

AnnsShadoW

博客园

首页

新随笔

联系

订阅

管理

# 步步深入：MySQL架构总览->查询执行流程->SQL解析顺序

**前言：**

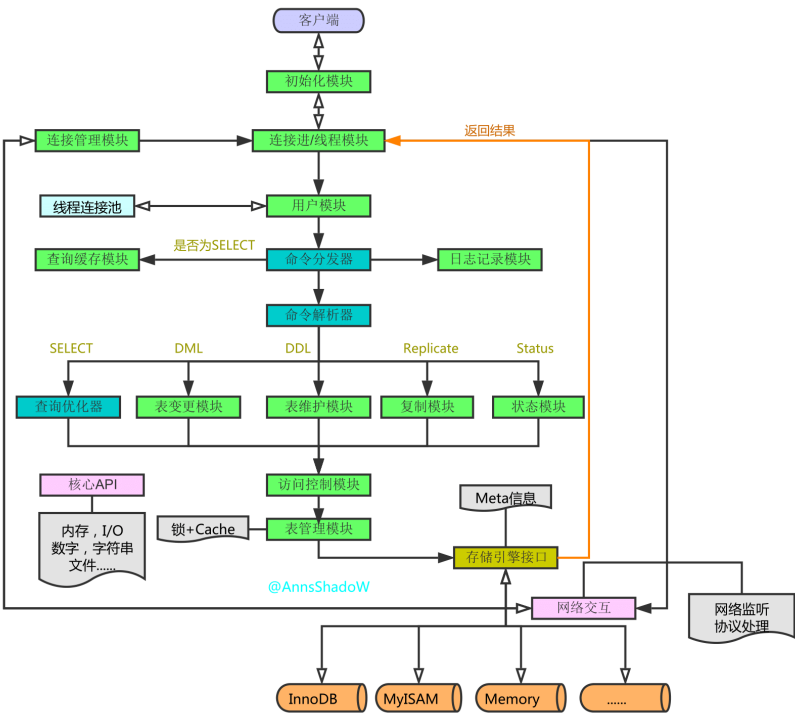
一直是想知道一条SQL语句是怎么被执行的，它执行的顺序是怎样的，然后查看总结各方资料，就有了下面这一篇博文了。

本文将从MySQL总体架构--->查询执行流程--->语句执行顺序来探讨一下其中的知识。

## 一、MySQL架构总览：

架构最好看图，再配上必要的说明文字。

下图根据参考书籍中一图为原本，再在其上添加上了自己的理解。



从上图中我们可以看到，整个架构分为两层，上层是MySQLD的被称为的‘SQL Layer’，下层是各种各样对上提供接口的存储引擎，被称为‘Storage Engine Layer’。其它各个模块和组件，从名字上就可以简单了解到它们的作用，这里就不再赘述了。

### 公告

昵称： AnnsShadoW  
园龄： 8年1个月  
粉丝： 49  
关注： 8  
[+加关注](#)

< 2023年2月 >

日	一	二	三	四	五	六
29	30	31	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	1	2	3	4
5	6	7	8	9	10	11

### 搜索

找找看

谷歌搜索

### 我的标签

linux(3)

vim(1)

life(1)

二、查询执行流程

下面再向前走一些，容我根据自己的认识说一下查询执行的流程是怎样的：

1.连接

- 1.1客户端发起一条Query请求，监听客户端的‘连接管理模块’接收请求
- 1.2将请求转发到‘连接进/线程模块’
- 1.3调用‘用户模块’来进行授权检查
- 1.4通过检查后，‘连接进/线程模块’从‘线程连接池’中取出空闲的被缓存的连接线程和客户端请求对接，如果失败则创建一个新的连接请求

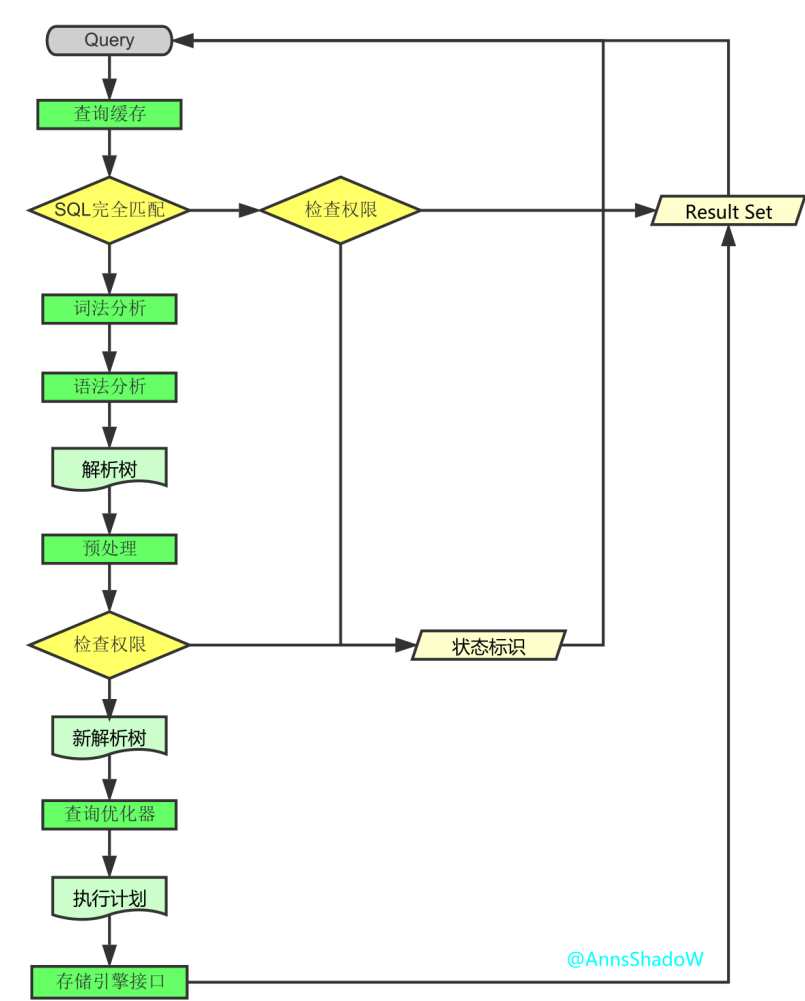
2.处理

- 2.1先查询缓存，检查Query语句是否完全匹配，接着再检查是否具有权限，都成功则直接取数据返回
- 2.2上一步有失败则转交给‘命令解析器’，经过词法分析，语法分析后生成解析树
- 2.3接下来是预处理阶段，处理解析器无法解决的语义，检查权限等，生成新的解析树
- 2.4再转交给对应的模块处理
- 2.5如果是SELECT查询还会经由‘查询优化器’做大量的优化，生成执行计划
- 2.6模块收到请求后，通过‘访问控制模块’检查所连接的用户是否有访问目标表和目标字段的权限
- 2.7有则调用‘表管理模块’，先是查看table cache中是否存在，有则直接对应的表和获取锁，否则重新打开表文件
- 2.8根据表的meta数据，获取表的存储引擎类型等信息，通过接口调用对应的存储引擎处理
- 2.9上述过程中产生数据变化的时候，若打开日志功能，则会记录到相应二进制日志文件中

3.结果

- 3.1Query请求完成后，将结果集返回给‘连接进/线程模块’
- 3.2返回的也可以是相应的状态标识，如成功或失败等
- 3.3‘连接进/线程模块’进行后续的清理工作，并继续等待请求或断开与客户端的连接

一图小总结



note(1)

english(1)

MapReduce(1)

php(1)

随笔分类 (19)

DataStructure && Algorithm(1)

C/C++(2)

CS(1)

DataBase(3)

Hadoop(1)

Life(4)

Linux(4)

PHP(1)

Shell(2)

随笔档案 (19)

2016年9月(3)

2016年4月(1)

2016年3月(1)

2016年1月(3)

2015年12月(3)

2015年11月(8)

三、SQL解析顺序

接下来再走一步，让我们看看一条SQL语句的前世今生。  
首先看一下示例语句



```
SELECT DISTINCT
    < select_list >
FROM
    < left_table > < join_type >
JOIN < right_table > ON < join_condition >
WHERE
    < where_condition >
GROUP BY
    < group_by_list >
HAVING
    < having_condition >
ORDER BY
    < order_by_condition >
LIMIT < limit_number >
```



然而它的执行顺序是这样的



```
1 FROM <left_table>
2 ON <join_condition>
3 <join_type> JOIN <right_table>
4 WHERE <where_condition>
5 GROUP BY <group_by_list>
6 HAVING <having_condition>
7 SELECT
8 DISTINCT <select_list>
9 ORDER BY <order_by_condition>
10 LIMIT <limit_number>
```



虽然自己没想到是这样的，不过一看还是很自然和谐的，从哪里获取，不断的过滤条件，要选择一样或不一样的，排好序，那才知道要取前几条呢。  
既然如此了，那就让我们一步步来看看其中的细节吧。

准备工作

1.创建测试数据库

```
create database testQuery
```

2.创建测试表



```
CREATE TABLE table1
(
    uid VARCHAR(10) NOT NULL,
    name VARCHAR(10) NOT NULL,
    PRIMARY KEY (uid)
) ENGINE=INNODB DEFAULT CHARSET=UTF8;

CREATE TABLE table2
(
    oid INT NOT NULL auto_increment,
    uid VARCHAR(10),
    PRIMARY KEY (oid)
) ENGINE=INNODB DEFAULT CHARSET=UTF8;
```

阅读排行榜

- 1. 步步深入：MySQL架构总览->查询执行流程->SQL解析顺序(50425)
- 2. 了解编译原理-笔记小结(6410)
- 3. [Hadoop]-从数据去重认识MapReduce (6295)
- 4. Linux换源+编译内核总结(2370)
- 5. C+命令行+方向键=简易版扫雷(2248)

评论排行榜

- 1. 步步深入：MySQL架构总览->查询执行流程->SQL解析顺序(14)
- 2. C+命令行+方向键=简易版扫雷(6)
- 3. [Hadoop]-从数据去重认识MapReduce (3)
- 4. [来自妹纸的挑战]-展开/还原多层链表 (2)
- 5. 图概PHP生命周期(1)

推荐排行榜

- 1. 步步深入：MySQL架构总览->查询执行流程->SQL解析顺序(32)
- 2. C+命令行+方向键=简易版扫雷(6)
- 3. 图概PHP生命周期(5)
- 4. [Hadoop]-从数据去重认识MapReduce (4)
- 5. Linux换源+编译内核总结(2)



3.插入数据

```
INSERT INTO table1(uid,name) VALUES('aaa','mike'),('bbb','jack'),
('ccc','mike'),('ddd','mike');

INSERT INTO table2(uid) VALUES('aaa'),('aaa'),('bbb'),('bbb'),('bbb'),
('ccc'),(NULL);
```

4.最后想要的结果



```
SELECT
    a.uid,
    count(b.oid) AS total
FROM
    table1 AS a
LEFT JOIN table2 AS b ON a.uid = b.uid
WHERE
    a.NAME = 'mike'
GROUP BY
    a.uid
HAVING
    count(b.oid) < 2
ORDER BY
    total DESC
LIMIT 1;
```



！现在开始SQL解析之旅吧！

1. FROM

当涉及多个表的时候，左边表的输出会作为右边表的输入，之后会生成一个虚拟表VT1。  
(1-J1)笛卡尔积  
计算两个相关联表的笛卡尔积(CROSS JOIN)，生成虚拟表VT1-J1。



```
mysql> select * from table1,table2;
+-----+-----+-----+-----+
| uid | name | oid | uid |
+-----+-----+-----+-----+
| aaa | mike | 1 | aaa |
| bbb | jack | 1 | aaa |
| ccc | mike | 1 | aaa |
| ddd | mike | 1 | aaa |
| aaa | mike | 2 | aaa |
| bbb | jack | 2 | aaa |
| ccc | mike | 2 | aaa |
| ddd | mike | 2 | aaa |
| aaa | mike | 3 | bbb |
| bbb | jack | 3 | bbb |
| ccc | mike | 3 | bbb |
| ddd | mike | 3 | bbb |
| aaa | mike | 4 | bbb |
| bbb | jack | 4 | bbb |
| ccc | mike | 4 | bbb |
| ddd | mike | 4 | bbb |
| aaa | mike | 5 | bbb |
| bbb | jack | 5 | bbb |
| ccc | mike | 5 | bbb |
```

最新评论

1. Re:步步深入：MySQL架构总览->查询执行流程->SQL解析顺序

很棒

--大王巡山巡山003

2. Re:步步深入：MySQL架构总览->查询执行流程->SQL解析顺序

很好

--南晓东

3. Re:步步深入：MySQL架构总览->查询执行流程->SQL解析顺序

@ProphetWC 是的 (1) FROM 子句 组装来自不同数据源的数据 (2) WHERE 子句 基于指定的条件对记录进行筛选 (3) GROUP BY 子句 将数据划分为多个分组 (4) 使用...

--学习周

4. Re:步步深入：MySQL架构总览->查询执行流程->SQL解析顺序

写的好！！

--dzx321

5. Re:步步深入：MySQL架构总览->查询执行流程->SQL解析顺序

@iame 没有反 select肯定是在having后执行的 不过在group by之后 如果有count,avg,sum,listagg等聚合函数会优先执行 然后再执行having...

--ProphetWC

```
| ddd | mike | 5 | bbb |
| aaa | mike | 6 | ccc |
| bbb | jack | 6 | ccc |
| ccc | mike | 6 | ccc |
| ddd | mike | 6 | ccc |
| aaa | mike | 7 | NULL |
| bbb | jack | 7 | NULL |
| ccc | mike | 7 | NULL |
| ddd | mike | 7 | NULL |
+-----+-----+-----+-----+
28 rows in set (0.00 sec)
```



(1-J2)ON过滤

基于虚拟表VT1-J1这一个虚拟表进行过滤，过滤出所有满足ON 谓词条件的列，生成虚拟表VT1-J2。

注意：这里因为语法限制，使用了'WHERE'代替，从中读者也可以感受到两者之间微妙的关系；

```
mysql> SELECT
-> *
-> FROM
-> table1,
-> table2
-> WHERE
-> table1.uid = table2.uid
-> ;
+-----+-----+-----+-----+
| uid | name | oid | uid |
+-----+-----+-----+-----+
| aaa | mike | 1 | aaa |
| aaa | mike | 2 | aaa |
| bbb | jack | 3 | bbb |
| bbb | jack | 4 | bbb |
| bbb | jack | 5 | bbb |
| ccc | mike | 6 | ccc |
+-----+-----+-----+-----+
6 rows in set (0.00 sec)
```



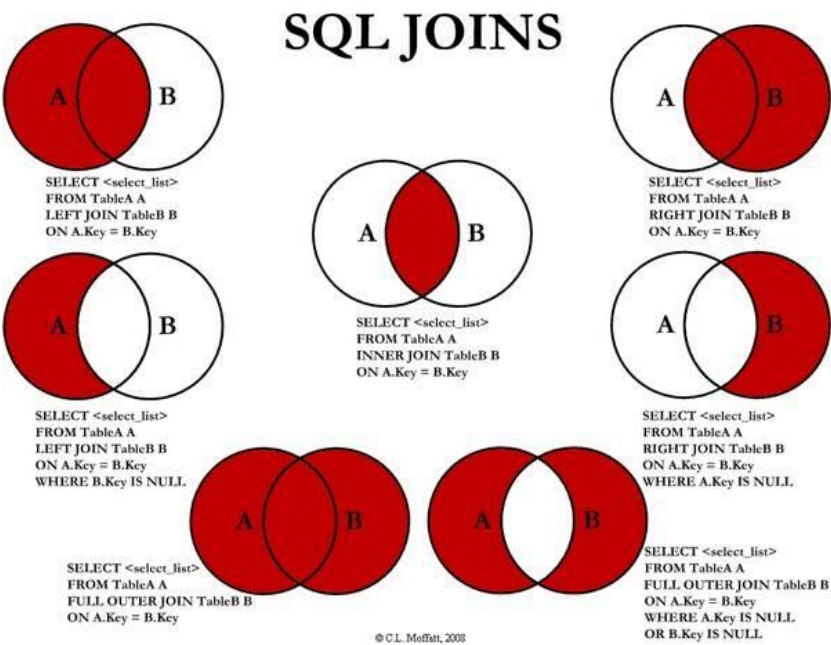
(1-J3)添加外部列

如果使用了外连接(LEFT,RIGHT,FULL)，主表（保留表）中的不符合ON条件的列也会被加入到VT1-J2中，作为外部行，生成虚拟表VT1-J3。

```
mysql> SELECT
-> *
-> FROM
-> table1 AS a
-> LEFT OUTER JOIN table2 AS b ON a.uid = b.uid;
+-----+-----+-----+-----+
| uid | name | oid | uid |
+-----+-----+-----+-----+
| aaa | mike | 1 | aaa |
| aaa | mike | 2 | aaa |
| bbb | jack | 3 | bbb |
| bbb | jack | 4 | bbb |
| bbb | jack | 5 | bbb |
| ccc | mike | 6 | ccc |
```

```
| ddd | mike | NULL | NULL |
+-----+-----+-----+
7 rows in set (0.00 sec)
```

下面从网上找到一张很形象的关于‘SQL JOINS’的解释图，如若侵犯了你的权益，请劳烦告知删除，谢谢。



2. WHERE

对VT1过程中生成的临时表进行过滤，满足WHERE子句的列被插入到VT2表中。

**注意：**  
此时因为分组，不能使用聚合运算；也不能使用SELECT中创建的别名；  
**与ON的区别：**  
如果有外部列，ON针对过滤的是关联表，主表（保留表）会返回所有的列；  
如果没有添加外部列，两者的效果是一样的；  
**应用：**  
对主表的过滤应该放在WHERE；  
对于关联表，先条件查询后连接则用ON，先连接后条件查询则用WHERE；

```
mysql> SELECT
-> *
-> FROM
-> table1 AS a
-> LEFT OUTER JOIN table2 AS b ON a.uid = b.uid
-> WHERE
-> a. NAME = 'mike';

| uid | name | oid | uid |
+-----+-----+-----+
| aaa | mike | 1 | aaa |
| aaa | mike | 2 | aaa |
| ccc | mike | 6 | ccc |
| ddd | mike | NULL | NULL |
+-----+-----+-----+
4 rows in set (0.00 sec)
```

3. GROUP BY

这个子句会把VT2中生成的表按照GROUP BY中的列进行分组。生成VT3表。

注意：


其后处理过程的语句，如SELECT,HAVING，所用到的列必须包含在GROUP BY中，对于没有出现的，得用聚合函数；

原因：

GROUP BY改变了对表的引用，将其转换为新的引用方式，能够对其进行下一级逻辑操作的列会减少；


我的理解是：

根据分组字段，将具有相同分组字段的记录归并成一条记录，因为每一个分组只能返回一条记录，除非是被过滤掉了，而不在分组字段里面的字段可能会有多个值，多个值是无法放进一条记录的，所以必须通过聚合函数将这些具有多值的列转换成单值；



```
mysql> SELECT
-> *
-> FROM
-> table1 AS a
-> LEFT OUTER JOIN table2 AS b ON a.uid = b.uid
-> WHERE
-> a. NAME = 'mike'
-> GROUP BY
-> a.uid;

+-----+-----+-----+-----+
| uid | name | oid | uid |
+-----+-----+-----+-----+
| aaa | mike | 1 | aaa |
| ccc | mike | 6 | ccc |
| ddd | mike | NULL | NULL |
+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```



4. HAVING

这个子句对VT3表中的不同的组进行过滤，只作用于分组后的数据，满足HAVING条件的子句被加入到VT4表中。



```
mysql> SELECT
-> *
-> FROM
-> table1 AS a
-> LEFT OUTER JOIN table2 AS b ON a.uid = b.uid
-> WHERE
-> a. NAME = 'mike'
-> GROUP BY
-> a.uid
-> HAVING
-> count(b.oid) < 2;

+-----+-----+-----+-----+
| uid | name | oid | uid |
+-----+-----+-----+-----+
| ccc | mike | 6 | ccc |
| ddd | mike | NULL | NULL |
+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```



5. SELECT

这个子句对SELECT子句中的元素进行处理，生成VT5表。

(5-J1)计算表达式 计算SELECT 子句中的表达式，生成VT5-J1

(5-J2)DISTINCT

寻找VT5-J1中的重复列，并删掉，生成VT5-J2

如果在查询中指定了DISTINCT子句，则会创建一张内存临时表（如果内存放不下，就需要存放在硬盘了）。这张临时表的表结构和上一步产生的虚拟表VT5是一样的，不同的是对进行DISTINCT操作的列增加了一个唯一索引，以此来除重复数据。



```
mysql> SELECT
-> a.uid,
-> count(b.oid) AS total
-> FROM
-> table1 AS a
-> LEFT OUTER JOIN table2 AS b ON a.uid = b.uid
-> WHERE
-> a. NAME = 'mike'
-> GROUP BY
-> a.uid
-> HAVING
-> count(b.oid) < 2;
```

```
+-----+-----+
| uid | total |
+-----+-----+
```

```
| ccc |    1 |
| ddd |    0 |
+-----+-----+
```

```
+-----+-----+
```

```
2 rows in set (0.00 sec)
```



## 6.ORDER BY

从VT5-J2中的表中，根据ORDER BY 子句的条件对结果进行排序，生成VT6表。

**注意：**

唯一可使用SELECT中别名的地方；



```
mysql> SELECT
-> a.uid,
-> count(b.oid) AS total
-> FROM
-> table1 AS a
-> LEFT OUTER JOIN table2 AS b ON a.uid = b.uid
-> WHERE
-> a. NAME = 'mike'
-> GROUP BY
-> a.uid
-> HAVING
-> count(b.oid) < 2
-> ORDER BY
-> total DESC;
```

```
+-----+-----+
| uid | total |
+-----+-----+
```

```
| ccc |    1 |
| ddd |    0 |
+-----+-----+
```

```
+-----+-----+
```

```
2 rows in set (0.00 sec)
```





7.LIMIT

LIMIT子句从上一步得到的VT6虚拟表中选出从指定位置开始的指定行数据。

注意：

offset和rows的正负带来的影响；

当偏移量很大时效率是很低的，可以这么做：

采用子查询的方式优化，在子查询里先从索引获取到最大id，然后倒序排，再取N行结果集

采用INNER JOIN优化，JOIN子句里也优先从索引获取ID列表，然后直接关联查询获得最终结果



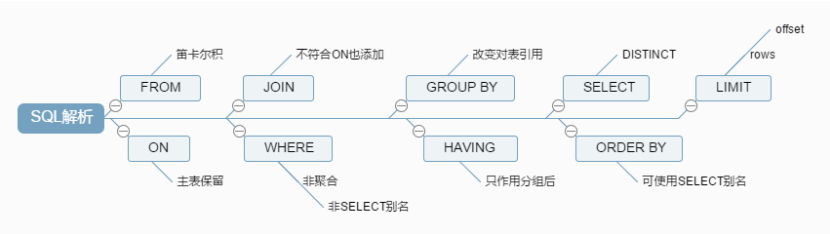
```
mysql> SELECT
-> a.uid,
-> count(b.oid) AS total
-> FROM
-> table1 AS a
-> LEFT JOIN table2 AS b ON a.uid = b.uid
-> WHERE
-> a. NAME = 'mike'
-> GROUP BY
-> a.uid
-> HAVING
-> count(b.oid) < 2
-> ORDER BY
-> total DESC
-> LIMIT 1;
```



uid	total
ccc	1

1 row in set (0.00 sec)

至此SQL的解析之旅就结束了，上图总结一下：



参考书籍：

《MySQL性能调优与架构实践》

《MySQL技术内幕：SQL编程》

尾声：

嗯，到这里这一次的深入了解之旅就差不多真的结束了，虽然也不是很深入，只是一些东西将其东拼西凑在一起而已，参考了一些以前看过的书籍，大师之笔果然不一样。而且在这过程中也是get到了蛮多东西的，最重要的是更进一步意识到，计算机软件世界的宏大呀~

另由于本人才疏学浅，其中难免存在纰漏错误之处，若发现劳烦告知修改，感谢~

如需转载，请保留AnnsShadoW和本文地址

<http://www.cnblogs.com/annsshadow/p/5037667.html>

分类： DataBase


好文要顶

关注我

收藏该文







AnnsShadoW

粉丝 - 49 关注 - 8

32

0

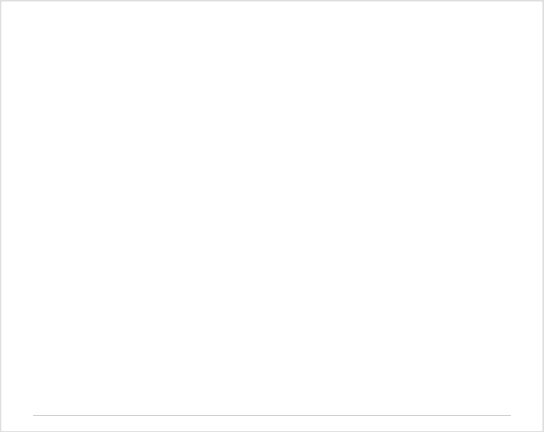
« 上一篇： [来自妹纸的挑战]-展开/还原多层链表

» 下一篇： 遇到shell重定向的一个奇怪问题：'消失'的标准输入！

posted @ 2015-12-10 23:03 AnnsShadoW 阅读(50425) 评论(14) 编辑 收藏 举报

刷新评论 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



编辑推荐：

· 从 C# 入门 Kafka

· C# 托管堆 遭破坏 问题溯源分析

· order by 语句怎么优化？

· SQLSERVER 事务日志的 LSN 到底是什么？

· 微服务架构学习与思考：从单体架构到微服务架构的演进历程

阅读排行：

· 这一年我们上线的自动化系统

· .Net6 使用 Ocelot + Consul 看这篇就够了

· WPF开发经验-实现一种三轴机械手控件

· C#开发PACS医学影像三维重建(十四):基于能量模型算法将曲面牙床展开至二维平面

· 让人眼前一亮的应用「GitHub 热点速览」