div>
</form>
```

内联表单

让表单左对齐浮动，并表现为 inline-block 内联块结构

```
<form class="form-inline">
```

注：当小于 768px，会恢复独占样式

表单合组

前后增加片段

```
<div class="input-group">
  <div class="input-group-addon">￥</div>
```

```
    <input type="text" class="form-control">
    <div class="input-group-addon">.00</div>
</div>
```

水平排列

让表单内的元素保持水平排列

```
<form class="form-horizontal">
    <div class="form-group">
        <label class="col-sm-2 control-label">电子邮件</label>
        <div class="col-sm-10">
            <input type="email" class="form-control" placeholder="请输入您的电子邮件">
        </div>
    </div>
</form>
```

注：这里用到了 col-sm 栅格系统，后面章节会重点讲解，而 control-label 表示和父元素样式同步。

复选框和单选框

设置复选框，在一行

```
<div class="checkbox">
        <label><input type="checkbox">体育</label>
</div>
<div class="checkbox">
        <label><input type="checkbox">音乐</label>
</div>

//设置禁用的复选框
<div class="checkbox disabled">
        <label><input type="checkbox" disabled>音乐</label>
</div>
//设置内联一行显示的复选框
<label class="checkbox-inline"><input type="checkbox">体育</label>
<label class="checkbox-inline disabled"><input type="checkbox" disabled>音乐</label>

//设置单选框
<div class="radio disabled">
        <label><input type="radio" name="sex" disabled>男</label>
</div>
```

下拉列表

设置下拉列表

```
<select class="form-control">
   <option>1</option>
   <option>2</option>
   <option>3</option>
   <option>4</option>
   <option>5</option>
</select>
```

校验状态

设置为错误状态

注：还有其他状态如下

样式	说明
has-error	错误状态
has-success	成功状态
has-warning	警告状态

label 标签同步相应状态

```
<label class="control-label">Input with success</label>
```

添加额外的图标

文本框右侧内置文本图标

```
<div class="form-group has-feedback">
        <label>电子邮件</label><input type="email" class="form-control">
        <span class="glyphicon glyphicon-ok form-control-feedback"></span>
</div>
```

样式	说明
glyphicon-ok	成功状态
glyphicon-warning-sign	警告状态
glyphicon-remove	错误状态

控制尺寸

从大到小

```
<input type="password" class="form-control input-lg">
<input type="password" class="form-control">
<input type="password" class="form-control input-sm">
```

注：也可以设置父元素 form-group-lg、form-group-sm，来调整。

第四节 图片

三种形状

```
<img src="img/pic.png" alt="图片" class="img-rounded">
<img src="img/pic.png" alt="图片" class="img-circle">
<img src="img/pic.png" alt="图片" class="img-thumbnail">
```

响应式图片

```
<img src="img/pic.png" alt="图片" class="img-responsive">
```

第五章 辅助类和响应式工具

第一节 辅助类

Bootstrap 在布局方面提供了一些细小的辅组样式，用于文字颜色以及背景色的设置、显示关闭图标等等。

情景文本颜色

样式名	描述
text-muted	柔和灰
text-primary	主要蓝
text-success	成功绿
text-info	信息蓝
text-warning	警告黄
text-danger	危险红

情景背景色

样式名	描述
bg-primary	主要蓝
bg-success	成功绿
bg-info	信息蓝
bg-warning	警告黄
bg-danger	危险红

关闭按钮

```
<button type="button" class="close">&times;</button>
```

三角符号

```
<span class="caret"></span>
```

快速浮动

```
<div class="pull-left">左边</div>
<div class="pull-right">右边</div>
注：这个浮动其实就是 float，只不过使用了!important 加强了优先级。
```

块级居中

```
<div class="center-block">居中</div>
```
注：就是 margin:x auto；并且设置了 display:block;。

清理浮动

```
<div class="clearfix"></div>
```
注：这个 div 可以放在需要清理浮动区块的前面即可。

显示和隐藏

```
<div class="show">show</div>
<div class="hidden">hidden</div>
```

第二节 响应式工具

类	超小屏幕 手机(<768px)	小屏幕 pad (>=768px)	中等屏幕 桌面（>=992px）	大屏幕 桌面（>=1200px）
.visible-xs-*	可见	隐藏	隐藏	隐藏
.visible-sm-*	隐藏	可见	隐藏	隐藏
.visible-md-*	隐藏	隐藏	可见	隐藏
.visible-lg-*	隐藏	隐藏	隐藏	可见
.hiddden=xs	隐藏	可见	可见	可见
.hidden-sm	可见	隐藏	可见	可见
.hidden-md	可见	可见	隐藏	可见
.hidden-lg	可见	可见	可见	隐藏

超小屏幕激活显示

```
<div class="visible-xs-block a">Bootstrap</div>
```

超小屏幕激活隐藏

```
<div class="hidden-xs a">Bootstrap</div>
```

注：对于显示的内容，有三种变体，分别为：block、inline-block、inline。

第六章 组件

第一节 图标菜单和按钮组件

小图标组件

Bootstrap 提供了免费的 263 个小图标，具体可以参考中文官网的组件 链接：http://v3.bootcss.com/components/#glyphicons。可以使用<i>或标签来配合使用，具体如下：

```
<i class="glyphicon glyphicon-star"></i>
<span class="glyphicon glyphicon-star"></span>

//也可以结合按钮
<button class="btn btn-default btn-lg">
        <span class="glyphicon glyphicon-star"></span>
</button>
<button class="btn btn-default btn">
        <span class="glyphicon glyphicon-star"></span>
</button>
<button class="btn btn-default btn-sm">
        <span class="glyphicon glyphicon-star"></span>
</button>
<button class="btn btn-default btn-xs">
        <span class="glyphicon glyphicon-star"></span>
</button>
```

下拉菜单组件

下拉菜单，就是点击一个元素或按钮，触发隐藏的列表显示出来

```
<div class="dropdown">
  <button class="btn btn-default" data-toggle="dropdown">
        下拉菜单
        <span class="caret"></span>
  </button>
  <ul class="dropdown-menu">
    <li><a href="#">首页</a></li>
    <li><a href="#">资讯</a></li>
    <li><a href="#">产品</a></li>
    <li><a href="#">关于</a></li>
  </ul>
</div>
```

按钮和菜单需要包裹在.dropdown 的容器里，而作为被点击的元素按钮需要设置 data-toggle="dropdown"才能有效。对于菜单部分，设置 class="dropdown-menu"才能自动隐藏并添加固定样式。设置 class="caret"表示箭头，可上可下。

```
//设置向上触发
<div class="dropup">

//菜单项居右对齐，默认值是 dropdown-menu-left
<ul class="dropdown-menu dropdown-menu-right">

//设置菜单的标题，不要加超链接
<li class="dropdown-header">网站导航</li>
//设置菜单的分割线

<li class="divider"></li>

//设置菜单的禁用项
<li class="disabled"><a href="#">产品</a></li>

//让菜单默认显示
<div class="dropdown open">
```

按钮组组件

按钮组就是多个按钮集成在一个容器里形成独有的效果

```
<div class="btn-group">
   <button type="button" class="btn btn-default">左</button>
   <button type="button" class="btn btn-default">中</button>
   <button type="button" class="btn btn-default">右</button>
</div>

//将多个按钮组整合起来便于管理
<div class="btn-toolbar">
   <div class="btn-group">
      <button type="button" class="btn btn-default">左</button>
      <button type="button" class="btn btn-default">中</button>
      <button type="button" class="btn btn-default">右</button>
   </div>
   <div class="btn-group">
      <button type="button" class="btn btn-default">1</button>
      <button type="button" class="btn btn-default">2</button>
      <button type="button" class="btn btn-default">3</button>
   </div>
</div>
```

```
//设置按钮组大小
<div class="btn-group btn-group-lg">
<div class="btn-group">
<div class="btn-group btn-group-sm">
<div class="btn-group btn-group-xs">

//嵌套一个分组，比如下拉菜单
<div class="btn-group">
   <button type="button" class="btn btn-default">左</button>
   <button type="button" class="btn btn-default">中</button>
   <button type="button" class="btn btn-default">右</button>

   <div class="btn-group">
      <button class="btn btn-default dropdown-toggle" data-toggle="dropdown">
         下拉菜单<span class="caret"></span>
      </button>
      <ul class="dropdown-menu">
         <li><a href="#">首页</a></li>
         <li><a href="#">资讯</a></li>
         <li><a href="#">产品</a></li>
         <li><a href="#">关于</a></li>
      </ul>
   </div>
</div>
```

注意：这里 中并没有实现 class="dropdown"，通过源码分析知道嵌套本身已经 有定位就不需要再设置。而右边的圆角只要多加一个 class="dropdown-toggle"即可。

设置按钮组垂直排列

```
<div class="btn-group-vertical">
```

设置两端对齐按钮组，使用标签

```
<div class="btn-group-justified">
   <a type="button" class="btn btn-default">左</a>
   <a type="button" class="btn btn-default">中</a>
   <a type="button" class="btn btn-default">右</a>
</div>
```

第二节 输入框和导航组件

输入框组件

文本输入框就是可以在元素前后加上文字或按钮，可以实现对表单控件的扩展。

```
//在左侧添加文字
<div class="input-group">
<span class="input-group-addon">@</span>
<input type="text" class="form-control">
</div>
//在右侧添加文字
<div class="input-group">
<input type="text" class="form-control">
<span class="input-group-addon">@163.com</span>
</div>
//在两侧添加文字
<div class="input-group">
<span class="input-group-addon">$</span>
<input type="text" class="form-control">
<span class="input-group-addon">.00</span>
</div>
//设置尺寸，另外三种分别是默认、xs、sm
<div class="input-group input-group-lg">
//左侧使用复选框和单选框
<div class="input-group">
<span class="input-group-addon"><input type="checkbox"></span>
<input type="text" class="form-control">
</div>
<div class="input-group">
<span class="input-group-addon"><input type="radio"></span>
<input type="text" class="form-control">
</div>
//左侧使用按钮
<div class="input-group">
<span class="input-group-btn">
<button type="button" class="btn btn-default">按钮</button>
</span>
<input type="text" class="form-control">
</div>
//左侧使用下拉菜单或分列式
<div class="input-group">
<span class="input-group-btn">
<button class="btn btn-default dropdown-toggle"
data-toggle="dropdown">
下拉菜单
<span class="caret"></span>
```

```
</button>
<ul class="dropdown-menu">
<li class="dropdown-header">网站导航</li>
<li><a href="#">首页</a></li>
<li><a href="#">资讯</a></li>
<li class="divider"><a href="#">产品</a></li>
<li class="disabled"><a href="#">关于</a></li>
</ul>
</span>
<input type="text" class="form-control">
</div>
```

导航组件

Bootstrap 提供了一组导航组件，用于实现 Web 页面的栏目操作。

```
//基本导航标签页
<ul class="nav nav-tabs">
   <li class="active"><a href="#">首页</a></li>
   <li><a href="#">资讯</a></li>
   <li><a href="#">产品</a></a></li>
   <li><a href="#">关于</a></li>
</ul>
//胶囊式导航
<ul class="nav nav-pills">
//垂直胶囊式导航
<ul class="nav nav-pills nav-stacked">
//导航两端对齐
<ul class="nav nav-tabs nav-justified">
//禁用导航中的项目
<li class="disabled"><a href="#">关于</a></li>

//带下拉菜单的导航
<ul class="nav nav-tabs">
   <li class="active"><a href="#">首页</a></li>
   <li><a href="#">资讯</a></li>
   <li class="dropdown">
      <a href="#" class="dropdown-toggle" data-toggle="dropdown">下拉菜单
         <span class="caret"></span>
      </a>
      <ul class="dropdown-menu">
         <li><a href="#">菜单一</a></li>
         <li><a href="#">菜单二</a></li>
      </ul>
   </li>
</ul>
```

导航条组件

导航条是网站中作为导航页头的响应式基础组件。

```
//基本格式
<nav class="navbar navbar-default">
...
</nav>
//反色调导航
<nav class="navbar navbar-inverse">
...
</nav>
//基本导航条，包含标题和列表
<nav class="navbar navbar-default">
<div class="container">
<div class="navbar-header">
<a href="#" class="navbar-brand">标题</a>
</div>
<ul class="nav navbar-nav">
<li class="active"><a href="#">首页</a></li>
<li><a href="#">资讯</a></li>
<li class="disabled"><a href="#">产品</a></li>
<li><a href="#">关于</a></li>
</ul>
</div>
</nav>
//导航条中使用表单
<form action="" class="navbar-form navbar-left">
<div class="input-group">
<input type="text" class="form-control">
<span class="input-group-btn">
<button type="submit" class="btn btn-default">提交</button>
</span>
</div>
</form>
//导航中使用按钮
<button class="btn btn-default navbar-btn">按钮</button>
//导航中使用对齐方式，left 和 right
<button class="btn btn-default navbar-btn navbar-right">按钮</button>
//导航中使用一段文本
<p class="navbar-text">我是一段文本</p>
//非导航链接，一般需要置入文本区域内
<a href="#" class="navbar-link">非导航链接</a>
//将导航固定在顶部，下面的内容会自动上移
<nav class="navbar navbar-default navbar-fixed-top">
//将导航补丁在底部
```

```
<nav class="navbar navbar-default navbar-fixed-bottom">
//静态导航，和页面等宽的导航条，去掉了圆角
<nav class="navbar navbar-default navbar-static-top">
```

第三节 路径分页标签和徽章组件

路径组件

路径组件也叫做面包屑导航。

```
//面包屑导航
<ol class="breadcrumb">
   <li><a href="#">首页</a></li>
   <li><a href="#">产品列表</a></li>
   <li class="active">韩版 2015 年羊绒毛衣</li>
</ol>
```

分页组件

分页组件可以提供带有展示页面的功能。

```
//默认分页
<ul class="pagination">
   <li><a href="#">&amp;laquo;</a></li>
   <li><a href="#">1</a></li>
   <li><a href="#">2</a></li>
   <li><a href="#">3</a></li>
   <li><a href="#">4</a></li>
   <li><a href="#">5</a></li>
   <li><a href="#">&amp;raquo;</a></li>
</ul>
//首选项和禁用
<li class="active"><a href="#">1</a></li>
<li class="disabled"><a href="#">2</a></li>

//设置尺寸，四种 lg、默认、sm 和 xs
<ul class="pagination pagination-lg">

//翻页效果
<ul class="pager">
   <li><a href="#">上一页</a></li>
   <li><a href="#">下一页</a></li>
</ul>
```

```
//对齐翻页链接
<ul class="pager">
   <li class="previous"><a href="#">上一页</a></li>
   <li class="next"><a href="#">下一页</a></li>
</ul>

//翻页项禁用
<li class="previous disabled"><a href="#">上一页</a></li>
```

标签

```
//在文本后面带上标签
<h3>标签 <span class="label label-default">new</span></h3>

//不同色调的标签
<h3>标签 <span class="label label-primary">new</span></h3>
<h3>标签 <span class="label label-success">new</span></h3>
<h3>标签 <span class="label label-info">new</span></h3>
<h3>标签 <span class="label label-warning">new</span></h3>
<h3>标签 <span class="label label-danger">new</span></h3>
```

徽章

```
//未读信息数量徽章
<a href="#">信息 <span class="badge">10</span></a>

//按钮中使用徽章
<button class="btn btn-success">提交 <span class="badge">3</span></button>

//激活状态自动适配色调
<ul class="nav nav-pills">
   <li class="active"><a href="#">首页 <span class="badge">2</span></a></li>
   <li><a href="#">资讯</a></li>
</ul>
```

第四节 巨幕页头缩略图和警告框组件

巨幕组件

巨幕组件主要是展示网站的关键性区域。

```
//在固定的范围内，有圆角
<div class="container">
   <div class="jumbotron">
```

```
        <h2>网站标题</h2>
        <p>这是一个学习性的网站！</p>
        <p><a href="#" class="btn btn-default">更多内容</a></p>
    </div>

        <div class="jumbotron">
      <div class="container">
                <h2>网站标题</h2>
                <p>这是一个学习性的网站！</p>
                <p><a href="#" class="btn btn-default">更多内容</a></p>
        </div>
        </div>
</div>
```

页头组件

```
//增加一些空间
<div class="page-header">
        <h1>大标题 <small>小标题</small></h1>
</div>
```

缩略图组件

```
//缩略图配合响应式
<div class="container">
   <div class="row">
      <div class="col-xs-6 col-md-3 col-sm-4">
         <div class="thumbnail">
                        <img src="img/pic.png" alt="">
                        </div>
                </div>
                <div class="col-xs-6 col-md-3 col-sm-4">
                        <div class="thumbnail">
                <img src="img/pic.png" alt="">
        </div>
                </div>
                <div class="col-xs-6 col-md-3 col-sm-4">
                        <div class="thumbnail">
                                <img src="img/pic.png" alt="">
                        </div>
                </div>
      <div class="col-xs-6 col-md-3 col-sm-4">
         <div class="thumbnail">
         <img src="img/pic.png" alt="">
         </div>
                </div>
         </div>
```

```
</div>


//自定义内容
<div class="container">
  <div class="row">
    <div class="col-xs-6 col-md-3 col-sm-4">
      <div class="thumbnail">
       <img src="img/pic.png" alt="">
        <div class="caption">
          <h3>图文并茂</h3>
          <p>这是一个图片结合文字的缩略图</p>
          <p><a href="#" class="btn btn-default">进入</a></p>
        </div>
            </div>
            </div>

            <div class="col-xs-6 col-md-3 col-sm-4">
                <div class="thumbnail">
                    <img src="img/pic.png" alt="">
                    <div class="caption">
                        <h3>图文并茂</h3>
                        <p>这是一个图片结合文字的缩略图</p>
                        <p><a href="#" class="btn btn-default">进入</a></p>
                    </div>
                </div>
            </div>

            <div class="col-xs-6 col-md-3 col-sm-4">
                <div class="thumbnail">
                    <img src="img/pic.png" alt="">
                    <div class="caption">
                        <h3>图文并茂</h3>
                        <p>这是一个图片结合文字的缩略图</p>
                        <p><a href="#" class="btn btn-default">进入</a></p>
                    </div>
                </div>
            </div>

    <div class="col-xs-6 col-md-3 col-sm-4">
                <div class="thumbnail">
                    <img src="img/pic.png" alt="">
                    <div class="caption">
                        <h3>图文并茂</h3>
                        <p>这是一个图片结合文字的缩略图</p>
                        <p><a href="#" class="btn btn-default">进入</a></p>
                    </div>
```

```
                    </div>
                </div>
            </div>
</div>
```

警告框组件

警告框组件是一组预定义消息。

```
//基本警告框
<div class="alert alert-success">Bootstrap</div>
<div class="alert alert-info">Bootstrap</div>
<div class="alert alert-warning">Bootstrap</div>
<div class="alert alert-danger">Bootstrap</div>

//带关闭的警告框
<div class="alert alert-success">
   Bootstrap
   <button type="button" class="close" data-dismiss="alert"><span>&times;</span></button>
</div>
//自动适配的超链接
<div class="alert alert-success">
        Bootstrap，请到官网 <a href="#" class="alert-link">下载</a>
</div>
```

第五节 进度条组件

进度条组件为当前工作流程或动作提供时时反馈。

```
//基本进度条
<div class="progress">
        <div class="progress-bar" style="width: 60%;">60%</div>
</div>

//最低值进度条
<div class="progress">
        <div class="progress-bar" style="min-width:20px">0%</div>
</div>

//结合情景的进度条
<div class="progress">
<div class="progress-bar progress-bar-success" style="min-width:20px;width:60%">60%</div>
</div>

//条纹状，IE10+支持
<div class="progress">
<div class="progress-bar progress-bar-success
```

```
progress-bar-striped" style="min-width:20px;width:60%">60%</div>
</div>

//动画效果
<div class="progress">
<div class="progress-bar progress-bar-success progress-bar-striped
active" style="min-width:20px;width:60%">60%</div>
</div>

//堆叠效果
<div class="progress">
<div class="progress-bar progress-bar-success"
style="min-width:20px;width:35%">35%</div>
<div class="progress-bar progress-bar-warning"
style="min-width:20px;width:20%">20%</div>
<div class="progress-bar progress-bar-danger"
style="min-width:20px;width:10%">10%</div>
</div>
```

第六节 列表组面板和嵌入组件

列表组组件

列表组组件用于显示一组列表的组件。

```
//基本实例
<ul class="list-group">
   <li class="list-group-item">1.这是起始</li>
   <li class="list-group-item">2.这是第二条数据</li>
   <li class="list-group-item">3.这是第三排信息</li>
   <li class="list-group-item">4.这是末尾</li>
</ul>
//列表项带勋章
<li class="list-group-item">1.这是起始<span class="badge">10</span></li>

//链接和首选
<div class="list-group">
   <a href="#" class="list-group-item active">1.这是起始<span class="badge">10</span></a>
   <a href="#" class="list-group-item">2.这是第二条数据</a>
   <a href="#" class="list-group-item">3.这是第三排信息</a>
   <a href="#" class="list-group-item">4.这是末尾</a>
</div>

//按钮式列表
```

```
<div class="list-group">
<button class="list-group-item active">1.这是起始 <span
class="badge">10</span></button>
<button class="list-group-item">2.这是第二条数据</button>
<button class="list-group-item">3.这是第三排信息</button>
<button class="list-group-item">4.这是末尾</button>
</div>
//设置项目被禁用
class="list-group-item disabled"

//情景类
<li class="list-group-item list-group-item-success">3.这是第三排信息</li>

//定制内容
<div class="list-group">
   <a href="#" class="list-group-item active">
      <h4>内容标题</h4>
      <p class="list-group-item-text">这里是相关内容详情！</p>
   </a>
   <a href="#" class="list-group-item">
      <h4>内容标题</h4>
      <p class="list-group-item-text">这里是相关内容详情！</p>
   </a>
   <a href="#" class="list-group-item">
      <h4>内容标题</h4>
      <p class="list-group-item-text">这里是相关内容详情！</p>
   </a>
</div>
```

面板组件

面板组件就是一个存放内容的容器组件。

```
//基本实例
<div class="panel panel-default">
        <div class="panel-body">这里是详细内容区！</div>
</div>

//带标题容器的面板
<div class="panel panel-default">
        <div class="panel-heading">面板标题</div>
        <div class="panel-body"></div>
</div>

//也可以设置标题元素
<div class="panel-heading">
```

```
        <h3 class="panel-title">面板标题</h3>
</div>

//带注脚的面板
<div class="panel-footer">这里是底部</div>

//情景效果：default、success、info、warning、danger、primary
<div class="panel panel-success">
        //表格类面板
   <div class="panel panel-default">
        <div class="panel-heading">表格标题</div>

   <div class="panel-body">
      <p>这里是表格标题的详细内容！</p>
        </div>

    <table class="table">
      <tr><th>1</th><th>2</th><th>3</th></tr>
      <tr><td>1</td><td>2</td><td>3</td></tr>
        </table>
</div>

//列表类面板
<div class="panel panel-default">
<div class="panel-heading">表格标题</div>

<div class="panel-body"><p>这里是表格标题的详细内容！</p></div>
        <ul class="list-group">
                <li class="list-group-item">1.这里是首页</li>
                <li class="list-group-item">2.这里是第二个项目</li>
                <li class="list-group-item">3.这里是第三个项目</li>
                <li class="list-group-item">4.这里是第四个项目</li>
        </ul>
</div>
```

第七节 模态框插件

基本使用

使用模态框的弹窗组件需要三层 div 容器元素，分别为 modal(模态声明层)、dialog(窗口声明层)、content(内容层)。在内容层里面，还有三层，分别为 header(头部)、body(主体)、footer(注脚)。

```
//基本实例
<!-- 模态声明，show 表示显示 -->
<div class="modal show" tabindex="-1">
  <!-- 窗口声明 -->
  <div class="modal-dialog">
    <!-- 内容声明 -->
    <div class="modal-content">
      <!-- 头部 -->
      <div class="modal-header">
        <button type="button" class="close" data-dismiss="modal"><span>&times;</span></
button>
        <h4 class="modal-title">会员登录</h4>
      </div>

      <!-- 主体 -->
      <div class="modal-body">
        <p>暂时无法登录会员</p>
      </div>

      <!-- 注脚 -->
      <div class="modal-footer">
        <button type="button" class="btn btn-default">注册</button>
        <button type="button" class="btn btn-primary">登录</button>
      </div>
    </div>
  </div>
</div>

<!--如果想让模态框自动隐藏，然后通过点击按钮弹窗，那么需要做如下操作。-->
<!--模态框去掉 show，增加一个 id-->

<div class="modal" id="myModal">
  <!--点击触发模态框显示-->
  <button class="btn btn-primary btn-lg" data-toggle="modal" data-target="#myModal">点击
弹窗</button>
  弹窗的大小有三种，默认情况下是正常，还有 lg(大)和 sm(小)
  <div class="modal-dialog modal-lg">

    <div class="modal-dialog sm-lg">

    可设置淡入淡出效果
    <div class="modal fade" id="myModal">

    //在主体部分使用栅格系统中的流体
    <!-- 主体 -->
    <div class="modal-body">
```

```
        <div class="container-fluid">
          <div class="row">
            <div class="col-md-4">1</div>
            <div class="col-md-4">1</div>
            <div class="col-md-4">1</div>
          </div>
        </div>
      </div>
    </div>
</div>
```

第八节 下拉菜单和滚动监听插件

下拉菜单

常规使用中，和组件方法一样，代码如下：

```
<div class="dropdown">
        <button class="btn btn-primary" data-toggle="dropdown">下拉菜单<span class="caret
"></span></button>
   <ul class="dropdown-menu">
      <li><a href="#">首页</a></li>
      <li><a href="#">产品</a></li>
      <li><a href="#">资讯</a></li>
      <li><a href="#">关于</a></li>
   </ul>
</div>
```

声明式用法的关键核心：

- 外围容器使用 class="dropdown"包裹；
- 内部点击按钮事件绑定 data-toggle="dropdown"；
- 菜单元素使用 class="dropdown-menu"。
- 如果按钮在容器外部，可以通过 data-target 进行绑定。

滚动监听

滚动监听插件是用来根据滚动条所处在的位置自动更新导航项目，显示导航项目高亮显示。

第九节 标签页和工具提示插件

标签页

标签页也就是通常所说的选项卡功能。

```
//基本用法
<ul class="nav nav-tabs">
   <li class="active"><a href="#html5" data-toggle="tab">HTML5</a></li>
   <li><a href="#bootstrap" data-toggle="tab">Bootstrap</a></li>
   <li><a href="#jquery" data-toggle="tab">jQuery</a></li>
   <li><a href="#extjs" data-toggle="tab">ExtJS</a></li>
</ul>
<div class="tab-content" style="padding: 10px;">
   <div class="tab-pane active" id="html5">...</div>
   <div class="tab-pane" id="bootstrap">...</div>
   <div class="tab-pane" id="jquery">...</div>
   <div class="tab-pane" id="extjs">...</div>
</div>
```

使用 JavaScript，直接使用 tab 方法。

```
$('#nav a').on('click', function (e) {
   e.preventDefault();
   $(this).tab('show');
});
```

工具提示

工具提示就是通过鼠标移动选定在特定的元素上时，显示相关的提示语。

```
<a href="#" data-toggle="tooltip" title="超文本标识符">HTML5</a>
```

JS 部分需要声明

```
$('#section').tooltip();
```

第十节 弹出框和警告框插件

弹出框

弹出框即点击一个元素弹出一个包含标题和内容的容器。

```
<button class="btn btn-lg btn-danger" type="button" data-toggle="popover" title="弹出框"
data-content="这是一个弹出框插件">点击弹出/隐藏弹出框</button>

//JavaScript 初始化
$('button').popover();
```

警告框

警告框即为点击小时的信息框。

```
//基本实例
<div class="alert alert-warning">
   <button class="close" type="button" data-dismiss="alert">
         <span>&times;</span>
   </button>
   <p>警告：您的浏览器不支持！</p>
</div>

//添加淡入淡出效果
<div class="alert alert-warning fade in">
如果用 JavaScript，可以代替 data-dismiss="alert"
```

JavaScript 方法

```
$('.close').on('click', function () {
         $('#alert').alert('close');
})
```

第十一节 按钮和折叠插件

按钮

可以通过按钮插件创建不同状态的按钮。

```
//单个切换。
<button class="btn btn-primary" data-toggle="button" autocomplete="off">单个切换</button>

//单选按钮
<div class="btn-group" data-toggle="buttons">
         <label for="" class="btn btn-primary active">
                  <input type="radio" name="sex" autocomplete="off" checked> 男
         </label>
         <label for="" class="btn btn-primary">
                  <input type="radio" name="sex" autocomplete="off"> 女
```

```
        </label>
</div>

//复选按钮
<div class="btn-group" data-toggle="buttons">
        <label for="" class="btn btn-primary active">
                <input type="checkbox" name="fa" autocomplete="off" checked>音乐
   </label>

   <label for="" class="btn btn-primary">
        <input type="checkbox" name="fa" autocomplete="off"> 体育
   </label>

   <label for="" class="btn btn-primary">
         <input type="checkbox" name="fa" autocomplete="off"> 美术
   </label>

   <label for="" class="btn btn-primary">
        <input type="checkbox" name="fa" autocomplete="off"> 电脑
   </label>
</div>

//加载状态
<button id="myButton" type="button" data-loading-text="Loading..." class="btn btn-primary" a
utocomplete="off">加载状态</button>

$('#myButton').on('click', function () {
   var btn = $(this).button('loading');
      setTimeout(function () {
      btn.button('reset');
   }, 1000);
});

//Button 插件中的 button 方法中有三个参数：toggle、reset、string(比如 loading、complet
e)。
// 可代替 data-toggle="button"
$('button').on('click', function () {
        $(this).button('toggle');
})
```

折叠

通过点击可以折叠内容。

```
//基本实例
<button class="btn btn-primary" data-toggle="collapse" data-target="#content" Bootstrap</butt
on>
```

```
<div class="collapse" id="content">
  <div class="well">
    Bootstrap 是 Twitter 推出的一个用于前端开发的开源工具包。它由
    Twitter 的设计师 Mark Otto 和 Jacob Thornton 合作开发,是一个 CSS/HTML 框架。目
    前,Bootstrap 最新版本为 3.0 。
  </div>
</div>
```

第十二节 轮播插件

轮播插件就是将几张同等大小的大图，按照顺序依次播放。

```
//基本实例。
<div id="myCarousel" class="carousel slide">
  <ol class="carousel-indicators">
    <li data-target="#myCarousel" data-slide-to="0"class="active"></li>
    <li data-target="#myCarousel" data-slide-to="1"></li>
    <li data-target="#myCarousel" data-slide-to="2"></li>
  </ol>

  <div class="carousel-inner">
    <div class="item active">
      <img src="img/slide1.png" alt="第一张">
    </div>
    <div class="item">
      <img src="img/slide2.png" alt="第二张">
    </div>

    <div class="item">
      <img src="img/slide3.png" alt="第三张">
    </div>
  </div>
  <a href="#myCarousel" data-slide="prev" class="carousel-controlleft">&lsaquo;</a>
      <a href="#myCarousel" data-slide="next" class="carousel-controlright">&rsaquo;</a>
</div>
```

data 属性解释：

- data-slide 接受关键字 prev 或 next，用来改变幻灯片相对于当前位置的位置；
- data-slide-to 来向轮播底部创建一个原始滑动索引，data-slide-to="2"将把滑动块移动到一个特定的索引，索引从 0 开始计数。
- data-ride="carousel"属性用户标记轮播在页面加载时开始动画播放。

轮播插件有三个自定义属性：

属性名称	描述
data-interval	默认值 5000，幻灯片的等待时间(毫秒)。如果为 false，轮播将不会自动开始循环。
data-pause	默认鼠标停留在幻灯片区域(hover)即暂停轮播，鼠标离开即启动轮播。
data-wrap	默认值 true，轮播是否持续循环。

如果在 JavaScript 调用就直接使用键值对方法，并去掉 data-；

```
//设置自定义属性
$('#myCarousel').carousel({
//设置自动播放/2 秒
interval : 2000,
//设置暂停按钮的事件
pause : 'hover',
//只播一次
wrap : false,
});
```

轮播插件还提供了一些方法，如下：

方法名称	描述
cycle	循环各帧(默认从左到右)
pause	停止轮播
number	轮播到指定的图片上(小标从 0 开始，类似数组)
prev	循环轮播到上一个项目
next	循环轮播到下一个项目

点击按钮执行

```
$('button').on('click', function () {
//点击后，自动播放
$('#myCarousel').carousel('cycle');
//其他雷同
}
```

第十三节 附加导航插件

附加导航即粘贴在屏幕某处实现锚点功能。

```
//基本实例。
<body data-spy="scroll" data-target="#myScrollspy">
  <div class="container">
    <div class="jumbotron" style="height:150px">
      <h1>Bootstrap Affix</h1>
    </div>
    <div class="row">
      <div class="col-xs-3" id="myScrollspy">
        <ul class="nav nav-pills nav-stacked" data-spy="affix" data-offset-top="150">
          <li class="active"><a href="#section-1">第一部分</a></li>
          <li><a href="#section-2">第二部分</a></li>
          <li><a href="#section-3">第三部分</a></li>
          <li><a href="#section-4">第四部分</a></li>
          <li><a href="#section-4">第五部分</a></li>
        </ul>
      </div>
      <div class="col-xs-9">
        <h2 id="section-1">第一部分</h2>
        <p>...</p>
        <h2 id="section-2">第二部分</h2>
        <p>...</p>
        <h2 id="section-3">第三部分</h2>
        <p>...</p>
        <h2 id="section-4">第四部分</h2>
        <p>...</p>
        <h2 id="section-5">第四部分</h2>
        <p>...</p>
      </div>
    </div>
  </div>
</div>
```

导航的 CSS 部分

```
ul.nav-pills {width: 200px;}
ul.nav-pills.affix{top: 30px;}
```

JavaScript 代替 data-spy="affix" data-offset-top="125"

```
$('#myAffix').affix({
  offset: {
        top: 150
  }
})
```