

公告

wiki和教程：www.5xclass.cn

免费教学视频：[B站：凸头统治地球](#)

高级专题教程：[网易云课堂：武沛齐](#)

聊技术，加武Sir微信

 武沛齐



扫一扫上面的二维码图案，加我微信

昵称： 武沛齐

园龄： 11年9个月

粉丝： 12988

关注： 44

+加关注

我的标签

- Python(17)
- ASP.NET MVC(15)
- python之路(7)
- Tornado源码分析(5)
- 每天一道Python面试题(5)
- crm项目(4)
- 面试都在问什么？(2)
- Python企业面试题讲解(1)
- Python面试315题(1)
- Python开源组件 - Tyrion(1)

积分与排名

积分 - 492822

排名 - 1343

随笔分类

- JavaScript(1)
- MVC(15)
- Python(17)
- 面试都在问什么系列？【图】(2)
- 其他(37)

随笔 - 140 文章 - 164 评论 - 1087 阅读 - 269万

Python开发【第十八篇】：MySQL（二）

视图

视图是一个虚拟表（非真实存在），其本质是【根据SQL语句获取动态的数据集，并为其命名】，用户使用只需使用【名称】即可获取结果集，并可以将其当作表来使用。

```
SELECT
*
FROM
(
    SELECT
        nid,
        NAME
    FROM
        tbl
    WHERE
        nid > 2
) AS A
WHERE
    A. NAME > 'alex';
```

1、创建视图

```
--格式：CREATE VIEW 视图名称 AS SQL语句
CREATE VIEW v1 AS
SELET nid,
    name
FROM
    A
WHERE
    nid > 4
```

2、删除视图

```
--格式：DROP VIEW 视图名称

DROP VIEW v1
```

3、修改视图

```
-- 格式：ALTER VIEW 视图名称 AS SQL语句
```

企业面试题及答案(1)
请求响应(6)
设计模式(9)
微软C#(34)

随机档案

- 2020年6月(1)
- 2020年5月(1)
- 2019年11月(1)
- 2019年10月(1)
- 2019年9月(4)
- 2018年12月(1)
- 2018年8月(1)
- 2018年5月(2)
- 2018年4月(1)
- 2017年8月(1)
- 2017年5月(1)
- 2017年3月(1)
- 2016年10月(1)
- 2016年7月(1)
- 2015年10月(1)
- 更多

相册

git(14)

最新评论

- 1. Re:django channels
2
--长街旧人,...
- 2. Re:Python生成随机验证码
多亏楼上评论 半天找不到这个文章 谢谢武老师 谢谢楼上
--sugaryy
- 3. Re:Python生成随机验证码
来下字体文件的
--Hinata-
- 4. Re:Python之路【第十七篇】： Django【进阶篇】
<script>
alert(123);
</script>
--柠檬の夏天
- 5. Re:【第4题】 什么是https
最好理解的解说, 赞
--华丽的肉虫子

```
ALTER VIEW v1 AS
SELET A.nid,
      B. NAME
FROM
      A
LEFT JOIN B ON A.id = B.nid
LEFT JOIN C ON A.id = C.nid
WHERE
      A.id > 2
AND C.nid < 5
```

4、使用视图

使用视图时，将其当作表进行操作即可，由于视图是虚拟表，所以无法使用其对真实表进行创建、更新和删除操作，仅能做查询用。

```
select * from v1
```

触发器

对某个表进行【增/删/改】操作的前后如果希望触发某个特定的行为时，可以使用触发器，触发器用于定制用户对表的行进行【增/删/改】前后的行为。

1、创建基本语法

```
# 插入前
CREATE TRIGGER tri_before_insert_tb1 BEFORE INSERT ON tb1 FOR EACH ROW
BEGIN
    ...
END

# 插入后
CREATE TRIGGER tri_after_insert_tb1 AFTER INSERT ON tb1 FOR EACH ROW
BEGIN
    ...
END

# 删除前
CREATE TRIGGER tri_before_delete_tb1 BEFORE DELETE ON tb1 FOR EACH ROW
BEGIN
    ...
END

# 删除后
CREATE TRIGGER tri_after_delete_tb1 AFTER DELETE ON tb1 FOR EACH ROW
BEGIN
    ...
END

# 更新前
CREATE TRIGGER tri_before_update_tb1 BEFORE UPDATE ON tb1 FOR EACH ROW
BEGIN
    ...
END

# 更新后
CREATE TRIGGER tri_after_update_tb1 AFTER UPDATE ON tb1 FOR EACH ROW
BEGIN
    ...
END
```

END



```
delimiter //
CREATE TRIGGER tri_before_insert_tb1 BEFORE INSERT ON tb1 FOR EACH ROW
BEGIN

IF NEW. NAME == 'alex' THEN

    INSERT INTO tb2 (NAME)
VALUES

    ('aa')

END
END//
delimiter ;
```



```
delimiter //
CREATE TRIGGER tri_after_insert_tb1 AFTER INSERT ON tb1 FOR EACH ROW
BEGIN

    IF NEW. num = 666 THEN

        INSERT INTO tb2 (NAME)
VALUES

            ('666'),
            ('666') ;

    ELSEIF NEW. num = 555 THEN

        INSERT INTO tb2 (NAME)
VALUES

            ('555'),
            ('555') ;

    END IF;
END//
delimiter ;
```



特别的：NEW表示即将插入的数据行，OLD表示即将删除的数据行。

2、删除触发器



```
DROP TRIGGER tri_after_insert_tb1;
```

3、使用触发器

触发器无法由用户直接调用，而知由于对表的【增/删/改】操作被动引发的。



```
insert into tb1(num) values(666)
```

存储过程

存储过程是一个SQL语句集合，当主动去调用存储过程时，其中内部的SQL语句会按照逻辑执行。

1、创建存储过程



```
-- 创建存储过程

delimiter //
create procedure p1()
BEGIN
    select * from t1;
END//
delimiter ;
```

```
-- 执行存储过程
```

```
call p1()
```



对于存储过程，可以接收参数，其参数有三类：

- in 仅用于传入参数用
- out 仅用于返回值用
- inout 既可以传入又可以当作返回值



```
-- 创建存储过程
delimiter \\\ncreate procedure p1(\n    in i1 int,\n    in i2 int,\n    inout i3 int,\n    out r1 int\n)\nBEGIN\n    DECLARE temp1 int;\n    DECLARE temp2 int default 0;\n\n    set temp1 = 1;\n\n    set r1 = i1 + i2 + temp1 + temp2;\n\n    set i3 = i3 + 100;\n\nend\\\ndelimiter ;
```

```
-- 执行存储过程\nset @t1 =4;\nset @t2 = 0;\nCALL p1 (1, 2 ,@t1, @t2);\nSELECT @t1,@t2;
```



```
delimiter //\ncreate procedure p1()\nbegin\n    select * from v1;\nend //\ndelimiter ;
```





```
delimiter //  
  
create procedure p2(  
    in n1 int,  
    inout n3 int,  
    out n2 int,  
)  
begin  
    declare temp1 int ;  
    declare temp2 int default 0;  
  
    select * from v1;  
    set n2 = n1 + 100;  
    set n3 = n3 + n1 + 100;  
end //  
delimiter ;
```



```
delimiter \\  
  
create PROCEDURE p1(  
    OUT p_return_code tinyint  
)  
BEGIN  
    DECLARE exit handler for sqlexception  
    BEGIN  
        -- ERROR  
        set p_return_code = 1;  
        rollback;  
    END;  
  
    DECLARE exit handler for sqlwarning  
    BEGIN  
        -- WARNING  
        set p_return_code = 2;  
        rollback;  
    END;  
  
    START TRANSACTION;  
    DELETE from tb1;  
    insert into tb2(name) values('seven');  
    COMMIT;  
  
    -- SUCCESS  
    set p_return_code = 0;  
  
    END\\  
delimiter ;
```



```
delimiter //  
  
create procedure p3()  
begin  
    declare ssid int; -- 自定义变量1  
    declare ssname varchar(50); -- 自定义变量2  
    DECLARE done INT DEFAULT FALSE;
```

```
DECLARE my_cursor CURSOR FOR select sid,sname
DECLARE CONTINUE HANDLER FOR NOT FOUND SET done=TRUE;

open my_cursor;
xxoo: LOOP
    fetch my_cursor into ssid,ssname;
    if done then
        leave xxoo;
    END IF;
    insert into teacher(tname) values(ssname);
end loop xxoo;
close my_cursor;
end //
delimiter ;
```



```
delimiter \
CREATE PROCEDURE p4 (
    in nid int
)
BEGIN
    PREPARE prod FROM 'select * from student where id=@nid';
    EXECUTE prod USING @nid;
    DEALLOCATE prepare prod;
END\
delimiter ;
```



2、删除存储过程



```
drop procedure proc_name;
```

3、执行存储过程



```
-- 无参数
call proc_name()

-- 有参数, 全in
call proc_name(1,2)

-- 有参数, 有in, out, inout
set @t1=0;
set @t2=3;
call proc_name(1,2,@t1,@t2)
```



```
#!/usr/bin/env python
# -*- coding:utf-8 -*-
import pymysql
```

```
conn = pymysql.connect(host='127.0.0.1', port=3306, user='root', passwd='123456',
                        cursor=pymysql.cursors.DictCursor)

# 执行存储过程
cursor.callproc('p1', args=(1, 22, 3, 4))

# 获取执行完存储的参数
cursor.execute("select @_p1_0,@_p1_1,@_p1_2,@_p1_3")
result = cursor.fetchall()

conn.commit()
cursor.close()
conn.close()

print(result)
```



MySQL中提供了许多内置函数，例如：



- CHAR_LENGTH(str)**

返回值为字符串str 的长度，长度的单位为字符。一个多字节字符算作一个单字符。
对于一个包含五个二字节字符集， LENGTH() 返回值为 10， 而CHAR_LENGTH() 的返回值为5。
- CONCAT(str1,str2,...)**

字符串拼接
如有任何一个参数为NULL ， 则返回值为 NULL。
- CONCAT_WS(separator,str1,str2,...)**

字符串拼接（自定义连接符）
CONCAT_WS() 不会忽略任何空字符串。（然而会忽略所有的 NULL）。
- CONV(N,from_base,to_base)**

进制转换
例如：
SELECT CONV('a',16,2); 表示将 a 由16进制转换为2进制字符串表示
- FORMAT(X,D)**

将数字X 的格式写为'#,###,###.##',以四舍五入的方式保留小数点后 D 位， 并
例如：
SELECT FORMAT(12332.1,4); 结果为: '12,332.1000'
- INSERT(str,pos,len,newstr)**

在str的指定位置插入字符串
pos: 要替换位置其实位置
len: 替换的长度
newstr: 新字符串
特别的：
如果pos超过原字符串长度，则返回原字符串
如果len超过原字符串长度，则由新字符串完全替换
- INSTR(str,substr)**

返回字符串 str 中子字符串的第一个出现位置。
- LEFT(str,len)**

返回字符串str 从开始的len位置的子序列字符。
- LOWER(str)**

变小写
- UPPER(str)**

变大写

`LTRIM(str)`
返回字符串 `str`，其引导空格字符被删除。

`RTRIM(str)`
返回字符串 `str`，结尾空格字符被删去。

`SUBSTRING(str,pos,len)`
获取字符串子序列

`LOCATE(substr,str,pos)`
获取子序列索引位置

`REPEAT(str,count)`
返回一个由重复的字符串`str` 组成的字符串，字符串`str`的数目等于`count`。
若 `count <= 0`,则返回一个空字符串。
若`str` 或 `count` 为 `NULL`，则返回 `NULL`。

`REPLACE(str,from_str,to_str)`
返回字符串`str` 以及所有被字符串`to_str`替代的字符串`from_str`。

`REVERSE(str)`
返回字符串 `str`，顺序和字符顺序相反。

`RIGHT(str,len)`
从字符串`str` 开始，返回从后边开始`len`个字符组成的子序列

`SPACE(N)`
返回一个由`N`空格组成的字符串。

`SUBSTRING(str,pos)`，`SUBSTRING(str FROM pos)` `SUBSTRING(str,pos,len)`
不带有`len` 参数的格式从字符串`str`返回一个子字符串，起始于位置 `pos`。带有`len`

```
mysql> SELECT SUBSTRING('Quadratically',5);  
-> 'ratically'
```

```
mysql> SELECT SUBSTRING('foobarbar' FROM 4);  
-> 'barbar'
```

```
mysql> SELECT SUBSTRING('Quadratically',5,6);  
-> 'ratica'
```

```
mysql> SELECT SUBSTRING('Sakila', -3);  
-> 'ila'
```

```
mysql> SELECT SUBSTRING('Sakila', -5, 3);  
-> 'aki'
```

```
mysql> SELECT SUBSTRING('Sakila' FROM -4 FOR 2);  
-> 'ki'
```

`TRIM([{BOTH | LEADING | TRAILING} [remstr] FROM] str)` `TRIM(remstr)`
返回字符串 `str`，其中所有`remstr` 前缀和/或后缀都被删除。若分类符`BOTH`、

```
mysql> SELECT TRIM(' bar ');  
-> 'bar'
```

```
mysql> SELECT TRIM(LEADING 'x' FROM 'xxxbarxxx');  
-> 'barxxx'
```

```
mysql> SELECT TRIM(BOTH 'x' FROM 'xxxbarxxx');  
-> 'bar'
```

```
mysql> SELECT TRIM(TRAILING 'xyz' FROM 'barxyz');  
-> 'barx'
```



更多函数：[中文猛击这里](#) OR [官方猛击这里](#)

1、自定义函数

```
delimiter \\  
create function f1(  
    i1 int,  
    i2 int)  
returns int  
BEGIN  
    declare num int;  
    set num = i1 + i2;  
    return(num);  
END \\  
delimiter ;
```

2、删除函数

```
drop function func_name;
```

3、执行函数

```
# 获取返回值  
declare @i VARCHAR(32);  
select UPPER('alex') into @i;  
SELECT @i;  
  
# 在查询中使用  
select f1(i1,nid) ,name from tb2;
```

事务

事务用于将某些操作的多个SQL作为原子性操作，一旦有某一个出现错误，即可回滚到原来的状态，从而保证数据库数据完整性。

```
delimiter \\  
create PROCEDURE p1(  
    OUT p_return_code tinyint  
)  
BEGIN  
    DECLARE exit handler for sqlexception  
    BEGIN  
        -- ERROR  
        set p_return_code = 1;  
        rollback;  
    END;  
  
    DECLARE exit handler for sqlwarning  
    BEGIN  
        -- WARNING  
        set p_return_code = 2;  
        rollback;
```

```
END;

START TRANSACTION;
DELETE from tbl;
insert into tb2(name) values ('seven');
COMMIT;

-- SUCCESS
set p_return_code = 0;

END\\
delimiter ;
```

```
1 set @i =0;
2 call p1(@i);
3 select @i;
```

索引

索引，是数据库中专门用于帮助用户快速查询数据的一种数据结构。类似于字典中的目录，查找字典内容时可以根据目录查找到数据的存放位置，然后直接获取即可。

1						30			
2									
3			10					40	
4									
5	5		15		35			66	
6									
7	1	6	11	19	21	39	55	100	

MySQL中常见索引有：

- 普通索引
- 唯一索引
- 主键索引
- 组合索引

1、普通索引

普通索引仅有一个功能：加速查询

```
create table in1(
    nid int not null auto_increment primary key,
    name varchar(32) not null,
    email varchar(64) not null,
    extra text,
    index ix_name (name)
)
```

```
create index index_name on table_name(column_name)
```

```
drop index_name on table_name;
```

```
show index from table_name;
```

注意：对于创建索引时如果是BLOB 和 TEXT 类型，必须指定length。

```
create index ix_extra on in1(extra(32));
```

2、唯一索引

唯一索引有两个功能：加速查询 和 唯一约束（可含null）

```
create table in1(  
    nid int not null auto_increment primary key,  
    name varchar(32) not null,  
    email varchar(64) not null,  
    extra text,  
    unique ix_name (name)  
)
```

```
create unique index 索引名 on 表名(列名)
```

```
drop unique index 索引名 on 表名
```

3、主键索引

主键有两个功能：加速查询 和 唯一约束（不可含null）

```
create table in1(  
    nid int not null auto_increment primary key,  
    name varchar(32) not null,  
    email varchar(64) not null,  
    extra text,  
    index ix_name (name)  
)  
  
OR  
  
create table in1(  
    nid int not null auto_increment,  
    name varchar(32) not null,  
    email varchar(64) not null,  
    extra text,  
    primary key(nid),  
    index ix_name (name)  
)
```

```
alter table 表名 add primary key(列名);
```

```
alter table 表名 drop primary key;  
alter table 表名 modify 列名 int, drop primary key;
```

4、组合索引

组合索引是将n个列组合成一个索引

其应用场景为：频繁的同时使用n列来进行查询，如：where n1 = 'alex' and n2 = 666。

```
create table in3(  
    nid int not null auto_increment primary key,  
    name varchar(32) not null,  
    email varchar(64) not null,  
    extra text  
)
```

```
create index ix_name_email on in3(name,email);
```

如上创建组合索引之后，查询：

- name and email -- 使用索引
- name -- 使用索引
- email -- 不使用索引

注意：对于同时搜索n个条件时，组合索引的性能好于多个单一索引合并。

其他

1、条件语句

```
delimiter \\  
CREATE PROCEDURE proc_if ()  
BEGIN  
  
    declare i int default 0;  
    if i = 1 THEN  
        SELECT 1;  
    ELSEIF i = 2 THEN  
        SELECT 2;  
    ELSE  
        SELECT 7;  
    END IF;  
  
END\\  
delimiter ;
```

2、循环语句

```
delimiter \\  
CREATE PROCEDURE proc_while ()  
BEGIN  
  
    DECLARE num INT ;  
    SET num = 0 ;  
    WHILE num < 10 DO  
        SELECT  
            num ;  
        SET num = num + 1 ;  
    END WHILE ;
```

```
END\\  
delimiter ;
```



```
delimiter \\  
CREATE PROCEDURE proc_repeat ()  
BEGIN  
  
    DECLARE i INT ;  
    SET i = 0 ;  
    repeat  
        select i;  
        set i = i + 1;  
        until i >= 5  
    end repeat;  
  
END\\  
delimiter ;
```



```
BEGIN  
  
declare i int default 0;  
loop_label: loop  
  
    set i=i+1;  
    if i<8 then  
        iterate loop_label;  
    end if;  
    if i>=10 then  
        leave loop_label;  
    end if;  
    select i;  
end loop loop_label;  
  
END
```



3、动态执行SQL语句



```
delimiter \\  
DROP PROCEDURE IF EXISTS proc_sql \\  
CREATE PROCEDURE proc_sql ()  
BEGIN  
    declare p1 int;  
    set p1 = 11;  
    set @p1 = p1;  
  
    PREPARE prod FROM 'select * from tb2 where nid > ?';  
    EXECUTE prod USING @p1;  
    DEALLOCATE prepare prod;  
  
END\\
```

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956 957 958 959 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979 980 981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997 998 999 1000

delimiter ;



作者: 武沛齐
出处: <http://www.cnblogs.com/wupeiqi/>
本文版权归作者和博客园共有, 欢迎转载, 但未经作者同意必须保留此段声明, 且在文章页面明显位置给出原文连接。

好文要顶

关注我

收藏该文

微信分享



武沛齐
粉丝 - 12988 关注 - 44

19

0

+加关注

posted @ 2016-07-28 07:12 武沛齐 阅读(28992) 评论(3) 编辑 收藏 举报

会员力量, 点亮园子希望

刷新页面 返回顶部

登录后才能查看或发表评论, 立即 [登录](#) 或者 [逛逛](#) 博客园首页

- 【推荐】博客园商业化之路-开篇: 开源的脚步, 商业化的出路
- 【推荐】园子周边第二季: 更大的鼠标垫, 没有logo的鼠标垫
- 【推荐】阿里云市场联合博客园推出开发者商店, 欢迎关注
- 【推荐】会员力量, 点亮园子希望, 期待您升级成为园子会员



编辑推荐:

- [async/await 贴脸输出, 这次你总该明白了](#)
- [WPF 随笔收录-实时绘制心率曲线](#)
- [「布局进阶」巧用 :has & drop-shadow 实现复杂布局效果](#)
- [\[Nano Framework ESP32篇\] WS2812 彩色灯带实验](#)
- [经过腾讯云这波故障, 我想表扬的点和学到的职场保命法则](#)

阿里云

由阿里云市场和博客园联合提供

开发者商店

AI、API、基础软件及服务,
产品折扣、先试后买!

立即查看 →

阅读排行:

- [在Windows电脑上快速运行AI大语言模型-Llama3](#)
- [.NET开源免费的跨平台框架 - MAUI \(附学习资料\)](#)
- [使用纯c#在本地部署多模态模型, 让本地模型也可以理解图像](#)
- [本地部署Llama3-8B/70B 并进行逻辑推理测试](#)
- [如何将 ASP.NET Core MVC 项目的视图分离到另一个项目](#)