DBMS\_JOB系统包是Oracle“任务队列”子系统的API编程接口。DBMS\_JOB包对于任务队列提供了下面这些功能：提交并且执行一个任务、改变任务的执行参数以及删除或者临时挂起任务等。

DBMS\_JOB包是由ORACLE\_HOME目录下的rdbms/admin子目录下的DBMSJOB.SQL和PRVTJOB.PLB 这两个脚本文件创建的。这两个文件被CATPROC.SQL脚本文件调用，而CATPROC.SQL这个文件一般是在数据库创建后立即执行的。脚本为DBMS\_JOB包创建了一个公共同义词，并给该包授予了公共的可执行权限，所以所有的Oracle用户均可以使用这个包。

下面几个数据字典视图是关于任务队列信息的，主要有DBA\_JOBS, USER\_JOBS和DBA\_JOBS\_RUNNING。这些字典视图是由名为CATJOBQ.SQL的脚本文件创建的。该脚本文件和创建DBMS\_JOB包的脚本文件一样在ORACLE\_HOME目录的rdbms/admin子目录中，同样也是由脚本文件CATPROC.SQL调用。

最后，要使任务队列能正常运行，还必须启动它自己专有的后台过程。启动后台过程是通过在初始化文件init\*.ora(实例不同，初始化文件名也略有不同)中设置初始化参数来进行的。下面就是该参数：

JOB\_QUEUE\_PROCESSES = n

其中，n可以是0到36之间的任何一个数。除了该参数以外，还有几个关于任务队列的初始化参数，本文后面将会对其进行详细讨论。

DBMS\_JOB包中包含有许多过程，见表1所示。

表1 DBMS\_JOB包

|  |  |  |
| --- | --- | --- |
| **名称** | **类型** | **描述** |
| DBMS\_JOB.ISUBMIT | 过程 | 提交一个新任务，用户指定一个任务号 |
| DBMS\_JOB.SUBMIT | 过程 | 提交一个新任务，系统指定一个任务号 |
| DBMS\_JOB.REMOVE | 过程 | 从队列中删除一个已经存在的任务 |
| DBMS\_JOB.CHANGE | 过程 | 更改用户设定的任务参数 |
| DBMS\_JOB.WHAT | 过程 | 更改PL/SQL任务定义 |
| DBMS\_JOB.NEXT\_DATE | 过程 | 更改任务下一次运行时间 |
| DBMS\_JOB.INTERVAL | 过程 | 更改任务运行的时间间隔 |
| DBMS\_JOB.BROKEN | 过程 | 将任务挂起，不让其重复运行 |
| DBMS\_JOB.RUN | 过程 | 在当前会话中立即执行任务 |
| DBMS\_JOB.USER\_EXPORT | 过程 | 创建文字字符串，用于重新创建一个任务 |

**三、DBMS\_JOB包参数**

DBMS\_JOB包中所有的过程都有一组相同的公共参数，用于定义任务，任务的运行时间以及任务定时运行的时间间隔。这些公共任务定义参数见表2所示。

表2 DBMS\_JOB过程的公共参数

|  |  |  |
| --- | --- | --- |
| **名称** | **类型** | **注释** |
| Job | BINARY\_INTEGER | 任务的唯一识别号 |
| What | VARCHAR2 | 作为任务执行的PL/SQL代码 |
| Next\_date | VARCHAR2 | 任务下一次运行的时间 |
| Interval | VARCHAR2 | 日期表达式，用来计算下一次任务运行的时间 |

下面我们来详细讨论这些参数的意义及用法。

**1、job**

参数job是一个整数，用来唯一地标示一个任务。该参数既可由用户指定也可由系统自动赋予，这完全取决于提交任务时选用了那一个任务提交过程。DBMS\_JOB.SUBMIT过程通过获得序列SYS.JOBSEQ的下一个值来自动赋予一个任务号。该任务号是作为一个OUT参数返回的，所以调用者随后可以识别出提交的任务。而DBMS\_JOB.ISUBMIT过程则由调用者给任务指定一个识别号，这时候，任务号的唯一性就完全取决于调用者了。

除了删除或者重新提交任务，一般来说任务号是不能改变的。即使当数据库被导出或者被导入这样极端的情况，任务号也将被保留下来。所以在执行含有任务的数据的导入/导出操作时很可能会发生任务号冲突的现象。

**2、what**

what参数是一个可以转化为合法PL/SQL调用的字符串，该调用将被任务队列自动执行。在what参数中，如果使用文字字符串，则该字符串必须用单引号括起来。 what参数也可以使用包含我们所需要字符串值的VARCHAR2变量。实际的PL/SQL调用必须用分号隔开。在PL/SQL调用中如果要嵌入文字字符串，则必须使用两个单引号。

what参数的长度在Oracle7.3中限制在2000个字节以内，在Oracle 8.0以后，扩大到了4000个字节，这对于一般的应用已完全足够。该参数的值一般情况下都是对一个PL/SQL存储过程的调用。在实际应用中，尽管可以使用大匿名Pl/SQL块，但建议大家最好不要这样使用。还有一个实际经验就是最好将存储过程调用封装在一个匿名块中，这样可以避免一些比较莫名错误的产生。我来举一个例子，一般情况下，what参数可以这样引用：

|  |
| --- |
| what =>’my\_procedure(parameter1);’ |

但是比较安全的引用，应该这样写：

|  |
| --- |
| what =>’begin my\_procedure(parameter1); end;’ |

任何时候，我们只要通过更改what参数就可以达到更改任务定义的目的。但是有一点需要注意，通过改变what参数来改变任务定义时，用户当前的会话设置也被记录下来并成为任务运行环境的一部分。如果当前会话设置和最初提交任务时的会话设置不同，就有可能改变任务的运行行为。意识到这个潜在的副作用是非常重要的，无论何时只要应用到任何DBMS\_JOB过程中的what参数时就一定要确保会话设置的正确。

**3、next\_date**

Next\_date参数是用来调度任务队列中该任务下一次运行的时间。这个参数对于DBMS\_JOB.SUBMIT和DBMS\_JOB.BROKEN这两个过程确省为系统当前时间，也就是说任务将立即运行。

当将一个任务的next\_date参数赋值为null时，则该任务下一次运行的时间将被指定为4000年1月1日，也就是说该任务将永远不再运行。在大多数情况下，这可能是我们不愿意看到的情形。但是，换一个角度来考虑，如果想在任务队列中保留该任务而又不想让其运行，将next\_date设置为null却是一个非常简单的办法。

Next\_date也可以设置为过去的一个时间。这里要注意，系统任务的执行顺序是根据它们下一次的执行时间来确定的，于是将next\_date参数设置回去就可以达到将该任务排在任务队列前面的目的。这在任务队列进程不能跟上将要执行的任务并且一个特定的任务需要尽快执行时是非常有用的。

**4、Interval**

Internal参数是一个表示Oracle合法日期表达式的字符串。这个日期字符串的值在每次任务被执行时算出，算出的日期表达式有两种可能，要么是未来的一个时间要么就是null。这里要强调一点：很多开发者都没有意识到next\_date是在一个任务开始时算出的，而不是在任务成功完成时算出的。

当任务成功完成时，系统通过更新任务队列目录表将前面算出的next\_date值置为下一次任务要运行的时间。当由interval表达式算出next\_date是null时，任务自动从任务队列中移出，不会再继续执行。因此，如果传递一个null值给interval参数，则该任务仅仅执行一次。

通过给interval参数赋各种不同的值，可以设计出复杂运行时间计划的任务。本文后面的“任务间隔和日期算法”将对interval表达式进行详细讨论，并给出一个实际有用interval表达式的例子。

**四、任务队列架构和运行环境**

任务队列在Oracle系统中其实是一个子系统，它具有自己特定的后台过程和目录表。该子系统设计的目的是为了能不在用户干预下自动运行PL/SQL过程。

**1、任务队列后台过程**

任务队列（SNP）后台过程随着Oracle实例的启动而同时启动。在文章前面已经谈到初始化文件init.ora中的参数JOB\_QUEUE\_PROCESSES，用来设置有几个队列过程。这里设置了几个过程，系统中就会有几个SNP过程被启动。JOB\_QUEUE\_PROCESSES这个参数，可以是0到36中的任何一个数，也就是说对于每个Oracle实例最多可以有36个SNP过程，也可以不支持队列过程（＝0）。在大多数操作系统中，SNP三个字母常作为过程名的一部分出现。如，在unix系统中，如果该Oracle实例名为ora8，有三个任务队列过程，则这三个任务队列过程名称为：

|  |
| --- |
| ora\_ora8\_snp0 ora\_ora8\_snp1 ora\_ora8\_snp2 |

SNP后台过程和其他的Oracle后台过程的一个重要区别就是杀掉一个SNP过程不会影响到Oracle实例。当一个任务队列过程失控或者消耗太多的资源时，就可以将其杀掉，当然这种情况不是经常遇到的。当一个SNP过程被杀掉或者失败时，Oracle就自动启动一个新的SNP过程来代替它。

**2、有关任务队列的初始化参数**

初始化文件init.ora中的几个参数控制着任务队列后台的运行，下面我们将对其进行详细讨论。

**（1）、JOB\_QUEUE\_INTERVAL**

任务队列过程定期唤醒并检查任务队列目录表是否有任务需要执行。参数JOB\_QUEUE\_INTERVAL决定SNP过程两次检查目录表之间“休眠”多长时间（单位为秒）。间隔设的太小会造成由于SNP过程不断检查目录表而导致不必要的系统吞吐量。相反如果间隔设得太大，SNP过程在特定的时间没有被唤醒，那个时间的任务就不会能被运行。最佳的时间间隔设置要综合考虑系统环境中不同的任务，60秒的确省设置可以满足大多数的应用。

**（2）、JOB\_QUEUE\_KEEP\_CONNECTIONS**

除了前面介绍的JOB\_QUEUE\_PROCESS和JOB\_QUEUE\_INTERVAL两个参数以外，影响SNP后台过程行为的第三个参数是JOB\_QUEUE\_KEEP\_CONNECTIONS。当该参数为TRUE时，SNP过程在两个任务的运行期间（也就是休眠期间），仍然和Oracle保持开放的连接。相反，如果为FALSE时，SNP过程将和数据库断开连接，当唤醒时刻到来时又重新连接并检查任务队列。

选择这两种方法中的那一种，主要是考虑任务队列的有效性和数据库关闭方法。长期保持连接的效率比较高，但任务队列会受到正常关闭数据库的影响。这是因为任务队列过程对于服务器管理器看来和一个普通用户的过程没有什么不同，而正常的关闭数据库需要让所有的用户都断开连接。而断开连接和重新连接又给数据库增加了负荷，但是可定期地使数据库没有可连接SNP过程，也就可以使数据库正常关闭。对于有很多任务或者是任务重复执行的时间间隔较短（一个小时或者更少）的环境，一般将JOB\_QUEUE\_KEEP\_CONNECTIOONS设置为TRUE，并修改关闭数据库的脚本为立即关闭。对于严格要求采用正常方式关闭的数据库或者是任务较少，重复间隔较长的环境，一般将该参数设置为FALSE。最好，要提醒一句，SNP过程仅在没有任何任务运行时才断开，这种情况下，那些需要比较长时间运行的任务SNP将在它们的生命周期内一致保持开放的连接，这就延迟了正常关闭数据库的时间。

**3、建立运行环境**

当SNP过程唤醒时，它首先查看任务队列目录中所有的任务是否当前的时间超过了下一次运行的日期时间。SNP检测到需要该时间立即执行的任务后，这些任务按照下一次执行日期的顺序依次执行。当SNP过程开始执行一个任务时，其过程如下：

1. 以任务所有者的用户名开始一个新的数据库会话。
2. 当任务第一次提交或是最后一次被修改时，更改会话NLS设置和目前就绪的任务相匹配。
3. 通过interval日期表达式和系统时间，计算下一次执行时间。
4. 执行任务定义的PL/SQL
5. 如果运行成功，任务的下一次执行日期（next\_date）被更新，否则，失败计数加1。
6. 经过JOB\_QUEUS\_INTERVAL秒后，又到了另一个任务的运行时间，重复上面的过程。

在前两步中，SNP过程创建了一个模仿用户运行任务定义的PL/SQL的会话环境。然而，这个模仿的运行环境并不是和用户实际会话环境完全一样，需要注意以下两点：第一，在任务提交时任何可用的非确省角色都将在任务运行环境中不可用。因此，那些想从非确省角色中取得权限的任务不能提交，用户确省角色的修改可以通过在任务未来运行期间动态修改来完成。第二，任何任务定义本身或者过程执行中需要的数据库联接都必须完全满足远程的用户名和密码。SNP过程不能在没有显式指明口令的情况下初始化一个远程会话。显然，SNP过程不能假定将本地用户的口令作为远程运行环境会话设置的一部分。

提交的任务如果运行失败会怎么样呢？当任务运行失败时，SNP过程在1分钟后将再次试图运行该任务。如果这次运行又失败了，下一次尝试将在2分钟后进行，再下一次在4分钟以后。任务队列每次加倍重试间隔直到它超过了正常的运行间隔。在连续16次失败后，任务就被标记为中断的（broken），如果没有用户干预，任务队列将不再重复执行。

**五、任务队列字典表和视图**

任务队列中的任务信息可以通过表3所示的几个字典视图来查看，这些视图是由CATJOBQ.sql脚本创建的。表4和5是各个视图每个字段的含义。

表3. 任务队列中关于任务的数据字典视图

|  |  |
| --- | --- |
| **视图名** | **描述** |
| DBA\_JOBS | 本数据库中定义到任务队列中的任务 |
| DBA\_JOBS\_RUNNING | 目前正在运行的任务 |
| USER\_JOBS | 当前用户拥有的任务 |

表4. DBA\_JOBS 和 USER\_JOBS.字典视图的字段含义

|  |  |  |
| --- | --- | --- |
| **字段（列）** | **类型** | **描述** |
| JOB | NUMBER | 任务的唯一标示号 |
| LOG\_USER | VARCHAR2(30) | 提交任务的用户 |
| PRIV\_USER | VARCHAR2(30) | 赋予任务权限的用户 |
| SCHEMA\_USER | VARCHAR2(30) | 对任务作语法分析的用户模式 |
| LAST\_DATE | DATE | 最后一次成功运行任务的时间 |
| LAST\_SEC | VARCHAR2(8) | 如HH24:MM:SS格式的last\_date日期的小时，分钟和秒 |
| THIS\_DATE | DATE | 正在运行任务的开始时间，如果没有运行任务则为null |
| THIS\_SEC | VARCHAR2(8) | 如HH24:MM:SS格式的this\_date日期的小时，分钟和秒 |
| NEXT\_DATE | DATE | 下一次定时运行任务的时间 |
| NEXT\_SEC | VARCHAR2(8) | 如HH24:MM:SS格式的next\_date日期的小时，分钟和秒 |
| TOTAL\_TIME | NUMBER | 该任务运行所需要的总时间，单位为秒 |
| BROKEN | VARCHAR2(1) | 标志参数，Y标示任务中断，以后不会运行 |
| INTERVAL | VARCHAR2(200) | 用于计算下一运行时间的表达式 |
| FAILURES | NUMBER | 任务运行连续没有成功的次数 |
| WHAT | VARCHAR2(2000) | 执行任务的PL/SQL块 |
| CURRENT\_SESSION\_LABEL | RAW MLSLABEL | 该任务的信任Oracle会话符 |
| CLEARANCE\_HI | RAW MLSLABEL | 该任务可信任的Oracle最大间隙 |
| CLEARANCE\_LO | RAW MLSLABEL | 该任务可信任的Oracle最小间隙 |
| NLS\_ENV | VARCHAR2(2000) | 任务运行的NLS会话设置 |
| MISC\_ENV | RAW(32) | 任务运行的其他一些会话参数 |

表 5. 视图DBA\_JOBS\_RUNNING的字段含义

|  |  |  |
| --- | --- | --- |
| **列** | **数据类型** | **描述** |
| SID | NUMBER | 目前正在运行任务的会话ID |
| JOB | NUMBER | 任务的唯一标示符 |
| FAILURES | NUMBER | 连续不成功执行的累计次数 |
| LAST\_DATE | DATE | 最后一次成功执行的日期 |
| LAST\_SEC | VARCHAR2(8) | 如HH24:MM:SS格式的last\_date日期的小时，分钟和秒 |
| THIS\_DATE | DATE | 目前正在运行任务的开始日期 |
| THIS\_SEC | VARCHAR2(8) | 如HH24:MM:SS格式的this\_date日期的小时，分钟和秒 |

**六、任务重复运行间隔和间隔设计算法**

任务重复运行的时间间隔取决于interval参数中设置的日期表达式。下面就来详细谈谈该如何设置interval参数才能准确满足我们的任务需求。一般来讲，对于一个任务的定时执行，有三种定时要求。

1. 在一个特定的时间间隔后，重复运行该任务。
2. 在特定的日期和时间运行任务。
3. 任务成功完成后，下一次执行应该在一个特定的时间间隔之后。

第一种调度任务需求的日期算法比较简单，即'SYSDATE+n',这里n是一个以天为单位的时间间隔。表6给出了一些这种时间间隔设置的例子。

表6 一些简单的interval参数设置例子

|  |  |
| --- | --- |
| **描述** | **Interval参数值** |
| 每天运行一次 | 'SYSDATE + 1' |
| 每小时运行一次 | 'SYSDATE + 1/24' |
| 每10分钟运行一次 | 'SYSDATE + 10/（60\*24）' |
| 每30秒运行一次 | 'SYSDATE + 30/(60\*24\*60)' |
| 每隔一星期运行一次 | 'SYSDATE + 7' |
| 不再运行该任务并删除它 | NULL |

表6所示的任务间隔表达式不能保证任务的下一次运行时间在一个特定的日期或者时间，仅仅能够指定一个任务两次运行之间的时间间隔。例如，如果一个任务第一次运行是在凌晨12点，interval指定为'SYSDATE + 1',则该任务将被计划在第二天的凌晨12点执行。但是，如果某用户在下午4点手工（DBMS\_JOB.RUN）执行了该任务，那么该任务将被重新定时到第二天的下午4点。还有一个可能的原因是如果数据库关闭或者说任务队列非常的忙以至于任务不能在计划的那个时间点准时执行。在这种情况下，任务将试图尽快运行，也就是说只要数据库一打开或者是任务队列不忙就开始执行，但是这时，运行时间已经从原来的提交时间漂移到了后来真正的运行时间。这种下一次运行时间的不断“漂移”是采用简单时间间隔表达式的典型特征。

第二种调度任务需求相对于第一种就需要更复杂的时间间隔（interval）表达式，表7是一些要求在特定的时间运行任务的interval设置例子。

表 7. 定时到特定日期或时间的任务例子

|  |  |
| --- | --- |
| **描述** | **INTERVAL参数值** |
| 每天午夜12点 | 'TRUNC(SYSDATE + 1)' |
| 每天早上8点30分 | 'TRUNC(SYSDATE + 1) + （8\*60+30）/(24\*60)' |
| 每星期二中午12点 | 'NEXT\_DAY(TRUNC(SYSDATE ), ''TUESDAY'' ) + 12/24' |
| 每个月第一天的午夜12点 | 'TRUNC(LAST\_DAY(SYSDATE ) + 1)' |
| 每个季度最后一天的晚上11点 | 'TRUNC(ADD\_MONTHS(SYSDATE + 2/24, 3 ), 'Q' ) -1/24' |
| 每星期六和日早上6点10分 | 'TRUNC(LEAST(NEXT\_DAY(SYSDATE, ''SATURDAY"), NEXT\_DAY(SYSDATE, "SUNDAY"))) + （6×60+10）/（24×60）' |

第三种调度任务需求无论通过怎样设置interval日期表达式也不能满足要求。这时因为一个任务的下一次运行时间在任务开始时才计算，而在此时是不知道任务在何时结束的。遇到这种情况怎么办呢？当然办法肯定是有的，我们可以通过为任务队列写过程的办法来实现。这里我只是简单介绍以下，可以在前一个任务队列执行的过程中，取得任务完成的系统时间，然后加上指定的时间间隔，拿这个时间来控制下一个要执行的任务。这里有一个前提条件，就是目前运行的任务本身必须要严格遵守自己的时间计划。

......目前，流行的主流数据库都拥有此项功能，最具代表性的是Microsoft SQL Server 7.0、Oracle8i/9i等。但是，要让Job工作，还需要我们加以配置才能实现。这些配置都有GUI操作。本文介绍Oracle9i下通过命令行实现Job配置......

众所周知，一般操作系统会提供定时执行任务的方法，例如：Unix平台上提供了让系统定时执行任务的命令Crontab。但是，对于某些需求，例如：一些对数据库表的操作，最为典型的是证券交易所每日收盘后的结算，它涉及大量的数据库表操作，如果仍然利用操作系统去定时执行，不仅需要大量的编程工作，而且还会出现用户不一致等运行错误，甚至导致程序无法执行。   
事实上，对于以上需求，我们可以利用数据库本身拥有的功能Job Queue（任务队列管理器）去实现。Job允许用户提前调度和安排某一任务，使其能在指定的时间点或时间段内自动执行一次或多次，由于任务在数据库中被执行，所以执行效率很高。   
Job允许我们定制任务的执行时间，并提供了灵活的处理方式，还可以通过配置，安排任务在系统用户访问量少的时段内执行，极大地提高了工作效率。例如，对于数据库日常的备份、更新、删除和复制等耗时长、重复性强的工作，以及电信增值短信业务中的定时PUSH，我们就可以利用Job去自动执行以减少工作量。  
目前，流行的主流数据库都拥有此项功能，最具代表性的是Microsoft SQL Server 7.0、Oracle8i/9i等。但是，要让Job工作，还需要我们加以配置才能实现。这些配置都有GUI操作。本文介绍Oracle9i下通过命令行实现Job配置。  
前提：写好的要定时执行的存储过程[不能带参数]。  
定义一个Job，执行间隔是需要注意的一件重要的事情。SYSDATE+1/24 是存储在dba\_jobs 视图中的间隔，它可以产生没小时一次的快照。可以将这个数据改变为不同的采样时间，在一天中有 24\*60 = 1440 分钟，可以使用这个数字调整执行次数。比方说：我希望在每10分钟获取一次快照，应该使用下列命令：  
execute dbms\_job.submit(  
    :jobno,      --作业编号  
    ‘sp;', --执行的过程  
    trunc(sysdate+10/1440,‘MI'),    --下次执行时间  
    ‘trunc(sysdate+10/1440,‘‘MI'')',    --间隔时间  
    true,        --no\_parse  
    :instno);  
1.创建JOB  
创建一个任务，执行间隔是每5分钟。  
    Variable v\_sn number;  
    Begin  
        Dbms\_job.submit(:v\_sn,  
            ‘p\_push\_send;',  
            trunc(sysdate+5/1440,‘MI'),  
            ‘trunc(sysdate+5/1440,‘‘MI'')');  
        commit;  
    end;  
    /  
删除一个任务：  
execute dbms\_job.remove(jobno);  
2.查询任务语句  
涉及两个表：user\_jobs及dba\_jobs\_running[视图dba\_jobs]  
select \* from user\_jobs;  
select \* from dba\_jobs\_running;  
select \* from dba\_jobs;  
3.必要的参数[修改initsid.ora参数]  
job\_queue\_processes = 4              --可执行作业个数  
job\_queue\_interval = 10              --默认间隔延迟时间10s  
job\_queue\_keep\_connections=true       --job保持正常连接  
修改可执行作业个数为20个：  
ALTER SYSTEM SET JOB\_QUEUE\_PROCESSES = 2;0  
修改取消限制模式：  
ALTER SYSTEM DISABLE RESTRICTED SESSION;  
4.相关的几个Job操作  
删除job:dbms\_job.remove(jobno);  
修改要执行的操作:dbms\_job.what(jobno,what);  
修改下次执行时间：dbms\_job.next\_date(job,next\_date);  
修改间隔时间：dbms\_job.interval(job,interval);  
停止job:dbms.broken(job,broken,nextdate);  
启动job:dbms\_job.run(jobno);  
5. 关于Job 的延迟  
Job都有不同程度的延迟，想完全排除这种误差最好使用操作系统的定时器crontab或者at，哈哈，开个玩笑。  
A时间重叠的问题[网友论]：  
比如我有两个JOB，都是在凌晨3点运行，那么如果其中一个在3点运行，那么另一个必须等待第一个JOB完成，然后才能执行。特别是有些象sysdate+1/24，这样每格一小时运行的JOB就更容易冲突；即使没有时间重叠，ORACLE也是按job\_queue\_internal（通常是1分钟）的间隔进行检查JOB队列，这样1点钟的作业正好在3：00：45才检查到，那么就会在3：00：45才执行该job.我们知道ORACLE JOB如果第一次执行失败，那么按一定的时间间隔再次启动该JOB直到成功，如果直到运行16次还是失败，那么就中断该JOB，所以实际运行的时间会进行推迟。  
B采用"精确定时"函数（从前面可能看到，其实是很难实现精确定时执行JOB的）  
我们可以采用如trunc(sysdate)+(1+24)/25或trunc(sysdate)+25/24。表示每天1点执行job，这样就不会受上次JOB延时的影响。（没测过）。

**oracle中的定时任务dbms\_job包的使用**

SVRMGR> select \* from dba\_jobs;

初始化相关参数job\_queue\_processes  
alter system set job\_queue\_processes=39 scope=spfile;//最大值不能超过1000  ;job\_queue\_interval = 10  //调度作业刷新频率秒为单位  
DBA\_JOBS describes all jobs in the database.   
USER\_JOBS describes all jobs owned by the current user

1  select job,what,to\_char(last\_date,'yyyy-mm-dd HH24:mi:ss'),to\_char(next\_date,'yyyy-mm-dd HH24:m),interval from dba\_jobs where job in (325,295)  
2 select job,what,last\_date,next\_date,interval from  dba\_jobs where job in (1,3);

查询job的情况。  
show paramter background\_dump\_dest.  
看alter.log 和trace

SVRMGR> select \* from dba\_jobs;

初始化相关参数job\_queue\_processes  
alter system set job\_queue\_processes=39 scope=spfile;//最大值不能超过1000

job\_queue\_interval = 10  //调度作业刷新频率秒为单位

DBA\_JOBS describes all jobs in the database.   
USER\_JOBS describes all jobs owned by the current user

1  select job,what,to\_char(last\_date,'yyyy-mm-dd HH24:mi:ss'),to\_char(next\_date,'yyyy-mm-dd HH24:m),interval from dba\_jobs where job in (325,295)  
2 select job,what,last\_date,next\_date,interval from  dba\_jobs where job in (1,3);

查询job的情况。  
show paramter background\_dump\_dest.  
看alter.log 和trace

请问我如何停止一个JOB  
SQL> exec dbms\_job.broken(1,true)

PL/SQL 过程已成功完成。  
SQL>commit  //必须提交否则无效

启动作业  
SQL> exec dbms\_job.broken(1,false)

PL/SQL 过程已成功完成。

停其他用户的job  
SQL>exec sys.dbms\_ijob.broken(98,true);  
SQL>commit;

============================

exec dbms\_job.broken(:job) 停止  
exec dbms\_job.broken(186,true) //标记位broken  
exec dbms\_job.broken(186,false)//标记为非broken  
exec dbms\_job.broken(186,false,next\_day(sysdate,'monday')) //标记为非broken,指定执行时间  
exec dbms\_job.remove(:job)；删除  
exec dbms\_job.remove(186);  
commit；

把一个broken job重新运行

三、查看相关job信息  
1、相关视图  
dba\_jobs  
all\_jobs  
user\_jobs  
dba\_jobs\_running 包含正在运行job相关信息

创建JOB  
variable jobno number;  
begin  
  dbms\_job.submit(:jobno, 'statspack.snap;', trunc(sysdate+1/24,'HH'), 'trunc(SYSDATE+1/24,''HH'')', TRUE, :instno);  
  commit;  
end;  
print jobno

例如，我们已经建立了一个存储过程，其名称为my\_job，在sql/plus中以scott用户身份登录，执行如下命令：   
sql> variable n number;   
sql> begin   
dbms\_job.submit(:n‘my\_job;’,sysdate,   
‘sysdate＋1/360’);   
commit;   
end;  
Sql> print :n;

系统提示执行成功。   
Sql> print :n;   
系统打印此任务的编号，例如结果为300。

简单例子  
一个简单例子：

创建测试表  
SQL> create table TEST(a date);

表已创建。

创建一个自定义过程  
SQL> create or replace procedure MYPROC as  
  2  begin  
  3  insert into TEST values(sysdate);  
  4  end;  
  5  /

过程已创建。

创建JOB  
SQL> variable job1 number;  
SQL>   
SQL> begin  
  2  dbms\_job.submit(:job1,'MYPROC;',sysdate,'sysdate+1/1440');　　--每天1440分钟，即一分钟运行test过程一次  
  3  end;  
  4  /

PL/SQL 过程已成功完成。

运行JOB  
SQL> begin  
  2  dbms\_job.run(:job1);  
  3  end;  
  4  /

PL/SQL 过程已成功完成。

SQL> select to\_char(a,'yyyy/mm/dd hh24:mi:ss') 时间 from TEST;

时间  
-------------------  
2001/01/07 23:51:21  
2001/01/07 23:52:22  
2001/01/07 23:53:24

删除JOB  
SQL> begin  
  2  dbms\_job.remove(:job1);  
  3  end;  
  4  /

PL/SQL 过程已成功完成。

＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝

a、利用dbms\_job.run()立即执行该job  
   sql>begin  
   sql>dbms\_job.run(:jobno) 该jobno为submit过程提交时返回的job number  
   sql>end;  
   sql>/  
b、利用dbms\_job.broken()重新将broken标记为false  
   sql>begin  
   sql>dbms\_job.broken (:job,false,next\_date)  
   sql>end;  
   sql>/

＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝  
SQL> create table a(a date);  
Table created

创建一个过程  
SQL> create or replace procedure test as  
  2  begin  
  3  insert into a values(sysdate);  
  4  end;  
  5  /  
Procedure created

提交作业  
SQL> declare  
  2  job1 number;   //定义一个数字型变量  
  3  begin  
  4  dbms\_job.submit(:job1,'test;',sysdate,'sysdate+1/1440');  //按分钟算一天1440分钟  
  5  end;  
  6  /  
PL/SQL procedure successfully completed  
job1  
---------  
4  
SQL> commit;  
Commit complete

运行作业  
SQL> begin  
  2  dbms\_job.run(4);  
  3  end;  
  4  /  
PL/SQL procedure successfully completed

删除作业  
SQL> begin  
  2  dbms\_job.remove(4);  
  3  end;  
  4  /  
PL/SQL procedure successfully completed  
SQL> commit;  
Commit complete

job change//修改作业  
execute dbms\_job.change(186,null,null,'sysdate+3');  
execute dbms\_job.change(186,'scott.test(update)');

DBA\_JOBS  
＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝  
字段（列）          类型                 描述   
JOB                NUMBER          任务的唯一标示号   
LOG\_USER           VARCHAR2(30)    提交任务的用户   
PRIV\_USER          VARCHAR2(30)    赋予任务权限的用户   
SCHEMA\_USER        VARCHAR2(30)    对任务作语法分析的用户模式   
LAST\_DATE          DATE            最后一次成功运行任务的时间   
LAST\_SEC           VARCHAR2(8)     如HH24:MM:SS格式的last\_date日期的小时，分钟和秒   
THIS\_DATE     DATE            正在运行任务的开始时间，如果没有运行任务则为null   
THIS\_SEC     VARCHAR2(8)     如HH24:MM:SS格式的this\_date日期的小时，分钟和秒   
NEXT\_DATE          DATE            下一次定时运行任务的时间   
NEXT\_SEC           VARCHAR2(8)     如HH24:MM:SS格式的next\_date日期的小时，分钟和秒   
TOTAL\_TIME         NUMBER          该任务运行所需要的总时间，单位为秒   
BROKEN             VARCHAR2(1)     标志参数，Y标示任务中断，以后不会运行   
INTERVAL           VARCHAR2(200)   用于计算下一运行时间的表达式   
FAILURES    NUMBER     任务运行连续没有成功的次数   
WHAT               VARCHAR2(2000)  执行任务的PL/SQL块   
CURRENT\_SESSION\_LABEL RAW          MLSLABEL 该任务的信任Oracle会话符   
CLEARANCE\_HI      RAW MLSLABEL     该任务可信任的Oracle最大间隙   
CLEARANCE\_LO      RAW              MLSLABEL 该任务可信任的Oracle最小间隙   
NLS\_ENV           VARCHAR2(2000)   任务运行的NLS会话设置   
MISC\_ENV          RAW(32)          任务运行的其他一些会话参数

描述                    INTERVAL参数值   
每天午夜12点            'TRUNC(SYSDATE + 1)'   
每天早上8点30分         'TRUNC(SYSDATE + 1) + （8\*60+30）/(24\*60)'   
每星期二中午12点         'NEXT\_DAY(TRUNC(SYSDATE ), ''TUESDAY'' ) + 12/24'   
每个月第一天的午夜12点    'TRUNC(LAST\_DAY(SYSDATE ) + 1)'   
每个季度最后一天的晚上11点 'TRUNC(ADD\_MONTHS(SYSDATE + 2/24, 3 ), 'Q' ) -1/24'   
每星期六和日早上6点10分    'TRUNC(LEAST(NEXT\_DAY(SYSDATE, ''SATURDAY"), NEXT\_DAY(SYSDATE, "SUNDAY"))) + （6×60+10）/（24×60）'

 要建一个触发器，在每天15：30的时候把wzcs.ckjl的表里面的cksj(出库时间)字段为当天的记录插入到rjck.rkjl里面！   
declare     
                jobid     number;     
                v\_sql     varchar2(2000);     
    begin     
                v\_sql:='begin     
                                      if     to\_char(sysdate,''HH24:MI'')=''15:30''     then     
                                            insert     into     rjck.rkjl(cksj)     select     cksj     from     wzcs.ckjl;     
                                            dbms\_output.put\_line(''inserted     success'');     
                                      end     if;     
                                      commit;     
                                exception     
                                      when     others     then     
                                      rollback;     
                                      dbms\_output.put\_line(SQLERRM);     
                                end;     
                              ';     
                dbms\_job.submit(jobid,v\_sql,sysdate,'sysdate+1/1440');     
                dbms\_job.run(jobid);     
                dbms\_output.put\_line('job     '||to\_char(jobid)||'     is     running');     
    end;