01----------MySQL
02----------jQuery
03----------xml+Tomcat
04----------Servlet
05----------JSP
06----------javaEE
07----------文件上传
08----------Cookie 和 Session
09----------Filter 和 ThreadLocal
10----------JSON 和 AJAX

<!--
MySQL:一种基于客户机——服务器的数据库管理系统，其他的比如微软的 SQL Server、收费的 Oracle

了解数据库的组成
|-- DB:数据库（database）：存储数据的“仓库”。保存有组织的数据的文件柜。
|-- DBMS:数据库管理系统（Database Management System）。数据操作的管家
|-- SQL:容器结构化查询语言（Structure Query Language）：专门用来与数据库通信的语言，管理过程。

了解 SQL 语句的组成
|-- DML（Data Manipulation Language):数据操纵语句，增、删、查、改
|-- DDL（Data Definition Language):数据定义语句，用于库和表的创建、修改、删除。
|-- DCL（Data Control Language):数据控制语句，用于定义用户的访问权限和安全级别。SQL 语言分类

数据库存储数据的方式（相当于一张信息表）
|--表，数据都存储在表中，相当于（类），所有表放在数据库里
|--表的列（字段），存储属性
|--表的行，存储具体的值，相当于（对象实例）

一、数据库的基本操作
1. 服务的启动与停止
启动：net start mysql
停止：net stop mysql

2. 数据库的登录与退出
mysql -u root -p 回车输入密码

3.使用某个库
use 库名

二、DQL，查询语句

1.基本查询

```
#基本查询
SELECT `first_name` FROM `employees`;
SELECT * FROM `employees`;全部元素
SELECT `first_name`,`last_name`,`salary` FROM `employees`;自助元素

#修改表头
SELECT `phone_number` AS "联系方式" FROM `employees`;

#去重
SELECT DISTINCT `department_id` AS "部门编号" FROM `employees`

#查询表的结构
DESC `employees`;
SHOW  COLUMNS FROM `employees`;

#拼接
拼接中如果有字段是 null，那么整个字段都会变成 null，用+的话会强制转换成数字先，文字则变成 0
SELECT CONCAT(`first_name`,`job_id`) "员工信息" FROM `employees`;
SELECT CONCAT(IFNULL(`commission_pct`,'空')) FROM `employees`;
```

2.条件查询

```
#条件查询 where+限定条件
|--- 条件运算符：><=  >=<= 不等于<>
|--- 逻辑运算符：not and or
|--- 模糊查询：
   |--- 含有 like，与通配符搭配使用,% 表示任意多个字符,_ 任意单个字符
        含有 a："%a%"、第二个字母为 a："_a"
   |--- in(列表)，in(100,200)
   |--- between ... and ...
   |--- is null

SELECT * FROM `employees` WHERE `salary`>10000;
SELECT * FROM `employees` WHERE `salary` BETWEEN 10000 AND 15000;
SELECT *  WHERE `commission_pct` IS NOT NULL;
SELECT * FROM `employees` WHERE NOT(`last_name` LIKE "%a%");
SELECT `job_id`,`first_name` FROM `employees` WHERE `first_name` LIKE "_a%";
SELECT `first_name`,`last_name`,`salary` FROM `employees` WHERE `salary`>12000;
SELECT `job_id`,CONCAT(`first_name`,`last_name`) AS "name",`salary` FROM `employees`
```

```
WHERE `job_id` =176;
  SELECT `first_name`,`last_name`,`salary` FROM `employees` WHERE NOT(`salary`>=5000 AND `salary`<=12000);
  SELECT `first_name`,`last_name` ,`department_id` FROM `employees` WHERE (`department_id`=20 OR `department_id`=50);
```

3.排序查询
 |--- order by 字段、函数、别名
 |--- asc 升序，desc 降序

```
  SELECT * FROM `employees` WHERE manager_id IN (100, 101, 201) ORDER BY `salary` DESC;
  SELECT LENGTH(`last_name`) AS "姓名
" ,`first_name`FROM `employees` ORDER BY LENGTH(`last_name`)ASC,`salary` DESC;
```

4.分组查询
 |--- 分组函数，sum 求和、avg 平均值、max 最大值 、min 最小值 、count 计算个数
    忽略 null,count(*)记总行数，结合去重 COUNT(DISTINCT salary)
 |--- 能按要求实现一组数据的统计查询，比如按部门排序

```
    SELECT SUM(`salary`),`department_id` FROM `employees` GROUP BY `department_id` ;
    SELECT COUNT(`department_id`),`department_id` FROM `employees` GROUP BY `department_id`;
    SELECT MAX(salary),`department_id` FROM `employees` GROUP BY `department_id`;

    SELECT MIN(salary),manager_id
    FROM employees
    WHERE manager_id IS NOT NULL
    GROUP BY manager_id
    HAVING MIN(salary)>=6000;
```

 |--- 补充：常见的单行函数：
   |--- 字符函数：length、concat、substr、instr、trim
        upper、lower、lpad、rpad、replace
   |--- 数学函数：round、ceil、floor、truncate、mod
   |--- 日期函数：now、curdate、curtime、year、month、monthname
        str_to_date、date_format
   |--- 其他函数： version、database、user
   |--- 控制函数：if、case(相当于 switch)

```
    #case 的两种用法
    SELECT
    CASE
    WHEN `salary`>2000 THEN `salary`*1.2
    WHEN `salary`<2000 THEN `salary`*2
    END AS '新工资'
    FROM `employees`;
```

```
SELECT `department_id`,`salary`,
CASE `department_id`
WHEN 30 THEN `salary`*1.2
WHEN 50 THEN `salary`*2
ELSE `salary`
#如果没有匹配的结果值，则返回结果为ELSE后的结果
#如果没有ELSE部分，则返回值为NULL。
END AS '新工资'
FROM `employees`;
```

5.多表查询（连接查询）

|--- 基本概念：

|--- 按年代分类：

- sql92 标准:仅仅支持内连接
- sql99 标准【推荐】：支持内连接+外连接（左外和右外）+交叉连接

|--- 按功能分类：

内连接：

等值连接：此表中的元素与另表中的元素相等

非等值连接

自连接：用的都是自身这张表，只是内容不同

外连接：

左外连接

右外连接

全外连接

交叉连接

|--- sql92 语法

-等值连接

```
SELECT d.department_name,d.`manager_id`,MIN(salary)
FROM employees e,`departments` d（文件名）
WHERE e.`department_id`=d.`department_id`（内联条件+过滤条件）
AND e.`commission_pct` IS NULL
GROUP BY d.`department_id`;
```

-自连接

```
SELECT e.manager_id AS 领导编号,m.job_id 领导姓名
FROM employees e,employees m
WHERE e.manager_id = m.employee_id;
```

|--- sql99 语法

|-- 内连接，与92内连接等效，提高分离性，便于阅读

select 查询列表

from 表 1 别名 【连接类型】
join 表 2 别名
on 连接条件
【where 筛选条件】
【group by 分组】
【having 筛选条件】
【order by 排序列表】

```
SELECT `last_name`,e.job_id,`salary`
FROM employees e
JOIN jobs j
ON e.job_id = j.job_id
WHERE salary>10000;
```

|-- 外连接
查询结果 = 内连接结果 + 主表中有而从表没有的记录（null 记录表示）
|-- 左外连接，left join 左边的是主表，相当于拼接成一张新的大表，属性的数量合起来了，再筛选
|-- 右外连接，right join 右边的是主表
|-- 全外连接=内连接的结果+表 1 中有但表 2 没有的+表 2 中有但表 1 没有的（取并集）

```
# 查询哪个部门没有员工，确定主表是部门，从表是员工
# 连接条件是部门 id 相等，筛选条件是新表中的员工号为 null
SELECT d.*,employee_id
FROM departments d
LEFT JOIN employees e
ON e.department_id = d.department_id
WHERE e.employee_id IS NULL;

#查询哪个城市没有部门，主表是城市，从表是部门
#连接条件是城市 id 相等，筛选条件是新表中的部门号为 null
SELECT l.*,d.department_id
FROM locations l
LEFT JOIN departments d
ON l.location_id = d.location_id
WHERE d.department_id IS NULL;

#查询部门名为 SAL 或 IT 的员工信息
#主表是部门，从表是员工表，连接条件是部门号相等，筛选条件是 name
SELECT d.department_name,last_name
FROM departments d
LEFT JOIN employees e
ON d.department_id = e.department_id
WHERE d.department_name IN('SAL','IT');
```

|-- 子查询（select 查询嵌套，双重查询）
  |--- 可以按子查询嵌套的位置、子查询的行数进行各种分类
```
SELECT last_name,job_id,salary
FROM employees
WHERE salary>(
    SELECT salary
    FROM employees
    WHERE employee_id = 143
) AND job_id = (
    SELECT job_id
    FROM employees
    WHERE employee_id = 141
);
```

|-- 分页查询：单页数据太多，满足分页显示的需求
放在语句的最后位置
|-- limit 起始位置 size(数量);
|-- limit (page-1)*size,size;

|-- 联合查询：多个表没有关联，需要将内容拼接在一起。union

三、DDL database define language 数据定义语言，操作库、表

1.库的管理
|-- 创建库
```
CREATE DATABASE IF NOT EXISTS Thinking;
```
|-- 删除库
```
DROP DATABASE IF EXISTS student;
```
|-- 修改库,低版本不支持
```
RENAME DATABASE thinking TO study;
```

2.表的管理
|-- 创建表： 表头 + 数据类型 + 约束条件（可选）
```
CREATE TABLE IF NOT EXISTS infor(
  subjects CHAR NOT NULL,
  number_all INT DEFAULT 10,
  student_name VARCHAR(20),
  #gender char  default '男'#报错，原因编码格式问题
  age INT PRIMARY KEY
);
```

|-- 约束条件，用做对表格内容的限制

NOT NULL：非空，用于保证该字段的值不能为空
DEFAULT:默认，用于保证该字段有默认值
PRIMARY KEY:主键，用于保证该字段的值具有唯一性，并且非空
UNIQUE:唯一，用于保证该字段的值具有唯一性，可以为空
CHECK:检查约束【mysql 中不支持】
FOREIGN KEY:外键，连接两表。比如此表中应用的另一张表的内容不能超过范围
AUTO_INCREMENT：自增长，一个表中只能设置一个自增长

|-- 基本数据类型
|-- 数字
int
float、double、decimal
|-- 字符
char
varchar(n)：定长字符串
text：长文本
|-- 日期
date：年月日
time：时分秒
datetime：年月日时分秒
timestamp：时间戳

|-- 删除表
DROP TABLE IF EXISTS infor;

|-- 增删查改

```
# 添加字段
alter table 表名 add 字段名称 数据类型;
ALTER TABLE infor ADD name_new CHAR;

# 删除字段
alter table 表名 drop 字段名称;
ALTER TABLE infor DROP student_name;

# 修改数据类型
alter table 表名 modify 列名称 新的数据类型;
ALTER TABLE infor MODIFY name_new INT;

# 修改列名
alter table 表名 change 旧的列名 新的列名称 新的类型;
ALTER TABLE infor CHANGE number_all total INT;

# 表建好后需要在添加外键或者级联操作,可以使用
```

alter table 表名 add constraint 外键名称(外键字段)
foreign key (外键字段) references 关系表名(关系表内字段)

|-- 复制
|--- 复制表的结构
CREATE TABLE mytable LIKE infor;

|--- 复制表的结构+内容
CREATE TABLE copy
SELECT * FROM infor;

|--- 复制部分表的结构，只有属性没有内容
CREATE TABLE copy2
SELECT age
FROM infor
WHERE 1=2;

四、DML 语句：data monipulate language 数据操作语言，直接对表中数据增删改

|-- 插入中文错误，数据库的编码格式错误，首先介绍一下编码格式的修改
MySQL 中默认字符集的设置有四级:服务器级，数据库级，表级,字段级
前三种均为默认设置，并不代表字段最终会使用这个字符集设置。
|-- 查询编码格式 SHOW CREATE TABLE <表名>;
/*
CREATE TABLE `infor` (
`subjects` varchar(30) CHARACTER SET latin1 DEFAULT NULL,
`total` int(11) DEFAULT NULL,
`name_new` int(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8
*/

|-- 修改全表的编码格式
ALTER TABLE infor CHARACTER SET utf8mb4;

|-- 修改列(字段)的编码格式
ALTER TABLE < 表 名 > MODIFY COLUMN < 字 段 名 >< 字 段 类型> CHARACTER SET utf8mb4【COLLATE utf8mb4_unicode_ci】;
ALTER TABLE infor MODIFY COLUMN subjects VARCHAR(30) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;

|-- 增：单个增、多个增
insert into 表名(列名,...) value (值 1,...);
INSERT INTO infor VALUE ('小米',67,1);

INSERT INTO infor VALUES ('桨板',120,3),('舟艇',130,4),('皮划艇',140,5);

|-- 删

1、单表的删除【★】

delete from 表名 where 筛选条件
DELETE FROM infor WHERE subjects = 's';

2、多表的删除【补充】

sql92 语法：
delete 表 1 的别名,表 2 的别名
from 表 1 别名,表 2 别名
where 连接条件
and 筛选条件;

sql99 语法：
delete 表 1 的别名,表 2 的别名
from 表 1 别名
inner|left|right join 表 2 别名 on 连接条件
where 筛选条件;

|-- 改

1.修改单表的记录★

update <表名> set 列=新值,列=新值,... where 筛选条件;
UPDATE infor SET name_new = 111 WHERE total = 67;

2.修改多表的记录【补充】
sql92 语法：
update 表 1 别名,表 2 别名
set 列=值,...
where 连接条件
and 筛选条件;

sql99 语法：
update 表 1 别名
inner|left|right join 表 2 别名
on 连接条件
set 列=值,...
where 筛选条件;

五、TCL：Transaction Control Language 事务控制语言
事务：相当于多线程问题，一个执行单元，要么全部执行，要么全部不执行。

|---事务的特性：ACID
原子性：一个事务不可再分割，要么都执行要么都不执行
一致性：一个事务执行会使数据从一个一致状态切换到另外一个一致状态
隔离性：与其他事务独立
事务的隔离级别：

	脏读	不可重复读	幻读
read uncommitted：	√	√	√
read committed：	×	√	√
repeatable read：	×	×	√
serializable	×	×	×

查看隔离级别 select @@tx_isolation;
设置隔离级别 set session|global transaction isolation level 隔离级别;

持久性：一个事务一旦提交，则会永久的改变数据库的数据.

|--- 事务的使用：
1. 设置自动提交功能为禁用 set autocommit=0;
2. 开启事务 start transaction;可选的
3. 编写事务中的 sql 语句(select insert update delete)
4. 结束事务
commit;提交事务
rollback;回滚事务

```
SET autocommit = 0;
START TRANSACTION;
UPDATE infor SET total = 60 WHERE subjects = '小米';
UPDATE infor SET total = 100 WHERE subjects = '桨板';

COMMIT;
ROLLBACK;
```

|--- 比较 delete 和 truncate
delete 删除可以撤销，删除某个内容
```
DELETE FROM infor WHERE subjects = '小米';
ROLLBACK;
```
truncate 删除不可撤销，删除所有内容

|--- savepoint 保存点： 自定义回滚的范围·
```
SET autocommit=0;
START TRANSACTION;
DELETE FROM account WHERE id=25;
```

SAVEPOINT a;#设置保存点
DELETE FROM account WHERE id=28;
ROLLBACK TO a;#回滚到保存点，此时 25 依旧被删除，但是 28 的数据可以恢复。

六、视图：通过表动态生成的数据，没有实际的表，但是保存了逻辑，可以复用。
|--- 创建视图：
create view 视图名
as
查询语句;

CREATE VIEW myview
AS
SELECT * FROM infor WHERE total>130;

|--- 修改视图：
create or replace view 视图名
as
查询语句;

|--- 删除视图 DROP VIEW 视图名
|--- 查看视图 SHOW CREATE VIEW 视图名;

|--- 修改视图内容： 部分基础的视图可以增删改查，但是大部分不能增删改
insert into
update
delete

七、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。实现了 SQL 代码的封装。
|--- MySQL 中的变量：
1. 系统变量：（默认是局部变量）
全局变量：global 关键字
会话变量：也就是局部变量，session 关键字

|--- 查看所有系统变量 show global|【session】variables;
SHOW GLOBAL VARIABLES;
SHOW 【session】 VARIABLES;

|--- 查看指定的系统变量的值 select @@global|【session】系统变量名;
SELECT @@session.tx_isolation;
SELECT @@global.autocommit;

|--- 为某个系统变量赋值
方式一：set global|【session】系统变量名=值;

```
SET GLOBAL autocommit = 0;
SET SESSION tx_isolation  = 'read-committed';
```

方式二：set @@global|【session】.系统变量名=值;

```
SET @@global.autocommit = 0;
```

2. 自定义变量：

用户变量: 针对于当前会话（连接）有效，作用域同于会话变量,重启失效

|--- 声明与赋值

#方式一：

```
SET @变量名 = 或 := 值;
SELECT @变量名:=值;

SET @m= 1;#创建赋值变量的三种方式
SET @n:=2;
SELECT @sum:=21;

SELECT @sum;#显示变量值
```

#方式二：

```
SELECT 字段 INTO @变量名
FROM 表;
```

局部变量：仅仅在定义它的 begin end 块中有效（相当于花括号里面）

declare 只能用于存储过程或函数，必须写在 begin end 之间的第一句话

|--- 创建：DECLARE 变量名 类型【DEFAULT 值】;

|--- 赋值，同用户变量

```
DECLARE m INT DEFAULT 1;
DECLARE f CHAR;
```

|--- 存储过程的使用

1.创建：类似于函数的定义过程

```
CREATE PROCEDURE 存储过程名(参数列表)
BEGIN
  存储过程体（一组合法的 SQL 语句）
END
```

2.参数列表，包含 参数模式(输入、返回) + 参数名 + 参数类型

举例：in id varchar(20)

1、参数模式：

in：调用该存储过程需要的传入值

out：存储过程的返回值
inout：该参数既需要传入值，又可以返回值

2、如果存储过程体仅仅只有一句话，begin end 可以省略
存储过程体中的每条 sql 语句的结尾要求必须加分号。

3、delimiter 重新设置结束符号(针对于 cmd 需要输入多条语句才结束的解决)
语法：delimiter 结束标记
delimiter $ 此后$既代表语句的结束符号

3.调用： call myfun(参数列表)$

4.删除存储过程：drop procedure 存储过程名

5.显示存储过程的所有信息： SHOW CREATE PROCEDURE myfun;

|--- 实际的使用一直报错：几个坑
① 存储过程的定义，必须在 cmd 里面输入
② begin end 里面的语句必须要带;正常的语法规则
③ 开始前设置好 delimiter 结束符号

|-- 空参使用存储过程：每调用一次，往 infor 表中插入一行数据。

```
DELIMITER $
CREATE PROCEDURE myfun()
BEGIN
INSERT INTO infor(subjects,total,name_new) VALUE('骑车',3000,100);
END $

call myfun()$
```

|-- in 模式使用,实现输入部门编号，查找所有该部门员工的信息

```
DELIMITER $
CREATE PROCEDURE myfound(IN department INT)
BEGIN
SELECT d.*,e.*
FROM employees e
JOIN departments d
ON d.department_id = e.department_id
WHERE  department= e.department_id;
END $

call myfound()$
```

|-- INOUT 模式的使用,输入输出的载体是同一个变量

|-- 先介绍 @ 的用处

@n，代表 n 是变量名，如果你不加的话，会认为这是一个列名，但是这列不存在，就报错了；

1 存储过程如果传进一个参数 int_id，值为 1，则在存储过程中 int_id 和@int_id 的值都是 1

2 如果在存储过程中 set @int_id2=1; 则@int_id2=1， int_id2 无关

3 如果在存储过程中 set int_id3=1，则@int_id3=NULL， int_id3 =1

4 @int_id3 不需要 declare 就可以使用，而 int_id3 必须先定义

|-- 示例，使用这个来实现对 a b 的改变

```
DELIMITER $
CREATE PROCEDURE myfun(INOUT mynumber INT,INOUT mynumber2 INT)
BEGIN
  #直接 set 表示这两个变量已经存在是直接赋值
  SET mynumber = mynumber * 5 ;
  SET mynumber2 = mynumber2 * 2 ;
END $

SET @a = 4$
SET @b = 8$
#@a,@b,表示传入的是变量 a，b 的值
CALL myfun(@a,@b)$
SELECT @m,@n$
```

八、函数

函数和存储过程，都需要在 cmd 中定义，不然会报错

几乎与存储过程一样，区别如下

存储过程可以有多个返回，所以常用作批处理的增删查改操作

函数只有一个返回值，一般用作对某个数据进行处理，得到返回值

|--- 定义语法：（注意是 returns ！！！）

```
CREATE FUNCTION 函数名(参数名 参数类型) RETURNS 返回类型
BEGIN
  函数体;
END
```

|--- 调用语法：SELECT 函数名(参数列表)

|--- 查看函数：SHOW CREATE FUNCTION myf3;

|--- 删除函数：DROP FUNCTION myf3;

|--- 无参函数

```
DELIMITER $
CREATE FUNCTION myf() RETURNS INT
```

```
BEGIN
    DECLARE c INT DEFAULT 0;
    SELECT COUNT(*) INTO c
    FROM `employees`;
    RETURN c;
END $
调用：SELECT myf()$
```

|--- 带参函数

```
DELIMITER $
CREATE FUNCTION myf2(num1 INT ,num2 INT) RETURNS INT
BEGIN
    DECLARE SUM INT DEFAULT 0;
    SET SUM= num1+num2;
    RETURN SUM;
END$
调用：SELECT myf2(5,6)$
```

九、分支循环结构

一. 分支结构

1. if 函数：if(条件，值 1，值 2)

位置：可以作为表达式放在任何位置

2. case 结构：

```
语法 1：
case 表达式或字段
when 值 1 then 语句 1;
when 值 2 then 语句 2；
..
else 语句 n;
end [case];

语法 2：
case
when 条件 1 then 语句 1;
when 条件 2 then 语句 2；
..
else 语句 n;
end [case];

位置：
可以放在任何位置，
如果放在 begin end 外面，作为表达式结合着其他语句使用
如果放在 begin end 里面，一般作为独立的语句使用
```

3. if 结构：
语法：
if 条件 1 then 语句 1;
elseif 条件 2 then 语句 2;
...
else 语句 n;
end if;

位置：只能放在 begin end 中

```
DELIMITER $
CREATE FUNCTION grade(g INT ) RETURNS CHAR
BEGIN
  DECLARE gra CHAR DEFAULT 'A';
  IF g>=90 THEN SET gra = 'A';
  ELSEIF g>=80 THEN SET gra = 'B';
  ELSEIF g>=70 THEN SET gra = 'C';
  ELSE SET gra = 'D';
  END IF;
  RETURN gra;
END$
```

二. 循环结构
位置：只能放在 begin end 中
1. while 结构：
语法：
【名称:】while 循环条件 do
  循环体
end while 【名称】;

2. loop 结构：
语法：
【名称：】loop
  循环体
end loop 【名称】;

3. repeat 结构：
语法：
【名称:】repeat
  循环体
until 结束条件
end repeat 【名称】;

4. 循环控制语句：
leave：类似于 break，用于跳出所在的循环
iterate：类似于 continue，用于结束本次循环，继续下一次

5. 三种循环对比：
(1)、这三种循环都可以省略名称，
但如果循环中添加了循环控制语句（leave 或 iterate）则必须添加名称
(2)、位于哪里？
loop 一般用于实现简单的死循环
while 先判断后执行
repeat 先执行后判断，无条件至少执行一次

```
DELIMITER $
CREATE PROCEDURE myp(IN countnum INT)
BEGIN
 DECLARE i INT DEFAULT 1;
 a:WHILE i<countnum DO
    INSERT INTO infor(subjects,name_new) VALUE ('lucy',i);
    IF i>20 THEN LEAVE a;
    END IF;
    SET i = i+1;
    END WHILE a;
END $

#创建表格，插入随机字符串
DELIMITER $
DROP TABLE IF EXISTS stringcontent$
CREATE TABLE stringcontent(
 id INT PRIMARY KEY AUTO_INCREMENT,
 content VARCHAR(26)
)$

DROP PROCEDURE IF EXISTS insert_string$
CREATE PROCEDURE insert_string(IN countnum INT)
BEGIN
 DECLARE str VARCHAR(26) DEFAULT 'abcdefghijklmnopqrstuvwxyz';
 DECLARE startindex INT DEFAULT 1;
 DECLARE i INT DEFAULT 1;
 WHILE i<countnum DO

   SET startindex = FLOOR(26*RAND()+1);
   INSERT INTO stringcontent(content) VALUE(SUBSTR(str,startindex));
   SET i = i+1;
```

```
    END WHILE;
  END$


-->

<!--
  jQuery:javaScript Query,目的是简化 javaScript 编程

  在事件响应的 function 函数中，有一个 this 对象。
  这个 this 对象是当前正在响应事件的 dom 对象
  return false；可以阻止元素的默认行为

  1.$
   是 jQuery 的核心函数，能完成 jQuery 的很多功能。$()就是调用$这个函数
   获取对象值：var name = $("#empName").val();
   直接剑建一个行标签对象：var $trobj=s(<tr>表格</tr>);

  2.jQuery 对象和 dom 对象
   javascrpit 出来的对象就是 DOM 对象，经过 jQuery 函数出来的就变成了 jQuery 对象
   jQuery 对象 = dom 对象的数组 +jQuery 提供的一系列功能函数。
   jQuery 通过数组下标取出就是 DOM 对象

  3.jQuery 选择器,同 javascript
    |-- 基本选择器
      $("#id"),id 选择器，返回对应 id 的对象
      $(".class")，类选择器，返回对应类的对象
      $("label")，标签选择器，返回对应标签的对象

    |-- 层级选择器
      ancestor descendant 后代选择器：在给定的祖先元素下匹配所有的后代元素
      parent>child 子元素选择器：在给定的父元素下匹配所有的子元素
      prev+next 相邻元素选择器：匹配所有紧接在 prev 元素后的 next 元素
      prev~sibings 之后的兄弟元素选择器：匹配 prev 元素之后的所有 siblings 元素

  4.jQuery 过滤器,类似于伪元素 ::first
    |-- 基本过滤器
      :first ， $("div:first")
      :last
      :not(selector)
      :even
      :odd
      :eq(index)
      :gt(index)
```

:lt(index)
:header
:animated

|-- 内容过滤器
:contains(text) ，$("div:contains('di')")，含有文本'di'的 div 元素
:empty
:has(selector)
:parent，$("div:parent")含有子元素的 div

|-- 可见性、属性、表单对象过滤器
:hidden 、:visible
[attribute=value]，$("div[title='test']")，属性 title 值等于'test'的 div 元素

|-- 元素筛选
first()
last()
hasClass(class)
closest(expr,[con]|obj|ele)1.6*
find(expr|obj|ele)

例如：select[name=sel01] :selected 表示名字叫 sel01 的选择框的被选元素
select[name=sel01] option 表示名字叫 sel01 的选择框的 option 元素

5. jQuery 的 DOM 属性操作
|--页面属性
html() 它可以设置和获取起始标签和结束标签中的内容。 同 innerHTML
text() 它可以设置和获取起始标签和结束标签中的文本。 同 innerText
val() 它可以设置和获取表单项的 value 属性值。 同 value
例如：
获取选中状态：$(":checkbox").val(["checkbox3","checkbox2"]);

|-- 获取设置属性的值
attr()
prop()
设置多选框，并设置选中状态为选中，$(":checkbox").prop("checked",true);

例如设置反选按钮：

```
$("#checkedRevBtn").click(function(){
    //用 each 来获取每一个的选中状态
    $("[name = 'items']").each(function () {
        this.checked = !this.checked;
```

```
            });
            //再将全选的按钮也设置上
            var stat = $("[name = 'items']:checked").length==4;
            $("#checkedAllBox").attr("checked",stat)

        });
    |-- 增删查改
        appendTo(content)
        prependTo(content)

        insertAfter(content)
        insertBefore(content)

        replaceWith(content|fn)
        replaceAll(selector)

        empty()
        remove([expr])

        例如：$(this).appendTo("select[name=sel02]");将 eah 中的对象转移到框 2 中去

6. jQuery 设置 CSS 动画效果，
    |-- 添加样式
        addClass() 添加样式，里面放在 CSS 中编写好的样式。
        removeClass() 删除样式
        toggleClass() 有就删除，没有就添加样式。
        offset() 获取和设置元素的坐标。

    |-- 添加动画效果，里面添加参数，设置动画时间和回调函数
        show() 将隐藏的元素显示
        hide() 将可见的元素隐藏。
        toggle() 可见就隐藏，不可见就显示。
        fadeIn() 淡入（慢慢可见）
        fadeOut() 淡出（慢慢消失）
        fadeTo() 在指定时长内慢慢的将透明度修改到指定的值。0 透明，1 完成可见，0.5 半透明
        fadeToggle() 淡入/淡出 切换

7. jQuery 的事件处理
    |-- 常用事件处理
        click() 它可以绑定单击事件，以及触发单击事件
        mouseover() 鼠标移入事件
        mouseout()  鼠标移出事件
        bind() 可以给元素一次性绑定一个或多个事件。
```

one() 使用上跟 bind 一样。但是 one 方法绑定的事件只会响应一次。
unbind() 跟 bind 方法相反的操作，解除事件的绑定
live() 也是用来绑定事件。
它可以用来绑定选择器匹配的所有元素的事件。
哪怕这个元素是后面动态创建出来的也有效

|-- 事件的冒泡
父子元素同时监听同一个事件。当触发子元素的事件的时候，同一个事件也被传递到了父元素的事件里去响应。
阻止事件冒泡
在子元素事件函数体内，return false;

|-- 获取事件对象
在给元素绑定事件的时候，参数列表中添加一个参数，function(event)，此 event 就是事件对象
-->

<!--
XML

XML 被设计用来传输和存储数据。与配置文件 properties 的作用相似，语法与 HTML 相似
HTML 被设计用来显示数据。

1. XML 标签
声明文件：<?xml version="1.0" encoding="utf-8"?>
没有预定义的标签，可以自定义
注意：大小写敏感、属性必须加引号，关闭标签不能掉、位置不能混乱

2. XML 元素
从（且包括）开始标签直到（且包括）结束标签的部分。如：<title>Harry Potter</title>

3. XML 中，尽量避免使用属性。优先使用元素。
属性不能包含多个值、不能包含树结构、不易扩展
元数据（有关数据的数据）应当存储为属性，而数据本身应当存储为元素
属性：<person sex="female">
元素：<sex>female</sex>

4. 解析 XML 的 dom4j,第三方解析技术
|-- 创建 Saxreader 对象。这个对象，用于读取 xml 文件，并创建 Document
Saxreader reader =new Saxreader O)
Document document =reader read(src/books.xml);

|-- 遍历标签获取所有标签中的内容
第一步，通过 Document 对象。拿到 XML 的根元素对象

```
            Element root =document.getrootelement();

        第二步，通过根元素对象。获取所有的book 标签对象元素
            List<element> books =rootelements("book");
            Element nameelement =book.element("name");

        第三步，通过 gettext()方法获取每一个元素的文本内容
            name = nameelement.gettext();
-->

<!--
    Tomcat

    1.javaweb
    JavaWeb 是指，所有通过Java 语言编写可以通过浏览器访问的程序的总称
    简单讲就是用 java 以及相关知识做网站开发
    javaWeb 容器 == 服务器,tomcat 就是一种轻量级的服务器

    2.常用的 web 资源
        静态资源： html、css、js、txt、mp4 视频 ,jpg 图片
        动态资源： jsp 页面、Servlet 程序

    3.安装目录介绍
        bin 专门用来存放 Tomcat 服务器的可执行程序
        conf 专门用来存放 Tocmat 服务器的配置文件
        lib 专门用来存放 Tomcat 服务器的 jar 包
        logs 专门用来存放 Tomcat 服务器运行时输出的日记信息
        temp 专门用来存放 Tomcdat 运行时产生的临时数据
        webapps 专门用来存放部署的 Web 工程。
        work 是 Tomcat 工作时的目录，用来存放 Tomcat 运行时 jsp 翻译为 Servlet 的源码，
和 Session 钝化的目录。

    4.IDEA 整合 Tomcat 服务器
      创建动态 web 工程

-->

<!--
    写在前面，理清关系。

    什么是 Web 服务器？
    Web 服务器的作用说穿了就是：将某个主机上的资源映射为一个 URL 供外界访问。

    什么是 Servlet 容器？
```

Servlet 容器，顾名思义里面存放着 Servlet 对象。

我们为什么能通过 Web 服务器映射的 URL 访问资源？都要经过以下步骤：
接收请求
处理请求
响应请求
于是把接收和响应两个步骤抽取成 Web 服务器，
但处理请求的逻辑是不同的，抽取出来做成 Servlet，交给程序员自己编写。
随后一些逻辑就从 Servlet 抽取出来，分担到 Service 和 Dao。
但是 Servlet 并不擅长往浏览器输出 HTML 页面，所以出现了 JSP。
等 Spring 家族出现后，Servlet 开始退居幕后，取而代之的是方便的 SpringMVC。

进入 Tomcat 阶段后，我们开始全面面向接口编程。

Servlet 的五个方法，最难的地方在于形参，然而 Tomcat 会事先把形参对象封装好传给我。
不需要写 TCP 连接数据库、解析 HTTP 请求、把结果转成 HTP 响应，和原始的、底层的解析、连接等没有丝毫关系。
request 对象和 response 对象就是搞定之后封装成的形参。

Servlet 虽然是个接口，但实现类只是个空壳，我们写点业务逻辑就好了。

总的来说， Tomcat 已经替我们完成了所有底层抽象操作，并且传入三个对象
ServletConfig、 ServletRequest、 ServletResponse。

ServletConfig：web.xml 中的配置直接经过 dom4j 解析 xml 得到对象，打包给我们

Request：HTTP 请求到了 Tomcat 后，Tomcat 通过字符串解析，把各个请求头（ Header),
请求地址（URL),请求参数（ Query String）都封装进了 Request 对象中。

Response, Servlet 逻辑处理后得到结果，最终通过 response. write()方法，
将结果写入 response 内部的缓冲区。 Tomcat 会在 servlet 处理结束后，拿到
response，遍历里面的信息，组装成 HTP 响应发给客户端。

Servlet 接口 5 个方法，其中 init、service、destroy 是生命周期方法。
init 和 destroy 各自只执 1 次，即 servlet 创建和销毁时。而 service 会在每次有
新请求到来时被调用。也就是说，我们主要的业务代码需要写在 servicer 中.但是，
浏览器发送请求最基本的有两种：Get/Post，于是我们必须在里面区分请求写处理逻辑。

在 Servlet 基础之上抽象类 Genericservlet 对其进行改良。但是依旧没有实现 service 类
于是抽象类 HttpServlet 继承 Genericservlet，实现 service 的 get、post 方法区分

Httpservlet 虽然在 servicer 中帮我们写了请求方式的判断。但是针对每一种请求，
业务逻辑代码是不同的， HttpServlet 无法知晓子类想干嘛，所以写成抽象类，要求实现
doGet()、doPost()两个方法，当子类重写该方法，整个业务代码就活了。

servlet 是啥？很简单，是一组接口，接口的作用是什么？规范
servlet 接口定义的是一套处理网络请求的规范，所有实现 servlet 的类，都需要实五个方法
其中最主要的是两个生命周期方法 init()和 destroy(),还有一个处理请求的 servce(),
也就是说，要回答这三个问题
你初始化时要做什么?
你销毁时要做什么?
你接受到请求时要做什么?

servlet 不会直接和客户端打交道！那请求怎么来到 servlet 呢？答案是 servlet 容器。
比如我们最常用的 tomcat,必须把 servlet 部署到一个容器中，不然 servlet 不会起作用。

tomcat 才是与客户端直接打交道的家伙，他监听了端口，请求过来后，根据 url 等信息，
确定要将请求交给哪个 servlet 去处理，然后调用那个 servlet 的 service 方法，
service 方法返回一个 response 对象，tomcat 再把这个 response 返回给客户端。

-->

<!--
Servlet 是 Javaee 规范之一。规范就是接口，是 Javaweb 三大组件之一。
Servlet 是运行在服务器上的一个 java 小程序，接收客户端发送过来的请求，并响应数据给客户端

注意 Servlet 的默认编码格式是"iso-8859-1"，如果中文乱码注意编码格式设置

|-- 三大组件分别是：
Servlet 程序
Filter 过滤器
Listener 监听器

1. 手动实现 Servlet 程序
|-- 编写一个类去实现 Servlet 接口
|-- 实现 service 方法，处理请求，并响应数据
|-- 在 web.xml 中去配置 servlet 程序的访问地址

2. Servlet 类的继承体系
|-- Servlet 是抽象接口
GenericServlet 实现 Servlet 接口
HttpServlet 继承 GenericServlet，实现了 service 方法

|-- 继承 HttpServlet 实现 Servlet 程序，重写 doGet 或 doPost 方法

|-- ServletConfig 类，Servlet 程序的配置信息类。
获取 servlet-name 的值，servletConfig.getServletName()
获取初始化参数 init-param，servletConfig.getInitParameter("username")
获取 ServletContext 对象，servletConfig.getServletContext()

|-- ServletContext 类
ServletContext context = getServletConfig().getServletContext();

获取 web.xml 中配置的上下文参数 context-param
获取当前的工程路径，context.getContextPath()
获取工程部署后在服务器硬盘上的绝对路径，context.getRealPath("/imgs/1.jpg")
像 Map 一样存取数据，context.setAttribute("key1", "value1");

3. HTTP 协议
客户端和服务器之间通信时，发送的数据，需要遵守的规则，叫 HTTP 协议
客户端给服务器发送数据叫请求。
服务器给客户端回传数据叫响应。

|-- 请求又分为 GET 请求，和 POST 请求两种
请求行：Get/06_servlet/a.html HTTP/1.1
请求的方式:Get
请求的资源路径:06_servlet/a.html
请求的协议的版本号:HTTP/1.1
请求头:包括数据类型、语言信息、浏览器信息等

|-- 响应
响应行：HTTP/1.1 200 ok
响应的协议:HTTP/1.1
响应状态码:200
响应状态描述符:OK
响应头：同理

|-- 常见响应码
200 表示请求成功
302 表示请求重定向
404 表示请求服务器已经收到了，但是你要的数据不存在（请求地址错误）
500 表示服务器已经收到请求，但是服务器内部错误（代码错误）

4. HttpServletResquest 类

html 存放在 web 目录下。a.html 页面 访问地址是： http://localhost:8080/工程路径/a.html

|-- 设置表单：<form action="http://localhost:8080/07_servlet/parameterServlet" method="get">

相当于规定你在页面上提交了数据，谁是处理数据的服务器。

|-- 浏览器使用 method 属性设置的方法将表单中的数据传送给服务器进行处理。
HTTP 请求的两种基本方法：POST 方法和 GET 方法。
|-- POST 方法，浏览器将会按照下面两步来发送数据。
首先，浏览器将与 action 属性中指定的表单处理服务器建立联系，
建立连接之后，浏览器就会按分段传输的方法将数据发送给服务器。
|-- GET 方法，建立连接，直接将数据直接附在表单的 URL 之后发送。
GET 和 POST 最直观的区别就是 GET 把参数包含在 URL 中，POST 通过 request body 传递参数。

|-- 配置 web.xml

```
<servlet>
<servlet-name>HelloServlet</servlet-name>,给调用的类取别名
<servlet-class>test.HelloServlet</servlet-class>,③ 该别名对应的真实的类
</servlet>

给 servlet 配置访问地址
<servlet-mapping>
  <servlet-name>HelloServlet</servlet-name>,② 响应之后调用哪个别名的类服务
  <url-pattern>/helloservlet</url-pattern>,① 访问哪个目录时响应该 servlet
</servlet-mapping>
```

|-- 服务类功能的实现，继承 HttpServlet,重写 doGet()和 doPost()作为方法实现的响应体

```
public class ResquestDemo extends HttpServlet {
@Override
protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
    super.doGet(req, resp);
}

@Override
protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
    super.doPost(req, resp);
}
}
```

5. HttpServletResponse 类
每次请求进来，Tomcat 服务器都会创建一个 Response 对象传递给 Servlet 程序去使用。

HttpServletRequest 表示请求过来的信息
HttpServletResponse 表示所有响应的信息
可以通过 HttpServletResponse 对象来设置返回给客户端的信息

|-- 输出流实现响应信息的回传，两个流不能同时使用
字节流 getOutputStream(); 常用于下载（传递二进制数据）
字符流 getWriter(); 常用于回传字符串（常用）

获取流之前先设置编码格式避免显示乱码的情况
resp.setContentType("text/html; charset=UTF-8");

PrintWriter writer = resp.getWriter();
writer.write("response's content!!!");

|-- 重定向：原来的网站已经迁移，转向新的地址
resp.sendRedirect("http://localhost:8080/03_servlet_war_exploded/helloservlet");

-->

<!--
JSP：java server pages。Java 的服务器页面。本质其实就是 Servlet。

作用：代替 Servlet 程序回传 html 页面的数据，简化 Servlets 输出 HTML 代码。
JSP = HTML + Java 片段，里面的代码是 java 和 html 的混合，可读性差。

第一次访问 jsp 页面的时候。Tomcat 服务器把 jsp 页面翻译成为一个 java 源文件。并且对它进行编译成为.class 字节码程序。而这个 java 源文件里面的内容是：
jsp 翻译出来的 java 类，它间接了继承了 HttpServlet 类。
也就是说，翻译出来的是一个 Servlet 程序

Servlet 程序的源代码底层实现，也是通过输出流把 html 页面数据回传给客户端。

|-- JSP 需要学:JSTL 和 EL 表达式
|-- EL 表达式：表达式语言（ Expression Language,EL),${}括起来的脚本，
用来读取数据，进行内容的显示
使用 EL 表达式可以方便地读取对象中的属性、提交的参数、Java Bean、甚至集合

|-- JSTL 全称为 JSP Standard Tag Library 即 JSP 标准标签库。

JSTL 提供了一系列的 JSP 标签，实现了基本的功能：集合的遍历、数据的输出、字符串的处理、数据的格式化等等

为什么要使用 JSTL?

EL 表达式不能遍历集合，做逻辑的控制。JSTL 与 HTML 代码十分类似，遵循 XML 标签语

法，使用 JSTL 让 JSP 页面显得整洁，可读性非常好，重用性高

|-- jsp 的三种语法

|-- jsp 头部的 page 指令,修改 jsp 页面中一些重要的属性，或者行为

<%@ page contentType="text/html;charset=UTF-8" language="java" %>

包括以下属性：

language 属性

contenttype 属性

pageencoding 属性

import 属性

autoflush 属性

buffer 属性

errorpage 属性

iserrorpage 属性

session 属性

extends 属性

|-- jsp 中的三种脚本

|-- 声明脚本：java 类的内部的代码都可以写，使用少

```
<%!
   java 代码（定义变量、静态代码块、方法、内部类）
%>

<%!
   private Integer id;
   private String name;
   private static Map<String,Object> map;
%>
```

|-- 表达式脚本（常用），会被翻译成为 out.print()输出到页面上

```
<%=表达式%>，

<%=12 %> <br>  =====》 相当于 out.print(12)
<%=12.12 %> <br>
<%="我是字符串" %> <br>
```

```
<%=map%><br>
<%=request.getParameter("username")%>
```

|-- 代码脚本,实现在 jsp 页面中，用 java 编写需要的功能。
可以多个代码脚本拼在一起、和表达式脚本一起组合使用，在 jsp 页面上输出数据
<% java 语句 %>

```
<% for (int j = 0; j < 10; j++) { %>
   <tr>
      <td>第 <%=j + 1%>行 </td>
   </tr>
<% } %>
```

|-- jsp 中的三种注释
|-- < !-- 这是 html 注释 -- >,会被翻译到 java 代码中输出到 html 页面中查看

|-- // 单行 java 注释, 会被翻译到 java 源代码中。
/* 多行 java 注释 */

|-- <%-- 这是 jsp 注释 --%>,jsp 注释在翻译的时候会直接被忽略掉,可以注掉 jsp 页面中所有代码

|-- 九大内置对象，可以在【代码脚本】中或【表达式脚本】中直接使用

request 对象：请求对象，可以获取请求信息
response 对象：响应对象。可以设置响应信息
pageContext 对象：当前页面上下文对象。可以在当前上下文保存属性信息
session 对象：会话对象。可以获取会话信息。
exception 对象：异常对象只有在 jsp 页面的 page 指令中设置 isErrorPage="true" 的时候才会存在
application 对象：ServletContext 对象实例，可以获取整个工程的一些信息。
config 对象：ServletConfig 对象实例，可以获取 Servlet 的配置信息
out 对象：输出流。
page 对象：表示当前 Servlet 对象实例（无用，用它不如使用 this 对象）。

其中 jsp 四大域对象，经常用来保存数据信息。
pageContext 可以保存数据在同一个 jsp 页面中使用
request 可以保存数据在同一个 request 对象中使用。经常用于在转发的时候传递数据
session 可以保存在一个会话中使用
application(ServletContext) 就是 ServletContext 对象

|-- 输出，jsp 翻译之后，底层源代码都是使用 out 来进行输出，

out.write() 输出字符串没有问题
out.print() 输出任意数据都没有问题（都转换成为字符串后调用的 write 输出）
在 jsp 页面中，可以统一使用 out.print()来进行输出

|-- jsp 的常用标签

|-- 静态包含--很常用
<%@ include file="" %>，把包含的内容原封拷贝到 service 中
file 属性指定你要包含的 jsp 页面的路径，地址中第一个斜杠 / 表示为 http://ip:port/工程路径/
如：<%@ include file="/include/footer.jsp"%>

2）动态包含--很少用
<jsp:include page=""></jsp:include>

3）页面转发--常用，page 属性设置请求转发的路径
例如：<jsp:forward page="/index.jsp"></jsp:forward>

-->

<!--
JavaBean :遵循“一定编程原则”的 Java 类既被称作 JavaBean。

|-- JavaBean 是一个遵循特定写法的可重用的 Java 类，必须符合特定的约定：
1、提供：public 无参构造函数；
2、所有属性私有化（private）；
3、提供私有化的属性的 getter()和 setter()方法
4、实现 Serializable 接口，用于实现 bean 的持久性

|-- Bean：组件技术
Bean 的含义是可重复使用的 Java 组件。就好像你做了一个扳手，而这个扳手会在很多地方被拿去用，这个扳子也提供多种功能(你可以拿这个扳手扳、锤、撬等等)，而这个扳手就是一个组件。使用它的对象只能通过接口来操作。

|-- Java Bean
是基于 Java 的组件模型，由属性、方法和事件 3 部分组成。
JavaBean 在 Java EE 开发中，通常用于封装数据，对于遵循以上写法的 JavaBean 组件，其它程序可以通过反射技术实例化 JavaBean 对象、获取 JavaBean 的属性
-->

<!--
JavaEE

当前工程的路径代表 web 的路径，http://localhost:8080/04_book/，已经表示到了 web 文件夹

|-- 三层架构
表示层（web 层）、业务逻辑层（service 层）、数据访问层（dao 层）。
分层的目的是为了解耦。解耦就是为了降低代码的耦合度。方便后期的维护和升级。

|-- web 层：
与客户端交互，包括获取用户请求，传递数据，封装数据，展示数据。
|-- service 层：
复杂的业务处理，包括各种实际的逻辑运算。
|-- dao 层：
与数据库进行交互，与数据库相关的代码在此处实现。

|-- 项目文件结构
|-- src：存放 java 源代码
web 层 com.myproject.web/servlet/controller
service 层 com.myproject.service Service 接口包
com.myproject.service.impl Service 接口实现类
dao 持久层 com.myproject.dao Dao 接口包
com.myproject.dao.impl Dao 接口实现类
实体 bean 对象 com.myproject.pojo/entity/domain/bean JavaBean 类
测试包 com.myproject.test/junit
工具类 com.myproject.utils

|-- web：存放静态资源
html
js
jsp
css
commom:公共静态文件资源（图片）

|-- web-inf:对外界隐藏的资源文件夹
lib 项目依赖包
web.xml 项目配置文件

|-- 开发步骤
1. 创建存储的项目数据库 book 和对应的用户表 t_user
2. 创建于数据库对应的 user 类即 javabase 类

3. 创建数据库连接池
lib 导包加入模块中
src 下复制 jdbc.properties，配置数据库连接

新建 jdbcUtils，提供数据库连接和关闭的方法。
在 test 中创建数据库连接工具包是否成功的测试代码 JdbcUtileTest

4. 编写 DAO，实现对数据库的增删查改操作
① 抽象基类 BaseDAO,里面包含数据库的增山查改操作
导入工具包:commons-dbutils-1.3.jar

② UserDAO 接口，里面设计查询用户名、用户名和密码、保存用户数据的几个空壳方法。

③ UserDAOImp 类，继承 BaseDAO 类，实现 UserDAO 接口，实现登录过程中对密码的校验和注册时保存用户信息的操作。

④ 编写 UserDAOImp 类的测试代码 UserDAOImpTest
IEDA 自动生成测试类代码：ctrl+shift+t

5. 编写 service，实现业务：登录、注册
① 编写 userService 空壳接口，表示要实现的业务逻辑
② 编写实现类 userServiceImp
里面直接调用 DAO 中实现的根据用户名查询、根据用户名、密码查询的数据库操作
③ 编写测试类 userServiceTest

6.编写 web 层
① 修改 regist.html 和 regist_success.html 页面
|--设置 base 路径，将相对路径固定到当前工程路径，注意，要与 tomcat 服务器中的名称一致：http://localhost:8080/04_book_war_exploded/

|-- 修改注册表单的提交地址（要加上项目名称）和请求方式
action = "/04_book_war_exploded/registerServlet", 注意这里的项目名称要与 tomcat 中的 application context 一致

② 编写 编写 RegistServlet 程序
分析实现的逻辑
1、获取请求的参数，username、passward
2、检查验证码是否正确
正确——>检查用户名是否可用
|-- 可用
调用 service 存到数据库
跳到注册成功页面 regist_ success.htm
|-- 不可用
跳回注册页面
不正确
跳回注册页
③ 编辑 web.xml 文件，设置 servlet 及其 maping
-->

<!--
文件的上传与下载

文件上传：

1.表单页面设置
|-- form 标签，method=post 请求，get 有字节大小限制
|-- encType 属性值必须为 multipart/form-data 值
表示提交的数据，以多段（每一个表单项一个数据段）的形式进行拼接，然后以二进制流的形式发送给服务器
|-- input type=file 添加上传的文件
|-- Servlet 程序）接收，处理上传的数据。

<form action=servlet 程序地址 method="post" enctype="multipart/form-data">
头像：<input type="file" name="photo" >
</form>

2.servlet 程序处理(字节流)
|-- 导入第三方的工具包，
commons-fileupload-1.2.1.jar
commons-io-1.4.jar

|-- ServletFileUpload 类，用于解析上传的数据。
isMultipartContent(HttpServletRequest request); 判断上传的数据格式是否是多段的格式。
List<FileItem> parseRequest(HttpServletRequest request) 解析上传的数据，返回表单项的 list

FileItem 类，表示每一个表单项。
isFormField() 判断是普通的表单(true)。还是上传的文件类型
getFieldName(),获取表单项的 name 属性值
getString(),获取当前表单项的值。
getName(),获取上传的文件名
write(file);,将上传的文件写到 参数 file 所指向的硬盘位置

创建 FileItemFactory 工厂实现类
FileItemFactory fileItemFactory = new DiskFileItemFactory();

创建用于解析上传数据的工具类 ServletFileUpload 类
ServletFileUpload servletFileUpload = new ServletFileUpload(fileItemFactory);

解析上传的数据，得到每一个表单项 FileItem

```
        List<FileItem> list = servletFileUpload.parseRequest(req);

        for (FileItem item:list) {
            //当不是普通表单项的时候进行读取
            if(!item.isFormField()){
                System.out.println("上传文件名是："+item.getName());
                item.write(new File("d:\\" + item.getName()));
            }
        }
-->



<!—
    Cookie 和 Session

    浏览器打开一个网页，用到的是 HTTP 协议，它是无状态的，这一次请求和上一次请求是没有任何关系的，互不认识的，没有关联的。好处是快速，但是不方便，于是基于这种需求就出现了 Cookie


    Cookie 是帮助 web 站 保留访问者信息的技术，是存储在用户电脑上的小文件，保存一些站点的用户数据，这样能够让服务器为这样的用户定制内容。一般保存时间为 7-30 天

    比如：免用户名登录。
    第一次登录成功之后，服务器给浏览器发送了用户名和密码的 cookie，浏览器存储
    下一次浏览器又要登录时，在发送请求头的时候一起把 cookie 发送给服务器
    服务器解析出直接填入用户名和密码实现免用户名登录。

    登录成功就设置 cookie
    if ("wzg168".equals(username) && "123456".equals(password)) {
        //登录 成功
        Cookie cookie = new Cookie("username", username);
        cookie.setMaxAge(60 * 60 * 24 * 7);//当前 Cookie 一周内有效
        resp.addCookie(cookie);
        System.out.println("登录 成功");
    } else {
        // 登录 失败
        System.out.println("登录 失败");
    }
    然后用户名回显
    用户名：<input type="text" name="username" value="${cookie.username.value}">
```

|-- 创建 Cookie

1.创建服务端，CookieServlet，服务端内部 createCookie()方法

创建 cookie,Cookie cookie =new Cookie(key1", "value1")

通知客户端保存 response.addCookie(cookie)

服务端给客户端返回响应头

2.客户端看响应头是否带有 cookie 信息

```
Cookie cookie = new Cookie("key4", "value4");
resp.addCookie(cookie);
```

|--服务器如何获取 Cookie

req.getCookies()；得到一个 cookie 的数组返回

```
Cookie[] cookies = req.getCookies();
```

|-- Cookie 值的修改

创建同样的 key 值，value 覆盖

找到对应的 cookie，在 setValue()方法设置新的值

```
if (name == null || cookies == null || cookies.length == 0) {
    return null;
}
for (Cookie cookie : cookies) {
    if (name.equals(cookie.getName())) {
        return cookie;
    }
}

if (cookie != null) {
    cookie.setValue("newValue2");
    resp.addCookie(cookie);
}
```

|-- Cookie 生命控制

setMaxAge()，设置生命周期，既什么时候被销毁

```
Cookie cookie = new Cookie("life3600", "life3600");
cookie.setMaxAge(60 * 60); // 设置 Cookie 一小时之后被删除。无效
resp.addCookie(cookie);
```

|-- Cookie 有效路径 Path 的设置

过滤哪些 Cookie 可以发送给服务器。哪些不发。

Cookie 虽好，但是存在于用户端，存储的尺寸大小有限，用户可见，并可以随意的修改，很不安全。那如何又要安全，又可以方便的全局读取信息呢？于是，这个时候，一种新的存储会话机制：Session 诞生了。

会话：打开浏览器到关闭浏览器的这个过程

Session 会话:每次我们访问一个页面，会自动生成一个 session_id 来标注是这次会话的唯一 ID，同时也会自动往 cookie 里写入，当这次会话没结束，再次访问的时候，服务器会去读取这个 cookie 是否有值有没过期，如果能够读取到，则继续用这个 session_id，如果没有，就会新生成一个 session_id，同时生成新的 cookie。

|--创建
request.getSession()
第一次调用是：创建 Session 会话
之后调用都是：获取前面创建好的 Session 会话对象。
isNew(); 判断到底是不是刚创建出来的（新的）
getId() 得到 Session 的会话 id 值。

|-- Session 域数据的存取
req.getSession().setAttribute("key1", "value1");
Object attribute = req.getSession().getAttribute("key1");

|--Session 生命周期控制，默认的超时时间长为 30 分钟。
setMaxInactiveInterval(int interval)，设置生命周期
invalidate() 让当前 Session 会话马上超时无效

-->

<!--
Filter 过滤器

1. JavaWeb 的三大组件之一。Servlet 程序、Listener 监听器、Filter 过滤器
2、Filter 过滤器它是 JavaEE 的规范。也就是接口
3、Filter 过滤器它的作用是：拦截请求，过滤响应

|-- Filter 过滤器的使用步骤：
1、编写一个类去实现 Filter 接口,注意是 javax.servlet 下的 Filter 接口
public class adminFilter implements Filter

2、实现过滤方法 doFilter()

@Override

public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain) throws IOException, ServletException {

//HttpServletRequest 继承了 ServletRequest ，比 ServletRequest 有更多的方法。如 getSession() 等方法。

HttpServletRequest httpServletRequest = (HttpServletRequest) servletRequest;

Object user = httpServletRequest.getSession().getAttribute("user");

if(user==null){

//用户未登录，跳转到登录界面

servletRequest.getRequestDispatcher("login.jsp").forward(servletRequest,servletResponse);

}else{

//用户已登录，放行

filterChain.doFilter(servletRequest,servletResponse);

}

}

3、到 web.xml 中去配置 Filter 的拦截路径

```
<filter>
    <filter-name>AdminFilter</filter-name>
    <filter-class>com.myFilter.filter.AdminFilter</filter-class>
</filter>

<filter-mapping>
    <filter-name>AdminFilter</filter-name>
    表示拦截所有访问工程路径下的 admin 包下的所有文件的请求
    拦截路径可以：精准匹配、目录匹配、后缀名匹配，可以有多个拦截路径
    <url-pattern>/admin/*</url-pattern>
    <url-pattern>/manager/*</url-pattern>
</filter-mapping>
```

|-- FilterConfig 类

是 Filter 过滤器的配置文件类，作用是获取 filter 过滤器的配置内容

|-- FilterChain 过滤器链

支持多个 Filter 组合拦截同一个文件请求，执行的优先顺序是由他们在 web.xml 中从上到下配置的顺序

ThreadLocal：

本质是 key 为线程，变量为 value 的一个 map

Web 应用程序就是典型的多任务应用，每个用户请求页面时，我们都会创建一个任务，然后，

通过线程池去执行这些任务。往往一个方法又会调用其他很多方法，这样会导致 User 传递到所有地方。这种在一个线程中，横跨若干方法调用，需要传递的对象，我们通常称之为上下文（Context），它是一种状态，可以是用户身份、任务信息等。给每个方法增加一个 context 参数非常麻烦，而且有些时候，如果调用链有无法修改源码的第三方库，User 对象就传不进去了。Java 标准库提供了一个特殊的 ThreadLocal，它可以在一个线程中传递同一个对象。设置一个 User 实例关联到 ThreadLocal 中，在移除之前，所有方法都可以随时获取到该 User 实例：

ThreadLocal 是一个关于创建线程局部变量的类。

通常情况下，我们创建的变量是可以被任何一个线程访问并修改的。而使用 ThreadLocal 创建的变量只能被当前线程访问，其他线程则无法访问和修改。

使用场景

1. 实现单个线程单例以及单个线程上下文信息存储，比如交易 id 等
2. 实现线程安全，非线程安全的对象使用 ThreadLocal 之后就会变得线程安全
3. 承载一些线程相关的数据，避免在方法中来回传递参数

使用：

使用 ThreadLocal 来确保所有 dao 操作都在同一个 Connection 连接对象中完成

加上 jdbc 中的事务，要么全部实现要么全部都失败，确保数据的安全与一致性。

特别注意：

ThreadLocal 一定要在 finally 中清除，这是因为当前线程执行完相关代码后，很可能会被重新放入线程池中，如果 ThreadLocal 没有被清除，该线程执行其他代码时，会把上一次的状态带进去。

-->

<!—

JSON：一种数据交换的格式

数据交换指的是客户端和服务器之间业务数据的传递格式，轻量级指的是跟 xml 做比较。

实质是键值对，{"key","值"}
json 本身是一个对象。
json 中的 key 我们可以理解为是对象中的一个属性。
json 中的 key 访问就跟访问对象的属性一样： json 对象.key

json 的存在有两种形式。
一种是：对象的形式存在，我们叫它 json 对象。
一种是：字符串的形式存在，我们叫它 json 字符串。

JSON 在 java 中的使用
1.导入 Gson 包，里面提供字符串与 json 对象相互转化的方法

2.toJson 方法可以把 java 对象转换成为 json 字符串
String personJsonString = gson.toJson(person);

3.fromJson 把 json 字符串转换回 Java 对象
Person person1 = gson.fromJson(personJsonString, Person.class);

4.List 或 Map 的互转，需要额外用到 TypeToken 类获取内部的类型
Map<Integer,Person> personMap2 = gson.fromJson(personMapJsonString, new TypeToken<HashMap<Integer,Person>>(){}.getType());

AJAX
【文章：https://www.cnblogs.com/qianguyihao/p/8485028.html】
ajax 是一种浏览器通过 js 异步发起请求，局部更新页面的技术。
同步就是整个处理过程顺序执行，执行的流程不能跨越。 异步则是只是发送了调用的指令，无需被调用的方法完全执行完毕；就可以继续执行下面的流程。

一个 Form 的提交，一旦用户点击“Submit”按钮，表单开始提交，浏览器就会刷新页面，然后在新页面里告诉操作是成功了还是失败了。如果不幸由于网络太慢或者其他原因，就会得到一个 404 页面。

这就是 Web 的运作原理：一次 HTTP 请求对应一个页面。

如果要让用户留在当前页面中，同时发出新的 HTTP 请求，就必须用 JavaScript 发送这个新请求，接收到数据后，再用 JavaScript 更新页面，这样一来，用户就感觉自己仍然停留在当前页面，但是数据却可以不断地更新。这就是 AJAX 做的事情

|-- JavaScript 中的原生 AJAX 创建操作
发送一个 HTTP 请求，需要创建 XMLHttpRequest 对象，打开新的 URL，最后发送请求。
1.创建 XMLHttpRequest
var xmlhttprequest = new XMLHttpRequest();
2.调用 open 方法设置请求参数
xmlhttprequest.open("GET","响应地址",同步还是异步)
3.AJAX 请求是异步执行的，也就是说，要通过回调函数获得响应。
发送前绑定 onreadystatechange 事件，处理请求完成后的操作。在回调函数中，通过 readyState === 4 判断请求是否完成，如果已完成，再根据 status === 200 判断是否是一个成功的响应。
function(data)中的参数就是 HTTP 请求中要传递的参数对象
//状态发生变化时，函数被回调
request.onreadystatechange = function () {
if (request.readyState === 4) { // 成功完成

```
                // 判断响应结果:
                if (request.status === 200) {
    4.调用 send 方法发送请求
        xmlhttprequest.send();


|-- jQuery 中的 AJAX 请求，进行新的封装
    $.ajax 方法
    $.get 方法和$.post 方法
    $.getJSON 方法

        url     表示请求的地址
        type    表示请求的类型 GET 或 POST 请求
        data    表示发送给服务器的数据
                格式有两种：
                    一：name=value&name=value
                    二：{key:value}
        success 请求成功，响应的回调函数
        dataType 响应的数据类型
                常用的数据类型有：
                    text 表示纯文本
                    xml 表示 xml 数据
                    json 表示 json 对象

    $("#ajaxBtn").click(function(){
        $.ajax({
            url:"http://localhost:8080/16_json_ajax_i18n/ajaxServlet",
            data:{action:"jQueryAjax"},
            type:"GET",
            success:function (data) {
                $("#msg").html(data.name);
            },
            dataType : "json"
        });
    });

    // ajax--get 请求
    $("#getBtn").click(function(){
        $.get("http://localhost:8080/16_json_ajax_i18n/ajaxServlet","action=jQueryGet",function (data) {
            $("#msg").html(" get 编号：" + data.id + " , 姓名：" + data.name);
        },"json");
      });

    // ajax--post 请求
```

```
    $("#postBtn").click(function(){
        $.post("http://localhost:8080/16_json_ajax_i18n/ajaxServlet","action=jQueryPost",function (data){
            $("#msg").html(" post 编号：" + data.id + " , 姓名：" + data.name);},"json");
    });

    // ajax--getJson 请求
    $("#getJSONBtn").click(function(){

$.getJSON("http://localhost:8080/16_json_ajax_i18n/ajaxServlet","action=jQueryGetJSON",function(data) {
            $("#msg").html(" getJSON 编号：" + data.id + " , 姓名：" + data.name);
        });
    });
```

表单序列化 serialize()可以把表单中所有表单项的内容都获取到，并以 name=value&name=value 的形式进行拼接

```
|--Servlet 与前端 ajax 的数据交互:
    1. 前端获取数据通过 ajax 的异步传输传至 servlet，类似 http 请求 servlet
    2. servlet 处理数据后，将数据转换成 json 格式，通过 response 回传至前端页面。
        Gson gson = new Gson();//利用 gson 将数据转换成 json 格式
        String json = gson.toJson(map);
        response.getWriter().write(json 数据);
    3. 传回的 json 数据触发 ajax 中的回调函数
    function(data){
        data 即为传回的 json 数据
    }



-->
```