

1. tar xpf proc-demo.tar
2. imp your_oracle_account_name/your_oracle_account_password@csdbora
3. make -f demo_proc64.mk ematch_job
- 4.
- 5.

```
<601> z_x3 at newfirebird.cs.txstate.edu: ls
a_search.php b_search.php ematch_job.pc sample3 sample3.c sample3.o sample3.pc
<602> z_x3 at newfirebird.cs.txstate.edu: ./sample3
```

```
Connected to ORACLE as user: Z_X3
```

Number	Employee	Salary
7369	SMITH	800.00
7499	ALLEN	1600.00
7521	WARD	1250.00
7566	JONES	2975.00
7654	MARTIN	1250.00

Number	Employee	Salary
7698	BLAKE	2850.00
7782	CLARK	2450.00
7788	SCOTT	3000.00
7839	KING	5000.00
7844	TURNER	1500.00

Number	Employee	Salary
7876	ADAMS	1100.00
7900	JAMES	950.00
7902	FORD	3000.00
7934	MILLER	1300.00

```
Au revoir.
```

```
<603> z_x3 at newfirebird.cs.txstate.edu: █
```

```
<639> z_x3 at newfirebird.cs.txstate.edu: ./sample3 deptno 20
```

Employee	Salary	commission
-----	-----	-----
SMITH	800.00	0.00
JONES	2975.00	0.00
SCOTT	3000.00	0.00
ADAMS	1100.00	0.00
FORD	3000.00	0.00

Au revoir.

```
<639> z_x3 at newfirebird.cs.txstate.edu: /home/Students/z_x3/public_html/demo/p
roc/unix-version/html/zebo/sample3 deptno 20
```

Employee	Salary	commission
-----	-----	-----
SMITH	800.00	0.00
JONES	2975.00	0.00
SCOTT	3000.00	0.00
ADAMS	1100.00	0.00
FORD	3000.00	0.00

Au revoir.

LIBRARY_PATH 和 LD_LIBRARY_PATH 是 Linux 下的两个环境变量，二者的含义和作用分别如下：

LIBRARY_PATH 环境变量用于在程序编译期间查找动态链接库时指定查找共享库的路径，例如，指定 gcc 编译需要用到的动态链接库的目录。设置方法如下（其中，LIBDIR1 和 LIBDIR2 为两个库目录）：

```
export LIBRARY_PATH=LIBDIR1:LIBDIR2:$LIBRARY_PATH
```

LD_LIBRARY_PATH 环境变量用于在程序加载运行期间查找动态链接库时指定除了系统默认路径之外的其他路径，注意，LD_LIBRARY_PATH 中指定的路径会在系统默认路径之前进行查找。设置方法如下（其中，LIBDIR1 和 LIBDIR2 为两个库目录）：

```
export LD_LIBRARY_PATH=LIBDIR1:LIBDIR2:$LD_LIBRARY_PATH
```

举个例子，我们开发一个程序，经常会需要使用某个或某些动态链接库，为了保证程序的可移植性，可以先将这些编译好的动态链接库放在自己指定的目录下，然后按照上述方式将这些目录加入到 LD_LIBRARY_PATH 环境变量中，这样自己的程序就可以动态链接后加载库文件运行了。

区别与使用：

开发时，设置 LIBRARY_PATH，以便 gcc 能够找到编译时需要的动态链接库。

发布时，设置 LD_LIBRARY_PATH，以便程序加载运行时能够自动找到需要的动态链接库。

Linux 环境变量名，该环境变量主要用于指定查找共享库(动态链接库)时除了默认路径之外的其他路径。(该路径在默认路径之前查找)

移植程序时的经常碰到需要使用一些特定的动态库，而这些编译好的动态库放在我们自己建立的目录里，这时可以将这些目录设置到 LD_LIBRARY_PATH 中。

当执行函数动态链接.so 时，如果此文件不在缺省目录下 '/usr/local/lib' and '/usr/lib'.

那么就需要指定环境变量 LD_LIBRARY_PATH

假如现在需要在已有的环境变量上添加新的路径名，则采用如下方式：

LD_LIBRARY_PATH=NEWDIRS:\$LD_LIBRARY_PATH. (newdirs 是新的路径串)

TWO_TASK 环境变量的作用，指出“在 Unix 和 Linux 环境下，可以设置 TWO_TASK 环境变量，当用户连接数据库且没有指定服务名时，会自动利用 TWO_TASK 的设置作为环境变量连接数据库。”，并用两个本地数据库为例说明了 TWO_TASK 的使用。

SID = identifies the database instance (database name + instance number). So if your database name is somedb and your instance number is 3, then your SID is somedb3.

DB Name = Name of the database (database can be shared b/t multiple instances)

DB Domain = Usually the same as your company domain (somecompany.com)

Global Database Name = Database name + database domain (somedb.somecompany.com)+

\$_SERVER is an array containing information such as headers, paths, and script locations. The entries in this array are created by the web server

- 1、\$_SERVER['PHP_SELF'] -- 获取当前正在执行脚本的文件名。
如:/PHP/SourceCode/08/04/Untitled-1.php
- 2、\$_SERVER['SERVER_PROTOCOL'] -- 请求页面时通信协议的名称和版本。例如，“HTTP/1.0”。
- 3、\$_SERVER['REQUEST_TIME'] -- 请求开始时的时间戳。从 PHP 5.1.0 起有效。和 time 函数效果一样。
- 4、\$_SERVER['argv'] -- 传递给该脚本的参数。
- 5、\$_SERVER['SERVER_NAME'] -- 返回当前主机名。如：<http://blog.sina.com.cn>
- 6、\$_SERVER['SERVER_SOFTWARE'] -- 服务器标识的字串，在响应请求时的头信息中给出。如 Microsoft-IIS/6.0
- 7、\$_SERVER['REQUEST_METHOD'] -- 访问页面时的请求方法。例如：“GET”、“HEAD”、“POST”、“PUT”。
- 8、\$_SERVER['QUERY_STRING'] -- 查询（query）的字符串（URL 中第一个问号 ? 之后的内容）。
- 9、\$_SERVER['DOCUMENT_ROOT'] -- 当前运行脚本所在的文档根目录。在服务器配置

文件中定义。如 E:\server

10、\$_SERVER['HTTP_ACCEPT'] -- 当前请求的 Accept: 头信息的内容。

11、\$_SERVER['HTTP_ACCEPT_CHARSET'] -- 当前请求的 Accept-Charset: 头信息的内容。例如: “iso-8859-1,*,utf-8”。

12、\$_SERVER['HTTP_ACCEPT_ENCODING'] -- 当前请求的 Accept-Encoding: 头信息的内容。例如: “gzip”。

13、\$_SERVER['HTTP_ACCEPT_LANGUAGE'] -- 当前请求的 Accept-Language: 头信息的内容。例如: “en”。

14、\$_SERVER['HTTP_CONNECTION'] -- 当前请求的 Connection: 头信息的内容。例如: “Keep-Alive”。

15、\$_SERVER['HTTP_HOST'] -- 当前请求的 Host: 头信息的内容。如: 192.168.2.53:8888

16、\$_SERVER['HTTP_REFERER'] -- 链接到当前页面的前一页面的 URL 地址。

17、\$_SERVER['HTTP_USER_AGENT'] -- 返回用户使用的浏览器信息。也可以使用 get_browser() 得到此信息。

18、\$_SERVER['HTTPS'] -- 如果通过 https 访问, 则被设为一个非空的值, 否则返回 off.

19、\$_SERVER['REMOTE_ADDR'] -- 正在浏览当前页面用户的 IP 地址。

20、\$_SERVER['REMOTE_HOST'] -- 正在浏览当前页面用户的主机名。反向域名解析基于该用户的 REMOTE_ADDR。如本地测试返回 127.0.0.1

21、\$_SERVER['REMOTE_PORT'] -- 用户连接到服务器时所使用的端口。我在本机测试没通过, 不知道什么原因。

22、\$_SERVER['SCRIPT_FILENAME'] -- 当前执行脚本的绝对路径名。如返回 E:\server\index.php

23、\$_SERVER['SERVER_ADMIN'] -- 该值指明了 Apache 服务器配置文件中的 SERVER_ADMIN 参数。如果脚本运行在一个虚拟主机上, 则该值是那个虚拟主机的值

24、\$_SERVER['SERVER_PORT'] -- 服务器所使用的端口。默认为“80”。如果使用 SSL 安全连接, 则这个值为用户设置的 HTTP 端口。

25、\$_SERVER['SERVER_SIGNATURE'] -- 包含服务器版本和虚拟主机名的字符串。

26、\$_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。这是在服务器进行虚拟到真实路径的映像后的结果。Apache 2 用户可以使用 httpd.conf 中的 AcceptPathInfo On 来定义 PATH_INFO。

27、\$_SERVER['SCRIPT_NAME'] -- 包含当前脚本的路径。这在页面需要指向自己时非常有用。**FILE** 包含当前文件的绝对路径和文件名（例如包含文件）。

28、\$_SERVER['REQUEST_URI'] -- 访问此页面所需的 URI。例如, “/index.html”。

29、\$_SERVER['PHP_AUTH_DIGEST'] -- 当作为 Apache 模块运行时, 进行 HTTP Digest 认证的过程中, 此变量被设置成客户端发送的“Authorization”HTTP 头内容（以便作进一步的认证操作）。

30、\$_SERVER['PHP_AUTH_USER'] -- 当 PHP 运行在 Apache 或 IIS（PHP 5 是 ISAPI）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。

31、\$_SERVER['PHP_AUTH_PW'] -- 当 PHP 运行在 Apache 或 IIS（PHP 5 是 ISAPI）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。

32、\$_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。

作者：烟雨弥漫了江南

链接：<https://www.jianshu.com/p/35153922aa0c>

来源：简书

著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

3, <http://www.biuuu.com/index.php?p=222&q=biuuu>

结果：

```
$_SERVER["QUERY_STRING"] = "p=222&q=biuuu"
$_SERVER["REQUEST_URI"] = "/index.php?p=222&q=biuuu"
$_SERVER["SCRIPT_NAME"] = "/index.php"
$_SERVER["PHP_SELF"] = "/index.php"
```

\$_SERVER["**QUERY_STRING**"]获取查询语句，实例中可知，获取的是?后面的值

\$_SERVER["**REQUEST_URI**"] 获取 <http://www.biuuu.com> 后面的值，包括/

\$_SERVER["**SCRIPT_NAME**"] 获取当前脚本的路径，如：index.php

\$_SERVER["**PHP_SELF**"] 当前正在执行脚本的文件名

[explode\(\)](#)：使用一个字符串分割另一个字符串

echo PHP_EOL; 换行符

C++ 数据结构

C/C++ 数组允许定义可存储相同类型数据项的变量，但是**结构**是 C++ 中另一种用户自定义的可用的数据类型，它允许您存储不同类型的数据项。

结构用于表示一条记录，假设您想要跟踪图书馆中书本的动态，您可能需要跟踪每本书的下列属性：

- Title：标题
- Author：作者
- Subject：类目
- Book ID：书的 ID

定义结构

为了定义结构，您必须使用 **struct** 语句。struct 语句定义了一个包含多个成员的新的数据类型，struct 语句的格式如下：

```
struct type_name {  
    member_type1 member_name1;  
    member_type2 member_name2;  
    member_type3 member_name3;  
    .  
    .  
} object_names;
```

type_name 是结构体类型的名称，**member_type1 member_name1** 是标准的变量定义，比如 `int i;` 或者 `float f;` 或者其他有效的变量定义。在结构定义的末尾，最后一个分号之前，您可以指定一个或多个结构变量，这是可选的。下面是声明一个结构体类型 **Books**，变量为 **book**：

```
struct Books  
{  
    char title[50];  
    char author[50];  
    char subject[100];  
    int book_id;  
} book;
```

访问结构成员

为了访问结构的成员，我们使用成员访问运算符（.）。成员访问运算符是结构变量名称和我们要访问的结构成员之间的一个句号。

下面的实例演示了结构的用法：

实例

```
#include <iostream>
#include <cstring>

using namespace std;

// 声明一个结构体类型 Books
struct Books
{
    char title[50];
    char author[50];
    char subject[100];
    int book_id;
};

int main( )
{
    Books Book1;          // 定义结构体类型 Books 的变量 Book1
    Books Book2;          // 定义结构体类型 Books 的变量 Book2

    // Book1 详述
    strcpy( Book1.title, "C++ 教程");
    strcpy( Book1.author, "Runoob");
    strcpy( Book1.subject, "编程语言");
    Book1.book_id = 12345;

    // Book2 详述
    strcpy( Book2.title, "CSS 教程");
    strcpy( Book2.author, "Runoob");
    strcpy( Book2.subject, "前端技术");
    Book2.book_id = 12346;

    // 输出 Book1 信息
    cout << "第一本书标题 : " << Book1.title << endl;
    cout << "第一本书作者 : " << Book1.author << endl;
    cout << "第一本书类目 : " << Book1.subject << endl;
    cout << "第一本书 ID : " << Book1.book_id << endl;


    // 输出 Book2 信息
    cout << "第二本书标题 : " << Book2.title << endl;
    cout << "第二本书作者 : " << Book2.author << endl;
    cout << "第二本书类目 : " << Book2.subject << endl;
    cout << "第二本书 ID : " << Book2.book_id << endl;

    return 0;
}
```

```
void main(int argc, char *argv[])
```

sqlplus Z_X3@CSDBORA

C 库函数 - strcmp()

 C 标准库 - <string.h>

描述

C 库函数 `int strcmp(const char *str1, const char *str2)` 把 `str1` 所指向的字符串和 `str2` 所指向的字符串进行比较。

声明

下面是 `strcmp()` 函数的声明。

```
int strcmp(const char *str1, const char *str2)
```

参数

- `str1` -- 要进行比较的第一个字符串。
- `str2` -- 要进行比较的第二个字符串。

返回值

该函数返回值如下：

- 如果返回值小于 0，则表示 `str1` 小于 `str2`。
- 如果返回值大于 0，则表示 `str1` 大于 `str2`。
- 如果返回值等于 0，则表示 `str1` 等于 `str2`。

Compile the `ematch_job.pc` file:

```
make -f demo_proc64.mk ematch_job
```

If you add your own application files, you have to modify the file `demo_proc64.mk` accordingly.

Just like we need to have `sample3` file, we use **make** to pre-compile the class

deptno =atoi(argv[2]); 把字符串转换成 int 整型数;

```
EXEC SQL DECLARE c1 CURSOR FOR
SELECT ename, sal, NVL(comm,0)
FROM emp
WHERE deptno = :deptno;

EXEC SQL OPEN c1;
```

oracle pro*c之 sqlca

转载

wenchao126

2013-01-16 11:42:19

© 2041

☆ 收藏

展开

sqlca

转自: <http://huangxiaojian9999.blog.163.com/blog/static/12129874220101103135228/>

如果需要更强大的错误处理机制, 那么嵌入的 SQL 接口提供了一个叫 `sqlca` 的全局变量, 它是一个有着下面定义的结构:

```
struct {
    char sqlcaid[8];
    long sqlabc;
    long sqlcode;
    struct {
        int sqlerrml;
        char sqlerrmc[70];
    } sqlerrm;
    char sqlerrp[8];
    long sqlerrd[6];
    char sqlwarn[8];
    char sqlstate[5];
} sqlca;
```

(在一个多线程的程序里, 每个线程自动获得自己的 `sqlca` 的拷贝。这个方式类似于处理标准 C 全局变量 `errno`。)

如果最后一条 SQL 语句成功, 如果适合该具体命令, 那么 `sqlca.sqlerrd[1]` 包含处理过的行的 OID, 而 `sqlca.sqlerrd[2]` 包含 处理或返回的行数。