# 计算机原理

## 2的幂数

if(n&(n-1)) then n不是2的幂数;

else n是2的幂数;

原理：如果n=2^K,那么n = 1000...0(k个0）,则n-1 = 111...0(k个1);相与之后则为0

如果 n!=2^k，那么

n跟（n-1)第一位都为1，则相与这后然后第一位为1，则不为0.

要判断某整数n是否是2的幂数，可以把此数与n - 1相与，如结果为0，则表示n为2的幂数，反之则不是。原理是2的x次幂的二进制表示必为最高位是1，低x位全为0；减1后则为x位，且全为1；相与的结果一定为0。

## 二进制、八进制、十六进制

**进制越大，数的表达长度也就越短**。2、8、16，分别是2的1次方，3次方，4次方。这一点使得三种进制之间可以非常直接地互相转换。8进制或16进制缩短了二进制数，但保持了二进制数的表达特点。

### 6.2  二、八、十六进制数转换到十进制数

6.2.1 二进制数转换为十进制数

二进制数第0位的权值是2的0次方，第1位的权值是2的1次方……

所以，设有一个二进制数：0110 0100，转换为10进制为：

下面是竖式：

0110 0100 换算成 十进制

第0位 0 \* 20  =  0

第1位 0 \* 21  =  0

第2位 1 \* 22  =  4

第3位 0 \* 23  =  0

第4位 0 \* 24  =  0

第5位 1 \* 25  = 32

第6位 1 \* 26  = 64

第7位 0 \* 27  =  0     ＋

---------------------------

              100

用横式计算为：

0 \* 20+ 0 \* 21 + 1 \* 22 + 1 \* 23 + 0 \* 24 + 1 \* 25 + 1 \* 26 + 0 \* 27 = 100

0乘以多少都是0，所以我们也可以直接跳过值为0的位：

1 \* 22 + 1 \* 23 +  1 \* 25 + 1 \* 26 = 100

**6.2.2 八进制数转换为十进制数**

八进制就是逢8进1。

八进制数采用 0～7这八数来表达一个数。

八进制数第0位的权值为8的0次方，第1位权值为8的1次方，第2位权值为8的2次方……

所以，设有一个八进制数：1507，转换为十进制为：

用竖式表示：

1507换算成十进制。

第0位 7 \* 80 = 7

第1位 0 \* 81 = 0

第2位 5 \* 82 = 320

第3位 1 \* 83 = 512   ＋

--------------------------

              839

同样，我们也可以用横式直接计算：

7 \* 80 + 0 \* 81 + 5 \* 82 + 1 \* 83 =839

结果是，八进制数 1507 转换成十进制数为 839

**6.2.3 八进制数的表达方法**

C,C++语言中，如何表达一个八进制数呢？如果这个数是 876,我们可以断定它不是八进制数，因为八进制数中不可能出7以上的阿拉伯数字。但如果这个数是123、是567，或12345670，那么它是八进制数还是10进制数，都有可能。

所以,C,C++规定，**一个数如果要指明它采用八进制，必须在它前面加上一个0**，如：123是十进制，但0123则表示采用八进制。这就是八进制数在C、C++中的表达方法。

由于C和C++都没有提供二进制数的表达方法，所以，这里所学的八进制是我们学习的，CtC++语言的数值表达的第二种进制法。

现在，对于同样一个数，比如是100，我们在代码中可以用平常的10进制表达，例如在变量初始化时：

int a = 100;

我们也可以这样写：

int a = 0144; //0144是八进制的100；一个10进制数如何转成8进制，我们后面会学到。

千万记住，用八进制表达时，你不能少了最前的那个0。否则计算机会通通当成10进制。不过，有一个地方使用八进制数时，却不能使用加0，那就是我们前面学的用于表达字符的“转义符”表达法。

**6.2.4 八进制数在转义符中的使用**

我们学过用一个转义符'\'加上一个特殊字母来表示某个字符的方法，如：'\n'表示换行(line)，而'\t'表示Tab字符，'\''则表示单引号。今天我们又学习了一种使用转义符的方法：转义符'\'后面接一个八进制数，用于表示ASCII码等于该值的字符。

比如，查一下[第5章中的ASCII码表](http://www.d2school.com/bcyl/bhcpp/newls/ls05.htm#美国信息交换标准码)，我们找到问号字符（?)的ASCII值是63，那么我们可以把它转换为八进值：77，然后用 '\77'来表示'?'。由于是八进制，所以本应写成 '\077'，但因为C,C++规定不允许使用斜杠加10进制数来表示字符，所以这里的0可以不写。

**6.2.5 十六进制数转换成十进制数**

2进制，用两个阿拉伯数字：0、1；

8进制，用八个阿拉伯数字：0、1、2、3、4、5、6、7；

10进制，用十个阿拉伯数字：0到9；

16进制，用十六个阿拉伯数字……等等，阿拉伯人或说是印度人，只发明了10个数字啊？

16进制就是逢16进1，但我们只有0~9这十个数字，所以我们**用A，B，C，D，E，F这五个字母来分别表示10，11，12，13，14，15**。字母不区分大小写。

十六进制数的第0位的权值为16的0次方，第1位的权值为16的1次方，第2位的权值为16的2次方……

所以，在第N（N从0开始）位上，如果是是数 X （X 大于等于0，并且X小于等于 15，即：F）表示的大小为 X \* 16的N次方。

假设有一个十六进数 2AF5, 那么如何换算成10进制呢？

用竖式计算：

2AF5换算成10进制:

第0位：  5 \* 160 = 5

第1位：  F \* 161 = 240

第2位：  A \* 162 = 2560

第3位：  2 \* 163 = 8192  ＋

-------------------------------------

                 10997

直接计算就是：

5 \* 160  + F \* 161 + A \* 162+2 \* 163 = 10997

(别忘了，在上面的计算中，A表示10，而F表示15)

现在可以看出，所有进制换算成10进制，关键在于各自的权值不同。

假设有人问你，十进数 1234 为什么是 一千二百三十四？你尽可以给他这么一个算式：

1234 = 1 \* 103 + 2 \* 102 + 3 \* 101 + 4 \* 100

**6.2.6  十六进制数的表达方法**

如果不使用特殊的书写形式，16进制数也会和10进制相混。随便一个数：9876，就看不出它是16进制或10进制。

C，C++规定，**16进制数必须以 0x开头**。比如 0x1表示一个16进制数。而1则表示一个十进制。另外如：0xff,0xFF,0X102A,等等。其中的x也也不区分大小写。(注意：0x中的0是数字0，而不是字母O)

以下是一些用法示例：

int a = 0x100F;

int b = 0x70 + a;

至此，我们学完了所有进制：10进制，8进制，16进制数的表达方式。最后一点很重要，C/C++中，10进制数有正负之分，比如12表示正12，而-12表示负12，；但**8进制和16进制只能用达无符号的正整数**，如果你在代码中里：-078，或者写：-0xF2,C,C++并不把它当成一个负数。

**6.2.7 十六进制数在转义符中的使用**

转义符也可以接一个16进制数来表示一个字符。如在6.2.4小节中说的 '?' 字符，可以有以下表达方式：

'?'     //直接输入字符

'\77'   //用八进制，此时可以省略开头的0

'\0x3F' //用十六进制

同样，这一小节只用于了解。除了空字符用八进制数 '\0' 表示以外，我们很少用后两种方法表示一个字符。

**6.3 十进制数转换到二、八、十六进制数**

**6.3.1 10进制数转换为2进制数**

给你一个十进制，比如：6，如果将它转换成二进制数呢？

10进制数转换成二进制数，这是一个连续除2的过程：

**把要转换的数，除以2，得到商和余数，**

**将商继续除以2，直到商为0。最后将所有余数倒序排列，得到数就是转换结果。**

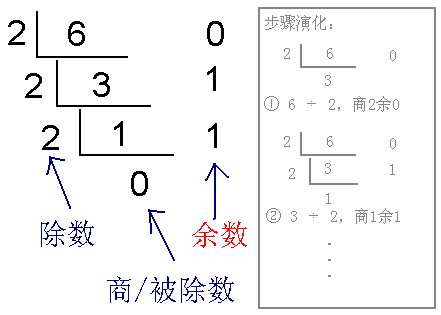
听起来有些糊涂？我们结合例子来说明。比如要转换6为二进制数。

6转换成二进制，结果是110。

把上面的一段改成用表格来表示，则为：

|  |  |  |  |
| --- | --- | --- | --- |
| 被除数 | 计算过程 | 商 | 余数 |
| 6 | 6/2 | 3 | 0 |
| 3 | 3/2 | 1 | 1 |
| 1 | 1/2 | 0 | 1 |

（在计算机中，÷用 / 来表示）



**6.3.2 10进制数转换为8、16进制数**

非常开心，10进制数转换成8进制的方法，和转换为2进制的方法类似，惟一变化：除数由2变成8。

来看一个例子，如何将十进制数120转换成八进制数。

用表格表示：

|  |  |  |  |
| --- | --- | --- | --- |
| 被除数 | 计算过程 | 商 | 余数 |
| 120 | 120/8 | 15 | 0 |
| 15 | 15/8 | 1 | 7 |
| 1 | 1/8 | 0 | 1 |

120转换为8进制，结果为：170。

非常非常开心，10进制数转换成16进制的方法，和转换为2进制的方法类似，惟一变化：除数由2变成16。

同样是120，转换成16进制则为：

|  |  |  |  |
| --- | --- | --- | --- |
| 被除数 | 计算过程 | 商 | 余数 |
| 120 | 120/16 | 7 | 8 |
| 7 | 7/16 | 0 | 7 |

120转换为16进制，结果为：78。

**6.4 二、十六进制数互相转换**

二进制和十六进制的互相转换比较重要。不过这二者的转换却不用计算，每个C，C++程序员都能做到看见二进制数，直接就能转换为十六进制数，反之亦然。

首先我们来看一个二进制数：1111，它是多少呢？

你可能还要这样计算：1 \* 20 + 1 \* 21 + 1 \* 22+ 1 \* 23 = 1 \* 1 + 1 \* 2 + 1 \* 4 + 1 \* 8 = 15。然而，由于1111才4位，所以我们必须直接记住它每一位的权值，并且是从高位往低位记，：8、4、2、1。即，最高位的权值为23＝ 8，然后依次是 22 ＝ 4，21＝2， 20 ＝ 1。

记住8421，对于任意一个4位的二进制数，我们都可以很快算出它对应的10进制值。

下面列出四位二进制数 xxxx 所有可能的值（中间略过部分）

仅4位的2进制数  快速计算方法   十进制值     十六进值

1111        = 8 + 4 + 2 + 1  = 15          F

1110        = 8 + 4 + 2 + 0  = 14          E

1101        = 8 + 4 + 0 + 1  = 13          D

1100        = 8 + 4 + 0 + 0  = 12          C

1011        = 8 + 4 + 0 + 1  = 11          B

1010        = 8 + 0 + 2 + 0  = 10          A

1001        = 8 + 0 + 0 + 1  = 10          9

....

0001        = 0 + 0 + 0 + 1  = 1           1

0000        = 0 + 0 + 0 + 0  = 0           0

**二进制数要转换为十六进制，就是以4位一段，分别转换为十六进制。**

如(上行为二制数，下面为对应的十六进制)：

1111 1101 ， 1010 0101 ， 1001 1011

 F    D   ，  A    5   ，  9    B

反过来，当我们看到 FD时，如何迅速将它转换为二进制数呢？

先转换F：

看到F，我们需知道它是15（可能你还不熟悉A～F这五个数），然后15如何用8421凑呢？应该是8 + 4 + 2 + 1，所以四位全为1 ：1111。

接着转换 D：

看到D，知道它是13，13如何用8421凑呢？应该是：8 + 2 + 1,即：1011。

所以,FD转换为二进制数，为： 1111 1011

由于十六进制转换成二进制相当直接，所以，我们需要将一个十进制数转换成2进制数时，也可以先转换成16进制，然后再转换成2进制。

**比如，十进制数 1234转换成二制数，如果要一直除以2，直接得到2进制数，需要计算较多次数。所以我们可以先除以16，得到16进制数:**

|  |  |  |  |
| --- | --- | --- | --- |
| 被除数 | 计算过程 | 商 | 余数 |
| 1234 | 1234/16 | 77 | 2 |
| 77 | 77/16 | 4 | 13 (D) |
| 4 | 4/16 | 0 | 4 |

结果16进制为： 0x4D2

**然后我们可直接写出0x4D2的二进制形式：** 0100 1011 0010。

其中对映关系为：

0100 -- 4

1011 -- D

0010 -- 2

同样，如果一个二进制数很长，我们需要将它转换成10进制数时，除了前面学过的方法是，我们还可以先将这个二进制转换成16进制，然后再转换为10进制。

下面举例一个int类型的二进制数：

01101101 11100101 10101111 00011011

我们按四位一组转换为16进制： 6D E5 AF 1B

**6.5 原码、反码、补码**

结束了各种进制的转换，我们来谈谈另一个话题：原码、反码、补码。

我们已经知道计算机中，所有数据最终都是使用二进制数表达。

比如，假设有一 int 类型的数，值为5，那么，我们知道它在计算机中表示为：

00000000 00000000 00000000 00000101

5转换成二制是101，不过int类型的数占用4字节（32位），所以前面填了一堆0。

现在想知道，-5在计算机中如何表示？

**在计算机中，负数以其正值的补码形式表达**。

什么叫补码呢？这得从原码，反码说起。

**原码：一个整数，按照绝对值大小转换成的二进制数，称为原码。**

比如 00000000 00000000 00000000 00000101 是 5的 原码。

**反码：将二进制数按位取反，所得的新二进制数称为原二进制数的反码。**

取反操作指：原为1，得0；原为0，得1。（1变0; 0变1）

比如：将00000000 00000000 00000000 00000101每一位取反，得11111111 11111111 11111111 11111010。

称：11111111 11111111 11111111 11111010 是 00000000 00000000 00000000 00000101 的反码。

反码是相互的，所以也可称：

11111111 11111111 11111111 11111010 和 00000000 00000000 00000000 00000101 互为反码。

**补码：反码加1称为补码。**

也就是说，要得到一个数的补码，先得到反码，然后将反码加上1，所得数称为补码。

比如：00000000 00000000 00000000 00000101 的反码是：11111111 11111111 11111111 11111010。

那么，补码为：

11111111 11111111 11111111 11111010 + 1 = 11111111 11111111 11111111 11111011

所以，-5 在计算机中表达为：11111111 11111111 11111111 11111011。转换为十六进制：0xFFFFFFFB。

再举一例，我们来看整数-1在计算机中如何表示。

假设这也是一个int类型，那么：

1、先取1的原码：00000000 00000000 00000000 00000001

2、得反码：     11111111 11111111 11111111 11111110

3、得补码：     11111111 11111111 11111111 11111111

可见，－1在计算机里用二进制表达就是全1。16进制为：0xFFFFFF。

### 二进制乘法

二进制乘法有点像十进制里的某些乘法运算，如

123\*111

我们可以这么算

123\*100+123\*10+123\*1=12300+1230+123=13530+123=13653

二进制乘法：

比如10010\*11010

可以这样算（如10111<<1000代表在10111后面添加3个零）

10010<<10000 =100100000

10010<< 1000 = 10010000

10010<< 10 = 100100

最后相加，得

100100000+10010000+100100

=110110000+100100

=111010100 这就是10010\*11010的结果~ 计算中，仅使用了加法和补零

## Base64

**Base64** is a group of similar encoding schemes that represent [binary data](http://en.wikipedia.org/wiki/Binary_data) in an ASCII string format by translating it into a [radix](http://en.wikipedia.org/wiki/Radix)-64 representation. The Base64 term originates from a specific [MIME content transfer encoding](http://en.wikipedia.org/wiki/MIME#Content-Transfer-Encoding).

Base64 encoding schemes are commonly used when there is a need to encode binary data that needs be stored and transferred over media that are designed to deal with textual data. This is to ensure that the data remains intact without modification during transport. Base64 is commonly used in a number of applications including [email](http://en.wikipedia.org/wiki/Email) via [MIME](http://en.wikipedia.org/wiki/MIME), and storing complex data in [XML](http://en.wikipedia.org/wiki/XML).

**Design**

The general rule is to choose a set of 64 characters that is both part of a subset common to most encodings, and also [printable](http://en.wikipedia.org/wiki/Printable_character). This combination leaves the data unlikely to be modified in transit through information systems, such as email, that were traditionally not [8-bit clean](http://en.wikipedia.org/wiki/8-bit_clean).[[1]](http://en.wikipedia.org/wiki/Base64#cite_note-0) For example, MIME's Base64 implementation uses A–Z, a–z, and 0–9 for the first 62 values. Other variations, usually derived from Base64, share this property but differ in the symbols chosen for the last two values; an example is [UTF-7](http://en.wikipedia.org/wiki/UTF-7).

**Examples**

the encoded value of *Man* is *TWFu*. Encoded in ASCII, *M*, *a*, *n* are stored as the bytes 77, 97, 110, which are, in 8-bit quantities, 01001101, 01100001, 01101110 in base 2. These three bytes are joined together into a 24 bit buffer producing 010011010110000101101110. Packs of 6 bits (6 bits have a maximum of 64 different binary values) are converted into numbers (in this case, there are 4 numbers in this 24-bit string), which are then converted to their corresponding values in Base64.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Text content | M | | | | | | | | a | | | | | | | | n | | | | | | | |
| ASCII | 77 | | | | | | | | 97 | | | | | | | | 110 | | | | | | | |
| Bit pattern | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 |
| Index | 19 | | | | | | 22 | | | | | | 5 | | | | | | 46 | | | | | |
| Base64-encoded | T | | | | | | W | | | | | | F | | | | | | u | | | | | |

As this example illustrates, Base64 encoding converts 3 [octets](http://en.wikipedia.org/wiki/Octet_(computing)) into 4 encoded characters.

The Base64 index table:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Value | Char |  | Value | Char |  | Value | Char |  | Value | Char |
| 0 | A | 16 | Q | 32 | g | 48 | w |
| 1 | B | 17 | R | 33 | h | 49 | x |
| 2 | C | 18 | S | 34 | i | 50 | y |
| 3 | D | 19 | T | 35 | j | 51 | z |
| 4 | E | 20 | U | 36 | k | 52 | 0 |
| 5 | F | 21 | V | 37 | l | 53 | 1 |
| 6 | G | 22 | W | 38 | m | 54 | 2 |
| 7 | H | 23 | X | 39 | n | 55 | 3 |
| 8 | I | 24 | Y | 40 | o | 56 | 4 |
| 9 | J | 25 | Z | 41 | p | 57 | 5 |
| 10 | K | 26 | a | 42 | q | 58 | 6 |
| 11 | L | 27 | b | 43 | r | 59 | 7 |
| 12 | M | 28 | c | 44 | s | 60 | 8 |
| 13 | N | 29 | d | 45 | t | 61 | 9 |
| 14 | O | 30 | e | 46 | u | 62 | + |
| 15 | P | 31 | f | 47 | v | 63 | / |

When the number of bytes to encode is not divisible by 3, that is there are only one or two bytes of input for the last block, then the following action is performed: Add extra bytes with value zero so there are three bytes, and perform the conversion to base64. If there was only one significant input byte, only the first two base64 digits are picked, and if there were two significant input bytes, the first three base64 digits are picked. '=' characters might be added to make the last block contain four base64 characters.

As a result: When the last group contains one octet the four least significant bits of the final 6-bit block are set to zero, and when the last group contains two octets the two least significant bits of the final 6-bit block are set to zero.

**[**[**edit**](http://en.wikipedia.org/w/index.php?title=Base64&action=edit&section=3)**]**Padding

The '==' sequence indicates that the last group contained only 1 byte, and '=' indicates that it contained 2 bytes. The example below illustrates how truncating the input of the whole of the above quote changes the output padding:

Input ends with: *any carnal plea****sure.*** Output ends with: YW55IGNhcm5hbCBwbGVh**c3VyZS4=**

Input ends with: *any carnal plea****sure*** Output ends with: YW55IGNhcm5hbCBwbGVh**c3VyZQ==**

Input ends with: *any carnal plea****sur*** Output ends with: YW55IGNhcm5hbCBwbGVh**c3Vy**

Input ends with: *any carnal plea****su*** Output ends with: YW55IGNhcm5hbCBwbGVh**c3U=**

Input ends with: *any carnal plea****s*** Output ends with: YW55IGNhcm5hbCBwbGVh**cw==**

The same characters will be encoded differently depending on their position within the three-octet group which is encoded to produce the four characters. For example

The Input: *plea****sure.*** Encodes to cGxlY**XN1cmUu**

The Input: *lea****sure.*** Encodes to bGVh**c3VyZS4=**

The Input: *ea****sure.*** Encodes to ZWF**zdXJlLg==**

The Input: *a****sure.*** Encodes to Y**XN1cmUu**

The Input: ***sure.*** Encodes to **c3VyZS4=**

The number of output bytes per input byte is approximately 4 / 3 (33% overhead) and converges to that value for large number of bytes. More specifically, given an input of *n* bytes, the output will be bytes long, including padding characters.

From a theoretical point of view the padding character is not needed, since the number of missing bytes can be calculated from the number of Base64 digits. In some implementations the padding character is mandatory to use, for others it is not used. One case where padding characters are required is when multiple Base64 encoded files are concatenated. The 2011 DEF-CON Capture the Flag (CTF) qualifiers[[6]](http://en.wikipedia.org/wiki/Base64" \l "cite_note-5) contained a puzzle with a file of concatenated Base64 encoded files.

**[**[**edit**](http://en.wikipedia.org/w/index.php?title=Base64&action=edit&section=4)**]**Decoding Base64 with padding

When decoding Base64 text 4 characters are typically converted back to 3 characters. The only exceptions are when padding characters exist. A single '=' indicates the 4 characters will decode to only 2 bytes, while 2 '='s indicates the 4 characters will decode to a single byte. This example illustrates:

Encoded text ends with: YW55IGNhcm5hbCBwbGVh**cw==** 2 '='s decodes to 1 character: *any carnal plea****s***

Encoded text ends with: YW55IGNhcm5hbCBwbGVh**c3U=** 1 '=' decodes to 2 characters: *any carnal plea****su***

Encoded text ends with: YW55IGNhcm5hbCBwbGVh**c3Vy** 0 '='s decodes to 3 characters: *any carnal plea****sur***

## BigEndian和LittleEndian

谈到字节序的问题，必然牵涉到两大CPU派系。那就是Motorola的PowerPC系列CPU和Intel的x86系列CPU。PowerPC系列采用big endian方式存储数据，而x86系列则采用little endian方式存储数据。

其实big endian是指低地址存放最高有效字节（MSB），而little endian则是低地址存放最低有效字节（LSB）。

用文字说明可能比较抽象，下面用图像加以说明。比如数字0x12345678在两种不同字节序CPU中的存储顺序如下所示：

Big Endian

低地址 高地址

----------------------------------------->

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

| 12 | 34 | 56 | 78 |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

Little Endian

低地址 高地址

----------------------------------------->

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

| 78 | 56 | 34 | 12 |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

从上面两图可以看出，采用big endian方式存储数据是符合我们人类的思维习惯的。

如果你写的程序只在单机环境下面运行，并且不和别人的程序打交道，那么你完全可以忽略字节序的存在。但是，如果你的程序要跟别人的程序产生交互呢？在这里我想说说两种语言。C/C++语言编写的程序里数据存储顺序是跟编译平台所在的CPU相关的，而JAVA编写的程序则唯一采用big endian方式来存储数据。试想，如果你用C/C++语言在x86平台下编写的程序跟别人的JAVA程序互通时会产生什么结果？就拿上面的0x12345678来说，你的程序传递给别人的一个数据，将指向0x12345678的指针传给了JAVA程序，由于JAVA采取big endian方式存储数据，很自然的它会将你的数据翻译为0x78563412。什么？竟然变成另外一个数字了？是的，就是这种后果。因此，在你的C程序传给JAVA程序之前有必要进行字节序的转换工作。

无独有偶，所有网络协议也都是采用big endian的方式来传输数据的。所以有时我们也会把big endian方式称之为网络字节序。当两台采用不同字节序的主机通信时，在发送数据之前都必须经过字节序的转换成为网络字节序后再进行传输。ANSI C中提供了下面四个转换字节序的宏。

public class BigEndianToLittleEndian {

public BigEndianToLittleEndian() {

}

public static byte[] intToByteArray(int i) {

byte[] byteArrayData = new byte[4];

byteArrayData[0] = (byte) (0xff & i);

byteArrayData[1] = (byte) (0xff & (i >> 8));

byteArrayData[2] = (byte) (0xff & (i >> 16));

byteArrayData[3] = (byte) (0xff & (i >> 24));

return byteArrayData;

}

public static byte[] floatToByteArray(float f) {

byte[] byteArrayData = new byte[4];

int i = Float.floatToIntBits(f);

byteArrayData[0] = (byte) (0xff & i);

byteArrayData[1] = (byte) (0xff & (i >> 8));

byteArrayData[2] = (byte) (0xff & (i >> 16));

byteArrayData[3] = (byte) (0xff & (i >> 24));

return byteArrayData;

}

public static byte[] stringToByteArray(String s) {

int len = s.length();

byte[] byteArrayData = new byte[len \* 2];

for (int i = 0, j = 0; i < len; i++, j++) {

byteArrayData[j] = (byte) (0xff & (int) (s.charAt(i)));

byteArrayData[++j] = (byte) (0xff & ((int) (s.charAt(i)) >> 8));

}

return byteArrayData;

}

}

## 回车、换行

In terms of ascii code, it's 3 -- since they're 10 and 13 respectively;-).

But seriously, there are many:

in Unix and all Unix-like systems, \n is the code for end-of-line, \r means nothing special

as a consequence, in C and most languages that somehow copy it (even remotely), \n is the standard escape sequence for end of line (translated to/from OS-specific sequences as needed)。

in old Mac systems (pre-OS X), \r was the code for end-of-line instead。

in Windows (and many old OSs), the code for end of line is 2 characters, `\r\n', in this order

as a (surprising;-) consequence (harking back to OSs much older than Windows), \r\n is the standard line-termination for text formats on the Internet

for electromechanical teletype-like "terminals", \r commands the carriage to go back leftwards until it hits the leftmost stop (a slow operation), \n commands the roller to roll up one line (a much faster operation) -- that's the reason you always have \r before \n, so that the roller can move while the carriage is still going leftwards!-)

for character-mode terminals (typically emulating even-older printing ones as above), in raw mode, \r and \n act similarly (except both in terms of the cursor, as there is no carriage or roller;-)

In practice, in the modern context of writing to a text file, you should always use \n (the underlying runtime will translate that if you're on a weird OS, e.g., Windows;-). The only reason to use \r is if you're writing to a character terminal (or more likely a "console window" emulating it) and want the next line you write to overwrite the last one you just wrote (sometimes used for goofy "ascii animation" effects of e.g. progress bars) -- this is getting pretty obsolete in a world of GUIs, though;-).

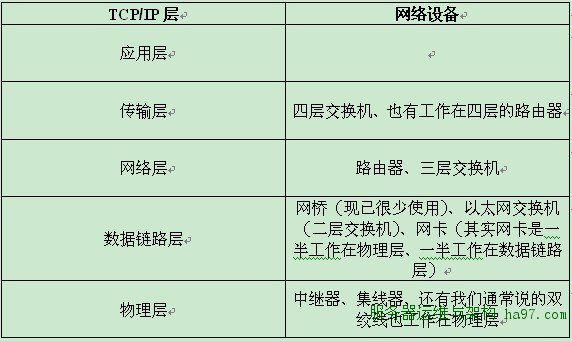
# 网络

## OSI七层与TCP/IP五层网络架构详解

**OSI和**[**TCP/IP**](http://www.ha97.com/tag/tcpip)**是很基础但又非常重要的网络基础知识。**

**（1）OSI七层模型**

OSI中的层 功能 TCP/IP协议族  
  
  
**（2）TCP/IP五层模型的协议**



**二、TCP/UDP协议**

TCP (Transmission Control Protocol)和UDP([User](http://www.ha97.com/tag/user) Datagram Protocol)协议属于传输层协议。其中TCP提供IP环境下的数据可靠传输，它提供的服务包括数据流传送、可靠性、有效流控、全双工操作和多路复 用。通过面向连接、端到端和可靠的数据包发送。通俗说，它是事先为所发送的数据开辟出连接好的通道，然后再进行数据发送；而UDP则不为IP提供可靠性、 流控或差错恢复功能。一般来说，TCP对应的是可靠性要求高的应用，而UDP对应的则是可靠性要求低、传输经济的应用。TCP支持的应用协议主要 有：Telnet、FTP、SMTP等；UDP支持的应用层协议主要有：NFS（网络文件系统）、SNMP（简单网络管理协议）、DNS（主域名称系 统）、TFTP（通用文件传输协议）等.  
TCP/IP协议与低层的数据链路层和物理层无关，这也是TCP/IP的重要特点  
  
**三、OSI的基本概念**  
OSI是Open [System](http://www.ha97.com/tag/system) Interconnect的缩写，意为开放式系统互联。  
**OSI七层参考模型的各个层次的划分遵循下列原则：**  
1、同一层中的各网络节点都有相同的层次结构，具有同样的功能。  
2、同一节点内相邻层之间通过接口（可以是逻辑接口）进行通信。  
3、七层结构中的每一层使用下一层提供的服务，并且向其上层提供服务。  
4、不同节点的同等层按照协议实现对等层之间的通信。

第一层：物理层（PhysicalLayer)，  
规定通信设备的机械的、电气的、功能的和过程的特性，用以建立、维护和拆除物理链路连接。具体地讲，机械 特性规定了网络连接时所需接插件的规格尺寸、引脚数量和排列情况等；电气特性规定了在物理连接上传输bit流时线路上信号电平的大小、阻抗匹配、传输速率 距离限制等；功能特性是指对各个信号先分配确切的信号含义，即定义了DTE和DCE之间各个线路的功能；规程特性定义了利用信号线进行bit流传输的一组 操作规程，是指在物理连接的建立、维护、交换信息是，DTE和DCE双放在各电路上的动作系列。在这一层，数据的单位称为比特（bit）。属于物理层定义的典型规范代表包括：EIA/TIA RS-232、EIA/TIA RS-449、V.35、RJ-45等。

第二层：数据链路层（DataLinkLayer):  
在物理层提供比特流服务的基础上，建立相邻结点之间的数据链路，通过差错控制提供数据帧（Frame）在信道上无差错的传输，并进行各电路上的动作系列。数据链路层在不可靠的物理介质上提供可靠的传输。该层的作用包括：物理地址寻址、数据的成帧、流量控制、数据的检错、重发等。在这一层，数据的单位称为帧（frame）。数据链路层协议的代表包括：SDLC、HDLC、PPP、STP、帧中继等。

第三层是网络层  
在 计算机网络中进行通信的两个计算机之间可能会经过很多个数据链路，也可能还要经过很多通信子网。网络层的任务就是选择合适的网间路由和交换结点， 确保数据及时传送。网络层将数据链路层提供的帧组成数据包，包中封装有网络层包头，其中含有逻辑地址信息- -源站点和目的站点地址的网络地址。如 果你在谈论一个IP地址，那么你是在处理第3层的问题，这是“数据包”问题，而不是第2层的“帧”。IP是第3层问题的一部分，此外还有一些路由协议和地 址解析协议（ARP）。有关路由的一切事情都在这第3层处理。地址解析和路由是3层的重要目的。网络层还可以实现拥塞控制、网际互连等功能。在这一层，数据的单位称为数据包（packet）。网络层协议的代表包括：IP、IPX、RIP、OSPF等。

第 四层是处理信息的传输层  
第4层的数据单元也称作数据包（packets）。但是，当你谈论TCP等具体的协议时又有特殊的叫法，TCP的数据单元称为段 （segments）而UDP协议的数据单元称为“数据报（datagrams）”。这个层负责获取全部信息，因此，它必须跟踪数据单元碎片、乱序到达的 数据包和其它在传输过程中可能发生的危险。第4层为上层提供端到端（最终用户到最终用户）的透明的、可靠的数据传输服务。所为透明的传输是指在通信过程中 传输层对上层屏蔽了通信传输系统的具体细节。传输层协议的代表包括：TCP、UDP、SPX等。

第五层是会话层  
这一层也可以称为会晤层或对话层，在会话层及以上的高层次中，数据传送的单位不再另外命名，而是统称为报文。会话层不参与具体的传输，它提供包括访问验证和会话管理在内的建立和维护应用之间通信的机制。如服务器验证用户登录便是由会话层完成的。

第六层是表示层  
这一层主要解决拥护信息的语法表示问题。它将欲交换的数据从适合于某一用户的抽象语法，转换为适合于OSI系统内部使用的传送语法。即提供格式化的表示和转换数据服务。数据的压缩和解压缩， 加密和解密等工作都由表示层负责。

第七层应用层  
应用层为操作系统或网络应用程序提供访问网络服务的接口。应用层协议的代表包括：Telnet、FTP、HTTP、SNMP等。

**除了层的数量之外，开放式系统互联（OSI）模型与TCP/IP协议有什么区别？**

开放式系统互联模型是一个参考标准，解释协议相互之间应该如何相互作用。TCP/IP协议是美国国防部发明的，是让互联网成为了目前这个样子的标准之一。开放式系统互联模型中没有清楚地描绘TCP/IP协议，但是在解释TCP/IP协议时很容易想到开放式系统互联模型。两者的主要区别如下：

TCP/IP协议中的应用层处理开放式系统互联模型中的第五层、第六层和第七层的功能。

TCP/IP协议中的传输层并不能总是保证在传输层可靠地传输数据包，而开放式系统互联模型可以做到。TCP/IP协议还提供一项名为UDP（用户数据报协议）的选择。UDP不能保证可靠的数据包传输。

TCP/IP协议与低层的数据链路层和物理层无关，这也是TCP/IP的重要特点。

OSI是Open System Interconnect的缩写，意为开放式系统互联。

# Design

[**【翻译】两种高性能I/O设计模式(Reactor/Proactor)的比较**](http://www.cppblog.com/pansunyou/archive/2011/01/26/io_design_patterns.html)

|  |  |
| --- | --- |
| **标题：** | **两种高性能I/O设计模式的比较** |
| **作者：** | Alexander Libman 、Vladimir Gilbourd |
| **原文：** | <http://www.artima.com/articles/io_design_patterns.html> |
| **时间：** | November 25, 2005 |
| **译者：** | 潘孙友 2010-01-26 于深圳 |

**综述**

这篇文章探讨并比较两种用于TCP服务器的高性能设计模式. 除了介绍现有的解决方案, 还提出了一种更具伸缩性,只需要维护一份代码并且跨平台的解决方案(含代码示例), 以及其在不同平台上的微调. 此文还比较了java,c#,c++对各自现有以及提到的解决方案的实现性能.

系统I/O 可分为阻塞型, 非阻塞同步型以及非阻塞异步型[[1](file:///E:\Projects\collection\资料\Design\【翻译】两种高性能I%20O设计模式(Reactor%20Proactor)的比较%20-%20半冷半暖秋天%20-%20C++博客.htm#resources), [2](file:///E:\Projects\collection\资料\Design\【翻译】两种高性能I%20O设计模式(Reactor%20Proactor)的比较%20-%20半冷半暖秋天%20-%20C++博客.htm#resources)]. 阻塞型I/O意味着控制权只有调用操作结束了才会回到调用者手里. 结果调用者被阻塞了, 这段时间了做不了任何其它事情. 更郁闷的是,在等待IO结果的时间里,调用者所在线程此时无法腾出手来去响应其它的请求，这真是太浪费资源了。拿read()操作来说吧, 调用此函数的代码会一直僵在此处直至它所读的socket缓存中有数据到来.

相比之下，非阻塞同步是会立即返回控制权给调用者的。调用者不需要等待，它从调用的函数获取两种结果：要么此次调用成功进行了;要么系统返回错误标识告诉调用者当前资源不可用，你再等等或者再试度看吧。比如read()操作, 如果当前socket无数据可读，则立即返回**EWOULBLOCK/EAGAIN**，告诉调用read()者"数据还没准备好，你稍后再试".

在非阻塞异步调用中，稍有不同。调用函数在立即返回时，还告诉调用者，这次请求已经开始了。系统会使用另外的资源或者线程来完成这次调用操作，并在完成的时候知会调用者（比如通过回调函数）。拿Windows的**ReadFile()**或者POSIX的**aio\_read()**来说,调用它之后，函数立即返回，操作系统在后台同时开始读操作。

在以上三种IO形式中，非阻塞异步是性能最高、伸缩性最好的。

这篇文章探讨不同的I/O利用机制并提供一种跨平台的设计模式(解决方案). 希望此文可以给于TCP高性能服务器开发者一些帮助，选择最佳的设计方案。下面我们会比较 Java, c#, C++各自对探讨方案的实现以及性能. 我们在文章的后面就不再提及阻塞式的方案了，因为阻塞式I/O实在是缺少可伸缩性，性能也达不到高性能服务器的要求。

**两种IO多路复用方案:Reactor and Proactor**

一般情况下，I/O 复用机制需要**事件分享器**(event demultiplexor [[1](file:///E:\Projects\collection\资料\Design\【翻译】两种高性能I%20O设计模式(Reactor%20Proactor)的比较%20-%20半冷半暖秋天%20-%20C++博客.htm#resources), [3](file:///E:\Projects\collection\资料\Design\【翻译】两种高性能I%20O设计模式(Reactor%20Proactor)的比较%20-%20半冷半暖秋天%20-%20C++博客.htm#resources)]). 事件分享器的作用，即将那些读写事件源分发给各读写事件的处理者，就像送快递的在楼下喊: 谁的什么东西送了, 快来拿吧。开发人员在开始的时候需要在分享器那里注册感兴趣的事件，并提供相应的处理者(event handlers)，或者是回调函数; 事件分享器在适当的时候会将请求的事件分发给这些handler或者回调函数.

涉及到事件分享器的两种模式称为：Reactor and Proactor [[1](file:///E:\Projects\collection\资料\Design\【翻译】两种高性能I%20O设计模式(Reactor%20Proactor)的比较%20-%20半冷半暖秋天%20-%20C++博客.htm#resources)]. Reactor模式是基于同步I/O的，而Proactor模式是和异步I/O相关的. 在Reactor模式中，事件分离者等待某个事件或者可应用或操作的状态发生（比如文件描述符可读写，或者是socket可读写）,事件分离者就把这个事件传给事先注册的事件处理函数或者回调函数，由后者来做实际的读写操作。

而在Proactor模式中，事件处理者(或者代由事件分离者发起)直接发起一个异步读写操作(相当于请求)，而实际的工作是由操作系统来完成的。发起时，需要提供的参数包括用于存放读到数据的缓存区，读的数据大小，或者用于存放外发数据的缓存区，以及这个请求完后的回调函数等信息。事件分离者得知了这个请求，它默默等待这个请求的完成，然后转发完成事件给相应的事件处理者或者回调。举例来说，在Windows上事件处理者投递了一个异步IO操作(称有overlapped的技术)，事件分离者等IOCompletion事件完成[[1](file:///E:\Projects\collection\资料\Design\【翻译】两种高性能I%20O设计模式(Reactor%20Proactor)的比较%20-%20半冷半暖秋天%20-%20C++博客.htm#resources)]. 这种异步模式的典型实现是基于操作系统底层异步API的，所以我们可称之为“系统级别”的或者“真正意义上”的异步，因为具体的读写是由操作系统代劳的。

举另外个例子来更好地理解Reactor与Proactor两种模式的区别。这里我们只关注read操作，因为write操作也是差不多的。下面是Reactor的做法：

* 某个事件处理者宣称它对某个socket上的读事件很感兴趣;
* 事件分离者等着这个事件的发生;
* 当事件发生了，事件分离器被唤醒，这负责通知先前那个事件处理者;
* 事件处理者收到消息，于是去那个socket上读数据了. 如果需要，它再次宣称对这个socket上的读事件感兴趣，一直重复上面的步骤;

下面再来看看真正意义的异步模式Proactor是如何做的：

* 事件处理者直接投递发一个写操作(当然，操作系统必须支持这个异步操作). 这个时候，事件处理者根本不关心读事件，它只管发这么个请求，它魂牵梦萦的是这个**写操作的完成事件**。这个处理者很拽，发个命令就不管具体的事情了，只等着别人（系统）帮他搞定的时候给他回个话。
* 事件分离者等着这个读事件的完成(比较下与Reactor的不同);
* 当事件分离者默默等待完成事情到来的同时，操作系统已经在一边开始干活了，它从目标读取数据，放入用户提供的缓存区中，最后通知事件分离者，这个事情我搞完了;
* 事件分享者通知之前的事件处理者: 你吩咐的事情搞定了;
* 事件处理者这时会发现想要读的数据已经乖乖地放在他提供的缓存区中，想怎么处理都行了。如果有需要，事件处理者还像之前一样发起另外一个写操作，和上面的几个步骤一样。

**现行做法**

开源C++开发框架 ACE[[1](file:///E:\Projects\collection\资料\Design\【翻译】两种高性能I%20O设计模式(Reactor%20Proactor)的比较%20-%20半冷半暖秋天%20-%20C++博客.htm#resources), [3](file:///E:\Projects\collection\资料\Design\【翻译】两种高性能I%20O设计模式(Reactor%20Proactor)的比较%20-%20半冷半暖秋天%20-%20C++博客.htm#resources)](Douglas Schmidt, et al.开发) 提供了大量平台独立的底层并发支持类(线程、互斥量等). 同时在更高一层它也提供了独立的几组C++类，用于实现Reactor及Proactor模式。 尽管它们都是平台独立的单元，但他们都提供了不同的接口.

ACE Proactor在MS-Windows上无论是性能还在健壮性都更胜一筹，这主要是由于Windows提供了一系列高效的底层异步API. [[4](file:///E:\Projects\collection\资料\Design\【翻译】两种高性能I%20O设计模式(Reactor%20Proactor)的比较%20-%20半冷半暖秋天%20-%20C++博客.htm#resources), [5](file:///E:\Projects\collection\资料\Design\【翻译】两种高性能I%20O设计模式(Reactor%20Proactor)的比较%20-%20半冷半暖秋天%20-%20C++博客.htm#resources)].

(这段可能过时了点吧) 不幸的是,并不是所有操作系统都为底层异步提供健壮的支持。举例来说, 许多Unix系统就有麻烦.因此, ACE Reactor可能是Unix系统上更合适的解决方案. 正因为系统底层的支持力度不一，为了在各系统上有更好的性能,开发者不得不维护独立的好几份代码: 为Windows准备的ACE Proactor以及为Unix系列提供的ACE Reactor.

就像我们提到过的，真正的异步模式需要操作系统级别的支持。由于事件处理者及操作系统交互的差异，为Reactor和Proactor设计一种通用统一的外部接口是非常困难的。这也是设计通行开发框架的难点所在。

**更好的解决方案**

在文章这一段时，我们将尝试提供一种融合了Proactor和Reactor两种模式的解决方案. 为了演示这个方案，我们将Reactor稍做调整，模拟成异步的Proactor模型(主要是在事件分离器里完成本该事件处理者做的实际读写工作,我们称这种方法为"**模拟异步**")。 下面的示例可以看看read操作是如何完成的:

* 事件处理者宣称对读事件感兴趣,并提供了用于存储结果的缓存区、读数据长度等参数;
* 调试者等待(比如通过select());
* 当有事件到来(即可读)，调试者被唤醒, 调试者去执行非阻塞的读操作(前面事件处理者已经给了足够的信息了)。读完后，它去通知事件处理者。
* 事件处理者这时被知会读操作已完成，它拥有完整的原先想要获取的数据了.

我们看到，通过为分离者(也就上面的调试者)添加一些功能，可以让Reactor模式转换为Proactor模式。所有这些被执行的操作,其实是和Reactor模型应用时完全一致的。我们只是把工作打散分配给不同的角色去完成而已。这样并不会有额外的开销，也不会有性能上的的损失，我们可以再仔细看看下面的两个过程，他们实际上完成了一样的事情：

标准的经典的 Reactor模式:

* 步骤 1) 等待事件 (Reactor 的工作)
* 步骤 2) 发"已经可读"事件发给事先注册的事件处理者或者回调 ( Reactor 要做的)
* 步骤 3) 读数据 (用户代码要做的)
* 步骤 4) 处理数据 (用户代码要做的)

模拟的Proactor模式:

* 步骤 1) 等待事件 (Proactor 的工作)
* 步骤 2) 读数据(看，这里变成成了让 Proactor 做这个事情)
* 步骤 3) 把数据已经准备好的消息给用户处理函数，即事件处理者(Proactor 要做的)
* 步骤 4) 处理数据 (用户代码要做的)

在没有底层异步I/O API支持的操作系统，这种方法可以帮我们隐藏掉socket接口的差异(无论是性能还是其它), 提供一个完全可用的统一**"异步接口"**。这样我们就可以开发真正平台独立的通用接口了。

**TProactor**

我们提出的TProactor方案已经由TerabitP/L [[6](file:///E:\Projects\collection\资料\Design\【翻译】两种高性能I%20O设计模式(Reactor%20Proactor)的比较%20-%20半冷半暖秋天%20-%20C++博客.htm#resources)]公司实现了. 它有两种实现: C++的和Java的.C++版本使用了ACE平台独立的底层元件，最终在所有操作系统上提供了统一的异步接口。

TProactor中最重要的组件要数Engine和WaitStrategy了. Engine用于维护异步操作的生命周期；而WaitStrategy用于管理并发策略. WaitStrategy和Engine一般是成对出现的, 两者间提供了良好的匹配接口.

Engines和等待策略被设计成高度可组合的(完整的实现列表请参照附录1)。TProactor是高度可配置的方案，通过使用异步内核API和同步Unix API(**select()**, **poll()**, /dev/poll (Solaris 5.8+), **port\_get** (Solaris 5.10),RealTime (RT) signals (Linux 2.4+), epoll (Linux 2.6), k-queue (FreeBSD) )，它内部实现了三种引擎(POSIX AIO, SUN AIO and Emulated AIO)并隐藏了六类等待策略。TProactor实现了和标准的 ACE Proactor一样的接口。这样一来，为不同平台提供通用统一的只有一份代码的跨平台解决方案成为可能。

Engines和WaitStrategies可以像乐高积木一样自由地组合，开发者可以在运行时通过配置参数来选择合适的内部机制（引擎和等待策略）。可以根据需求设定配置，比如连接数，系统伸缩性，以及运行的操作系统等。如果系统支持相应的异步底层API，开发人员可以选择真正的异步策略，否则用户也可以选择使用模拟出来的异步模式。所有这一切策略上的实现细节都不太需要关注，我们看到的是一个可用的异步模型。

举例来说，对于运行在Sun Solaris上的HTTP服务器，如果需要支持大量的连接数，/dev/poll或者port\_get()之类的引擎是比较合适的选择；如果需要高吞吐量，那使用基本select()的引擎会更好。由于不同选择策略内在算法的问题，像这样的弹性选择是标准ACE Reactor/Proactor模式所无法提供的（见附录2）。

在性能方面，我们的测试显示，模拟异步模式并未造成任何开销，没有变慢，反倒是性能有所提升。根据我们的测试结果，TProactor相较标签的ACE Reactor在Unix/Linux系统上有大约10-35%性能提升,而在Windows上差不多(测试了吞吐量及响应时间)。

**性能比较 (JAVA / C++ / C#).**

除了C++,我们也在Java中实现了TProactor. JDK1.4中, Java仅提供了同步方法, 像C中的select() [[7](file:///E:\Projects\collection\资料\Design\【翻译】两种高性能I%20O设计模式(Reactor%20Proactor)的比较%20-%20半冷半暖秋天%20-%20C++博客.htm#resources), [8](file:///E:\Projects\collection\资料\Design\【翻译】两种高性能I%20O设计模式(Reactor%20Proactor)的比较%20-%20半冷半暖秋天%20-%20C++博客.htm#resources)]. Java TProactor基于Java的非阻塞功能(java.nio包),类似于C++的TProactor使用了select()引擎.

图1、2显示了以 bits/sec为单位的传输速度以及相应的连接数。这些图比较了以下三种方式实现的echo服务器：标准ACE Reactor实现(基于RedHat Linux9.0)、TProactor C++/Java实现(Microsoft Windows平台及RedHat v9.0), 以及C#实现。测试的时候，三种服务器使用相同的客户端疯狂地连接，不间断地发送固定大小的数据包。

这几组测试是在相同的硬件上做的，在不同硬件上做的相对结果对比也是类似。

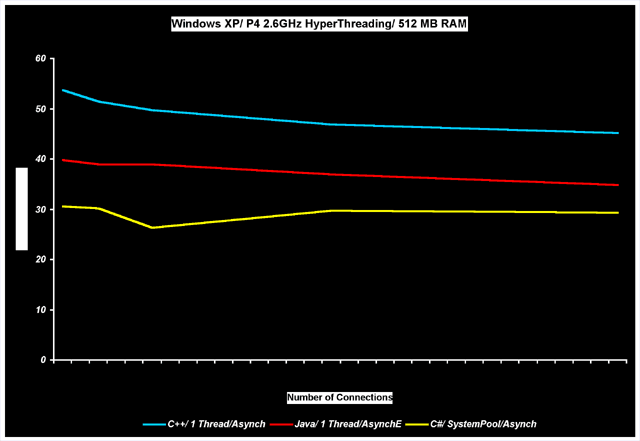


图 1. Windows XP/P4 2.6GHz HyperThreading/512 MB RAM.

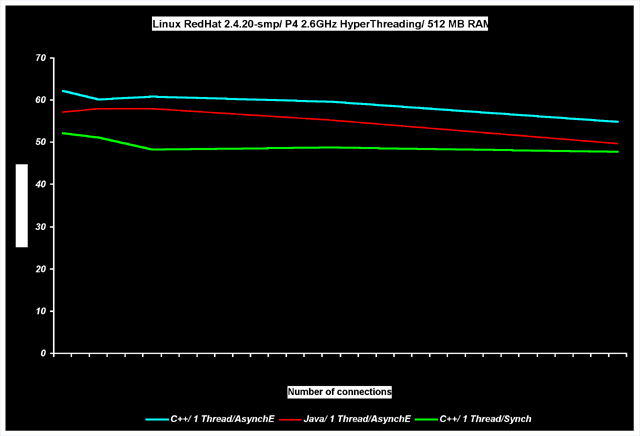


图 2. Linux RedHat 2.4.20-smp/P4 2.6GHz HyperThreading/512 MB RAM.

**用户代码示例**

下面是TProactor Java实现的echo服务器代码框架。总的来说，开发者只需要实现两个接口：一是OpRead，提供存放读结果的缓存；二是OpWrite，提供存储待写数据的缓存区。同时，开发者需要通过回调onReadComplated()和onWriteCompleted()实现协议相关的业务代码。这些回调会在合适的时候被调用.

class EchoServerProtocol implements AsynchHandler

{

AsynchChannel achannel = null;

EchoServerProtocol( Demultiplexor m, SelectableChannel channel )

throws Exception

{

this.achannel = new AsynchChannel( m, this, channel );

}

public void start() throws Exception

{

// called after construction

System.out.println( Thread.currentThread().getName() +

": EchoServer protocol started" );

achannel.read( buffer);

}

public void onReadCompleted( OpRead opRead ) throws Exception

{

if ( opRead.getError() != null )

{

// handle error, do clean-up if needed

System.out.println( "EchoServer::readCompleted: " +

opRead.getError().toString());

achannel.close();

return;

}

if ( opRead.getBytesCompleted () <= 0)

{

System.out.println("EchoServer::readCompleted: Peer closed "

+ opRead.getBytesCompleted();

achannel.close();

return;

}

ByteBuffer buffer = opRead.getBuffer();

achannel.write(buffer);

}

public void onWriteCompleted(OpWrite opWrite)

throws Exception

{

// logically similar to onReadCompleted

...

}

}

**结束语**

TProactor为多个平台提供了一个通用、弹性、可配置的高性能通讯组件，所有那些在附录2中提到的问题都被很好地隐藏在内部实现中了。

从上面的图中我们可以看出C++仍旧是编写高性能服务器最佳选择，虽然Java已紧随其后。然而因为Java本身实现上的问题，其在Windows上表现不佳(这已经应该成为历史了吧)。

需要注意的是，以上针对Java的测试，都是以裸数据的形式测试的，未涉及到数据的处理(影响性能)。

纵观AIO在Linux上的快速发展[[9](file:///E:\Projects\collection\资料\Design\【翻译】两种高性能I%20O设计模式(Reactor%20Proactor)的比较%20-%20半冷半暖秋天%20-%20C++博客.htm#resources)], 我们可以预计Linux内核API将会提供大量更加强健的异步API, 如此一来以后基于此而实现的新的Engine/等待策略将能轻松地解决能用性方面的问题，并且这也能让标准ACE Proactor接口受益。

**附录 I**

TProactor中实现的Engines 和 等待策略

|  |  |  |
| --- | --- | --- |
| **引擎类型** | **等待策略** | **操作系统** |
|  |  |  |
| POSIX\_AIO (true async) aio\_read()/aio\_write() | aio\_suspend() Waiting for RT signal Callback function | POSIX complained UNIX (not robust) POSIX (not robust) SGI IRIX, LINUX (not robust) |
| SUN\_AIO (true async) aio\_read()/aio\_write() | aio\_wait() | SUN (not robust) |
| Emulated Async Non-blocking read()/write() | select() poll() /dev/poll Linux RT signals Kqueue | generic POSIX Mostly all POSIX implementations SUN Linux FreeBSD |

**附录 II**

所有同步等待策略可划分为两组:

* edge-triggered (e.g. Linux实时信号) - signal readiness only when socket became ready (changes state);
* level-triggered (e.g. select(), poll(), /dev/poll) - readiness at any time.

让我们看看这两组的一些普遍的逻辑问题:

* edge-triggered group: after executing I/O operation, the demultiplexing loop can lose the state of socket readiness. Example: the "read" handler did not read whole chunk of data, so the socket remains still ready for read. But the demultiplexor loop will not receive next notification.
* level-triggered group: when demultiplexor loop detects readiness, it starts the write/read user defined handler. But before the start, it should remove socket descriptior from theset of monitored descriptors. Otherwise, the same event can be dispatched twice.
* Obviously, solving these problems adds extra complexities to development. All these problems were resolved internally within TProactor and the developer should not worry about those details, while in the synch approach one needs to apply extra effort to resolve them.

**资源**

[1] Douglas C. Schmidt, Stephen D. Huston "C++ Network Programming." 2002, Addison-Wesley ISBN 0-201-60464-7

[2] W. Richard Stevens "UNIX Network Programming" vol. 1 and 2, 1999, Prentice Hill, ISBN 0-13- 490012-X

[3] Douglas C. Schmidt, Michael Stal, Hans Rohnert, Frank Buschmann "Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects, Volume 2" Wiley & Sons, NY 2000

[4] INFO: Socket Overlapped I/O Versus Blocking/Non-blocking Mode. Q181611. Microsoft Knowledge Base Articles.

[5] Microsoft MSDN. I/O Completion Ports.  
[http://msdn.microsoft.com/library/default.asp?url=/library/en- us/fileio/fs/i\_o\_completion\_ports.asp](file:///E:\Projects\collection\资料\Design\%20http:\msdn.microsoft.com\library\default.asp?url=/library/en-us/fileio/fs/i_o_completion_ports.asp)

[6] TProactor (ACE compatible Proactor).  
[www.terabit.com.au](file:///E:\Projects\collection\资料\Design\www.terabit.com.au)

[7] JavaDoc java.nio.channels  
<http://java.sun.com/j2se/1.4.2/docs/api/java/nio/channels/package-summary.html>

[8] JavaDoc Java.nio.channels.spi Class SelectorProvider   
<http://java.sun.com/j2se/1.4.2/docs/api/java/nio/channels/spi/SelectorProvider.html>

[9] Linux AIO development   
<http://lse.sourceforge.net/io/aio.html>, and  
<http://archive.linuxsymposium.org/ols2003/Proceedings/All-Reprints/Reprint-Pulavarty-OLS2003.pdf>

更多

Ian Barile "I/O Multiplexing & Scalable Socket Servers", 2004 February, DDJ

Further reading on event handling  
[- http://www.cs.wustl.edu/~schmidt/ACE-papers.html](http://www.cs.wustl.edu/~schmidt/ACE-papers.html)

The Adaptive Communication Environment  
<http://www.cs.wustl.edu/~schmidt/ACE.html>

Terabit Solutions  
<http://terabit.com.au/solutions.php>

# Examples of GoF Design Patterns-Java

[**Creational patterns**](http://en.wikipedia.org/wiki/Creational_pattern)

[**Abstract factory**](http://en.wikipedia.org/wiki/Abstract_factory_pattern) **(recognizeable by creational methods returning an abstract/interface type)**

* [java.util.Calendar#getInstance()](http://docs.oracle.com/javase/6/docs/api/java/util/Calendar.html#getInstance%28%29)
* [java.util.Arrays#asList()](http://docs.oracle.com/javase/6/docs/api/java/util/Arrays.html#asList%28T...%29)
* [java.util.ResourceBundle#getBundle()](http://docs.oracle.com/javase/6/docs/api/java/util/ResourceBundle.html#getBundle%28java.lang.String%29)
* [java.net.URL#openConnection()](http://docs.oracle.com/javase/6/docs/api/java/net/URL.html#openConnection%28%29)
* [java.sql.DriverManager#getConnection()](http://docs.oracle.com/javase/6/docs/api/java/sql/DriverManager.html#getConnection%28java.lang.String%29)
* [java.sql.Connection#createStatement()](http://docs.oracle.com/javase/6/docs/api/java/sql/Connection.html#createStatement%28%29)
* [java.sql.Statement#executeQuery()](http://docs.oracle.com/javase/6/docs/api/java/sql/Statement.html#executeQuery%28java.lang.String%29)
* [java.text.NumberFormat#getInstance()](http://docs.oracle.com/javase/6/docs/api/java/text/NumberFormat.html#getInstance%28%29)
* [java.lang.management.ManagementFactory](http://docs.oracle.com/javase/6/docs/api/java/lang/management/ManagementFactory.html) (all getXXX() methods)
* [java.nio.charset.Charset#forName()](http://docs.oracle.com/javase/6/docs/api/java/nio/charset/Charset.html#forName%28java.lang.String%29)
* [javax.xml.parsers.DocumentBuilderFactory#newInstance()](http://docs.oracle.com/javase/6/docs/api/javax/xml/parsers/DocumentBuilderFactory.html#newInstance%28%29)
* [javax.xml.transform.TransformerFactory#newInstance()](http://docs.oracle.com/javase/6/docs/api/javax/xml/transform/TransformerFactory.html#newInstance%28%29)
* [javax.xml.xpath.XPathFactory#newInstance()](http://docs.oracle.com/javase/6/docs/api/javax/xml/xpath/XPathFactory.html#newInstance%28%29)
* [java.net.URLStreamHandlerFactory#createURLStreamHandler(String)](http://docs.oracle.com/javase/6/docs/api/java/net/URLStreamHandlerFactory.html) (Returns singleton object per protocol)
* Also Scripting and JDBC API.

[**Builder**](http://en.wikipedia.org/wiki/Builder_pattern) **(recognizeable by creational methods returning the instance itself)**

* [java.lang.StringBuilder#append()](http://docs.oracle.com/javase/6/docs/api/java/lang/StringBuilder.html#append%28boolean%29) (unsynchronized)
* [java.lang.StringBuffer#append()](http://docs.oracle.com/javase/6/docs/api/java/lang/StringBuffer.html#append%28boolean%29) (synchronized)
* [java.nio.ByteBuffer#put()](http://docs.oracle.com/javase/6/docs/api/java/nio/ByteBuffer.html#put%28byte%29) (also on [CharBuffer](http://docs.oracle.com/javase/6/docs/api/java/nio/CharBuffer.html#put%28char%29), [ShortBuffer](http://docs.oracle.com/javase/6/docs/api/java/nio/ShortBuffer.html#put%28short%29), [IntBuffer](http://docs.oracle.com/javase/6/docs/api/java/nio/IntBuffer.html#put%28int%29), [LongBuffer](http://docs.oracle.com/javase/6/docs/api/java/nio/LongBuffer.html#put%28long%29), [FloatBuffer](http://docs.oracle.com/javase/6/docs/api/java/nio/FloatBuffer.html#put%28float%29) and [DoubleBuffer](http://docs.oracle.com/javase/6/docs/api/java/nio/DoubleBuffer.html#put%28double%29))
* [javax.swing.GroupLayout.Group#addComponent()](http://docs.oracle.com/javase/6/docs/api/javax/swing/GroupLayout.Group.html#addComponent%28java.awt.Component%29)
* All implementations of [java.lang.Appendable](http://docs.oracle.com/javase/6/docs/api/java/lang/Appendable.html)

[**Factory method**](http://en.wikipedia.org/wiki/Factory_method_pattern) **(recognizeable by creational methods returning a concrete type)**

* [java.lang.Object#toString()](http://docs.oracle.com/javase/6/docs/api/java/lang/Object.html#toString%28%29) (overrideable in all subclasses)
* [java.lang.Class#newInstance()](http://docs.oracle.com/javase/6/docs/api/java/lang/Class.html#newInstance%28%29)
* [java.lang.Integer#valueOf(String)](http://docs.oracle.com/javase/6/docs/api/java/lang/Integer.html#valueOf%28java.lang.String%29) (also on [Boolean](http://docs.oracle.com/javase/6/docs/api/java/lang/Boolean.html#valueOf%28java.lang.String%29), [Byte](http://docs.oracle.com/javase/6/docs/api/java/lang/Byte.html#valueOf%28java.lang.String%29), [Character](http://docs.oracle.com/javase/6/docs/api/java/lang/Character.html#valueOf%28java.lang.String%29), [Short](http://docs.oracle.com/javase/6/docs/api/java/lang/Short.html#valueOf%28java.lang.String%29), [Long](http://docs.oracle.com/javase/6/docs/api/java/lang/Long.html#valueOf%28java.lang.String%29), [Float](http://docs.oracle.com/javase/6/docs/api/java/lang/Float.html#valueOf%28java.lang.String%29) and [Double](http://docs.oracle.com/javase/6/docs/api/java/lang/Double.html#valueOf%28java.lang.String%29))
* [java.lang.Class#forName()](http://docs.oracle.com/javase/6/docs/api/java/lang/Class.html#forName%28java.lang.String%29)
* [java.lang.reflect.Array#newInstance()](http://docs.oracle.com/javase/6/docs/api/java/lang/reflect/Array.html#newInstance%28java.lang.Class,%20int%29)
* [java.lang.reflect.Constructor#newInstance()](http://docs.oracle.com/javase/6/docs/api/java/lang/reflect/Constructor.html#newInstance%28java.lang.Object...%29)

[**Prototype**](http://en.wikipedia.org/wiki/Prototype_pattern) **(recognizeable by creational methods returning a *different* instance of itself with the same properties)**

* [java.lang.Object#clone()](http://docs.oracle.com/javase/6/docs/api/java/lang/Object.html#clone%28%29) (the class has to implement [java.lang.Cloneable](http://docs.oracle.com/javase/6/docs/api/java/lang/Cloneable.html))

[**Singleton**](http://en.wikipedia.org/wiki/Singleton_pattern) **(recognizeable by creational methods returning the *same* instance (usually of itself) everytime)**

* [java.lang.Runtime#getRuntime()](http://docs.oracle.com/javase/6/docs/api/java/lang/Runtime.html#getRuntime%28%29)
* [java.awt.Desktop#getDesktop()](http://docs.oracle.com/javase/6/docs/api/java/awt/Desktop.html#getDesktop%28%29)

[**Structural patterns**](http://en.wikipedia.org/wiki/Structural_pattern)

[**Adapter**](http://en.wikipedia.org/wiki/Adapter_pattern) **(recognizeable by creational methods taking an instance of *different* abstract/interface type and returning an implementation of own/another abstract/interface type which *decorates/overrides* the given instance)**

* [java.io.InputStreamReader(InputStream)](http://docs.oracle.com/javase/6/docs/api/java/io/InputStreamReader.html#InputStreamReader%28java.io.InputStream%29) (returns a Reader)
* [java.io.OutputStreamWriter(OutputStream)](http://docs.oracle.com/javase/6/docs/api/java/io/OutputStreamWriter.html#OutputStreamWriter%28java.io.OutputStream%29) (returns a Writer)
* [javax.xml.bind.annotation.adapters.XmlAdapter#marshal()](http://docs.oracle.com/javase/6/docs/api/javax/xml/bind/annotation/adapters/XmlAdapter.html#marshal%28BoundType%29) and [#unmarshal()](http://docs.oracle.com/javase/6/docs/api/javax/xml/bind/annotation/adapters/XmlAdapter.html#unmarshal%28ValueType%29)

[**Bridge**](http://en.wikipedia.org/wiki/Bridge_pattern) **(recognizeable by creational methods taking an instance of *different* abstract/interface type and returning an implementation of own abstract/interface type which *delegates/uses* the given instance)**

* None comes to mind yet. A fictive example would be new LinkedHashMap(LinkedHashSet<K>, List<V>) which returns an unmodifiable linked map which doesn't clone the items, but *uses* them. The [java.util.Collections#newSetFromMap()](http://docs.oracle.com/javase/6/docs/api/java/util/Collections.html#newSetFromMap%28java.util.Map%29) and [singletonXXX()](http://docs.oracle.com/javase/6/docs/api/java/util/Collections.html#singleton%28T%29) methods however comes close.

[**Composite**](http://en.wikipedia.org/wiki/Composite_pattern) **(recognizeable by behavioral methods taking an instance of *same* abstract/interface type into a tree structure)**

* [java.awt.Container#add(Component)](http://docs.oracle.com/javase/6/docs/api/java/awt/Container.html#add%28java.awt.Component%29) (practically all over Swing thus)
* [javax.faces.component.UIComponent#getChildren()](http://docs.oracle.com/javaee/6/api/javax/faces/component/UIComponent.html#getChildren%28%29) (practically all over JSF UI thus)

[**Decorator**](http://en.wikipedia.org/wiki/Decorator_pattern) **(recognizeable by creational methods taking an instance of *same* abstract/interface type which adds additional behaviour)**

* All subclasses of [java.io.InputStream](http://docs.oracle.com/javase/6/docs/api/java/io/InputStream.html), [OutputStream](http://docs.oracle.com/javase/6/docs/api/java/io/OutputStream.html), [Reader](http://docs.oracle.com/javase/6/docs/api/java/io/Reader.html) and [Writer](http://docs.oracle.com/javase/6/docs/api/java/io/Writer.html) have a constructor taking an instance of same type.
* [java.util.Collections](http://docs.oracle.com/javase/6/docs/api/java/util/Collections.html), the [checkedXXX()](http://docs.oracle.com/javase/6/docs/api/java/util/Collections.html#checkedCollection%28java.util.Collection,%20java.lang.Class%29), [synchronizedXXX()](http://docs.oracle.com/javase/6/docs/api/java/util/Collections.html#synchronizedCollection%28java.util.Collection%29) and [unmodifiableXXX()](http://docs.oracle.com/javase/6/docs/api/java/util/Collections.html#unmodifiableCollection%28java.util.Collection%29) methods.
* [javax.servlet.http.HttpServletRequestWrapper](http://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpServletRequestWrapper.html) and [HttpServletResponseWrapper](http://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpServletResponseWrapper.html)

[**Facade**](http://en.wikipedia.org/wiki/Facade_pattern) **(recognizeable by behavioral methods which internally uses instances of *different* independent abstract/interface types)**

* [javax.faces.context.FacesContext](http://docs.oracle.com/javaee/6/api/javax/faces/context/FacesContext.html), it internally uses among others the abstract/interface types [LifeCycle](http://docs.oracle.com/javaee/6/api/javax/faces/lifecycle/Lifecycle.html), [ViewHandler](http://docs.oracle.com/javaee/6/api/javax/faces/application/ViewHandler.html), [NavigationHandler](http://docs.oracle.com/javaee/6/api/javax/faces/application/NavigationHandler.html) and many more without that the enduser has to worry about it (which are however overrideable by injection).
* [javax.faces.context.ExternalContext](http://docs.oracle.com/javaee/6/api/javax/faces/context/ExternalContext.html), which internally uses [ServletContext](http://docs.oracle.com/javaee/6/api/javax/servlet/ServletContext.html), [HttpSession](http://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpSession.html), [HttpServletRequest](http://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpServletRequest.html), [HttpServletResponse](http://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpServletResponse.html), etc.

[**Flyweight**](http://en.wikipedia.org/wiki/Flyweight_pattern) **(recognizeable by creational methods returning a cached instance, a bit the "multiton" idea)**

* [java.lang.Integer#valueOf(int)](http://docs.oracle.com/javase/6/docs/api/java/lang/Integer.html#valueOf%28int%29) (also on [Boolean](http://docs.oracle.com/javase/6/docs/api/java/lang/Boolean.html#valueOf%28boolean%29), [Byte](http://docs.oracle.com/javase/6/docs/api/java/lang/Byte.html#valueOf%28byte%29), [Character](http://docs.oracle.com/javase/6/docs/api/java/lang/Character.html#valueOf%28char%29), [Short](http://docs.oracle.com/javase/6/docs/api/java/lang/Short.html#valueOf%28short%29), [Long](http://docs.oracle.com/javase/6/docs/api/java/lang/Long.html#valueOf%28long%29), [Float](http://docs.oracle.com/javase/6/docs/api/java/lang/Float.html#valueOf%28float%29) and [Double](http://docs.oracle.com/javase/6/docs/api/java/lang/Double.html#valueOf%28double%29))
* Interned Strings or CellRender in JTable for Flyweight Pattern (Also think about various pools - Thread pools, connection pools, EJB object pools - Flyweight is really about management of shared resources)

[**Proxy**](http://en.wikipedia.org/wiki/Proxy_pattern) **(recognizeable by creational methods which returns an implementation of given abstract/interface type which in turn *delegates/uses* a *different* implementation of given abstract/interface type)**

* [java.lang.reflect.Proxy](http://docs.oracle.com/javase/6/docs/api/java/lang/reflect/Proxy.html)
* [java.rmi.\*](http://docs.oracle.com/javase/6/docs/api/java/rmi/package-summary.html), the whole API actually.

The Wikipedia example is IMHO a bit poor, lazy loading has actually completely nothing to do with the proxy pattern at all.

[**Behavioral patterns**](http://en.wikipedia.org/wiki/Behavioral_pattern)

[**Chain of responsibility**](http://en.wikipedia.org/wiki/Chain_of_responsibility_pattern) **(recognizeable by behavioral methods which (indirectly) invokes the same method in *another* implementation of *same* abstract/interface type in a queue)**

* [java.util.logging.Logger#log()](http://docs.oracle.com/javase/6/docs/api/java/util/logging/Logger.html#log%28java.util.logging.Level,%20java.lang.String%29)
* [javax.servlet.Filter#doFilter()](http://docs.oracle.com/javaee/6/api/javax/servlet/Filter.html#doFilter%28javax.servlet.ServletRequest,%20javax.servlet.ServletResponse,%20javax.servlet.FilterChain%29)
* The Java 1.0 event model is an example of Chain of Responsibility, as are Servlet Filters.

[**Command**](http://en.wikipedia.org/wiki/Command_pattern) **(recognizeable by behavioral methods in an abstract/interface type which invokes a method in an implementation of a *different* abstract/interface type which has been *encapsulated* by the command implementation during its creation)**

* All implementations of [java.lang.Runnable](http://docs.oracle.com/javase/6/docs/api/java/lang/Runnable.html)
* All implementations of [javax.swing.Action](http://docs.oracle.com/javase/6/docs/api/javax/swing/Action.html)
* TextComponent's Undo/Redo

[**Interpreter**](http://en.wikipedia.org/wiki/Interpreter_pattern) **(recognizeable by behavioral methods returning a *structurally* different instance/type of the given instance/type; note that parsing/formatting is not part of the pattern, determining the pattern and how to apply it is)**

* [java.util.Pattern](http://docs.oracle.com/javase/6/docs/api/java/util/regex/Pattern.html)
* [java.text.Normalizer](http://docs.oracle.com/javase/6/docs/api/java/text/Normalizer.html)
* All subclasses of [java.text.Format](http://docs.oracle.com/javase/6/docs/api/java/text/Format.html)
* All subclasses of [javax.el.ELResolver](http://docs.oracle.com/javaee/6/api/javax/el/ELResolver.html)
* RegEx (java.util.regex.) and SQL (java.sql.) API.

[**Iterator**](http://en.wikipedia.org/wiki/Iterator_pattern) **(recognizeable by behavioral methods sequentially returning instances of a *different* type from a queue)**

* All implementations of [java.util.Iterator](http://docs.oracle.com/javase/6/docs/api/java/util/Iterator.html) (thus among others also [java.util.Scanner](http://docs.oracle.com/javase/6/docs/api/java/util/Scanner.html)!).
* All implementations of [java.util.Enumeration](http://docs.oracle.com/javase/6/docs/api/java/util/Enumeration.html)
* Iterator pattern in Collections Framework

[**Mediator**](http://en.wikipedia.org/wiki/Mediator_pattern) **(recognizeable by behavioral methods taking an instance of different abstract/interface type (usually using the command pattern) which delegates/uses the given instance)**

* [java.util.Timer](http://docs.oracle.com/javase/6/docs/api/java/util/Timer.html) (all scheduleXXX() methods)
* [java.util.concurrent.Executor#execute()](http://docs.oracle.com/javase/6/docs/api/java/util/concurrent/Executor.html#execute%28java.lang.Runnable%29)
* [java.util.concurrent.ExecutorService](http://docs.oracle.com/javase/6/docs/api/java/util/concurrent/ExecutorService.html) (the invokeXXX() and submit() methods)
* [java.util.concurrent.ScheduledExecutorService](http://docs.oracle.com/javase/6/docs/api/java/util/concurrent/ScheduledExecutorService.html) (all scheduleXXX() methods)
* [java.lang.reflect.Method#invoke()](http://docs.oracle.com/javase/6/docs/api/java/lang/reflect/Method.html#invoke%28java.lang.Object,%20java.lang.Object...%29)

[**Memento**](http://en.wikipedia.org/wiki/Memento_pattern) **(recognizeable by behavioral methods which internally changes the state of the *whole* instance)**

* [java.util.Date](http://docs.oracle.com/javase/6/docs/api/java/util/Date.html) (the setter methods do that, Date is internally represented by a long value)
* All implementations of [java.io.Serializable](http://docs.oracle.com/javase/6/docs/api/java/io/Serializable.html)
* All implementations of [javax.faces.component.StateHolder](http://docs.oracle.com/javaee/6/api/javax/faces/component/StateHolder.html)

[**Observer (or Publish/Subscribe)**](http://en.wikipedia.org/wiki/Observer_pattern) **(recognizeable by behavioral methods which invokes a method on an instance of *another* abstract/interface type, depending on own state)**

* [java.util.Observer](http://docs.oracle.com/javase/6/docs/api/java/util/Observer.html)/[java.util.Observable](http://docs.oracle.com/javase/6/docs/api/java/util/Observable.html) (rarely used in real world though)
* All implementations of [java.util.EventListener](http://docs.oracle.com/javase/6/docs/api/java/util/EventListener.html) (practically all over Swing thus)
* [javax.servlet.http.HttpSessionBindingListener](http://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpSessionBindingListener.html)
* [javax.servlet.http.HttpSessionAttributeListener](http://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpSessionAttributeListener.html)
* [javax.faces.event.PhaseListener](http://docs.oracle.com/javaee/6/api/javax/faces/event/PhaseListener.html)

[**State**](http://en.wikipedia.org/wiki/State_pattern) **(recognizeable by behavioral methods which changes its behaviour depending on the instance's state which can be controlled externally)**

* [javax.faces.lifecycle.LifeCycle#execute()](http://docs.oracle.com/javaee/6/api/javax/faces/lifecycle/Lifecycle.html#execute%28javax.faces.context.FacesContext%29) (controlled by [FacesServlet](http://docs.oracle.com/javaee/6/api/javax/faces/webapp/FacesServlet.html), the behavior is dependent on current phase (state) of JSF lifecycle)

[**Strategy**](http://en.wikipedia.org/wiki/Strategy_pattern) **(recognizeable by behavioral methods in an abstract/interface type which invokes a method in an implementation of a *different* abstract/interface type which has been *passed-in* as method argument into the strategy implementation)**

* [java.util.Comparator#compare()](http://docs.oracle.com/javase/6/docs/api/java/util/Comparator.html#compare%28T,%20T%29), executed by among others Collections#sort().
* [javax.servlet.http.HttpServlet](http://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpServlet.html), the service() and all doXXX() methods take HttpServletRequest and HttpServletResponse and the implementor has to process them (and not to get hold of them as instance variables!).
* [javax.servlet.Filter#doFilter()](http://docs.oracle.com/javaee/6/api/javax/servlet/Filter.html#doFilter%28javax.servlet.ServletRequest,%20javax.servlet.ServletResponse,%20javax.servlet.FilterChain%29)
* Layout Managers in AWT/Swing are an example of Strategy

[**Template method**](http://en.wikipedia.org/wiki/Template_method_pattern) **(recognizeable by behavioral methods which already have a "default" behaviour definied by an abstract type)**

* All non-abstract methods of [java.io.InputStream](http://docs.oracle.com/javase/6/docs/api/java/io/InputStream.html), [java.io.OutputStream](http://docs.oracle.com/javase/6/docs/api/java/io/OutputStream.html), [java.io.Reader](http://docs.oracle.com/javase/6/docs/api/java/io/Reader.html) and [java.io.Writer](http://docs.oracle.com/javase/6/docs/api/java/io/Writer.html).
* All non-abstract methods of [java.util.AbstractList](http://docs.oracle.com/javase/6/docs/api/java/util/AbstractList.html), [java.util.AbstractSet](http://docs.oracle.com/javase/6/docs/api/java/util/AbstractSet.html) and [java.util.AbstractMap](http://docs.oracle.com/javase/6/docs/api/java/util/AbstractMap.html).
* [javax.servlet.http.HttpServlet](http://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpServlet.html), all the doXXX() methods by default sends a HTTP 405 "Method Not Allowed" error to the response. You're free to implement none or any of them.

[**Visitor**](http://en.wikipedia.org/wiki/Visitor_pattern) **(recognizeable by two *different* abstract/interface types which has methods definied which takes each the *other* abstract/interface type; the one actually calls the method of the other and the other executes the desired strategy on it)**

* [javax.lang.model.element.AnnotationValue](http://docs.oracle.com/javase/6/docs/api/javax/lang/model/element/AnnotationValue.html) and [AnnotationValueVisitor](http://docs.oracle.com/javase/6/docs/api/javax/lang/model/element/AnnotationValueVisitor.html)
* [javax.lang.model.element.Element](http://docs.oracle.com/javase/6/docs/api/javax/lang/model/element/Element.html) and [ElementVisitor](http://docs.oracle.com/javase/6/docs/api/javax/lang/model/element/ElementVisitor.html)
* [javax.lang.model.type.TypeMirror](http://docs.oracle.com/javase/6/docs/api/javax/lang/model/type/TypeMirror.html) and [TypeVisitor](http://docs.oracle.com/javase/6/docs/api/javax/lang/model/type/TypeVisitor.html)

# 数据库

## 基础

### 事务

ACID:Atomic、Consistent、Isolated、Durable

事务都应该具备ACID特征。所谓ACID是Atomic（原子性），Consistent（一致性），Isolated（隔离性），Durable（持续性）四个词的首字母所写，下面以“银行转帐”为例来分别说明一下它们的含义：

原子性：组成事务处理的语句形成了一个逻辑单元，不能只执行其中的一部分。换句话说，事务是不可分割的最小单元。比如：银行转帐过程中，必须同时从一个帐户减去转帐金额，并加到另一个帐户中，只改变一个帐户是不合理的。

一致性：在事务处理执行前后，数据库是一致的。也就是说，事务应该正确的转换系统状态。比如：银行转帐过程中，要么转帐金额从一个帐户转入另一个帐户，要么两个帐户都不变，没有其他的情况。

隔离性：一个事务处理对另一个事务处理没有影响。就是说任何事务都不可能看到一个处在不完整状态下的事务。比如说，银行转帐过程中，在转帐事务没有提交之前，另一个转帐事务只能处于等待状态。

持续性：事务处理的效果能够被永久保存下来。反过来说，事务应当能够承受所有的失败，包括服务器、进程、通信以及媒体失败等等。比如：银行转帐过程中，转帐后帐户的状态要能被保存下来。

隔离级别：

隔离级别决定了一个session中的事务可能对另一个session的影响、并发session对数据库的操作、一个session中所见数据的一致性

ANSI标准定义了4个隔离级别，

1. READ UNCOMMITTED：最低级别的隔离，通常又称为dirty read，它允许一个事务读取还没commit的数据，这样可能会提高性能，但是dirty read可能不是我们想要的
2. READ COMMITTED：在一个事务中只允许已经commit的记录可见，如果session中select还在查询中，另一session此时insert一条记录，则新添加的数据不可见
3. REPEATABLE READ：在一个事务开始后，其他session对数据库的修改在本事务中不可见，直到本事务commit或rollback。在一个事务中重复select的结果一样，除非本事务中update数据库。
4. SERIALIZABLE：最高级别的隔离，只允许事务串行执行。为了达到此目的，数据库会锁住每行已经读取的记录，其他session不能修改数据直到前一事务结束，事务commit或取消时才释放锁。

### Closing BasicDataSource doesn't deregister JDBC driver, causing memory leak

BasicDataSource's method close() doesn't deregister JDBC driver. This causes permgen memory leaks in web server environments, during context reloads. For example, using Tomcat 6.0.26 with Spring, and BasicDataSource declared in Spring context, there is a message printed at web application reload:

SEVERE: A web application registered the JBDC driver [com.mysql.jdbc.Driver] but failed to unregister it when the web application was stopped. To prevent a memory leak, the JDBC Driver has been forcibly unregistered.

I was able to fix it by overriding close method this way:

public class XBasicDataSource extends BasicDataSource {

@Override

public synchronized void close() throws SQLException {

DriverManager.deregisterDriver(DriverManager.getDriver(url));

super.close();

}

}

but I think it should be probably the default behavior of BasicDataSource. Or perhaps there should be some flag/setting on BasicDataSource, named "deregisterDriverAtClose" or so.

### JDBC获取DB元数据

|  |
| --- |
| package com.util.jdbc;  import java.sql.Connection;  import java.sql.Driver;  import java.sql.DriverManager;  import java.sql.PreparedStatement;  import java.sql.ResultSet;  import java.sql.ResultSetMetaData;  import java.sql.SQLException;  import java.util.Enumeration;  import org.apache.log4j.Logger;  /\*\*  \*/  public class JDBCMsg {  private static Logger log = Logger.getLogger(JDBCMsg.class);    public static void main(String[] args) throws SQLException {  //rs connection 未关闭  //getAllDriverMsg();  //getDBParameterMetaData();  demoDB();  }    /\*\*  \* 取一个数据库中所有表的信息  \* @throws SQLException  \*/  public static void demoDB() throws SQLException{  Connection conn = JDBCUtil.getConnection();  log.info("###### DatabaseMetaData关于数据库的整体综合信息====");  java.sql.DatabaseMetaData dbmd = conn.getMetaData();  log.info("数据库产品名: " + dbmd.getDatabaseProductName());  log.info("数据库是否支持事务: " + dbmd.supportsTransactions());  log.info("数据库产品的版本号:"+dbmd.getDatabaseProductVersion());  log.info("数据库的默认事务隔离级别:"+dbmd.getDefaultTransactionIsolation());  log.info("支持批量更新:"+dbmd.supportsBatchUpdates());  log.info("DBMS 的 URL:"+dbmd.getURL());  log.info("数据库的已知的用户名称:"+dbmd.getUserName());  log.info("数据库是否处于只读模式:"+dbmd.isReadOnly());  log.info("数据库是否支持为列提供别名:"+dbmd.supportsColumnAliasing());  log.info("是否支持指定 LIKE 转义子句:"+dbmd.supportsLikeEscapeClause());  log.info("是否为外连接提供受限制的支持:"+dbmd.supportsLimitedOuterJoins());  log.info("是否允许一次打开多个事务:"+dbmd.supportsMultipleTransactions());  log.info("是否支持 EXISTS 表达式中的子查询:"+dbmd.supportsSubqueriesInExists());  log.info("是否支持 IN 表达式中的子查询:"+dbmd.supportsSubqueriesInIns());  log.info("是否支持给定事务隔离级别:"+dbmd.supportsTransactionIsolationLevel(1));  log.info("此数据库是否支持事务:"+dbmd.supportsTransactions());  log.info("此数据库是否支持 SQL UNION:"+dbmd.supportsUnion());  log.info("此数据库是否支持 SQL UNION ALL:"+dbmd.supportsUnionAll());  log.info("此数据库是否为每个表使用一个文件:"+dbmd.usesLocalFilePerTable());  log.info("此数据库是否将表存储在本地文件中:"+dbmd.usesLocalFiles());  log.info("底层数据库的主版本号:"+dbmd.getDatabaseMajorVersion());  log.info("底层数据库的次版本号:"+dbmd.getDatabaseMinorVersion());    log.info("JDBC 驱动程序的主版本号:"+dbmd.getJDBCMajorVersion());  log.info("JDBC 驱动程序的次版本号:"+dbmd.getJDBCMinorVersion());  log.info("JDBC 驱动程序的名称:"+dbmd.getDriverName());  log.info("JDBC 驱动程序的 String 形式的版本号:"+dbmd.getDriverVersion());    log.info("可以在不带引号的标识符名称中使用的所有“额外”字符:"+dbmd.getExtraNameCharacters());  log.info("用于引用 SQL 标识符的字符串:"+dbmd.getIdentifierQuoteString());  log.info("允许用于类别名称的最大字符数:"+dbmd.getMaxCatalogNameLength());  log.info("允许用于列名称的最大字符数:"+dbmd.getMaxColumnNameLength());  log.info("允许在 GROUP BY 子句中使用的最大列数:"+dbmd.getMaxColumnsInGroupBy());  log.info("允许在 SELECT 列表中使用的最大列数:"+dbmd.getMaxColumnsInSelect());  log.info("允许在表中使用的最大列数:"+dbmd.getMaxColumnsInTable());  log.info("数据库的并发连接的可能最大数:"+dbmd.getMaxConnections());  log.info("允许用于游标名称的最大字符数:"+dbmd.getMaxCursorNameLength());  log.info("在同一时间内可处于开放状态的最大活动语句数:"+dbmd.getMaxStatements());    //获取所有表 new String[]{"TABLE"}  //String[] type = {"TABLE","VIEW"} null  log.info("###### 获取表的信息");  ResultSet tSet = dbmd.getTables(null, "%", "%", new String[]{"TABLE","VIEW"});  while (tSet.next()) {  log.info(tSet.getRow()+"\_表类别:"+tSet.getString("TABLE\_CAT")+"\_表模式:"+tSet.getString("TABLE\_SCHEM")  +"\_表名称:"+tSet.getString("TABLE\_NAME")+"\_表类型:"+tSet.getString("TABLE\_TYPE")  //+"\n\_表的解释性注释:"+tSet.getString("REMARKS")+"\_类型的类别:"+tSet.getString("TYPE\_CAT")  //+"\n\_类型模式:"+tSet.getString("TYPE\_SCHEM")+"\_类型名称:"+tSet.getString("TYPE\_NAME")  //+"\n\_有类型表的指定'identifier'列的名称:"+tSet.getString("SELF\_REFERENCING\_COL\_NAME")  //+"\n\_指定在 SELF\_REFERENCING\_COL\_NAME 中创建值的方式:"+tSet.getString("REF\_GENERATION")  );  //2\_表类别:MANOR\_表模式:PUBLIC\_表名称:SYS\_RESOURCE\_表类型:TABLE  String tableName = tSet.getString(3);  String sql = "select \* from " + tableName;  ResultSet rsSet = conn.createStatement().executeQuery(sql);  ResultSetMetaData rsData = rsSet.getMetaData();  for (int i = 1; i <= rsData.getColumnCount(); i++) {  log.info("==列的信息:获取SQL语句的列名:"+rsData.getColumnName(i)+"("+rsData.getColumnLabel(i)+","+rsData.getColumnType(i)+","+rsData.getColumnClassName(i)+")"  +" 列宽"+rsData.getPrecision(i)+" 大小写敏感"+rsData.isCaseSensitive(i)+" isReadOnly:"+rsData.isReadOnly(i));  //==列的信息:获取SQL语句的列名:LIMITLEVER(LIMITLEVER,5,java.lang.Short) 列宽5 大小写敏感true isReadOnly:false  }    }  tSet.close();    log.info("###### 获取当前数据库所支持的SQL数据类型");  ResultSet tableType = dbmd.getTypeInfo();  while(tableType.next()){  log.info("数据类型名:"+tableType.getString(1)  +",短整型的数:"+tableType.getString(2)  +",整型的数:"+tableType.getString(3)  +",最小精度:"+tableType.getString(14)  +",最大精度:"+tableType.getString(15));  //数据类型名:TIMESTAMP,短整型的数:93,整型的数:23,最小精度:0,最大精度:10  //数据类型名:VARCHAR,短整型的数:12,整型的数:2147483647,最小精度:0,最大精度:0  }    log.info("###### 表的主键列信息");  ResultSet primaryKey = dbmd.getPrimaryKeys("MANOR","PUBLIC","SYS\_ROLE\_RES");  while(primaryKey.next()){  log.info("表名:"+primaryKey.getString("TABLE\_NAME")+",列名:"+primaryKey.getString("COLUMN\_NAME")  +" 主键名:"+primaryKey.getString("PK\_NAME"));  //表名:SYS\_ROLE\_RES,列名:SYS\_RES\_ID 主键名:CONSTRAINT\_9  //表名:SYS\_ROLE\_RES,列名:SYS\_ROLE\_ID 主键名:CONSTRAINT\_9  }    log.info("###### 表的外键列信息");  ResultSet foreinKey = dbmd.getImportedKeys("MANOR","PUBLIC","SYS\_ROLE\_RES");  while(foreinKey.next()){  log.info("主键名:"+foreinKey.getString("PK\_NAME")+",外键名:"+foreinKey.getString("FKCOLUMN\_NAME")  +",主键表名:"+foreinKey.getString("PKTABLE\_NAME")+",外键表名:"+foreinKey.getString("FKTABLE\_NAME")  +",外键列名:"+foreinKey.getString("PKCOLUMN\_NAME")+",外键序号:"+foreinKey.getString("KEY\_SEQ"));  //主键名:PRIMARY\_KEY\_95,外键名:SYS\_RES\_ID,主键表名:SYS\_RESOURCE,外键表名:SYS\_ROLE\_RES,外键列名:ID,外键序号:1  //主键名:PRIMARY\_KEY\_A,外键名:SYS\_ROLE\_ID,主键表名:SYS\_ROLE,外键表名:SYS\_ROLE\_RES,外键列名:ID,外键序号:1  }    log.info("###### 获取数据库中允许存在的表类型");  ResultSet tableTypes = dbmd.getTableTypes();  while(tableTypes.next()){  log.info("类型名:"+tableTypes.getString(1));  /\*\* H2  类型名:SYSTEM TABLE  类型名:TABLE  类型名:TABLE LINK  类型名:VIEW  \*/  }    //此外还可以获取索引等的信息  conn.close();  }    /\*\*  \* PreparedStatement 信息  \* ResultSetMetaData 信息  \* @throws SQLException  \*/  public static void getDBParameterMetaData() throws SQLException{  Connection conn = JDBCUtil.getConnection(); //id,name  PreparedStatement pre = conn.prepareStatement("SELECT \* FROM SYS\_APPTYPE where id = ?");  pre.setInt(1, 3);  java.sql.ParameterMetaData pmd = pre.getParameterMetaData();  log.info("参数的个数:"+pmd.getParameterCount());  log.info("获取指定参数的 SQL 类型:"+pmd.getParameterType(1));  log.info("culomn的参数类型:"+pmd.getParameterTypeName(1));  log.info("Java 类的完全限定名称:"+pmd.getParameterClassName(1));  log.info("获取指定参数的模式:"+pmd.getParameterMode(1));  log.info("获取指定参数的指定列大小:"+pmd.getPrecision(1));  log.info("获取指定参数的小数点右边的位数:"+pmd.getScale(1));  log.info("是否允许在指定参数中使用 null 值:"+pmd.isNullable(1));  log.info("指定参数的值是否可以是带符号的数字:"+pmd.isSigned(1));    //获取结果集元数据  ResultSet rs = pre.executeQuery();  while (rs.next()) {  log.info(rs.getString(1)+"\_\_\_"+rs.getString(2));  }  rs.close();  }  /\*\*  \* 获取所有Driver信息  \*/  public static void getAllDriverMsg(){  Enumeration<Driver> drivers = DriverManager.getDrivers();  while(drivers.hasMoreElements()) {  Driver d = drivers.nextElement();  log.info(d.getClass().getName()+"\_"+d.getMajorVersion());  }  }  } |

### jdbc性能优化

系统性能.

少用Metadata方法

与其它的JDBC方法相比, 由ResultSet对象生成的metadata对象的相对来说是很慢的. 应用程序应该缓存从ResultSet返回的metadata信息，避免多次不必要的执行这个操作.

几乎没有哪一个JDBC应用程序不用到metadata，虽然如此，你仍可以通过少用它们来改善系统性能. 要返回JDBC规范规定的结果集的所有列信息, 一个简单的metadata的方法调用可能会使JDBC驱动程序去执行很复杂的查询甚至多次查询去取得这些数据. 这些细节上的SQL语言的操作是非常消耗性能的.

应用程序应该缓存这些metadata信息. 例如, 程序调用一次getTypeInfo方法后就将这些程序所依赖的结果信息缓存. 而任何程序都不大可能用到这些结果信息中的所有内容，所以这些缓存信息应该是不难维护的.

避免null参数

在metadata的方法中使用null参数或search patterns是很耗时的. 另外, 额外的查询会导致潜在的网络交通的增加. 应尽可能的提供一些non-null的参数给metadata方法.

因为metadata的方法很慢, 应用程序要尽可能有效的调用它们. 许多应用程序只传递少量的non-null参数给这些方法.

例如:

ResultSet WSrs = WSc.getTables (null, null, "WSTable", null);

应该这样:

ResultSet WSrs = WSc.getTables ("cat1", "johng", "WSTable", "TABLE");

在第一个getTables()的调用中, 程序可能想知道表'WSTable'是否存在. 当然, JDBC驱动程序会逐个调用它们并且会解译不同的请求. JDBC驱动程序会解译请求为: 返回所有的表, 视图, 系统表, synonyms, 临时表, 或存在于任何数据库类别任何Schema中的任何别名为'WSTable'的对象.

第二个getTables()的调用会得到更正确的程序想知道的内容. JDBC驱动程序会解译这个请求为: 返回当前数据库类别中所有存在于'johng'这个schema中的所有表.

很显然, JDBC驱动程序处理第二个请求比处理第一个请求更有效率一些.

有时, 你所请求信息中的对象有些信息是已知的. 当调用metadata方法时, 程序能传送到驱动程序的的任何有用信息都可以导致性能和可靠性的改善.

使用'哑元'(dummy)查询确定表的特性

要避免使用getColumns()去确定一个表的特性. 而应该使用一个‘哑元’查询来使用getMetadata()方法.

请考虑这样一个程序, 程序中要允许用户选取一些列. 我们是否应该使用getColumns()去返回列信息给用户还是以一个'哑元'查询来调用getMetadata()方法呢?

案例 1: GetColumns 方法

ResultSet WSrc = WSc.getColumns (... "UnknownTable" ...);

// getColumns()会发出一个查询给数据库系统

. . .

WSrc.next();

string Cname = getString(4);

. . .

// 用户必须从反复从服务器获取N行数据

// N = UnknownTable的列数

案例 2: GetMetadata 方法

// 准备'哑元'查询

PreparedStatement WSps = WSc.prepareStatement

("SELECT \* from UnknownTable WHERE 1 = 0");

// 查询从来没有被执行，只是被预储

ResultSetMetaData WSsmd=WSps.getMetaData();

int numcols = WSrsmd.getColumnCount();

...

int ctype = WSrsmd.getColumnType(n)

...

// 获得了列的完整信息

在这两个案例中, 一个查询被传送到服务器. 但在案例1中, 查询必须被预储和执行, 结果的描述信息必须确定(以传给getColumns()方法), 并且客户端必须接收一个包含列信息的结果集. 在案例2中, 只要准备一个简单的查询并且只用确定结果描述信息. 很显然, 案例2执行方式更好一些.

这个讨论有点复杂, 让我们考虑一个没有本地化支持prepared statement的DBMS服务器. 案例1的性能没有改变, 但案例2中, 因为'哑元'查询必须被执行而不是被预储使得它的性能增强了一些. 因为查询中的WHERE子句总是为FALSE, 查询在不用存取表的数据情况的下会生成没有数据的结果集. 在这种情况下，第二种方式当然比第一种方式好一些.

总而言之，总是使用ResultSet的metadata方法去获取列信息，像列名,列的数据类型,列的数据精度和长度等. 当要求的信息无法从ResultSet的metadata中获取时才去用getColumns()方法(像列的缺省值这些信息等).

获取数据

要有效的获取数据，就只需返回你需要的数据, 以及很多用效的方法. 本节的指导原则将帮助你使用JDB获取数据时优化系统性能.

获取长数据

如非必要, 应用程序不应请求长的数据, 因为长的数据通过网络传输会非常慢和消耗资源.

大多数用户并不想看到大堆的数据. 如果用户不想处理这些长数据, 那么程序应能够再次查询数据库, 在SELECT子句中指定需要的列名. 这种方式允许一般用户获取结果集而不用消耗昂贵的网络流量.

虽然最好的方式是不要将长数据包括在SELECT子句的列名中，但还是有一些应用程序在发送查询给JDBC驱动程序时并没有在SELECT子句中明确指出列名 (确切一点, 有些程序发送这样的查询： select \* from

...). 如果SELECT子句的列名中包含长数据, 许多JDBC驱动程序必须在查询时重新获取数据, 甚至在ResultSet中这些长数据并没有被程序用到. 在可能情况下，开发者应该试着去实现一种不需获取所有列数据的方法.

例如，看以下的JDBC代码:

ResultSet rs = stmt.executeQuery (

"select \* from Employees where SSID = '999-99-2222'");

rs.next();

string name = rs.getString (4);

要记住JDBC驱动程序没有知觉. 当查询被执行时它不知道哪些列是程序所要的. 驱动程序只知道应用程序能请求任意的列. 当JDBC驱动程序处理 rs.next() 请求时, 它可能会跨过网络从数据库服务器至少返回一行结果. 在这种情况下, 每个结果行会包含所有的列数据– 如果Employees表有一列包含员工相片的话它也会被包含在结果里面. 限制SELECT子句的列名列表并且只包含有用的列名，会减少网络流量及更快的查询性能.

另外，虽然getClob()和getBlob()方法可以允许应用程序去如何控制获取长数据, 但开发者必须认识到在许多情况下, JDBC驱动程序缺少真正的LOB定位器的支持. 像这种情况下，在暴露getClob和getBlob方法给开发者之前驱动程序必须经过网络获取所有的长数据.

减少获取的数据量

有时必须要获取长数据. 这时, 要注意的是大多数用户并不想在屏幕上看到100k甚至更多的文字.

要减少网络交通和改善性能, 通过调用setMaxRows(), SetMaxFieldSize及SetFetchSize()方法, 你可以减少取获取的数据量. 另一种方式是减少数据的列数. 如果驱动程序允许你定义packet的大小, 使用最小的packet尺寸会适合你的需要.

记住: 要小心的返回只有你需要的行和列数据. 当你只需要2列数据而你却返回的5列数据时，性能会降低 – 特别是不需要的行中包含有长数据时.

选择合适的数据类型

接收和发送某些数据可能代价昂贵. 当你设计一个schema时, 应选择能被最有效地处理的数据类型. 例如, 整型数就比浮点数或实数处理起来要快一些. 浮点数的定义是按照数据库的内部规定的格式, 通常是一种压缩格式. 数据必须被 解压和转换到另外种格式, 这样它才能被数据的协议处理.

获取ResultSet

由于数据库系统对可滚动光标的支持有限, 许多JDBC驱动程序并没有实现可滚动光标. 除非你确信数据库支持可滚动光标的结果集, 否则不要调用rs.last()和rs.getRow()方法去找出数据集的最大行数. 因为JDBC驱动程序模拟了可滚动光标, 调用rs.last()导致了驱动程序透过网络移到了数据集的最后一行. 取而代之, 你可以用ResultSet遍历一次计数或者用SELECT查询的COUNT函数来得到数据行数.

通常情况下，请不要写那种依赖于结果集行数的代码, 因为驱动程序必须获取所有的数据集以便知道查询会返回多少行数据.

选用JDBC对象和方法

本节的指导原则将帮助你在选用JDBC的对象和方法时哪些会有最好的性能.

在存储过程中使用参数标记作为参数

当调用存储过程时, 应总是使用参数标记(?)来代替字面上的参数. JDBC驱动能调用数据库的存储过程, 也能被其它的SQL查询执行, 或者直接通过远程进程调用(RPC)的方式执行. 当你将存储过程作为一个SQL查询执行时, 数据库要解析这个查询语句, 校验参数并将参数转换为正确的数据类型.

要记住, SQL语句总是以字符串的形式送到数据库, 例如, “{call getCustName (12345)}”. 在这里, 即使程序中将参数作为整数赋给了getSustName, 而实现上参数还是以字符串的形式传给了服务器. 数据库会解析这个SQL查询, 并且根据metadata来决定存储过程的参数类型, 然后分解出参数"12345", 然后在最终将存储过程作为一个SQL查询执行之前将字串'12345’转换为整型数.

按RPC方式调用时, 之前那种SQL字符串的方式要避免使用. 取而代之, JDBC驱动会构造一个网络packet, 其中包含了本地数据类型的参数，然后执行远程调用.

案例 1

在这个例子中, 存储过程不能最佳的使用RPC. 数据库必须将这作为一个普通的语言来进行解析，校验参数类型并将参数转换为正确的数据类型，最后才执行这个存储过程.

CallableStatement cstmt = conn.prepareCall (

"{call getCustName (12345)}");

ResultSet rs = cstmt.executeQuery ();

案例 2

在这个例子中, 存储过程能最佳的执行RPC. 因为程序避免了字面的的参数, 使用特殊的参数来调用存储过程, JDBC驱动能最好以RPC方式直接来执行存储过程. SQL语言上的处理在这里被避免并且执行也得到很大的改善.

CallableStatement cstmt - conn.prepareCall (

"{call getCustName (?)}");

cstmt.setLong (1,12345);

ResultSet rs = cstmt.executeQuery();

使用Statement而不是PreparedStatement对象

JDBC驱动的最佳化是基于使用的是什么功能. 选择PreparedStatement还是Statement取决于你要怎么使用它们. 对于只执行一次的SQL语句选择Statement是最好的. 相反, 如果SQL语句被多次执行选用PreparedStatement是最好的.

PreparedStatement的第一次执行消耗是很高的. 它的性能体现在后面的重复执行. 例如, 假设我使用Employee ID, 使用prepared的方式来执行一个针对Employee表的查询. JDBC驱动会发送一个网络请求到数据解析和优化这个查询. 而执行时会产生另一个网络请求. 在JDBC驱动中，减少网络通讯是最终的目的. 如果我的程序在运行期间只需要一次请求, 那么就使用Statement. 对于Statement, 同一个查询只会产生一次网络到数据库的通讯.

对于使用PreparedStatement池的情况下, 本指导原则有点复杂. 当使用PreparedStatement池时, 如果一个查询很特殊, 并且不太会再次执行到, 那么可以使用Statement. 如果一个查询很少会被执行,但连接池中的Statement池可能被再次执行, 那么请使用PreparedStatement. 在不是Statement池的同样情况下, 请使用Statement.

使用PreparedStatement的Batch功能

Update大量的数据时, 先Prepare一个INSERT语句再多次的执行, 会导致很多次的网络连接. 要减少JDBC的调用次数改善性能, 你可以使用PreparedStatement的AddBatch()方法一次性发送多个查询给数据库. 例如, 让我们来比较一下下面的例子.

例 1: 多次执行Prepared Statement

PreparedStatement ps = conn.prepareStatement(

"INSERT into employees values (?, ?, ?)");

for (n = 0; n < 100; n++) {

ps.setString(name[n]);

ps.setLong(id[n]);

ps.setInt(salary[n]);

ps.executeUpdate();

}

例 2: 使用Batch

PreparedStatement ps = conn.prepareStatement(

"INSERT into employees values (?, ?, ?)");

for (n = 0; n < 100; n++) {

ps.setString(name[n]);

ps.setLong(id[n]);

ps.setInt(salary[n]);

ps.addBatch();

}

ps.executeBatch();

在例 1中, PreparedStatement被用来多次执行INSERT语句. 在这里, 执行了100次INSERT操作, 共有101次网络往返. 其中,1次往返是预储statement, 另外100次往返执行每个迭代. 在例2中, 当在100次INSERT操作中使用addBatch()方法时, 只有两次网络往返. 1次往返是预储statement, 另一次是执行batch命令. 虽然Batch命令会用到更多的数据库的CPU周期, 但是通过减少网络往返，性能得到提高. 记住, JDBC的性能最大的增进是减少JDBC驱动与数据库之间的网络通讯.

选择合适的光标类型

选择适用的光标类型以最大限度的适用你的应用程序. 本节主要讨论三种光标类型的性能问题.

对于从一个表中顺序读取所有记录的情况来说, Forward-Only型的光标提供了最好的性能. 获取表中的数据时, 没有哪种方法比使用Forward-Only型的光标更快. 但不管怎样, 当程序中必须按无次序的方式处理数据行时, 这种光标就无法使用了.

对于程序中要求与数据库的数据同步以及要能够在结果集中前后移动光标, 使用JDBC的Scroll-Insensitive型光标是较理想的选择. 此类型的光标在第一次请求时就获取了所有的数据(当JDBC驱动采用'lazy'方式获取数据时或许是很多的而不是全部的数据)并且储存在客户端. 因此, 第一次请求会非常慢, 特别是请求长数据时会理严重. 而接下来的请求并不会造成任何网络往返(当使用'lazy'方法时或许只是有限的网络交通) 并且处理起来很快. 因为第一次请求速度很慢, Scroll-Insensitive型光标不应该被使用在单行数据的获取上. 当有要返回长数据时, 开发者也应避免使用Scroll-Insensitive型光标, 因为这样可能会造成内存耗尽. 有些Scroll-Insensitive型光标的实现方式是在数据库的临时表中缓存数据来避免性能问题, 但多数还是将数据缓存在应用程序中.

Scroll-Sensitive型光标, 有时也称为Keyset-Driven光标, 使用标识符, 像数据库的ROWID之类. 当每次在结果集移动光标时, 会重新该标识符的数据. 因为每次请求都会有网络往返, 性能可能会很慢. 无论怎样, 用无序方式的返回结果行对性能的改善是没有帮助的.

现在来解释一下这个, 来看这种情况. 一个程序要正常的返回1000行数据到程序中. 在执行时或者第一行被请求时, JDBC驱动不会执行程序提供的SELECT语句. 相反, 它会用键标识符来替换SELECT查询, 例如, ROWID. 然后修改过的查询都会被驱动程序执行，跟着会从数据库获取所有1000个键值. 每一次对一行结果的请求都会使JDBC驱动直接从本地缓存中找到相应的键值, 然后构造一个包含了'WHERE ROWID=？'子句的最佳化查询, 再接着执行这个修改过的查询, 最后从服务器取得该数据行.

当程序无法像Scroll-Insensitive型光标一样提供足够缓存时, Scroll-Sensitive型光标可以被替代用来作为动态的可滚动的光标.

使用有效的getter方法

JDBC提供多种方法从ResultSet中取得数据, 像getInt(), getString(), 和getObject()等等. 而getObject()方法是最泛化了的, 提供了最差的性能。 这是因为JDBC驱动必须对要取得的值的类型作额外的处理以映射为特定的对象. 所以就对特定的数据类型使用相应的方法.

要更进一步的改善性能, 应在取得数据时提供字段的索引号, 例如, getString(1), getLong(2), 和getInt(3)等来替代字段名. 如果没有指定字段索引号, 网络交通不会受影响, 但会使转换和查找的成本增加. 例如, 假设你使用getString("foo") ... JDBC驱动可能会将字段名转为大写(如果需要), 并且在到字段名列表中逐个比较来找到"foo"字段. 如果可以, 直接使用字段索引, 将为你节省大量的处理时间.

例如, 假设你有一个100行15列的ResultSet, 字段名不包含在其中. 你感兴趣的是三个字段 EMPLOYEENAME (字串型), EMPLOYEENUMBER (长整型), 和SALARY (整型). 如果你指定getString(“EmployeeName”), getLong(“EmployeeNumber”), 和getInt(“Salary”), 查询旱每个字段名必须被转换为metadata中相对应的大小写, 然后才进行查找. 如果你使用getString(1), getLong(2), 和getInt(15). 性能就会有显著改善.

获取自动生成的键值

有许多数据库提供了隐藏列为表中的每行记录分配一个唯一键值. 很典型, 在查询中使用这些字段类型是取得记录值的最快的方式, 因为这些隐含列通常反应了数据在磁盘上的物理位置. 在JDBC3.0之前, 应用程序只可在插入数据后通过立即执行一个SELECT语句来取得隐含列的值.

例如:

//插入行

int rowcount = stmt.executeUpdate (

"insert into LocalGeniusList (name) values ('Karen')");

// 现在为新插入的行取得磁盘位置 - rowid

ResultSet rs = stmt.executeQuery (

"select rowid from LocalGeniusList where name = 'Karen'");

这种取得隐含列的方式有两个主要缺点. 第一, 取得隐含列是在一个独立的查询中, 它要透过网络送到服务器后再执行. 第二, 因为不是主键, 查询条件可能不是表中的唯一性ID. 在后面一个例子中, 可能返回了多个隐含列的值, 程序无法知道哪个是最后插入的行的值.

(译者：由于不同的数据库支持的程度不同，返回rowid的方式各有差异。在SQL Server中，返回最后插入的记录的id可以用这样的查询语句：SELECT @IDENTITY )

JDBC3.0规范中的一个可选特性提供了一种能力, 可以取得刚刚插入到表中的记录的自动生成的键值.

例如:

int rowcount = stmt.executeUpdate (

"insert into LocalGeniusList (name) values ('Karen')",

// 插入行并返回键值

Statement.RETURN\_GENERATED\_KEYS);

ResultSet rs = stmt.getGeneratedKeys ();

// 得到生成的键值

现在, 程序中包含了一个唯一性ID, 可以用来作为查询条件来快速的存取数据行, 甚至于表中没有主键的情况也可以.

这种取得自动生成的键值的方式给JDBC的开发者提供了灵活性, 并且使存取数据的性能得到提升.

### PreparedStatement

对于预先准备语句, 我们应该使用参数化的查询. 这样允许数据库重用已经存在的访问 方案, 从而减轻数据库的负担. 这样的缓冲区是这个数据库范围的, 所以你可以安排你所有的 应用程序, 使用相似的参数化的 SQL, 就会提高这样的缓冲区方案的效率, 因为一个应用程序 可以使用另一个应用程序的语句. 一个应用服务器的优势也在于此, 因为访问数据库的逻辑应该集中在数据访问层上(OR 映射, 实体 bean 或者直接 JDBC)，最后, 预先准备语句的正确使用也让你利用应用程序服务器的预先准备语句的缓冲区的好处. 提高你的应用程序的性能, 因为应用程序通过对以前的预先准备语句的重用减少 JDBC 驱动程序调用的次数. 这样使它能和胖客户端的效率竞争, 并且去掉了不能保持一个长期 连接的坏处. 如果你使用参数化的预先准备语句, 就可以提高数据库和你的服务器端的代码的效率. 这些提高 都会允许你的应用程序提高性能。

在JDBC应用中,如果你已经是稍有水平开发者,你就应该始终以PreparedStatement代替Statement.也就是说,在任何时候都不要使用Statement

一.代码的可读性和可维护性.

虽然用PreparedStatement来代替Statement会使代码多出几行,但这样的代码无论从可读性还是可维护性上来说.都比直接用Statement的代码高很多档次:

stmt.executeUpdate("insert into tb\_name (col1,col2,col2,col4) values ('"+var1+"','"+var2+"',"+var3+",'"+var4+"')");

perstmt = con.prepareStatement("insert into tb\_name (col1,col2,col2,col4) values (?,?,?,?)");

perstmt.setString(1,var1);

perstmt.setString(2,var2);

perstmt.setString(3,var3);

perstmt.setString(4,var4);

perstmt.executeUpdate();

不用我多说,对于第一种方法.别说其他人去读你的代码,就是你自己过一段时间再去读,都会觉得伤心.

二.PreparedStatement尽最大可能提高性能.

每一种数据库都会尽最大努力对预编译语句提供最大的性能优化.因为预编译语句有可能被重复调用.所以语句在被DB的编译器编译后的执行代码被缓存下来,那么下次调用时只要是相同的预编译语句就不需要编译,只要将参数直接传入编译过的语句执行代码中(相当于一个涵数)就会得到执行.这并不是说只有一个Connection中多次执行的预编译语句被缓存,而是对于整个DB中,只要预编译的语句语法和缓存中匹配.那么在任何时候就可以不需要再次编译而可以直接执行.而statement的语句中,即使是相同一操作,而由于每次操作的数据不同所以使整个语句相匹配的机会极小,几乎不太可能匹配.比如:

insert into tb\_name (col1,col2) values ('11','22');

insert into tb\_name (col1,col2) values ('11','23');

即使是相同操作但因为数据内容不一样,所以整个个语句本身不能匹配,没有缓存语句的意义.事实是没有数据库会对普通语句编译后的执行代码缓存.这样每执行一次都要对传入的语句编译一次.

当然并不是所有预编译语句都一定会被缓存,数据库本身会用一种策略,比如使用频度等因素来决定什么时候不再缓存已有的预编译结果.以保存有更多的空间存储新的预编译语句.

三.最重要的一点是极大地提高了安全性.

即使到目前为止,仍有一些人连基本的恶义SQL语法都不知道.

String sql = "select \* from tb\_name where name= '"+varname+"' and passwd='"+varpasswd+"'";

如果我们把[' or '1' = '1]作为varpasswd传入进来.用户名随意,看看会成为什么?

select \* from tb\_name = '随意' and passwd = '' or '1' = '1';

因为'1'='1'肯定成立,所以可以任何通过验证.更有甚者:

把[';drop table tb\_name;]作为varpasswd传入进来,则:

select \* from tb\_name = '随意' and passwd = '';drop table tb\_name;有些数据库是不会让你成功的,但也有很多数据库就可以使这些语句得到执行.

而如果你使用预编译语句.你传入的任何内容就不会和原来的语句发生任何匹配的关系.(前提是数据库本身支持预编译,可能没有什么服务端数据库不支持编译了,只有少数的桌面数据库,就是直接文件访问的那些)只要全使用预编译语句,你就用不着对传入的数据做任何过虑.而如果使用普通的statement,有可能要对drop,;等做费尽心机的判断和过虑.

## MySQL

### 存储过程之事务管理

存储程序提供了一个绝佳的机制来定义、封装和管理事务。

#### 1，MySQL的事务支持

MySQL的事务支持不是绑定在MySQL服务器本身，而是与存储引擎相关：

Java代码

1.MyISAM：不支持事务，用于只读程序提高性能

2.InnoDB：支持ACID事务、行级锁、并发

3.Berkeley DB：支持事务

MySQL的InnoDB支持ANSI标准定义的4个隔离级别。

可以使用如下语句设置MySQL的session隔离级别：

SET TRANSACTION ISOLATION LEVEL {READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ | SERIALIZABLE}

MySQL默认的隔离级别是REPEATABLE READ，在设置隔离级别为READ UNCOMMITTED或SERIALIZABLE时要小心，READ UNCOMMITTED会导致数据完整性的严重问题，而SERIALIZABLE会导致性能问题并增加死锁的机率

事务管理语句：

1.START TRANSACTION：开始事务，autocommit设为0，如果已经有一个事务在运行，则会触发一个隐藏的COMMIT

2.COMMIT：提交事务，保存更改，释放锁

3.ROLLBACK：回滚本事务对数据库的所有更改，然后结束事务，释放锁

4.SAVEPOINT savepoint\_name：创建一个savepoint识别符来ROLLBACK TO SAVEPOINT

5.ROLLBACK TO SAVEPOINT savepoint\_name：回滚到从savepoint\_name开始对数据库的所有更改，这样就允许回滚事务中的一部分，保证更改的一个子集被提交

6.SET TRANSACTION：允许设置事务的隔离级别

7.LOCK TABLES：允许显式的锁住一个或多个table，会隐式的关闭当前打开的事务，建议在执行LOCK TABLES语句之前显式的commit或rollback。我们一般所以一般在事务代码里不会使用LOCK TABLES

#### 2，定义事务

MySQL默认的行为是在每条SQL语句执行后执行一个COMMIT语句，从而有效的将每条语句独立为一个事务。

在复杂的应用场景下这种方式就不能满足需求了。

为了打开事务，允许在COMMIT和ROLLBACK之前多条语句被执行，我们需要做以下两步：

1, 设置MySQL的autocommit属性为0，默认为1

2，使用START TRANSACTION语句显式的打开一个事务

如果已经打开一个事务，则SET autocommit=0不会起作用，因为START TRANSACTION会隐式的提交session中所有当前的更改，结束已有的事务，并打开一个新的事务。

#### **3，使用Savepoint**

使用savepoint回滚难免有些性能消耗，一般可以用IF改写   
savepoint的良好使用的场景之一是“嵌套事务”，你可能希望程序执行一个小的事务，但是不希望回滚外面更大的事务

#### 4，事务和锁

事务的ACID属性只能通过限制数据库的同步更改来实现，从而通过对修改数据加锁来实现。   
直到事务触发COMMIT或ROLLBACK语句时锁才释放。   
缺点是后面的事务必须等前面的事务完成才能开始执行，吞吐量随着等待锁释放的时间增长而递减。   
MySQL/InnoDB通过行级锁来最小化锁竞争。这样修改同一table里其他行的数据没有限制，而且读数据可以始终没有等待。   
可以在SELECT语句里使用FOR UPDATE或LOCK IN SHARE MODE语句来加上行级锁

SELECT select\_statement options [FOR UPDATE|LOCK IN SHARE MODE]

SELECT select\_statement options [FOR UPDATE|LOCK IN SHARE MODE]

FOR UPDATE会锁住该SELECT语句返回的行，其他SELECT和DML语句必须等待该SELECT语句所在的事务完成   
LOCK IN SHARE MODE同FOR UPDATE，但是允许其他session的SELECT语句执行并允许获取SHARE MODE锁   
  
死锁：   
死锁发生于两个事务相互等待彼此释放锁的情景   
当MySQL/InnoDB检查到死锁时，它会强制一个事务rollback并触发一条错误消息   
对InnoDB而言，所选择的rollback的事务是完成工作最少的事务（所修改的行最少）

死锁在任何数据库系统里都可能发生，但是对MySQL/InnoDB这种行级锁数据库而言可能性相对较少。   
可以通过使用一致的顺序来锁row或table以及让事务保持尽可能短来减少死锁的频率。   
如果死锁不容易debug，你可以向你的程序中添加一些逻辑来处理死锁并重试事务，但这部分代码多了以后很难维护   
所以，比较好的避免死锁的方式是在做任何修改之前按一定的顺序添加行级锁，这样就能避免死锁:

设置死锁ttl: innodb\_lock\_wait\_timeout，默认为50秒   
如果你在一个事务中混合使用InnoDB和非InnoDB表，则MySQL不能检测到死锁，此时会抛出“lock wait timeuot”1205错误   
  
乐观所和悲观锁策略：   
悲观锁：在读取数据时锁住那几行，其他对这几行的更新需要等到悲观锁结束时才能继续   
乐观所：读取数据时不锁，更新时检查是否数据已经被更新过，如果是则取消当前更新   
一般在悲观锁的等待时间过长而不能接受时我们才会选择乐观锁

#### 5，事务设计指南

1，保持事务短小

2，尽量避免事务中rollback

3，尽量避免savepoint

4，默认情况下，依赖于悲观锁

5，为吞吐量要求苛刻的事务考虑乐观锁

6，显示声明打开事务

7，锁的行越少越好，锁的时间越短越好

### 3个引擎MyISAM,HEAP,InnoDB,ISAM

ISAM

ISAM是一个定义明确且历经时间考验的数据表格管理方法，它在设计之时就考虑到数据库被查询的次数要远大于更新的次数。因此，ISAM执行读取操作的速度很快，而且不占用大量的内存和存储资源。ISAM的两个主要不足之处在于，它不支持事务处理，也不能够容错：如果你的硬盘崩溃了，那么数据文件就无法恢复了。如果你正在把ISAM用在关键任务应用程序里，那就必须经常备份你所有的实时数据，通过其复制特性，MySQL能够支持这样的备份应用程序。

MyISAM

MyISAM是MySQL的ISAM扩展格式和缺省的数据库引擎。除了提供ISAM里所没有的索引和字段管理的大量功能，MyISAM还使用一种表格锁定的机制，来优化多个并发的读写操作。其代价是你需要经常运行OPTIMIZE TABLE命令，来恢复被更新机制所浪费的空间。MyISAM还有一些有用的扩展，例如用来修复数据库文件的MyISAMChk工具和用来恢复浪费空间的 MyISAMPack工具。

MyISAM强调了快速读取操作，这可能就是为什么MySQL受到了Web开发如此青睐的主要原因：在Web开发中你所进行的大量数据操作都是读取操作。所以，大多数虚拟主机提供商和Internet平台提供商(Internet Presence Provider，IPP)只允许使用MyISAM格式。

HEAP

HEAP允许只驻留在内存里的临时表格。驻留在内存里让HEAP要比ISAM和MyISAM都快，但是它所管理的数据是不稳定的，而且如果在关机之前没有进行保存，那么所有的数据都会丢失。在数据行被删除的时候，HEAP也不会浪费大量的空间。HEAP表格在你需要使用SELECT表达式来选择和操控数据的时候非常有用。要记住，在用完表格之后就删除表格。让我再重复一遍：在你用完表格之后，不要忘记删除表格。

InnoDB和Berkley DB

InnoDB和Berkley DB(BDB)数据库引擎都是造就MySQL灵活性的技术的直接产品，这项技术就是MySQL++ API。在使用MySQL的时候，你所面对的每一个挑战几乎都源于ISAM和MyISAM数据库引擎不支持事务处理也不支持外来键。尽管要比ISAM和 MyISAM引擎慢很多，但是InnoDB和BDB包括了对事务处理和外来键的支持，这两点都是前两个引擎所没有的。

### 管理

#### mysql导入.sql文件

1.导出整个数据库

mysqldump -u 用户名 -p 数据库名 > 导出的文件名

mysqldump -u wcnc -p smgp\_apps\_wcnc > wcnc.sql

2.导出一个表

mysqldump -u 用户名 -p 数据库名 表名> 导出的文件名

mysqldump -u wcnc -p smgp\_apps\_wcnc users> wcnc\_users.sql

3.导出一个数据库结构

mysqldump -u wcnc -p -d --add-drop-table smgp\_apps\_wcnc >d:\wcnc\_db.sql

-d 没有数据 --add-drop-table 在每个create语句之前增加一个drop table

4.导入数据库

常用source 命令

进入mysql数据库控制台，

如mysql -u root -p

mysql>use 数据库

然后使用source命令，后面参数为脚本文件(如这里用到的.sql)

mysql>source d:\wcnc\_db.sql

## Oracle

### 索引

#### 第一讲、索引并非总是最佳选择

　　如果发现Oracle 在有索引的情况下，没有使用索引，这并不是Oracle 的优化器出错。在有些情况下，Oracle 确实会选择全表扫描（Full Table Scan）,而非索引扫描（Index Scan）。这些情况通常有：   
  
　　1. 表未做statistics, 或者 statistics 陈旧，导致 Oracle 判断失误。   
　　2. 根据该表拥有的记录数和数据块数，实际上全表扫描要比索引扫描更快。   
  
　　对第1种情况，最常见的例子，是以下这句sql 语句：   
select count(\*) from mytable;   
  
　　在未作statistics 之前，它使用全表扫描，需要读取6000多个数据块（一个数据块是8k）, 做了statistics 之后，使用的是 INDEX (FAST FULL SCAN) ，只需要读取450个数据块。但是，statistics 做得不好，也会导致Oracle 不使用索引。   
  
　　第2种情况就要复杂得多。一般概念上都认为索引比表快，比较难以理解什么情况下全表扫描要比索引扫描快。为了讲清楚这个问题，这里先介绍一下Oracle 在评估使用索引的代价（cost）时两个重要的数据：CF(Clustering factor) 和 FF(Filtering factor).   
　　CF: 所谓 CF, 通俗地讲，就是每读入一个索引块，要对应读入多少个数据块。   
　　FF: 所谓 FF, 就是该sql 语句所选择的结果集，占总的数据量的百分比。   
  
　　大约的计算公式是：FF \* (CF + 索引块个数) ，由此估计出，一个查询， 如果使用某个索引，会需要读入的数据块块数。需要读入的数据块越多，则 cost 越大，Oracle 也就越可能不选择使用 index. （全表扫描需要读入的数据块数等于该表的实际数据块数）   
  
　　其核心就是， CF 可能会比实际的数据块数量大。CF 受到索引中数据的排列方式影响，通常在索引刚建立时，索引中的记录与表中的记录有良好的对应关系，CF 都很小；在表经过大量的插入、修改后，这种对应关系越来越乱，CF 也越来越大。此时需要 DBA 重新建立或者组织该索引。   
  
　　如果某个sql 语句以前一直使用某索引，较长时间后不再使用，一种可能就是 CF 已经变得太大，需要重新整理该索引了。   
  
　　FF 则是Oracle 根据 statistics 所做的估计。比如, mytables 表有32万行，其主键myid的最小值是1，最大值是409654，考虑以下sql 语句：   
  
Select \* from mytables where myid>=1; 和   
  
Select \* from mytables where myid>=400000   
  
　　这两句看似差不多的 sql 语句，对Oracle 而言，却有巨大的差别。因为前者的 FF 是100%， 而后者的 FF 可能只有 1%。如果它的CF 大于实际的数据块数，则Oracle 可能会选择完全不同的优化方式。而实际上，在我们的数据库上的测试验证了我们的预测. 以下是在HP 上执行时它们的 explain plan:   
  
　　第一句：   
  
SQL> select \* from mytables where myid>=1;   
　　已选择325917行。   
  
Execution Plan   
----------------------------------------------------------   
0 SELECT STATEMENT Optimizer=CHOOSE (Cost=3132 Card=318474 Byt es=141402456)   
1 0 TABLE ACCESS (FULL) OF 'MYTABLES' (Cost=3132 Card=318474 Byt es=141402456)   
Statistics   
----------------------------------------------------------   
7 recursive calls   
89 db block gets   
41473 consistent gets   
19828 physical reads   
0 redo size   
131489563 bytes sent via SQL\*Net to client   
1760245 bytes received via SQL\*Net from client   
21729 SQL\*Net roundtrips to/from client   
1 sorts (memory)   
0 sorts (disk)   
325917 rows processed   
　　第二句：   
Execution Plan   
----------------------------------------------------------   
0 SELECT STATEMENT Optimizer=CHOOSE (Cost=346 Card=663 Bytes=2 94372)   
1 0 TABLE ACCESS (BY INDEX ROWID) OF 'MYTABLES' (Cost=346 Card=663   
Bytes=294372)   
2 1 INDEX (RANGE SCAN) OF 'PK\_MYTABLES' (UNIQUE) (Cost=5 Card=663)   
Statistics   
----------------------------------------------------------   
1278 recursive calls   
0 db block gets   
6647 consistent gets   
292 physical reads   
0 redo size   
3544898 bytes sent via SQL\*Net to client   
42640 bytes received via SQL\*Net from client   
524 SQL\*Net roundtrips to/from client   
1 sorts (memory)   
0 sorts (disk)   
7838 rows processed   
  
　　显而易见，第1句没有使用索引，第2句使用了主键索引pk\_mytables. FF的巨大影响由此可见一斑。由此想到，我们在写sql 语句时，如果预先估计一下 FF, 你就几乎可以预见到 Oracle 会否使用索引。

#### 第二讲、索引也有好坏

　　索引有 B tree 索引， Bitmap 索引， Reverse b tree 索引， 等。最常用的是 B tree 索引。 B 的全称是Balanced , 其意义是，从 tree 的 root 到任何一个leaf ，要经过同样多的 level. 索引可以只有一个字段（Single column）, 也可以有多个字段（Composite）,最多32个字段，8I 还支持 Function-based index. 许多developer 都倾向于使用单列B 树索引。   
  
　　所谓索引的好坏是指：   
  
　　1，索引不是越多越好。特别是大量从来或者几乎不用的索引，对系统只有损害。OLTP系统每表超过5个索引即会降低性能，而且在一个sql 中， Oracle 从不能使用超过 5个索引。   
　　2，很多时候，单列索引不如复合索引有效率。   
　　3，用于多表连结的字段，加上索引会很有作用。   
  
　　那么，在什么情况下单列索引不如复合索引有效率呢？有一种情况是显而易见的，那就是，当sql 语句所查询的列，全部都出现在复合索引中时，此时由于 Oracle 只需要查询索引块即可获得所有数据，当然比使用多个单列索引要快得多。（此时，这种优化方式被称为 Index only access path）   
  
　　除此之外呢？我们还是来看一个例子吧：   
　　在 HP（Oracle 8.1.7） 上执行以下语句：   
select count(1) from mytabs where coid>=130000 and issuedate >= to\_date ('2001-07-20', 'yyyy-mm-dd')。   
  
　　一开始，我们有两个单列索引：I\_mytabs1(coid), I\_mytabs2(issuedate), 下面是执行情况：   
COUNT(1)   
----------   
6427   
Execution Plan   
----------------------------------------------------------   
0 SELECT STATEMENT Optimizer=CHOOSE (Cost=384 Card=1 Bytes=11)   
1 0 SORT (AGGREGATE)   
2 1 TABLE ACCESS (BY INDEX ROWID) OF 'T\_MYTABS' (Cost=384 Card   
=126 Bytes=1386)   
3 2 INDEX (RANGE SCAN) OF 'I\_MYTABS2' (NON-UNIQUE) (Cost=11   
Card=126)   
Statistics   
----------------------------------------------------------   
  
172 recursive calls   
  
1 db block gets   
  
5054 consistent gets   
  
2206 physical reads   
  
0 redo size   
  
293 bytes sent via SQL\*Net to client   
  
359 bytes received via SQL\*Net from client   
  
2 SQL\*Net roundtrips to/from client   
  
5 sorts (memory)   
  
0 sorts (disk)   
  
1 rows processed   
  
　　可以看到，它读取了7000个数据块来获得所查询的 6000多行。   
  
　　现在，去掉这两个单列索引，增加一个复合索引I\_mytabs\_test ( coid, issuedate), 重新执行，结果如下：   
  
COUNT(1)   
  
----------   
  
6436   
Execution Plan   
  
----------------------------------------------------------   
  
0 SELECT STATEMENT Optimizer=CHOOSE (Cost=3 Card=1 Bytes=11)   
  
1 0 SORT (AGGREGATE)   
  
2 1 INDEX (RANGE SCAN) OF 'I\_MYTABS\_TEST' (NON-UNIQUE) (Cost=3 Card=126 Bytes=1386)   
  
Statistics   
  
----------------------------------------------------------   
  
806 recursive calls   
  
5 db block gets   
  
283 consistent gets   
  
76 physical reads   
  
0 redo size   
  
293 bytes sent via SQL\*Net to client   
  
359 bytes received via SQL\*Net from client   
  
2 SQL\*Net roundtrips to/from client   
  
3 sorts (memory)   
  
0 sorts (disk)   
  
1 rows processed   
  
　　可以看到，这次只读取了300个数据块。   
  
　　7000块对300块，这就是在这个例子中，单列索引与复合索引的代价之比。这个例子提示我们， 在许多情况下，单列索引不如复合索引有效率。   
  
　　可以说，在索引的设置问题上，其实有许多工作可以做。正确地设置索引，需要对应用进行总体的分析。

#### 第三讲、索引再好，不用也是白搭

　　抛开前面所说的，假设你设置了一个非常好的索引，任何傻瓜都知道应该使用它，但是Oracle 却偏偏不用，那么，需要做的第一件事情，是审视你的 sql 语句。   
  
　　Oracle 要使用一个索引，有一些最基本的条件：   
  
　　1， where 子句中的这个字段，必须是复合索引的第一个字段；   
  
　　2， where 子句中的这个字段，不应该参与任何形式的计算   
  
　　具体来讲，假设一个索引是按 f1, f2, f3的次序建立的，现在有一个 sql 语句, where 子句是 f2 = : var2, 则因为 f2 不是索引的第1个字段，无法使用该索引。   
  
　　第2个问题，则在我们之中非常严重。以下是从 实际系统上面抓到的几个例子：   
  
Select jobid from mytabs where isReq='0' and to\_date (updatedate) >= to\_Date ( '2001-7-18', 'YYYY-MM-DD')；   
  
………   
  
　　以上的例子能很容易地进行改进。请注意这样的语句每天都在我们的系统中运行，消耗我们有限的cpu 和 内存资源。   
  
　　除了1，2这两个我们必须牢记于心的原则外，还应尽量熟悉各种操作符对 Oracle 是否使用索引的影响。这里我只讲哪些操作或者操作符会显式（explicitly）地阻止 Oracle 使用索引。以下是一些基本规则：   
  
　　1， 如果 f1 和 f2 是同一个表的两个字段，则 f1>2, f1>=f2, f1   
  
　　2， f1 is null, f1 is not null, f1 not in, f1 !=, f1 like ‘%pattern%’;   
  
　　3， Not exist   
  
　　4， 某些情况下，f1 in 也会不用索引；   
  
　　对于这些操作，别无办法，只有尽量避免。比如，如果发现你的 sql 中的 in 操作没有使用索引，也许可以将 in 操作改成 比较操作 + union all。笔者在实践中发现很多时候这很有效。   
  
　　但是，Oracle 是否真正使用索引，使用索引是否真正有效，还是必须进行实地的测验。合理的做法是，对所写的复杂的 sql, 在将它写入应用程序之前，先在产品数据库上做一次explain . explain 会获得Oracle 对该 sql 的解析（plan）,可以明确地看到 Oracle 是如何优化该 sql 的。   
  
　　如果经常做 explain, 就会发现，喜爱写复杂的 sql 并不是个好习惯，因为过分复杂的sql 其解析计划往往不尽如人意。事实上，将复杂的 sql 拆开，有时候会极大地提高效率，因为能获得很好的优化。当然这已经是题外话了

#### ORACLE里按用户名重建索引的方法

[轉貼]在ORACLE里按用户名重建索引的方法   
  
如果你管理的ORACLE数据库下某些应用项目有大量的修改删除操作, 数据索引是需要周期性的重建的.  
它不仅可以提高查询性能, 还能增加索引表空间空闲空间大小.   
在ORACLE里大量删除记录后, 表和索引里占用的数据块空间并没有释放.   
重建索引可以释放已删除记录索引占用的数据块空间.   
转移数据, 重命名的方法可以重新组织表里的数据.  
下面是可以按ORACLE用户名生成重建索引的SQL脚本:  
---------------------------------------------------------------------------  
SET ECHO OFF;  
SET FEEDBACK OFF;  
SET VERIFY OFF;  
SET PAGESIZE 0;  
SET TERMOUT ON;  
SET HEADING OFF;  
ACCEPT username CHAR PROMPT 'Enter the index username: ';   
spool /oracle/rebuild\_&username.sql;  
  
SELECT   
'REM +-----------------------------------------------+' || chr(10) ||  
'REM | INDEX NAME : ' || owner || '.' || segment\_name   
|| lpad('|', 33 - (length(owner) + length(segment\_name)) )  
|| chr(10) ||  
'REM | BYTES : ' || bytes   
|| lpad ('|', 34-(length(bytes)) ) || chr(10) ||  
'REM | EXTENTS : ' || extents   
|| lpad ('|', 34-(length(extents)) ) || chr(10) ||  
'REM +-----------------------------------------------+' || chr(10) ||  
'ALTER INDEX ' || owner || '.' || segment\_name || chr(10) ||  
'REBUILD ' || chr(10) ||  
'TABLESPACE ' || tablespace\_name || chr(10) ||  
'STORAGE ( ' || chr(10) ||  
' INITIAL ' || initial\_extent || chr(10) ||  
' NEXT ' || next\_extent || chr(10) ||  
' MINEXTENTS ' || min\_extents || chr(10) ||  
' MAXEXTENTS ' || max\_extents || chr(10) ||  
' PCTINCREASE ' || pct\_increase || chr(10) ||  
');' || chr(10) || chr(10)  
FROM dba\_segments  
WHERE segment\_type = 'INDEX'  
AND owner='&username'  
ORDER BY owner, bytes DESC;  
  
spool off;  
  
-----------------------------------------------------------------------------  
  
如果你用的是WINDOWS系统, 想改变输出文件的存放目录, 修改spool后面的路径成:  
spool c:\oracle\rebuild\_&username.sql;  
  
如果你只想对大于max\_bytes的索引重建索引, 可以修改上面的SQL语句:   
在AND owner='&username' 后面加个限制条件 AND bytes> &max\_bytes   
  
如果你想修改索引的存储参数, 在重建索引rebuild\_&username.sql里改也可以.   
比如把pctincrease不等于零的值改成是零.  
  
生成的rebuild\_&username.sql文件我们需要来分析一下, 它们是否到了需要重建的程度：  
  
分析索引，看是否碎片严重  
  
SQL>ANALYZE INDEX &index\_name VALIDATE STRUCTURE;  
  
col name heading 'Index Name' format a30  
col del\_lf\_rows heading 'Deleted|Leaf Rows' format 99999999  
col lf\_rows\_used heading 'Used|Leaf Rows' format 99999999  
col ratio heading '% Deleted|Leaf Rows' format 999.99999  
  
SELECT name,  
del\_lf\_rows,  
lf\_rows - del\_lf\_rows lf\_rows\_used,  
to\_char(del\_lf\_rows / (lf\_rows)\*100,'999.99999') ratio  
FROM index\_stats where name = upper('&index\_name');  
  
当删除的比率大于15 - 20% 时，肯定是需要索引重建的.   
  
经过删改后的rebuild\_&username.sql文件我们可以放到ORACLE的定时作业里:   
比如一个月或者两个月在非繁忙时间运行.  
  
如果遇到ORA-00054错误, 表示索引在的表上有锁信息, 不能重建索引.   
那就忽略这个错误, 看下次是否成功.   
  
对那些特别忙的表要区别对待, 不能用这里介绍的方法,   
还要把它们的索引从rebuild\_&username.sql里删去.

表與索引專題總結  
[http://www.cnoug.org/viewthread. ... hlight=%CB%F7%D2%FD](http://www.cnoug.org/viewthread.php?tid=20711&highlight=%CB%F7%D2%FD)  
索引建立速度   
[http://www.cnoug.org/viewthread. ... hlight=%CB%F7%D2%FD](http://www.cnoug.org/viewthread.php?tid=26282&highlight=%CB%F7%D2%FD)  
使用特殊索引的注意事项   
[http://www.cnoug.org/viewthread. ... hlight=%CB%F7%D2%FD](http://www.cnoug.org/viewthread.php?tid=23690&highlight=%CB%F7%D2%FD)  
如何监控索引的使用  
[http://www.cnoug.org/viewthread. ... hlight=%CB%F7%D2%FD](http://www.cnoug.org/viewthread.php?tid=18673&highlight=%CB%F7%D2%FD)  
把一个表空间的所有索引换到另一个表空间  
[http://www.cnoug.org/viewthread. ... hlight=%CB%F7%D2%FD](http://www.cnoug.org/viewthread.php?tid=19697&highlight=%CB%F7%D2%FD)  
索引的三個問題   
[http://www.cnoug.org/viewthread. ... hlight=%CB%F7%D2%FD](http://www.cnoug.org/viewthread.php?tid=12565&highlight=%CB%F7%D2%FD)

### 分析函数Rank, Dense\_rank, row\_number

1.使用rownum为记录排名

2.使用分析函数来为记录排名

3.使用分析函数为记录进行分组排名

一、使用rownum为记录排名：

在前面一篇《Oracle开发专题之：分析函数》，我们认识了分析函数的基本应用，现在我们再来考虑下面几个问题：

①对所有客户按订单总额进行排名

②按区域和客户订单总额进行排名

③找出订单总额排名前13位的客户

④找出订单总额最高、最低的客户

⑤找出订单总额排名前25%的客户

按照前面第一篇文章的思路，我们只能做到对各个分组的数据进行统计，如果需要排名的话那么只需要简单地加上rownum不就行了吗？事实情况是否如此想象般简单，我们来实践一下。

【1】测试环境：

SQL> desc user\_order;

Name Null? Type

----------------------------------------- -------- ----------------------------

REGION\_ID NUMBER(2)

CUSTOMER\_ID NUMBER(2)

CUSTOMER\_SALES NUMBER

【2】测试数据：

SQL> select \* from user\_order order by customer\_sales;

REGION\_ID CUSTOMER\_ID CUSTOMER\_SALES

---------- ----------- --------------

5 1 151162

10 29 903383

6 7 971585

10 28 986964

9 21 1020541

9 22 1036146

8 16 1068467

6 8 1141638

5 3 1161286

5 5 1169926

8 19 1174421

7 12 1182275

7 11 1190421

6 10 1196748

6 9 1208959

10 30 1216858

5 2 1224992

9 24 1224992

9 23 1224992

8 18 1253840

7 15 1255591

7 13 1310434

10 27 1322747

8 20 1413722

6 6 1788836

10 26 1808949

5 4 1878275

7 14 1929774

8 17 1944281

9 25 2232703

30 rows selected.

注意这里有3条记录的订单总额是一样的。假如我们现在需要筛选排名前12位的客户，如果使用rownum会有什么样的后果呢？

SQL> select rownum, t.\* from (select \* from user\_order order by customer\_sales desc) t

where rownum <= 12 order by customer\_sales desc;

ROWNUM REGION\_ID CUSTOMER\_ID CUSTOMER\_SALES

---------- ---------- ----------- --------------

1 9 25 2232703

2 8 17 1944281

3 7 14 1929774

4 5 4 1878275

5 10 26 1808949

6 6 6 1788836

7 8 20 1413722

8 10 27 1322747

9 7 13 1310434

10 7 15 1255591

11 8 18 1253840

12 5 2 1224992

12 rows selected.

很明显假如只是简单地按rownum进行排序的话，我们漏掉了另外两条记录(参考上面的结果)。

二、使用分析函数来为记录排名：

针对上面的情况，Oracle从8i开始就提供了3个分析函数：rand，dense\_rank，row\_number来解决诸如此类的问题，下面我们来看看这3个分析函数的作用以及彼此之间的区别：

**Rank，Dense\_rank，Row\_number函数为每条记录产生一个从1开始至N的自然数，N的值可能小于等于记录的总数。这3个函数的唯一区别在于当碰到相同数据时的排名策略。**

①ROW\_NUMBER：

Row\_number函数返回一个唯一的值，当碰到相同数据时，排名按照记录集中记录的顺序依次递增。

②DENSE\_RANK：

Dense\_rank函数返回一个唯一的值，除非当碰到相同数据时，此时所有相同数据的排名都是一样的。

③RANK：

Rank函数返回一个唯一的值，除非遇到相同的数据时，此时所有相同数据的排名是一样的，同时会在最后一条相同记录和下一条不同记录的排名之间空出排名。

这样的介绍有点难懂，我们还是通过实例来说明吧，下面的例子演示了3个不同函数在遇到相同数据时不同排名策略：

SQL> select region\_id, customer\_id, sum(customer\_sales) total,

2 rank() over(order by sum(customer\_sales) desc) rank,

3 dense\_rank() over(order by sum(customer\_sales) desc) dense\_rank,

4 row\_number() over(order by sum(customer\_sales) desc) row\_number

5 from user\_order

6 group by region\_id, customer\_id;

REGION\_ID CUSTOMER\_ID TOTAL RANK DENSE\_RANK ROW\_NUMBER

---------- ----------- ---------- ---------- ---------- ----------

8 18 1253840 11 11 11

5 2 1224992 12 12 12

9 23 1224992 12 12 13

9 24 1224992 12 12 14

10 30 1216858 15 13 15

30 rows selected.

请注意上面的绿色高亮部分，这里生动的演示了3种不同的排名策略：

①对于第一条相同的记录，3种函数的排名都是一样的：12

②当出现第二条相同的记录时，Rank和Dense\_rank依然给出同样的排名12；而row\_number则顺延递增为13，依次类推至第三条相同的记录

③当排名进行到下一条不同的记录时，可以看到Rank函数在12和15之间空出了13,14的排名，因为这2个排名实际上已经被第二、三条相同的记录占了。而Dense\_rank则顺序递增。row\_number函数也是顺序递增

比较上面3种不同的策略，我们在选择的时候就要根据客户的需求来定夺了：

①假如客户就只需要指定数目的记录，那么采用row\_number是最简单的，但有漏掉的记录的危险

②假如客户需要所有达到排名水平的记录，那么采用rank或dense\_rank是不错的选择。至于选择哪一种则看客户的需要，选择dense\_rank或得到最大的记录

三、使用分析函数为记录进行分组排名：

上面的排名是按订单总额来进行排列的，现在跟进一步：假如是为各个地区的订单总额进行排名呢？这意味着又多了一次分组操作：对记录按地区分组然后进行排名。幸亏Oracle也提供了这样的支持，我们所要做的仅仅是在over函数中order by的前面增加一个分组子句：partition by region\_id。

SQL> select region\_id, customer\_id,

sum(customer\_sales) total,

2 rank() over(partition by region\_id

order by sum(customer\_sales) desc) rank,

3 dense\_rank() over(partition by region\_id

order by sum(customer\_sales) desc) dense\_rank,

4 row\_number() over(partition by region\_id

order by sum(customer\_sales) desc) row\_number

5 from user\_order

6 group by region\_id, customer\_id;

REGION\_ID CUSTOMER\_ID TOTAL RANK DENSE\_RANK ROW\_NUMBER

---------- ----------- ---------- ---------- ---------- ----------

5 4 1878275 1 1 1

5 2 1224992 2 2 2

5 5 1169926 3 3 3

6 6 1788836 1 1 1

6 9 1208959 2 2 2

6 10 1196748 3 3 3

30 rows selected.

现在我们看到的排名将是基于各个地区的，而非所有区域的了！Partition by 子句在排列函数中的作用是将一个结果集划分成几个部分，这样排列函数就能够应用于这各个子集。

### 分析函数Top/Bottom N、First/Last、NTile

1.带空值的排列

2.Top/Bottom N查询

3.First/Last排名查询

4.按层次查询

一、带空值的排列：

在前面《Oracle开发专题之：分析函数2(Rank、Dense\_rank、row\_number)》一文中，我们已经知道了如何为一批记录进行全排列、分组排列。假如被排列的数据中含有空值呢？

SQL> select region\_id, customer\_id,

2 sum(customer\_sales) cust\_sales,

3 sum(sum(customer\_sales)) over(partition by region\_id) ran\_total,

4 rank() over(partition by region\_id

5 order by sum(customer\_sales) desc) rank

6 from user\_order

7 group by region\_id, customer\_id;

REGION\_ID CUSTOMER\_ID CUST\_SALES RAN\_TOTAL RANK

---------- ----------- ---------- ---------- ----------

10 31 6238901 1

10 26 1808949 6238901 2

10 27 1322747 6238901 3

10 30 1216858 6238901 4

10 28 986964 6238901 5

10 29 903383 6238901 6

我们看到这里有一条记录的CUST\_TOTAL字段值为NULL，但居然排在第一名了！显然这不符合情理。所以我们重新调整完善一下我们的排名策略，看看下面的语句：

SQL> select region\_id, customer\_id,

2 sum(customer\_sales) cust\_total,

3 sum(sum(customer\_sales)) over(partition by region\_id) reg\_total,

4 rank() over(partition by region\_id

order by sum(customer\_sales) desc NULLS LAST) rank

5 from user\_order

6 group by region\_id, customer\_id;

REGION\_ID CUSTOMER\_ID CUST\_TOTAL REG\_TOTAL RANK

---------- ----------- ---------- ---------- ----------

10 26 1808949 6238901 1

10 27 1322747 6238901 2

10 30 1216858 6238901 3

10 28 986964 6238901 4

10 29 903383 6238901 5

10 31 6238901 6

绿色高亮处，NULLS LAST/FIRST告诉Oracle让空值排名最后后第一。

注意是NULLS，不是NULL。

二、Top/Bottom N查询：

在日常的工作生产中，我们经常碰到这样的查询：找出排名前5位的订单客户、找出排名前10位的销售人员等等。现在这个对我们来说已经是很简单的问题了。下面我们用一个实际的例子来演示：

【1】找出所有订单总额排名前3的大客户：

SQL> select \*

SQL> from (select region\_id,

SQL> customer\_id,

SQL> sum(customer\_sales) cust\_total,

SQL> rank() over(order by sum(customer\_sales) desc NULLS LAST) rank

SQL> from user\_order

SQL> group by region\_id, customer\_id)

SQL> where rank <= 3;

REGION\_ID CUSTOMER\_ID CUST\_TOTAL RANK

---------- ----------- ---------- ----------

9 25 2232703 1

8 17 1944281 2

7 14 1929774 3

SQL>

【2】找出每个区域订单总额排名前3的大客户：

SQL> select \*

2 from (select region\_id,

3 customer\_id,

4 sum(customer\_sales) cust\_total,

5 sum(sum(customer\_sales)) over(partition by region\_id) reg\_total,

6 rank() over(partition by region\_id

order by sum(customer\_sales) desc NULLS LAST) rank

7 from user\_order

8 group by region\_id, customer\_id)

9 where rank <= 3;

REGION\_ID CUSTOMER\_ID CUST\_TOTAL REG\_TOTAL RANK

---------- ----------- ---------- ---------- ----------

5 4 1878275 5585641 1

5 2 1224992 5585641 2

5 5 1169926 5585641 3

6 6 1788836 6307766 1

6 9 1208959 6307766 2

6 10 1196748 6307766 3

7 14 1929774 6868495 1

7 13 1310434 6868495 2

7 15 1255591 6868495 3

8 17 1944281 6854731 1

8 20 1413722 6854731 2

8 18 1253840 6854731 3

9 25 2232703 6739374 1

9 23 1224992 6739374 2

9 24 1224992 6739374 2

10 26 1808949 6238901 1

10 27 1322747 6238901 2

10 30 1216858 6238901 3

18 rows selected.

三、First/Last排名查询：

想象一下下面的情形：找出订单总额最多、最少的客户。按照前面我们学到的知识，这个至少需要2个查询。第一个查询按照订单总额降序排列以期拿到第一名，第二个查询按照订单总额升序排列以期拿到最后一名。是不是很烦？因为Rank函数只告诉我们排名的结果，却无法自动替我们从中筛选结果。

幸好Oracle为我们在排列函数之外提供了两个额外的函数：first、last函数，专门用来解决这种问题。还是用实例说话： SQL> select min(customer\_id)

2 keep (dense\_rank first order by sum(customer\_sales) desc) first,

3 min(customer\_id)

4 keep (dense\_rank last order by sum(customer\_sales) desc) last

5 from user\_order

6 group by customer\_id;

FIRST LAST

---------- ----------

31 1

这里有几个看起来比较疑惑的地方：

①为什么这里要用min函数

②Keep这个东西是干什么的

③fist/last是干什么的

④dense\_rank和dense\_rank()有什么不同，能换成rank吗？

首先解答一下第一个问题：min函数的作用是用于当存在多个First/Last情况下保证返回唯一的记录。假如我们去掉会有什么样的后果呢？ SQL> select keep (dense\_rank first order by sum(customer\_sales) desc) first,

2 keep (dense\_rank last order by sum(customer\_sales) desc) last

3 from user\_order

4 group by customer\_id;

select keep (dense\_rank first order by sum(customer\_sales) desc) first,

\*

ERROR at line 1:

ORA-00907: missing right parenthesis

接下来看看第2个问题：keep是干什么用的？从上面的结果我们已经知道Oracle对排名的结果只“保留”2条数据，这就是keep的作用。告诉Oracle只保留符合keep条件的记录。

那么什么才是符合条件的记录呢？这就是第3个问题了。dense\_rank是告诉Oracle排列的策略，first/last则告诉最终筛选的条件。

第4个问题：如果我们把dense\_rank换成rank呢？ SQL> select min(region\_id)

2 keep(rank first order by sum(customer\_sales) desc) first,

3 min(region\_id)

4 keep(rank last order by sum(customer\_sales) desc) last

5 from user\_order

6 group by region\_id;

select min(region\_id)

\*

ERROR at line 1:

ORA-02000: missing DENSE\_RANK

四、按层次查询：

现在我们已经见识了如何通过Oracle的分析函数来获取Top/Bottom N，第一个，最后一个记录。有时我们会收到类似下面这样的需求：找出订单总额排名前1/5的客户。

很熟悉是不？我们马上会想到第二点中提到的方法，可是rank函数只为我们做好了排名，并不知道每个排名在总排名中的相对位置，这时候就引入了另外一个分析函数NTile，下面我们就以上面的需求为例来讲解一下：

SQL> select region\_id,

2 customer\_id,

3 ntile(5) over(order by sum(customer\_sales) desc) til

4 from user\_order

5 group by region\_id, customer\_id;

REGION\_ID CUSTOMER\_ID TILE

---------- ----------- ----------

10 31 1

9 25 1

10 26 1

6 6 1

8 18 2

5 2 2

9 23 3

6 9 3

7 11 3

5 3 4

6 8 4

8 16 4

6 7 5

10 29 5

5 1 5

Ntil函数为各个记录在记录集中的排名计算比例，我们看到所有的记录被分成5个等级，那么假如我们只需要前1/5的记录则只需要截取TILE的值为1的记录就可以了。假如我们需要排名前25%的记录(也就是1/4)那么我们只需要设置ntile(4)就可以了。

参考资料：《Mastering Oracle SQL》(By Alan Beaulieu, Sanjay Mishra O'Reilly June 2004 0-596-00632-2)

### lag（arg1,arg2,arg3):

arg1是从其他行返回的表达式

arg2是希望检索的当前行分区的偏移量。是一个正的偏移量，时一个往回检索以前的行的数目。

arg3是在arg2表示的数目超出了分组的范围时返回的值。

select deptno,ename,sal,lag(ename,1,null) over(partition by deptno order by ename) as lag\_ from scott.emp order by deptno;

执行结果：

DEPTNO ENAME SAL LAG\_

---------- -------------------- ---------- --------------------

10 CLARK 2450

10 KING 5000 CLARK

10 MILLER 1300 KING

20 ADAMS 1100

20 FORD 3000 ADAMS

20 JONES 2975 FORD

20 SCOTT 3000 JONES

20 SMITH 800 SCOTT

30 ALLEN 1600

30 BLAKE 2850 ALLEN

30 JAMES 950 BLAKE

30 MARTIN 1250 JAMES

30 TURNER 1500 MARTIN

30 WARD 1250 TURNER

已选择14行。

### 比较DATE、TIMESTAMP

A Comparison of Oracle's DATE and TIMESTAMP Datatypes

[James F. Koopmann](http://it.toolbox.com/people/jkoopmann/) posted 11/26/2005 | [Comments](http://it.toolbox.com/blogs/database-solutions/a-comparison-of-oracles-date-and-timestamp-datatypes-6681#commentsform) (15)

If you want to store date and time information in Oracle, you really only have two different options for the column's datatype. Lets take a quick look at these two datatypes and what they offer.

DATE datatype

This is the datatype that we are all too familiar with when we think about representing date and time values. It has the ability to store the month, day, year, century, hours, minutes, and seconds. It is typically good for representing data for when something has happened or should happen in the future. The problem with the DATE datatype is its' granularity when trying to determine a time interval between two events when the events happen within a second of each other. This issue is solved later in this article when we discuss the TIMESTAMP datatype. In order to represent the date stored in a more readable format, the TO\_CHAR function has traditionally been wrapped around the date as in [Listing A](file:///E:\Projects\collection\资料\DB\Oracle\misc\A%20Comparison%20of%20Oracle's%20DATE%20and%20TIMESTAMP%20Datatypes.htm#A).

LISTING A:

Formatting a date

SQL> SELECT TO\_CHAR(date1,'MM/DD/YYYY HH24:MI:SS') "Date" FROM date\_table;

Date

---------------------------

06/20/2003 16:55:14

06/26/2003 11:16:36

About the only trouble I have seen people get into when using the DATE datatype is doing arithmetic on the column in order to figure out the number of years, weeks, days, hours, and seconds between two dates. What needs to be realized when doing the calculation is that when you do subtraction between dates, you get a number that represents the number of days. You should then multiply that number by the number of seconds in a day (86400) before you continue with calculations to determine the interval with which you are concerned. Check out [Listing B](file:///E:\Projects\collection\资料\DB\Oracle\misc\A%20Comparison%20of%20Oracle's%20DATE%20and%20TIMESTAMP%20Datatypes.htm#B) for my solution on how to extract the individual time intervals for a subtraction of two dates. I am aware that the fractions could be reduced but I wanted to show all the numbers to emphasize the calculation.

LISTING B:

*Determine the interval breakdown between two dates for a DATE datatype*

1 SELECT TO\_CHAR(date1,'MMDDYYYY:HH24:MI:SS') date1,

2 TO\_CHAR(date2,'MMDDYYYY:HH24:MI:SS') date2,

3 trunc(86400\*(date2-date1))-

4 60\*(trunc((86400\*(date2-date1))/60)) seconds,

5 trunc((86400\*(date2-date1))/60)-

6 60\*(trunc(((86400\*(date2-date1))/60)/60)) minutes,

7 trunc(((86400\*(date2-date1))/60)/60)-

8 24\*(trunc((((86400\*(date2-date1))/60)/60)/24)) hours,

9 trunc((((86400\*(date2-date1))/60)/60)/24) days,

10 trunc(((((86400\*(date2-date1))/60)/60)/24)/7) weeks

11\* FROM date\_table

DATE1 DATE2 SECONDS MINUTES HOURS DAYS WEEKS

----------------- ----------------- ---------- ---------- ---------- ---------- ----------

06202003:16:55:14 07082003:11:22:57 43 27 18 17 2

06262003:11:16:36 07082003:11:22:57 21 6 0 12 1

TIMESTAMP datatype

One of the main problems with the DATE datatype was its' inability to be granular enough to determine which event might have happened first in relation to another event. Oracle has expanded on the DATE datatype and has given us the TIMESTAMP datatype which stores all the information that the DATE datatype stores, but also includes fractional seconds. If you want to convert a DATE datatype to a TIMESTAMP datatype format, just use the CAST function as I do [in Listing C](file:///E:\Projects\collection\资料\DB\Oracle\misc\A%20Comparison%20of%20Oracle's%20DATE%20and%20TIMESTAMP%20Datatypes.htm#C). As you can see, there is a fractional seconds part of '.000000' on the end of this conversion. This is only because when converting from the DATE datatype that does not have the fractional seconds it defaults to zeros and the display is defaulted to the default timestamp format (NLS\_TIMESTAMP\_FORMAT). If you are moving a DATE datatype column from one table to a TIMESTAMP datatype column of another table, all you need to do is a straight INSERTSELECT FROM and Oracle will do the conversion for you. Look at [Listing D](file:///E:\Projects\collection\资料\DB\Oracle\misc\A%20Comparison%20of%20Oracle's%20DATE%20and%20TIMESTAMP%20Datatypes.htm#D)for a formatting of the new TIMESTAMP datatype where everything is the same as formatting the DATE datatype as we did in [Listing A](file:///E:\Projects\collection\资料\DB\Oracle\misc\A%20Comparison%20of%20Oracle's%20DATE%20and%20TIMESTAMP%20Datatypes.htm#A). Beware while the TO\_CHAR function works with both datatypes, the TRUNC function will not work with a datatype of TIMESTAMP. This is a clear indication that the use of TIMESTAMP datatype should explicitly be used for date and times where a difference in time is of utmost importance, such that Oracle won't even let you compare like values. If you wanted to show the fractional seconds within a TIMESTAMP datatype, look at [Listing E](file:///E:\Projects\collection\资料\DB\Oracle\misc\A%20Comparison%20of%20Oracle's%20DATE%20and%20TIMESTAMP%20Datatypes.htm#E). In Listing E, we are only showing 3 place holders for the fractional seconds.

LISTING C:

Convert DATE datatype to TIMESTAMP datatype

SQL> SELECT CAST(date1 AS TIMESTAMP) "Date" FROM t;

Date

-----------------------------------------------------

20-JUN-03 04.55.14.000000 PM

26-JUN-03 11.16.36.000000 AM

LISTING D:

*Formatting of the TIMESTAMP datatype*

1 SELECT TO\_CHAR(time1,'MM/DD/YYYY HH24:MI:SS') "Date" FROM date\_table

Date

-------------------

06/20/2003 16:55:14

06/26/2003 11:16:36

LISTING E:

Formatting of the TIMESTAMP datatype with fractional seconds

1 SELECT TO\_CHAR(time1,'MM/DD/YYYY HH24:MI:SS:FF3') "Date" FROM date\_table

Date

-----------------------

06/20/2003 16:55:14:000

06/26/2003 11:16:36:000

Calculating the time difference between two TIMESTAMP datatypes is much easier than the old DATE datatype. Look at what happens when you just do straight subtraction of the columns in [Listing F](file:///E:\Projects\collection\资料\DB\Oracle\misc\A%20Comparison%20of%20Oracle's%20DATE%20and%20TIMESTAMP%20Datatypes.htm#F). As you can see, the results are much easier to recognize, 17days, 18hours, 27minutes, and 43seconds for the first row of output. This means no more worries about how many seconds in a day and all those cumbersome calculations. And therefore the calculations for getting the weeks, days, hours, minutes, and seconds becomes a matter of picking out the number by using the SUBSTR function as can be seen in [Listing G](file:///E:\Projects\collection\资料\DB\Oracle\misc\A%20Comparison%20of%20Oracle's%20DATE%20and%20TIMESTAMP%20Datatypes.htm#G).

LISTING F:

*Straight subtraction of two TIMESTAMP datatypes*

1 SELECT time1, time2, (time2-time1)

2\* FROM date\_table

TIME1 TIME2 (TIME2-TIME1)

------------------------------ ---------------------------- ----------------------

06/20/2003:16:55:14:000000 07/08/2003:11:22:57:000000 +000000017 18:27:43.000000

06/26/2003:11:16:36:000000 07/08/2003:11:22:57:000000 +000000012 00:06:21.000000

LISTING G:

Determine the interval breakdown between two dates for a TIMESTAMP datatype

1 SELECT time1,

2 time2,

3 substr((time2-time1),instr((time2-time1),' ')+7,2) seconds,

4 substr((time2-time1),instr((time2-time1),' ')+4,2) minutes,

5 substr((time2-time1),instr((time2-time1),' ')+1,2) hours,

6 trunc(to\_number(substr((time2-time1),1,instr(time2-time1,' ')))) days,

7 trunc(to\_number(substr((time2-time1),1,instr(time2-time1,' ')))/7) weeks

8\* FROM date\_table

TIME1 TIME2 SECONDS MINUTES HOURS DAYS WEEKS

------------------------- -------------------------- ------- ------- ----- ---- -----

06/20/2003:16:55:14:000000 07/08/2003:11:22:57:000000 43 27 18 17 2

06/26/2003:11:16:36:000000 07/08/2003:11:22:57:000000 21 06 00 12 1

System Date and Time

In order to get the system date and time returned in a DATE datatype, you can use the SYSDATE function such as :

SQL> SELECT SYSDATE FROM DUAL;

In order to get the system date and time returned in a TIMESTAMP datatype, you can use the SYSTIMESTAMP function such as:

SQL> SELECT SYSTIMESTAMP FROM DUAL;

You can set the initialization parameter FIXED\_DATE to return a constant value for what is returned from the SYSDATE function. This is a great tool for testing date and time sensitive code. Just beware that this parameter has no effect on the SYSTIMESTAMP function. This can be seen in [Listing H](file:///E:\Projects\collection\资料\DB\Oracle\misc\A%20Comparison%20of%20Oracle's%20DATE%20and%20TIMESTAMP%20Datatypes.htm#H).

LISTING H:

*Setting FIXED\_DATE and effects on SYSDATE and SYSTIMESTAMP*

SQL> ALTER SYSTEM SET fixed\_date = '2003-01-01-10:00:00';

System altered.

SQL> select sysdate from dual;

SYSDATE

---------

01-JAN-03

SQL> select systimestamp from dual;

SYSTIMESTAMP

---------------------------------------------------------

09-JUL-03 11.05.02.519000 AM -06:00

When working with date and time, the options are clear. You have at your disposal the DATE and TIMESTAMP datatypes. Just be aware, while there are similarities, there are also differences that could create havoc if you try to convert to the more powerful TIMESTAMP datatype. Each of the two has strengths in simplicity and granularity. Choose wisely.

### Converting LONG to VARCHAR2

#### 方法1

You cannot do string functions on LONG columns with regular SQL. You will

have to use PL/SQL. First, if you expect values to be more than 2000

characters, then you have two choices:

1) Switch to Oracle8, where the varchar2 fields can have 4000

characters. Oracle7 has a 2000 limit.

2) Stay in Oracle7 and break the LONG field into two varchar2(2000)

fields.

The only way to convert LONG columns is in PL/SQL. Look at the following

example, which determines the length of the LONG field:

SET SERVEROUTPUT ON SIZE 10000;

DECLARE

long\_var LONG;

BEGIN

SELECT text\_column INTO long\_var

FROM table\_with\_long

WHERE rownum < 2;

DBMS\_OUTPUT.PUT\_LINE('The length is '||LENGTH(long\_var));

END;

/

Basically, you define a variable as the LONG type, then SELECT the column

INTO the variable. Finally, it is output to the user. SET SERVEROUTPUT ON

SIZE 10000 allows spooling from the PUT\_LINE to go to the screen.

You can use a similar method to select the LONG into a varchar field. The

following example puts the first 2000 characters into TABLE\_B, which for

our purposes has one column, TEXT\_FIELD:

DECLARE

long\_var LONG;

var\_var VARCHAR2(2000);

BEGIN

SELECT text\_column INTO long\_var

FROM table\_with\_long

WHERE rownum < 2;

var\_var := substr(long\_var,1,2000);

INSERT INTO table\_b

VALUES (var\_var);

END;

/

#### 方法2

Can anyone tell me how to read a LONG RAW column into a varchar2 variable ? I tried using UTL\_RAW.CAST\_TO\_VARCHAR2 packaged procedure but getting the following error : ORA-00997: illegal use of LONG datatype

1. long raw is much much longer Oracle datatype than Oracle varchar2 datatype. You cannot put a larger value in a smaller variable; i.e. if you declare a varchar2 variable and try to populate it wih long raw data it won't work.   
   Also, none of the varchar2 or string functions (per example, length, instring, trunc, decode, ect) will not work for long raw datatype.
2. Actually, It does not matter which way I get the information from LONG RAW. But I need to write the value from this column into a text file.
3. Set long 10000000;  
   spool filename;  
   select column\_name  
   from table;  
     
   This will put the contents of the column into the filename you specify.
4. That will work for LONG datatype column. But LONG RAW stores the value in HEX format. So if i spool the output it shows only a single character 'A' in the file. This won't be a problem if we had BLOB instead of LONG RAW. Anyhow I am planning to change the datatype.
5. Try the following code using ULT\_RAW if you haven't done so.  
     
   -------------------------------------------------------------  
     
   /\*Problem description\*/   
   To convert long raw to varchar   
     
   /\*proposed solution\*/   
     
     
   1. Create utl\_raw package specification in $ORACLE\_HOME\rdbms\admin\utlraw.sql.   
   2. create utl\_raw package body in $ORACLE\_HOME\rdbms\admin\prvtrawb.plb   
   3. When you want to convert raw to character , fetch into a cursor and then convert.   
     
   declare   
   a varchar(255);   
   b long raw;   
   cursor c1 is select long\_raw from raw\_table;   
   begin   
   open c1;   
   loop   
   fetch c1 into b ;   
   exit when c1%notfound;   
   a:=UTL\_RAW.CAST\_TO\_VARCHAR2(b);   
   dbms\_output.put\_line(a);   
   end loop;   
   end;
6. Thanks for the code snippet. It works... But one small issue... My long raw column contains data length > 32760 chars. In PL/SQL varchar2 variable I am able to declare MAXimum of 32760 chars. Can you give me a hint on how can we read the complete column ( I think we have data upto 1 GB ) for each row.   
     
   Is there a way to read LONG RAW column as chunks of 32760 characters ? If i can do that, then I can use UTL\_FILE ( again imposes a limit of 1024 chars for each write) But if I can read from LONG RAW as chunks of 32760 characters I can probably use UTL\_FILE procedures to dump it to a OS file after a sequence of SUBSTR().
7. Use substr to loop through in smaller chunks if the column is more than 32767. See the code below. It does not have looping logic. Needs to be added. Also, you need to concatenate and write to a file.  
     
   set serveroutput on size 256000000  
   declare   
   a varchar(32767);   
   cursor c1 is select idt\_rec from ids\_upd\_sync\_nsa;   
   begin   
   for r2 in c1 loop  
   a:=UTL\_RAW.CAST\_TO\_VARCHAR2(substr(r2.idt\_rec,2000));   
   dbms\_output.put\_line(a);   
   dbms\_output.put\_line(r2.idt\_rec);  
   dbms\_output.put\_line(length(a));  
   end loop;   
   end;   
   /

#### 方法3

Plesae need support ,how can use instr and substr function for column with long datatype  
  
select substr(rec,1,(instr(rec,'?',1))) from F\_DE\_O\_HISTORY\_QUEUE\_PRE\_TST2

rec column is long,When execute this select message displayed as ORA-00932: inconsistent datatypes: expected NUMBER got LONG

回复1

If your long columns contain less than 32k bytes you can write a simple pl/sql that does the job:

SQL> select \* from F\_DE\_O\_HISTORY\_QUEUE\_PRE\_TST2;

REC

-------------------------------------------------------------------------------

euwitrwr?sdjklgflsdlg

SQL> create or replace function mysubstr(a varchar2) **return** varchar2 is

2 x varchar2(32767);

3 begin

4 select rec

5 into x

6 from F\_DE\_O\_HISTORY\_QUEUE\_PRE\_TST2

7 where rowid=a;

8

9 **return** substr(x,1,(instr(x,'?',1)));

10 end;

11 /

Function created.

SQL> select mysubstr(rowid) from F\_DE\_O\_HISTORY\_QUEUE\_PRE\_TST2;

MYSUBSTR(ROWID)

-------------------------------------------------------------------------------

euwitrwr?

Otherwise you must use TO\_LOB to insert your long content in a LOB DB object and DBMS\_LOB instr and substr...  
  
Max  
[My Italian Oracle blog|http://oracleitalia.wordpress.com/2010/01/17/supporto-di-xml-schema-in-oracle-xmldb/]

回复2

You can't use any function against a LONG. That's one of the reasons why its better to use CLOBS.

Since you are stuck with a legacy LONG there are some things you can do.

P. Forstmann's idea to use a CLOB in a GTT is good. A variant on this might be to see if you can translate the long to a temporary CLOB inline using TO\_LOB if your version of the database supports that function as Massimo suggested.

Another, more painful solution would be to use DBMS\_SQL to read the LONG in chunks and process it that way. There should be notes on how to do this in My Oracle Support (if you have an account), possibly here on OTN and probably generally on the internet.

### rownum和rowid的区别

在查询中，我们可以注意到，类似于“select xx from table where rownum < n”(n>1)这样的查询是有正确含义的，而“select xx from table where rownum = n”这样的查询只在n=1的时候成立，“select xx from table where rownum > n”(n>1)这样的查询只能得到一个空集。另外“select xx from table where rownum > 0”这个查询会返回所有的记录。这是为什么呢?原因就在于Oracle对rownum的处理上，rownum是在得到结果集的时候产生的，用于标记结果集中结果顺序的一个字段，这个字段被称为“伪数列”，也就是事实上不存在的一个数列。它的特点是按顺序标记，而且是逐次递加的，换句话说就是只有有rownum=1的记录，才可能有rownum=2的记录。

让我们回头来分析一下在where中使用rownum作为查询条件的情况。在rownum取=1，或者rownum <= n (n>1)的时候，没有问题。那么为什么当条件为rownum = n或者rownum >= n时明明有数据却只能得到一个空集呢?假设我们的查询条件为rownum = 2，那么在查询出的第一条记录的时候，oracle标记此条记录rownum为1，结果发现和rownum=2的条件不符，于是结果集为空。写到这里，我忽然有一个有趣的想法：假如有一条查询语句为select xx,yy from table where zz > 20 and rownum < 10，那么在执行的时候，是先按照zz>20的条件查询出一个结果集，然后按照rownum取出前10条返回?还是在按照zz>20的条件先查询，然后有一个记录就标记一个rownum，到rownum<10的时候就停止查询?我觉得应该是后者，也就是在执行语句的时候，不是做full scan，而是取够数据就停止查询。要验证这个想法应该很简单，找一个数据量非常大的表进行查询就可以了。可惜目前我没有这样的表。

我们可以看出，直接使用rownum是要受到限制的。但是很容易遇到这样的需求“查出符合条件的第xx条到第xx条记录”，比如页面的分页处理。这个时候如何构造出适合自己的结果集?嗯，墙边那位说全取出来手工挑选的哥们可以拉出去了。当然这样做也是可以的，但是前提是整个数据集的数据条数不多的情况下。假如遇到上十万百条的数据，全部取出来的话，用户就不用干别的事情了。这个时候用户应该怎么做呢?当然就是要用到我们介绍的rownum拉!rownum不是个“伪数列”么，好说，我们现在把它弄成一个实在的字段就可以了。

具体做法就是利用子查询，在构建临时表的时候，把rownum也一起构造进去。比如“select xx,yy from (select xx,yy,rownum as xyz from table where zz >20) where xyz between 10 and 20”这样就可以了。另外使用oracle提供的结果集处理函数minus也可以做到，例如“select xx,yy from table where zz > 20 and rownum <20 minus select xx,yy from table where zz>20 and rownum <10”，但是使用minus好像比使用子查询更加消耗资源。

和rownum相似，oracle还提供了另外一个伪数列：rowid。不过rowid和rownum不同，一般说来每一行数据对应的rowid是固定而且唯一的，在这一行数据存入数据库的时候就确定了。可以利用rowid来查询记录，而且通过rowid查询记录是查询速度最快的查询方法。(这个我没有试过，另外要记住一个长度在18位，而且没有太明显规律的字符串是一个很困难的事情，所以我个人认为利用rowid查询记录的实用性不是很大)rowid只有在表发生移动(比如表空间变化，数据导入/导出以后)，才会发生变化。

### 左、右连接

关于oracle外连接的若干问题

内连接（inner join）、全连接（full join）、左连接（left join）、右连接（right join）

Oracle的左连接和右连接

PL-SQL中，左连接和右连接以如下方式来实现

查看如下语句：

SELECT emp\_name, dept\_name FORM Employee, Department WHERE Employee.emp\_deptid(+) = Department.deptid

此SQL文使用了右连接，即“(+)”所在位置的另一侧为连接的方向，右连接说明等号右侧的所有记录均会被显示， 无论其在左侧是否得到匹配，也就是说上例中无论会不会出现某个部门没有一个员工的情况，这个部门的名字都会在查询结果中出现。

反之：

SELECT emp\_name, dept\_name FORM Employee, Department WHERE Employee.emp\_deptid = Department.deptid(+)

则是左连接，无论这个员工有没有一个能在Department表中得到匹配的部门号，这个员工的记录都会被显示

一般的相等连接：

select \* from a, b where a.id = b.id;

这个就属于内连接。

对于外连接：

Oracle中可以使用“(+) ”来表示，9i可以使用LEFT/RIGHT/FULL OUTER JOIN

LEFT OUTER JOIN：左外关联

SELECT e.last\_name, e.department\_id, d.department\_name

FROM employees e

LEFT OUTER JOIN departments d

ON (e.department\_id = d.department\_id);

等价于

SELECT e.last\_name, e.department\_id, d.department\_name

FROM employees e, departments d

WHERE e.department\_id=d.department\_id(+)

结果为：所有员工及对应部门的记录，包括没有对应部门编号department\_id的员工记录。

RIGHT OUTER JOIN：右外关联

SELECT e.last\_name, e.department\_id, d.department\_name

FROM employees e

RIGHT OUTER JOIN departments d

ON (e.department\_id = d.department\_id);

等价于

SELECT e.last\_name, e.department\_id, d.department\_name

FROM employees e, departments d

WHERE e.department\_id(+)=d.department\_id

结果为：所有员工及对应部门的记录，包括没有任何员工的部门记录。

FULL OUTER JOIN：全外关联

SELECT e.last\_name, e.department\_id, d.department\_name

FROM employees e

FULL OUTER JOIN departments d

ON (e.department\_id = d.department\_id);

结果为：所有员工及对应部门的记录，包括没有对应部门编号department\_id的员工记录和没有任何员工的部门记录。

### 数据库导出

exp CSATEST/CSATEST FULL=y FILE=dba.dmp LOG=dba.log

exp UNEVIOX/UNEVIOX@CSATEST file=dba.dmp full=y

conn UNEVIOX/UNEVIOX@CSATEST AS sysdba

conn sys/root@CSATEST as sysdba

exp UNEVIOX/UNEVIOX@CSATEST file=dba.dmp full=y

expdp system/oracle directory=backup dumpfile=alldb.dmp full=y

### GROUP函数

#### 相关函数

Oracle的group by除了基本用法以外，还有3种扩展用法，分别是rollup、cube、grouping sets。

假设有一个表test，有A、B、C、D、E5列。

1） 如果使用group by rollup(A,B,C)，首先会对(A、B、C)进行GROUP BY，然后对(A、B)进行GROUP BY，然后是(A)进行GROUP BY，最后对全表进行GROUP BY操作。roll up的意思是“卷起”，这也可以帮助我们理解group by rollup就是对选择的列从右到左以一次少一列的方式进行grouping直到所有列都去掉后的grouping(也就是全表grouping)，对于n个参数的rollup，有n+1次的grouping。以下2个sql的结果集是一样的：

Select A,B,C,sum(E) from test group by rollup(A,B,C)和

Select A,B,C,sum(E) from test group by A,B,C

union all

Select A,B,null,sum(E) from test group by A,B

union all

Select A,null,null,sum(E) from test group by A

union all

Select null,null,null,sum(E) from test

2) cube的意思是立方，对cube的每个参数，都可以理解为取值为参与grouping和不参与grouping两个值的一个维度，然后所有维度取值组合的集合就是grouping的集合，对于n个参数的cube，有2^n次的grouping。如果使用group by cube(A,B,C), ，则首先会对(A、B、C)进行GROUP BY，然后依次是(A、B)，(A、C)，(A)，(B、C)，(B)，(C)，最后对全表进行GROUP BY操作，一共是2^3=8次grouping。同rollup一样，也可以用基本的group by加上结果集的union all写出一个与group by cube结果集相同的sql：

Select A,B,C,sum(E) from test group by cube(A,B,C)；

Select A,B,C,sum(E) from test group by A,B,C

union all

Select A,B,null,sum(E) from test group by A,B

union all

Select A,null,C,sum(E) from test group by A,C

union all

Select A,null,null,sum(E) from test group by A

union all

Select null,B,C,sum(E) from test group by B,C

union all

Select null,B,null,sum(E) from test group by B

union all

Select null,null,C,sum(E) from test group by C

union all

Select null,null,null,sum(E) from test

3) grouping sets就是对参数中的每个参数做grouping，也就是有几个参数做几次grouping, 例如使用group by grouping sets(A,B,C)，则对(A),(B),(C)进行group by，如果使用group by grouping sets((A,B),C),则对(A,B),(C)进行group by。甚至grouping by grouping set(A,A)都是语法允许的，也就是对(A)进行2次group by, grouping sets的参数允许重复

需要说明的几点：

1） 在rollup和cube的说明中分别给出了用基本group by加结果集union all给出了结果集相同的sql，但这只是为了理解的方便而给出的sql，并不说明rollup和cube与基本group by加结果集union all等价。实际上两者的内部机制是安全不一样的，前者除了写法简洁以外，运行时不需多次扫描表，效率远比后者高。

2） 3种扩展用法的参数可以是源表中的某一个具体的列，也可以是若干列经过计算而形成的一个新列（比如说A+B，A||B），也可以是这两种列的一个集合（例如（A+B，C）），对于grouping set更是特殊，可以是空集合()，表示对全表进行group by。

3） Group by的基本用法以及这3种扩展用法可以组合使用，也就是说可以出现group by A,rollup(A,B)这样的用法，oracle将对出现在group by中的每种用法的grouping列集合做笛卡尔积然后对其中的每一个元素做group by。这话说起来挺绕口，举例说明吧，group by A,rollup(A,B)，基本用法的grouping集合是(A),rollup(A,B)的grouping集合是((A,B),(A),()),两个集合的笛卡尔积集合是((A,A,B),(A,A),(A))，所以会首先对(A,A,B)做group by，然后对(A,A)做group by，最后对(A)做group by。实际上对(A,A,B)做group by和对(A,B)做group by两者是完全等价的(group by A,A,B结果和group by A,B完全一样)，同理对(A,A)做group by和对(A)做group by也是等价的。简化后的结果就是首先对(A,B)做group by，然后对(A)做group by，最后再对(A)做group by。下面给出两个等价的sql以便理解：

Select A,B,sum(E) from test1 group by A,rollup(A,B);

Select A,B,sum(E) from test1 group by A,B

Union all

Select A,null,sum(E) from test1 group by A

Union all

Select A,null,sum(E) from test1 group by A;

相关函数grouping(),grouping\_id(),group\_id()的用法：

1） grouping():参数只有一个，而且必须为group by中出现的某一列，表示结果集的一行是否对该列做了grouping。对于对该列做了grouping的行而言，grouping()=0，反之为1；

2） grouping\_id():参数可以是多个，但必须为group by中出现的列。Grouping\_id()的返回值其实就是参数中的每列的grouping()值的二进制向量，例如如果grouping(A)=1，grouping(B)=0，则grouping\_id(A,B)的返回值就是二进制的10，转成10进制就是2。

3） group\_id():无参数。见上面的说明3），group by对某些列的集合会进行重复的grouping，而实际上绝大多数情况下对结果集中的这些重复行是不需要的，那就必须有办法剔出这些重复grouping的行。当结果集中有n条重复grouping而形成的行时，每行的group\_id()分别是0,1,…,n，这样我们在条件中加入一个group\_id()<1就可以剔出这些重复grouping的行了。

扩展用法及相关函数的应用例子：

数据库ACCINF表结构为：区域号area\_no，机构号org\_no，账户号acc\_no，账户余额acc\_bal，现有一个 报表要求先按机构统计每个机构的总余额，最后是所有区域的总余额，也就是说报表格式为：

区域1 机构1 余额数

区域1 机构2 余额数

区域2 机构3 余额数

区域2 机构4 余额数

总合计 余额数

以往的sql是通过先计算出各机构的总余额，然后在程序里将所有的余额相加计算出所有区域的总余额。但使用扩展用法之后的sql就非常简单了：

Select nvl(area\_no,’总合计’),org\_no,sum(acc\_bal) from acc\_inf group by rollup((area\_no,org\_no));

如果使用上面的函数，还可以有其它的写法，一个例子是：

Select nvl(area\_no,’总合计’),org\_no,sum(acc\_bal) from acc\_inf group by rollup(area\_no,org\_no) having (grouping\_id(area\_no,org\_no)=0 or grouping\_id(area\_no,org\_no)=3)

#### over partition by与group by 的区别

over partition by 与 group by 都是与统计类函数用，这两个有什么区别呢？

目前我只知道一个这样的区别：

比如有一张表saraly:

CREATE TABLE SALARY AS SELECT 'A' NAME,10 DEPT,1000 SALARY FROM DUAL

UNION ALL SELECT 'B',10,2000 FROM DUAL

UNION ALL SELECT 'C' ,20,1500 FROM DUAL

UNION ALL SELECT 'D',20,3000 FROM DUAL

UNION ALL SELECT 'E',10,1000 FROM DUAL;

NAME DEPT SALARY

A 10 1000

B 10 2000

C 20 1500

D 20 3000

E 10 1000

用over partition by 我就可以查询到每位员工本来的具体信息和它所在部门的总工资：

select name,dept,salary,sum(salary) over (partition by dept) total\_salary from salary;

name dept salary tatal\_salary

A 10 1000 4000

B 10 2000 4000

E 10 1000 4000

C 20 1500 4500

D 20 3000 4500

用goup by 就没办法做到这点，只能查询到每个部门的总工资：

select dept,sum(salary) total\_salary from salary group by dept

dept total\_salary

10 4000

20 4500

另外over partion by 还可以做到查询每位员工占部门总工资的百分比：

select name,dept,salary,salary\*100/sum(salary) over (partition by dept) percent from salary;

name dept salary percent

A 10 1000 25

B 10 2000 50

E 10 1000 25

C 20 1500 33.3333333333333

D 20 3000 66.6666666666667

over 是分析函数

group是分组

select deptno,ename,sal,

sum(sal) over (partition by deptno order by ename) --部门连续求和, --各部门的薪水"连续"求和

sum(sal) over (partition by deptno) 部门总和, -- 部门统计的总和，同一部门总和不变

100\*round(sal/sum(sal) over (partition by deptno),4) "部门份额(%)",

sum(sal) over (order by deptno,ename) 连续求和, --所有部门的薪水"连续"求和

sum(sal) over () 总和, -- 此处sum(sal) over () 等同于sum(sal)，所有员工的薪水总和

100\*round(sal/sum(sal) over (),4) "总份额(%)"

from emp

CREATE TABLE EMP ( ENAME CHAR(1), DEPTNO NUMBER, SAL NUMBER );

insert into EMP (ENAME, DEPTNO, SAL) values ('A', 10, 1000);

insert into EMP (ENAME, DEPTNO, SAL) values ('B', 10, 2000);

insert into EMP (ENAME, DEPTNO, SAL) values ('C', 20, 1500);

insert into EMP (ENAME, DEPTNO, SAL) values ('D', 20, 3000);

insert into EMP (ENAME, DEPTNO, SAL) values ('E', 10, 1000);

#### GROUP\_ID函数使用

GROUP\_ID distinguishes duplicate groups resulting from a GROUP BY specification. It is useful in filtering out duplicate groupings from the query result. It returns an Oracle NUMBER to uniquely identify duplicate groups. This function is applicable only in a SELECT statement that contains a GROUP BY clause.

If n duplicates exist for a particular grouping, then GROUP\_ID returns numbers in the range 0 to n-1.

GROUP\_ID函数能够区分GROUP BY表达式所产生的复制组。从查询结果中过滤复制组时显得特别有用。

它返回一个ORACLE NUMBER类型的数值来唯一标示每一个复制组。这个函数只是在包含GROUP BY 从句的select语句中使用。

Examples

The following example assigns the value 1 to the duplicate co.country\_region grouping from a query on the sample tables sh.countries and sh.sales:

SELECT co.country\_region, co.country\_subregion,

SUM(s.amount\_sold) "Revenue",

GROUP\_ID() g

FROM sales s, customers c, countries co

WHERE s.cust\_id = c.cust\_id AND

c.country\_id = co.country\_id AND

s.time\_id = '1-JAN-00' AND

co.country\_region IN ('Americas', 'Europe')

GROUP BY co.country\_region,

ROLLUP (co.country\_region, co.country\_subregion);

COUNTRY\_REGION COUNTRY\_SUBREGION Revenue G

-------------------- -------------------- ---------- ----------

Americas Northern America 220844 0

Americas Southern America 10872 0

Europe Eastern Europe 12751 0

Europe Western Europe 558686 0

Americas 231716 0

Europe 571437 0

Americas 231716 1

Europe 571437 1

To ensure that only rows with GROUP\_ID < 1 are returned, add the following HAVING clause to the end of the statement :

HAVING GROUP\_ID() < 1

#### Grouping函数使用：

GROUPING distinguishes superaggregate rows from regular grouped rows. GROUP BY extensions such as ROLLUP and CUBE produce superaggregate rows where the set of all values is represented by null. Using the GROUPING function, you can distinguish a null representing the set of all values in a superaggregate row from a null in a regular row.

The expr in the GROUPING function must match one of the expressions in the GROUP BY clause. The function returns a value of 1 if the value of expr in the row is a null representing the set of all values. Otherwise, it returns zero. The datatype of the value returned by the GROUPING function is Oracle NUMBER. Please refer to the SELECT group\_by\_clause for a discussion of these terms.

Examples

In the following example, which uses the sample tables hr.departments and hr.employees, if the GROUPING function returns 1 (indicating a superaggregate row rather than a regular row from the table), then the string "All Jobs" appears in the "JOB" column instead of the null that would otherwise appear:

SELECT DECODE(GROUPING(department\_name), 1, 'All Departments',

department\_name) AS department,

DECODE(GROUPING(job\_id), 1, 'All Jobs', job\_id) AS job,

COUNT(\*) "Total Empl", AVG(salary) \* 12 "Average Sal"

FROM employees e, departments d

WHERE d.department\_id = e.department\_id

GROUP BY ROLLUP (department\_name, job\_id);

DEPARTMENT JOB Total Empl Average Sal

------------------------------ ---------- ---------- -----------

Accounting AC\_ACCOUNT 1 99600

Accounting AC\_MGR 1 144000

Accounting All Jobs 2 121800

Administration AD\_ASST 1 52800

Administration All Jobs 1 52800

Executive AD\_PRES 1 288000

Executive AD\_VP 2 204000

Executive All Jobs 3 232000

Finance FI\_ACCOUNT 5 95040

Finance FI\_MGR 1 144000

Finance All Jobs 6 103200

#### GROUPING\_ID函数使用：

GROUPING\_ID returns a number corresponding to the GROUPING bit vector associated with a row. GROUPING\_ID is applicable only in a SELECT statement that contains a GROUP BY extension, such as ROLLUP or CUBE, and a GROUPING function. In queries with many GROUP BY expressions, determining the GROUP BY level of a particular row requires many GROUPING functions, which leads to cumbersome SQL. GROUPING\_ID is useful in these cases.

GROUPING\_ID is functionally equivalent to taking the results of multiple GROUPING functions and concatenating them into a bit vector (a string of ones and zeros). By using GROUPING\_ID you can avoid the need for multiple GROUPING functions and make row filtering conditions easier to express. Row filtering is easier with GROUPING\_ID because the desired rows can be identified with a single condition of GROUPING\_ID = n. The function is especially useful when storing multiple levels of aggregation in a single table.

Examples

The following example shows how to extract grouping IDs from a query of the sample table sh.sales:

SELECT channel\_id, promo\_id, sum(amount\_sold) s\_sales,

GROUPING(channel\_id) gc,

GROUPING(promo\_id) gp,

GROUPING\_ID(channel\_id, promo\_id) gcp,

GROUPING\_ID(promo\_id, channel\_id) gpc

FROM sales

WHERE promo\_id > 496

GROUP BY CUBE(channel\_id, promo\_id);

C PROMO\_ID S\_SALES GC GP GCP GPC

- ---------- ---------- ---------- ---------- ---------- ----------

C 497 26094.35 0 0 0 0

C 498 22272.4 0 0 0 0

C 499 19616.8 0 0 0 0

C 9999 87781668 0 0 0 0

C 87849651.6 0 1 1 2

I 497 50325.8 0 0 0 0

I 498 52215.4 0 0 0 0

I 499 58445.85 0 0 0 0

I 9999 169497409 0 0 0 0

I 169658396 0 1 1 2

P 497 31141.75 0 0 0 0

P 498 46942.8 0 0 0 0

P 499 24156 0 0 0 0

P 9999 70890248 0 0 0 0

P 70992488.6 0 1 1 2

S 497 110629.75 0 0 0 0

S 498 82937.25 0 0 0 0

S 499 80999.15 0 0 0 0

S 9999 267205791 0 0 0 0

S 267480357 0 1 1 2

T 497 8319.6 0 0 0 0

T 498 5347.65 0 0 0 0

T 499 19781 0 0 0 0

T 9999 28095689 0 0 0 0

T 28129137.3 0 1 1 2

497 226511.25 1 0 2 1

498 209715.5 1 0 2 1

499 202998.8 1 0 2 1

9999 623470805 1 0 2 1

624110031 1 1 3 3

### DECODE & CASE Functions

Decode and Case are very similar in their appearance but can produce very different results.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | **Version 11.1** | |  | | |  |  | | --- | --- | | **Note:** Decode and Case are very similar in their appearance but can produce very different results. | | | Demo Tables & Data | |  |  |  | | --- | --- | --- | | [Airplanes Table](file:///E:\Projects\collection\资料\DB\Oracle\files\airplanes.sql) | [Locations Table](file:///E:\Projects\collection\资料\DB\Oracle\files\locations.sql) | [All other tables](file:///E:\Projects\collection\资料\DB\Oracle\files\build01.sql) | | | DECODE (overload 1) | standard.decode(expr NUMBER, pat NUMBER, res NUMBER) RETURN NUMBER; | | DECODE (overload 2) | standard.decode( expr NUMBER, pat NUMBER, res VARCHAR2 CHARACTER SET ANY\_CS) return VARCHAR2 CHARACTER SET res%CHARSET; | | DECODE (overload 3) | standard.decode(expr NUMBER, pat NUMBER, res DATE) RETURN DATE; | | DECODE (overload 4) | standard.decode( expr VARCHAR2 CHARACTER SET ANY\_CS, pat VARCHAR2 CHARACTER SET expr%CHARSET, res NUMBER) RETURN NUMBER; | | DECODE (overload 5) | standard.decode( expr VARCHAR2 CHARACTER SET ANY\_CS, pat VARCHAR2 CHARACTER SET expr%CHARSET, res VARCHAR2 CHARACTER SET ANY\_CS) RETURN VARCHAR2 CHARACTER SET res%CHARSET; | | DECODE (overload 6) | standard.decode( expr VARCHAR2 CHARACTER SET ANY\_CS, pat VARCHAR2 CHARACTER SET expr%CHARSET, res DATE) RETURN DATE; | | DECODE (overload 7) | standard.decode(expr DATE, pat DATE, res NUMBER) RETURN NUMBER; | | DECODE (overload 8) | standard.decode( expr DATE, pat DATE, res VARCHAR2 CHARACTER SET ANY\_CS) RETURN VARCHAR2 CHARACTER SET res%CHARSET; | | DECODE (overload 9) | standard.decode( expr DATE, pat DATE, res DATE) RETURN DATE; | | DECODE (overload 10) | standard.decode(expr OBJECT, pat OBJECT, res OBJECT) RETURN OBJECT; | | DECODE (overload 11) | standard.decode(expr UNDEFINED, pat UNDEFINED, res UNDEFINED) RETURN UNDEFINED; | |  | | | **Decode Built-in Function** | | | Simple DECODE | SELECT DECODE (value, <if this value>, <return this value>) FROM dual; | | SELECT program\_id,  DECODE(customer\_id, 'AAL', 'American Airlines') AIRLINE, delivered\_date FROM airplanes WHERE ROWNUM < 11; | | More Complex DECODE | SELECT DECODE (value,<if this value>,<return this value>, < if this value>,<return this value>, **....**) FROM dual; | | SELECT program\_id,  DECODE(customer\_id, 'AAL', 'American Airlines', 'ILC', 'Intl. Leasing Corp.', 'NWO', 'Northwest Orient', 'SAL', 'Southwest Airlines', 'SWA', 'Sweptwing Airlines', 'USAF', 'U.S. Air Force') AIRLINE, delivered\_date FROM airplanes WHERE ROWNUM < 11; | | DECODE with DEFAULT | SELECT DECODE (value,<if this value>,<return this value>, <if this value>,<return this value>, **....** <otherwise this value>) FROM dual; | | SELECT program\_id,  DECODE(customer\_id, 'AAL', 'American Airlines', 'ILC', 'Intl. Leasing Corp.', 'NWO', 'Northwest Orient', 'SAL', 'Southwest Airlines', 'SWA', 'Sweptwing Airlines', 'USAF', 'United States Air Force', 'Not Known') AIRLINE, delivered\_date FROM airplanes WHERE ROWNUM < 11; | | **Note:** The following crosstabulation is the standard for 10g or earlier. In 11g use the PIVOT and UNPIVOT operators | | | Simple DECODE Crosstab  Note how each decode only looks at a single possible value and turns it into a new column | SELECT program\_id, DECODE(customer\_id, 'AAL', 'AAL') AMERICAN, DECODE(customer\_id, 'DAL', 'DAL') DELTA, DECODE(customer\_id, 'NWO', 'NWO') NORTHWEST, DECODE(customer\_id, 'ILC', 'ILC') INTL\_LEASING FROM airplanes WHERE rownum < 20; | | DECODE as an in-line view with crosstab summation | The above DECODE, in blue, used as an in-line view | | SELECT program\_id, COUNT (AMERICAN) AAL, COUNT (DELTA) DAL, COUNT (NORTHWEST) NWO, COUNT(INTL\_LEASING) ILC FROM **(** SELECT program\_id, DECODE(customer\_id, 'AAL', 'AAL') AMERICAN, DECODE(customer\_id, 'DAL', 'DAL') DELTA, DECODE(customer\_id, 'NWO', 'NWO') NORTHWEST, DECODE(customer\_id, 'ILC', 'ILC') INTL\_LEASING FROM airplanes**)** GROUP BY program\_id; | | Query for DECODE demo | CREATE TABLE stores ( store\_name VARCHAR2(20), region\_dir NUMBER(5), region\_mgr NUMBER(5), store\_mgr1 NUMBER(5), store\_mgr2 NUMBER(5), asst\_storemgr1 NUMBER(5), asst\_storemgr2 NUMBER(5), asst\_storemgr3 NUMBER(5));  INSERT INTO stores  VALUES ('San Francisco',100,200,301,302,401,0,403);  INSERT INTO stores VALUES ('Oakland',100,200,301,0,404,0,0);  INSERT INTO stores VALUES ('Palo Alto',100,200,0,305,0,405,406);  INSERT INTO stores VALUES ('Santa Clara',100,250,0,306,0,0,407); COMMIT;  SELECT DECODE(asst\_storemgr1, 0, DECODE(asst\_storemgr2, 0, DECODE(asst\_storemgr3, 0, 0, asst\_storemgr3), asst\_storemgr2), asst\_storemgr1) ASST\_MANAGER, DECODE(store\_mgr1,0, DECODE(store\_mgr2,0, 0, store\_mgr2), store\_mgr1) STORE\_MANAGER, REGION\_MGR, REGION\_DIR FROM stores; | | DECODE with Summary Function | SELECT SUM(CA\_COUNT) CA, SUM(TX\_COUNT) TX FROM ( SELECT state, DECODE(state, 'CA', COUNT(\*), 0) CA\_COUNT, DECODE(state, 'TX', COUNT(\*), 0) TX\_COUNT FROM locations GROUP BY state); | | DECODE in the WHERE clause | set serveroutput on  DECLARE posn PLS\_INTEGER := 0; empid PLS\_INTEGER := 178; x NUMBER; BEGIN SELECT NVL(SUM(ah.quantity \* ah.saleprice \* ap.payoutpct), 0) INTO x FROM accessoryhistory ah, payoutpercentage ap,  sku s, store st WHERE empid = DECODE(posn,  0, st.areadir,  1, st.areamgr,  2, NVL(st.storemgr1, st.storemgr2), 3, NVL(st.asstmgr1, NVL(st.asstmgr2, st.asstmgr3))) AND ah.statustype IN ('ACT', 'DEA') AND ah.store = st.store AND s.dbid = ah.dbid AND s.sku = ah.sku AND ap.productgroup = s.productgroup AND ap.position = posn;  dbms\_output.put\_line(x); END; / | | DECODE Altered WHERE Clause  Thanks to HJL | CREATE TABLE test ( pubdate DATE, compdate DATE, valuecol NUMBER(5));  INSERT INTO test VALUES (TRUNC(SYSDATE), TRUNC(SYSDATE+300), 1); INSERT INTO test VALUES (TRUNC(SYSDATE-300), TRUNC(SYSDATE), 9); COMMIT;  SELECT \* FROM test;  CREATE OR REPLACE PROCEDURE testproc ( StartDate DATE, EndDate DATE, DateType IN VARCHAR2) IS  i PLS\_INTEGER; BEGIN SELECT valuecol INTO i FROM test WHERE DECODE(DateType, 'AA',compdate, 'BB', pubdate, compdate) <= EndDate AND DECODE(DateType, 'AA', compdate, 'BB', pubdate, compdate) >= StartDate;  dbms\_output.put\_line(TO\_CHAR(i)); END testproc; /  set serveroutput on  exec testproc(TRUNC(SYSDATE), TRUNC(SYSDATE), 'BB'); | |  | | | **CASE** | | | Simple CASE Demo | SELECT CASE WHEN (<column\_value>**=** <value>) THEN WHEN (<column\_value> **=** <value>) THEN ELSE <value> FROM <table\_name>; | | SELECT line\_number,  CASE WHEN (line\_number = 1) THEN 'One' WHEN (line\_number = 2) THEN 'Two' ELSE 'More Than Two' END AS RESULTSET FROM airplanes; | | More Complex CASE Demo With Between | SELECT CASE WHEN (<column\_value> BETWEEN <value>  AND <value>) THEN WHEN (<column\_value> BETWEEN <value> AND <value>) THEN ELSE <value> FROM <table\_name>; | | SELECT line\_number,  CASE WHEN (line\_number BETWEEN 1 AND 10) THEN 'One' WHEN (line\_number BETWEEN 11 AND 100) THEN 'Big' ELSE 'Bigger' END FROM airplanes; | | More Complex CASE Demo With Booleans | SELECT CASE WHEN (<column\_value> **<=** <value>) THEN WHEN (<column\_value> **<=** <value>) THEN ELSE <value> FROM <table\_name>; | | SELECT line\_number, CASE WHEN (line\_number < 10) THEN 'Ones' WHEN (line\_number < 100) THEN 'Tens' WHEN (line\_number < 1000) THEN 'Hundreds' ELSE 'Thousands'  END RESULT\_SET FROM airplanes; | | The above demo turned into a view | CREATE OR REPLACE VIEW line\_number\_view AS SELECT line\_number, CASE WHEN (line\_number < 10) THEN 'Ones' WHEN (line\_number < 100) THEN 'Tens' WHEN (line\_number < 1000) THEN 'Hundreds' ELSE 'Thousands' END RESULT\_SET FROM airplanes; | | CASE with BOOLEANS | set serveroutput on  DECLARE boolvar BOOLEAN := TRUE; BEGIN dbms\_output.put\_line(CASE boolvar WHEN TRUE THEN 'TRUE' WHEN FALSE THEN 'FALSE' END); END; / | |  | | | **CASE - DECODE Comparison** | | | The same functionality written using both functions | SELECT parameter, DECODE(SIGN(parameter-1000),-1,'C','P') AS BAND FROM parameter\_table;  SELECT parameter, CASE WHEN parameter < 1000 THEN 'C' ELSE 'P' END AS BAND FROM parameter\_table; | | Another example using SIGN | SELECT min\_cached, COUNT(\*), ROUND(AVG(executions),2) FROM ( SELECT DECODE(min\_cached,  0, '1) 00-01 min', 1, '2) 01-02min', 2, '2) 01-02min', DECODE(SIGN(min\_cached -6), -1, '3) 03-05min', DECODE(SIGN(min\_cached -16), -1, '4) 06-15min', DECODE(SIGN(min\_cached -31), -1, '5) 16-30min', DECODE(SIGN(min\_cached -61), -1, '6) 31-60min', DECODE(SIGN(min\_cached-121), -1, '7) 1-2hrs', '8) 2 hrs+ ')))))) min\_cached, executions FROM ( SELECT ROUND((SYSDATE - TO\_DATE(first\_load\_time,'YYYY-MM-DD/HH24:MI:SS'))\*24\*60) min\_cached, executions FROM gv$sqlarea WHERE parsing\_user\_id != 0) ) GROUP BY min\_cached | | |  | | |  | | --- | | **Related Topics** | | [PIVOT](file:///E:\Projects\collection\资料\DB\Oracle\misc\ora_operators.html#oppv) | | [Select](file:///E:\Projects\collection\资料\DB\Oracle\misc\select.html) | | [String Functions](file:///E:\Projects\collection\资料\DB\Oracle\misc\string_func.html) | | [UN](file:///E:\Projects\collection\资料\DB\Oracle\misc\ora_operators.html)[PIVOT](file:///E:\Projects\collection\资料\DB\Oracle\misc\ora_operators.html#oppv) | | |  | |
|  |

### 锁

oracle 解锁

--查看锁

select t2.\* from v$locked\_object t1,v$session t2

where t1.session\_id=t2.sid order by t2.logon\_time;

--alter system kill session 'sid,serial#';

--把锁给KILL掉

alter system kill session '146,21177';

根据你用第1条代码查得到的结果，来kill 锁，注意看第2条注释部分的内容。

### Oracle Errors

#### ORA-01810 Error

|  |  |
| --- | --- |
| Error: | ORA-01810: format code appears twice |
| Cause: | You tried to use the [to\_date](http://www.techonthenet.com/oracle/functions/to_date.php) function in a query, but you used a format code twice in the date format. |
| Action: | The options to resolve this Oracle error are:   1. Re-write the to\_date function so that you only use each format code once. Examples of format codes are:  |  |  | | --- | --- | | **Format Code** | **Explanation** | | YEAR | Year, spelled out | | YYYY | 4-digit year | | MM | Month (01-12; JAN = 01). | | MON | Abbreviated name of month. | | MONTH | Name of month, padded with blanks to length of 9 characters. | | D | Day of week (1-7). | | DAY | Name of day. | | DD | Day of month (1-31). | | DDD | Day of year (1-366). | | DY | Abbreviated name of day. | | HH | Hour of day (1-12). | | HH12 | Hour of day (1-12). | | HH24 | Hour of day (0-23). | | MI | Minute (0-59). | | SS | Second (0-59). | | SSSSS | Seconds past midnight (0-86399). |   For example, if you tried to execute the following SELECT statement:  **SELECT to\_date('2004/12/14 4:29 PM', 'YYYY/MM/DD HH:MM PM' ) FROM dual;**  You would receive the following error message:  E:\Projects\collection\资料\DB\Oracle\misc\ORA-01810 format code appears twice_files\ora01810_001.png  Some people mistakenly use the **MM** format code to represent minutes, thus using the MM format for both the months and the minutes.  You could correct this SELECT statement as follows:  **SELECT to\_date('2004/12/14 4:29 PM', 'YYYY/MM/DD HH:MI PM' ) FROM dual;** |

### 删除重复数据

做项目的时候，一位同事导数据的时候，不小心把一个表中的数据全都搞重了，也就是说，这个表里所有的记录都有一条重复的。这个表的数据是千万级的，而且是生产系统。也就是说，不能把所有的记录都删除，而且必须快速的把重复记录删掉。

对此，总结了一下删除重复记录的方法，以及每种方法的优缺点。

为了陈诉方便，假设表名为Tbl，表中有三列col1，col2，col3，其中col1，col2是主键，并且，col1，col2上加了索引。

**1、通过创建临时表**

可以把数据先导入到一个临时表中，然后删除原表的数据，再把数据导回原表，SQL语句如下：

creat table tbl\_tmp (select distinct\* from tbl);truncate table tbl;//清空表记录insert into tbl select \* from tbl\_tmp;//将临时表中的数据插回来。

这种方法可以实现需求，但是很明显，对于一个千万级记录的表，这种方法很慢，在生产系统中，这会给系统带来很大的开销，不可行。

**2、利用rowid**

在oracle中，每一条记录都有一个rowid，rowid在整个数据库中是唯一的，rowid确定了每条记录是oracle中的哪一个数据文件、块、行上。在重复的记录中，可能所有列的内容都相同，但rowid不会相同。SQL语句如下：

delete from tbl where rowid in (select a.rowid from tbl a, tbl b where a.rowid>b.rowid and a.col1=b.col1 and a.col2 = b.col2)

如果已经知道每条记录只有一条重复的，这个sql语句适用。但是如果每条记录的重复记录有N条，这个N是未知的，就要考虑适用下面这种方法了。

**3、利用max或min函数**

这里也要使用rowid，与上面不同的是结合max或min函数来实现。SQL语句如下

delete from tbl awhere rowid not in (select max(b.rowid) from tbl b where a.col1=b.col1 and a.col2 = b.col2);//这里max使用min也可以

或者用下面的语句

delete from tbl awhere rowid<(select max(b.rowid) from tbl b where a.col1=b.col1 and a.col2 = b.col2);//这里如果把max换成min的话，前面的where子句中需要把"<"改为">"

跟上面的方法思路基本是一样的，不过使用了group by，减少了显性的比较条件，提高效率。SQL语句如下：

deletefrom tbl where rowid not in (select max(rowid) from tbl tgroup by t.col1, t.col2);delete from tbl where (col1, col2) in (select col1,col2 from tblgroup bycol1,col2havingcount(\*) >1)and rowidnotin(selectnin(rowid)fromtblgroup bycol1,col2havingcount(\*) >1)

还有一种方法，对于表中有重复记录的记录比较少的，并且有索引的情况，比较适用。假定col1，col2上有索引，并且tbl表中有重复记录的记录比较少，SQL语句如下4、利用group by，提高效率

原文地址：<http://dev.csdn.net/author/Rayfly/b850da865b3b4cc89008a2004dad56fc.html>[gigi_miao](http://tech.163.com/)

### 修改字符集

Oracle修改字符集2010-07-15 16:54今天导入一个数据文件到ORACLE，导入后发现中文都变成乱码了，检查发现原来是ORACLE的数据库的编码的问题，按以下思路修改编码为UTF8后，数据重新导入就OK了！

Oracle修改字符集

2.3oracle数据库的字符集更改

A、oracle server 端 字符集查询

select userenv(‘language’) from dual

其中NLS\_CHARACTERSET 为server端字符集

NLS\_LANGUAGE 为 server端字符显示形式

B、查询oracle client端的字符集

$echo $NLS\_LANG

如果发现你select 出来的数据是乱码，请把client端的字符集配置成与linux操作系统相同的字符集。如果还是有乱码，则有可能是数据库中的数据存在问题，或者是oracle服务端的配置存在问题。

C、server端字符集修改

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* 更改字符集步骤方法(WE8ISO8859P1 --> ZHS16GBK) \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

SQL>

将数据库启动到RESTRICTED模式下做字符集更改：

SQL> conn /as sysdba

Connected.

SQL> shutdown immediate;

Database closed.

Database dismounted.

ORACLE instance shut down.

SQL> startup mount

ORACLE instance started.

Total System Global Area 236000356 bytes

Fixed Size 451684 bytes

Variable Size 201326592 bytes

Database Buffers 33554432 bytes

Redo Buffers 667648 bytes

Database mounted.

SQL> ALTER SYSTEM ENABLE RESTRICTED SESSION;

System altered.

SQL> ALTER SYSTEM SET JOB\_QUEUE\_PROCESSES=0;

System altered.

SQL> ALTER SYSTEM SET AQ\_TM\_PROCESSES=0;

System altered.

SQL> alter database open;

Database altered.

SQL> ALTER DATABASE CHARACTER SET ZHS16GBK;

ALTER DATABASE CHARACTER SET ZHS16GBK

\*

ERROR at line 1:

ORA-12712: new character set must be a superset of old character set

提示我们的字符集：新字符集必须为旧字符集的超集，这时我们可以跳过超集的检查做更改：

SQL> ALTER DATABASE character set INTERNAL\_USE ZHS16GBK;

Database altered.

SQL> select \* from v$nls\_parameters;

略

19 rows selected.

重启检查是否更改完成：

SQL> shutdown immediate;

Database closed.

Database dismounted.

ORACLE instance shut down.

SQL> startup

ORACLE instance started.

Total System Global Area 236000356 bytes

Fixed Size 451684 bytes

Variable Size 201326592 bytes

Database Buffers 33554432 bytes

Redo Buffers 667648 bytes

Database mounted.

Database opened.

SQL> select \* from v$nls\_parameters;

略

19 rows selected.

我们看到这个过程和之前ALTER DATABASE CHARACTER SET操作的内部过程是完全相同的，也就是说INTERNAL\_USE提供的帮助就是使

Oracle数据库绕过了子集与超集的校验.

这一方法在某些方面是有用处的，比如测试；应用于产品环境大家应该格外小心，除了你以外，没有人会为此带来的后果负责:

结语(我们不妨再说一次):

对于DBA来说，有一个很重要的原则就是:不要把你的数据库置于危险的境地！

这就要求我们，在进行任何可能对数据库结构发生改变的操作之前，先做有效的备份，很多DBA没有备份的操作中得到了惨痛的教训

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

D、client端字符集修改

在 /home/oracle与 /root用户目录下的.bash\_profile中

添加或修改 export NLS\_LANG="AMERICAN\_AMERICA.UTF8" 语句

关闭当前ssh窗口。

注意 NLS\_LANG变量一定要配置正确否则会引起sqlplus 失效

### number精度

oracle中number型字段长度、精度及实际存储状态测试

相关知识回顾:

有效数位：从左边第一个不为0的数算起，小数点和负号不计入有效位数。

number(p,s)

p:1~38

s:-84~127

p>0，对s分2种情况：

1. s>0

精确到小数点右边s位，并四舍五入。然后检验有效数位是否<=p；如果s>p，小数点右边至少有s-p个0填充。

2. s<0

精确到小数点左边s位，并四舍五入。然后检验有效数位是否<=p+|s|

案例测试：

hongsy@test>desc hongsy;

名称 是否为空? 类型

----------------------------------------------------- -------- ------------

NUM1 NUMBER

NUM2 NUMBER(2)

NUM3 NUMBER(5,3)

NUM4 NUMBER(5,-2)

hongsy@test>insert into hongsy(num1) values(1111111111);

已创建 1 行。

hongsy@test>commit;

提交完成。

hongsy@test>insert into hongsy(num2) values(11);

已创建 1 行。

hongsy@test>insert into hongsy(num2) values(111);

insert into hongsy(num2) values(111)

\*

ERROR 位于第 1 行:

ORA-01438: 值大于此列指定的允许精确度

hongsy@test>insert into hongsy(num2) values(-11);

已创建 1 行。

hongsy@test>insert into hongsy(num2) values(-111);

insert into hongsy(num2) values(-111)

\*

ERROR 位于第 1 行:

ORA-01438: 值大于此列指定的允许精确度

hongsy@test>insert into hongsy(num2) values(-11.1);

已创建 1 行。

hongsy@test>insert into hongsy(num2) values(-11.1111);

已创建 1 行。

hongsy@test>commit;

提交完成。

hongsy@test>select num2 from hongsy;

NUM2

----------

11

-11

-11

-11

已选择8行。

hongsy@test>insert into hongsy(num3) values(111);

insert into hongsy(num3) values(111)

\*

ERROR 位于第 1 行:

ORA-01438: 值大于此列指定的允许精确度

hongsy@test>insert into hongsy(num3) values(11);

已创建 1 行。

hongsy@test>insert into hongsy(num3) values(-11);

已创建 1 行。

hongsy@test>insert into hongsy(num3) values(-111);

insert into hongsy(num3) values(-111)

\*

ERROR 位于第 1 行:

ORA-01438: 值大于此列指定的允许精确度

hongsy@test>insert into hongsy(num3) values(11.111);

已创建 1 行。

hongsy@test>insert into hongsy(num3) values(11.1111);

已创建 1 行。

hongsy@test>commit;

提交完成。

hongsy@test>select num3 from hongsy;

NUM3

----------

11

-11

11.111

11.111

已选择12行。

hongsy@test>insert into hongsy(num4) values(1111);

已创建 1 行。

hongsy@test>insert into hongsy(num4) values(111111);

已创建 1 行。

hongsy@test>insert into hongsy(num4) values(11111111);

insert into hongsy(num4) values(11111111)

\*

ERROR 位于第 1 行:

ORA-01438: 值大于此列指定的允许精确度

hongsy@test>commit;

提交完成。

hongsy@test>select num4 from hongsy;

NUM4

----------

1100

111100

已选择14行。

hongsy@test>insert into hongsy(num4) values(1111111);

已创建 1 行。

hongsy@test>commit;

提交完成。

hongsy@test>select num4 from hongsy;

NUM4

----------

1100

111100

1111100

已选择15行。

hongsy@test>create table hongsy1 (num1 number(39));

create table hongsy1 (num1 number(39))

\*

ERROR 位于第 1 行:

ORA-01727: 数字精度说明符超出范围（1 到 38）

hongsy@test>create table hongsy1 (num1 number(38,128));

create table hongsy1 (num1 number(38,128))

\*

ERROR 位于第 1 行:

ORA-01728: 数字标度说明符超出范围（-84 到 127）

### 游标

SQL是用于访问Oracle数据库的语言，PL/SQL扩展和加强了SQL的功能，它同时引入了更强的程序逻辑, 下面在本文中将对游标的使用进行一下讲解,希望可以和大家共同学习进步。

　　游标字面理解就是游动的光标。游标是SQL的一个内存工作区，由系统或用户以变量的形式定义。在某些情况下，需要把数据从存放在磁盘的表中调到计算机内存中进行处理，最后将处理结果显示出来或最终写回数据库。这样数据处理的速度才会提高，否则频繁的磁盘数据交换会降低效率。用数据库语言来描述游标就是映射在结果集中一行数据上的位置实体，有了游标，用户就可以访问结果集中的任意一行数据了，将游标放置到某行后，即可对该行数据进行操作，例如提取当前行的数据等。

　　游标有两种类型：显式游标和隐式游标。游标一旦打开，数据就从数据库中传送到游标变量中，然后应用程序再从游标变量中分解出需要的数据，并进行处理。

　　当系统使用一个隐式游标时，可以通过隐式游标的属性来了解操作的状态和结果，进而控制程序的流程。隐式游标可以使用名字SQL来访问，但要注意，通过SQL游标名总是只能访问前一个处理操作或单行SELECT操作的游标属性。所以通常在刚刚执行完操作之后，立即使用SQL游标名来访问属性。游标的属性有四种,分别是SQL %ISOPEN，SQL %FOUND，SQL %NOTFOUND，SQL %ROWCOUNT。

　　SQL%ISOPEN返回的类型为布尔型,判断游标是否被打开，如果打开%ISOPEN等于true,否则等于false,即执行过程中为真,结束后为假。

　　SQL%NOTFOUND返回值为布尔型,判断游标所在的行是否有效，如果有效，则%FOUNDD等于true，否则等于false,即与%FOUND属性返回值相反。

　　SQL%FOUND返回值的类型为布尔型，值为TRUE代表插入 删除 更新或单行查询操作成功。

　　SQL%ROWCOUNT返回值类型为整型，返回当前位置为止游标读取的记录行数,即成功执行的数据行数。 示例代码如下：

set serveroutput on;

　　declare

　　varno varchar2(20);

　　varprice varchar2(20);

　　CURSOR mycur(vartype number) is

　　select emp\_no,emp\_zc from cus\_emp\_basic

　　where com\_no = vartype;

　　begin

　　if mycur%isopen = false then

　　open mycur(043925);

　　end if;

　　fetch mycur into varno,varprice;

　　while mycur%found

　　loop

　　dbms\_output.put\_line(varno||','||varprice);

　　if mycur%rowcount=2 then

　　exit;

　　end if;

　　fetch mycur into varno,varprice;

　　end loop;

　　close mycur;

　　end;

　　记录并不保存在数据库中，它与变量一样，保存在内存空间中，在使用记录时候，要首先定义记录结构，然后声明记录变量。可以把PL/SQL记录看作是一个用户自定义的数据类型。

　　游标for循环是显示游标的一种快捷使用方式，它使用for循环依次读取结果集中的行数据，当form循环开始时，游标自动打开(不需要open)，每循环一次系统自动读取游标当前行的数据(不需要fetch)，当退出for循环时，游标被自动关闭(不需要使用close)。使用游标for循环的时候不能使用open语句，fetch语句和close语句，否则会产生错误。

set serveroutput on;

　　declare

　　cursor mycur(vartype number)is

　　select emp\_no,emp\_zc from cus\_emp\_basic

　　where com\_no=vartype;

　　begin

　　for person in mycur(000627) loop

　　dbms\_output.put\_line('编号:'||person.emp\_no||',住址:'||person.emp\_zc);

　　end loop;

　　end;

　　显式游标的使用总共分4个步骤：

　　1.声明游标

　　在DECLEAR部分按以下格式声明游标：

　　CURSOR 游标名[(参数1 数据类型[参数2 数据类型...])]

　　IS SELECT语句;

　　参数是可选部分，所定义的参数可以出现在SELECT语句的WHERE子句中。如果定义了参数，则必须在打开游标时传递相应的实际参数。

　　SELECT语句是对表或视图的查询语句，甚至也可以是联合查询。可以带WHERE条件、ORDER BY或GROUP BY等子句，但不能使用INTO子句。在SELECT语句中可以使用在定义游标之前定义的变量。

　　例：

DELCARE

　　CURSOR C\_EMP IS SELECT empno,ename,salary

　　FROM emp 　　WHERE salary>1500

　　ORDER BY ename;

　　........

　　BEGIN

　　在游标定义中SELECT语句中不一定非要表可以是视图，也可以从多个表或视图中选择.

　　2.打开游标

　　使用游标中的值之前应该首先打开游标，打开游标初始化查询处理。打开游标的语法

　　OPEN 游标名[(实际参数1[实际参数2，...])];

　　打开游标时，SELECT语句的查询结果就被传送到了游标工作区。

　　例：

　　OPEN C\_EMP;

　　3.提取数据

　　从游标得到一行数据使用FETCH命令。每一次提取数据后，游标都指向结果集的下一

　　FETCH cursor\_name INTO variable[,variable,...]

　　对于SELECT定义的游标的每一列，FETCH变量列表都应该有一个变量与之相对应，变

　　在可执行部分，按以下格式将游标工作区中的数据取到变量中。提取操作必须在打开游标之后进行。

　　FETCH 游标名 INTO 变量名1[变量名2，...];

　　或

　　FETCH 游标名 INTO 记录变量;

　　游标打开后有一个指针指向数据区，FETCH语句一次返回指针所指的一行数据，要返回多行需重复执行，可以使用循环语句来实现。控制循环可以通过判断游标的属性来进行。

　　下面对这两种格式进行说明：

　　第一种格式中的变量名是用来从游标中接收数据的变量，需要事先定义。变量的个数和类型应与SELECT语句中的字段变量的个数和类型一致。

　　第二种格式一次将一行数据取到记录变量中，需要使用%ROWTYPE事先定义记录变量，这种形式使用起来比较方便，不必分别定义和使用多个变量。

　　定义记录变量的方法如下：

　　变量名 表名|游标名%ROWTYPE;

　　其中的表必须存在，游标名也必须先定义。

4.关闭游标

　　CLOSE 游标名;

　　例：

　　CLOSE C\_EMP;

　　显式游标打开后，必须显式地关闭。游标一旦关闭，游标占用的资源就被释放，游标变成无效，必须重新打开才能使用。

　　Oracle支持动态SELECT语句和动态游标，动态的方法大大扩展了程序设计的能力。

　　对于查询结果为一行的SELECT语句，可以用动态生成查询语句字符串的方法，在程序执行阶段临时地生成并执行，语法是：

　　execute immediate 查询语句字符串 into 变量1[,变量2，...];

　　在变量声明部分定义的游标是静态的，不能在程序运行过程中修改。虽然可以通过参数传递来取得不同的数据，但还是有很大的局限性。通过采用动态游标，可以在程序运行阶段随时生成一个查询语句作为游标。要使用动态游标需要先定义一个游标类型，然后声明一个游标变量，游标对应的查询语句可以在程序的执行过程中动态地说明。

### 修改用户密码

有时候我们可能不知道一个用户的密码，但是又需要以这个用户做一些操作，又不能去修改掉这个用户的密码，这个时候，就可以利用一些小窍门，来完成操作。  
　　具体操作过程如下：  
　　SQL\*Plus: Release 9.2.0.5.0 - Production on 星期日 11月 21 13:32:34 2004  
　　Copyright (c) 1982, 2002, oracle Corporation. All rights reserved.  
　　SQL> connect sys/oracle as sysdba  
　　已连接。  
　　SQL> select username,password from dba\_users;  
　　USERNAME PASSWORD  
　　------------------------------ ------------------------------  
　　SYS 8A8F025737A9097A  
　　SYSTEM 2D594E86F93B17A1

WMSYS 7C9BA362F8314299  
　　已选择6行。  
　　SQL> connect system/oracle  
　　已连接。  
　　SQL> connect sys/oracle as sysdba  
　　已连接。  
　　修改用户system密码为manager  
　　SQL> alter user system identified by manager;  
　　用户已更改。  
　　SQL> select username,password from dba\_users;  
　　USERNAME PASSWORD  
　　------------------------------ ------------------------------  
　　SYS 8A8F025737A9097A  
　　SYSTEM D4DF7931AB130E37

　　WMSYS 7C9BA362F8314299  
　　已选择6行。  
　　SQL> connect system/manager  
　　已连接。  
　　然后此时可以做想要做的任何操作了  
　　SQL> connect sys/oracle as sysdba  
　　已连接。  
　　修改用户system密码为以前的值  
　　SQL> alter user system identified by values '2D594E86F93B17A1';  
　　用户已更改。  
　　SQL> connect system/oracle  
　　已连接。  
　　SQL> connect sys/oracle as sysdba  
　　已连接。  
　　SQL> connect system/manager  
　　ERROR:  
　　ORA-01017: invalid username/password; logon denied  
　　警告: 您不再连接到 oracle。

### 数据批量操作

[**ORACLE批量更新三种方法比较**](http://www.blogjava.net/cheneyfree/archive/2007/12/28/171310.html)

在大型的数据库应用中，我们经常会有针对表与表之间的关键建进行字段更新，那么在这个时候，我们就不能写简单的update来实现更新操作，而要针对具体的数据量来进行批量的update,下面几个例子是常用的SQL,将其做个对比，欢迎大家提出更好，更高效的SQL实现。

数据库：Oracle 9i  测试工具：PL/SQL

定义2张测试表：T1,T2  
T1--大表 10000条 T1\_FK\_ID   
T2--小表 5000条  T2\_PK\_ID  
T1通过表中字段ID与T2的主键ID关联

模拟数据如下：

--T2有5000条记录  
create table T2 as select rownum id, a.\* from all\_objects a where 1=0;   
//T2表的字段和all\_objects表字段类型以及默认值一致，但索引初始化了，需要重新设置  
  
--创建主键ID,向T2表copy数据  
alter table T2 add constraint T2\_PK\_ID primary key (ID);  
insert /\*+ APPEND \*/ into T2 select rownum id, a.\* from all\_objects a where rownum<=5000;  
   
--T1有10000条记录             
create table T1 as select rownum sid, T2.\* from T2 where 1=0;  
  
-- 创建外键ID,向T1表copy数据  
alter table T1 add constraint T1\_FK\_ID foreign key (ID) references t2 (ID);  
insert /\*+ APPEND \*/ into T1 select rownum sid, T2.\* from T2;  
insert /\*+ APPEND \*/ into T1 select rownum sid, T2.\* from T2;  
  
--更新Subobject\_Name字段，初始为NULL  
update T2 set T2.Subobject\_Name='StevenHuang'

需求：我们希望能把T1表的Subobject\_Name字段也全部更新成'StevenHuang'，也就是说T1的10000条记录都会得到更新，以下SQL语句均在PL/SQL命令窗口测试。  
  
方法一:  
写PL/SQL,开cursor

Nonedeclare   
None l\_varID **varchar2**(**20**);  
None l\_varSubName **varchar2**(**30**);  
None cursor mycur is select T2.Id,T2.Subobject\_Name from T2;  
Nonebegin   
None open mycur;   
None loop  
None      fetch mycur into l\_varID,l\_varSubName;  
None      exit when mycur %notfound;  
None      update T1 set T1.Subobject\_Name = l\_varSubName where T1.ID = l\_varID;  
None end loop;  
None close mycur;  
Noneend;

---耗时39.716s  
显然这是最传统的方法，如果数据量巨大的话（4000万记），还会报”snapshot too old”错误退出,PL/SQL工具会挂掉  
  
方法二:  
用loop循环，分批update

Nonedeclare   
None  i **number**;  
None  j **number**;  
Nonebegin  
None  i := **1**;  
None  j := **0**;  
Noneselect count(\*) into j from T1;  
None  loop  
None    exit when i > j;  
None    update T1 set T1.Subobject\_Name = (select T2.Subobject\_Name from T2 where T1.ID = T2.ID) where T1.ID >= i and T1.ID <   
None  
None(i + **1000**);  
None    i := i + **1000**;  
None  end loop;  
Noneend;  
None

--耗时0.656s，这里一共循环了10次，如果数据量巨大的话，虽然能够完成任务，但是速度还是不能令人满意。(例如我们将T1--大表增大到100000记录 T2--小表增大到50000记录,将耗时10.139s)   
  
方法三:  
--虚拟一张表来进行操作，在数据量大的情况下效率比方法二高很多.   
   **注：此语句下T1,T2表中必须有相应的主外建关联，否则sql编译不能通过**.

Noneupdate (select T1.Subobject\_Name A1,T2.Subobject\_Name B1 from T1,T2 where T1.ID=T2.ID) set A1=B1;   
None

--耗时3.234s (T1--大表增大到100000记录 T2--小表增大到50000记录)

方法四.

--由于UPDATE是比较消耗资源的操作，会有redo和undo操作，在这个例子里面我们可以换用下面的方法，创建一张新表，因为采用insert比update快的多，之后你会有一张旧表和一张新表，然后要怎么做就具体情况具体分析了~~~~~

create table T3 as select \* from T1 where rownum<1;

alter table T3 nologging;

insert /\*+ APPEND \*/ into T3

select T1.\* from T1,T2 where T1.ID=T2.ID;

--耗时0.398s (T1--大表增大到100000笔 T2--小表增大到50000笔)

**\*以上所有操作都已经将分析执行计划所需的时间排除在外**

oracle批量导入测试数据

一、Test Windos方式  
declare  
        maxrecords constant int:=1000;  
        i int :=1;  
    begin  
        for i in 1..maxrecords loop  
           insert into test2  
           (id, name)  
           values  
           (test2\_seq.nextval, to\_char('9999'+i));  
        end loop;  
    dbms\_output.put\_line(' 成功录入数据！ ');  
    commit;  
    end;   
二、从已有表中往入另一张表导数据  
create or replace procedure TestProc is  
begin  
  for c in(select id, name from test2) loop  
  insert into test  
    (id,   
     name)  
  values  
    (test\_seq.nextval,   
     c.name);  
  end loop;  
end TestProc;

## SQL Server

### truncate、delete

truncate table tb 'tb为表名

这样就把数据清空，同时IDENTITY 重新计数。

而delete只是清空数据，IDENTITY在原来基础上继续增长。

# 工具：

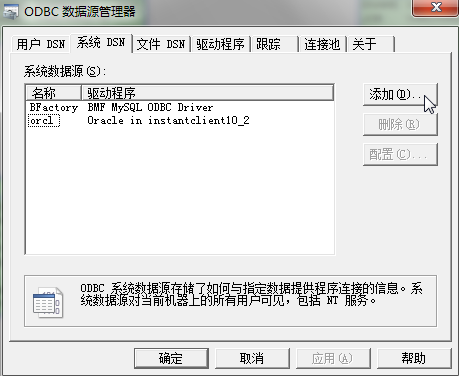
## 配置Power Designer连接Oracle10g数据库

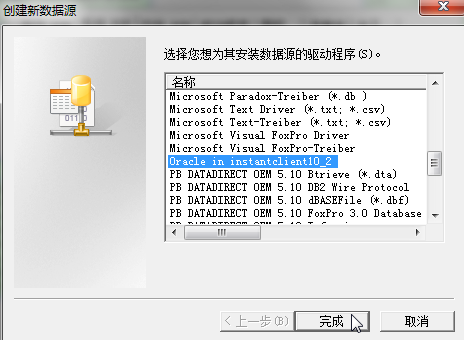
首先，需要下载ODBC 驱动程序 instantclient-odbc-win32-10.2.0.4.zip,解压缩到目录里,下载页面是

<http://www.oracle.com/technology/software/tech/oci/instantclient/htdocs/winsoft.html>;

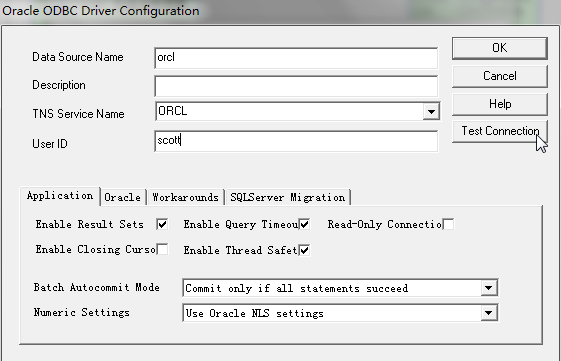
然后执行odbc\_install.exe,这样就在操作系统里安装了odbc for oracle10g的驱动。

打开PD，菜单Database\Configure Connections...  
选择系统DSN，添加，如下图所示

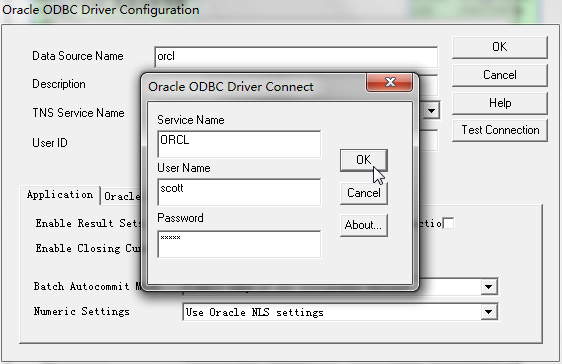
  
选择Oracle in instantclient10\_2，点击完成

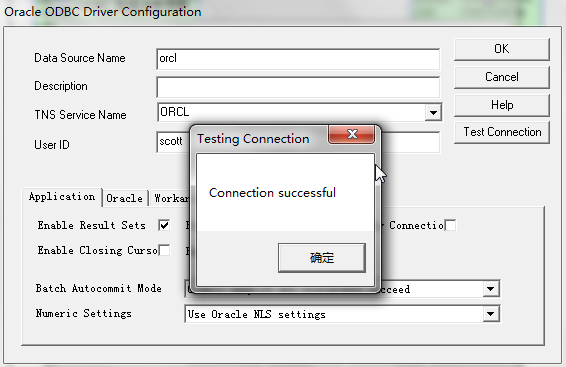


在Oracle ODBC Driver Configuration页面填写必要信息，如下图所示

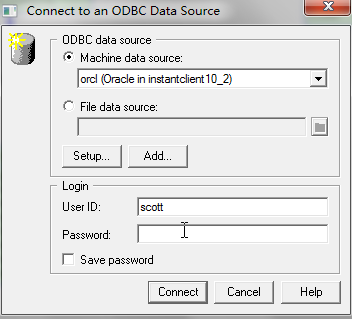


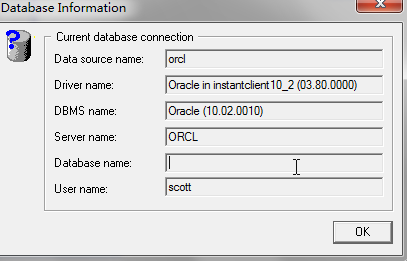
完成后测试连接，输入密码，成功后退出。





选择PD的菜单Database\Connect...，在打开的窗口选择新建立的DSN：orcl，输入密码，确定即可连接



完成后可以通过PD菜单Database\Connection Information...查看连接信息。  


本例在PD12 + Oracle10gR2测试成功。

# 其他

1. Java

(top class in Java)

.java.lang.Object

.StingBuilder, StringBuffer(capacity) vs String

.Error,Exception

.cost of synchronization

.inline final

.public,protected,private

.shallow/deep copy object

.coupling, cohesion;Fan-In vs Fan-Out

.hashCode, ==, equal;

.Thread, Extend Thread, Runnable

.ThreadPool

.primitive (int, char),Integer

.Serializable

.cost of create Object

.double, float;

.class load

.HashMap, Hashtable, HashSet

+=, long l = l + int

JDK 1.5 new feature (concurrentMap)

GC

XML concept

2. Design

3. SQL

3. XML filter

access control

JProbe,JMeter

----

http://www.javaperformancetuning.com/tips/

http://c2.com/cgi/wiki?CouplingAndCohesion

http://www.javaworld.com/javaworld/jw-03-2008/jw-03-osgi1.html

http://www.javaworld.com/jw-12-1996/jw-12-sockets.html

http://www.javaworld.com/javaworld/jw-01-2008/jw-01-jpa1.html

http://www.javaworld.com/javaworld/javaqa/2003-08/01-qa-0808-property.html

http://www.javaworld.com/javaworld/jw-10-2008/jw-10-rest-series-1.html

http://java.sun.com/docs/books/jls/third\_edition/html/j3IX.html

http://java.sun.com/j2se/1.5.0/docs/

http://java.sun.com/docs/books/jvms/second\_edition/html/VMSpecTOC.doc.html

http://java.sun.com/docs/books/jls/index.html