第7章 数据库作业和文件输入输出

本章将讨论PL/SQL内置包DBMS_JOB和UTL_FILE的特性。PL/SQL2.2以上版本提供的包DBMS_JOB支持存储过程在系统的管理下周期性自动运行而无须用户的介入。而 PL/SQL2.3以上版本提供的包UTL_FILE则扩充了读写系统文件的功能。这两个包提供的功能使 PL/SQL具备了与其他第三代程序设计语言相同的处理能力。

7.1 数据库作业

PL/SQL 2.2及更高版本

在PL/SQL2.2及更高版本下，我们可以指定PL/SQL程序在指定时间运行，支持这种定时运行功能的是包DBMS_JOB提供的作业序列。Oracle中作业的运行是通过将该作业和说明该作业运行运行方式的参数共同提交给作业序列实现的。有关当前运行的作业，上一个提交的作业的成功或失败状态等信息可以在数据字典中找到（请参阅 7.1.4节）。

Oracle 8及更高版本

读者需要注意的是，Oracle8及更高版本支持的高级查询功能提供了比包 DBMS_JOB功能更强大的PL/SQL查询功能。有关该功能的细节，请参考 Oracle文档资料。

7.1.1 后台进程

一个Oracle实例是由运行在系统上的各种进程所组成，不同的进程负责实现不同的数据库操作，如把数据库记录读入内存，把数据库记录写回磁盘，以及把数据归档到脱机存储器中。除了管理数据库的处理外，数据库系统还具有叫做 SNP的进程。这些后台进程除了要处理快照（Snapshot）的自动刷新外，还要通过DBMS_JOB来管理访问作业队列的访问通道。

象其他的数据库进程运行方式一样，SNP进程也运行在后台。然而，与数据库进程不同的是，如果SNP进程失败的话，Oracle就将重新启动该进程并不影响数据库的其他进程。如果其他的数据库进程失败的话，这些失败的进程将会使数据库停止运行。SNP周期性地激活来检查作业序列。如果论到某个作业运行，则SNP进程就在启动该进程运行后再进入睡眠状态。一个给定的进程只能一次运行一个作业。在 Oracle7系统下，系统限定的最大 SNP进程的数目是 10个（从SNP0-SNP9），因此，系统中可同时运行的最大作业的数目也是 10个。在Oracle8系统中，SNP进程最大数目增加到了36个，其进程号是SNP0-SNP9,SNPA-SNPZ。

数据库初始化文件（ init.ora）中有两个参数用来控制 SNP进程的属性。这两个参数是JOB_QUEUE_PROCESSES和JOB_QUEUE_INTERVAL，下面的表7-1是该参数的说明。值得注意的是，如果将JOB_QUEUE_PROCESSES设置为0的话，系统将禁止作业运行。由于每个进程都将在查询新的作业之前按 JOB_QUEUE_INTERVAL指定的时间（以秒为单位）进行睡眠，所以参数JOB_QUEUE_INTERVAL就指定了运行两个作业之间的最小时间间隔。上述两个参数都不能使用ALTER SYSTEM或ALTER SESSION进行动态修改，因此用户必须先对数据库初始化

文件进行修改并重启数据库才能使修改的参数生效。

注意 Oracle7.3以及更高版本中已经不在使用 Oracle7.2版使用的初始化参数 JOB_QUEUE_KEEP_CONNECTIONS。从Oracle7.3版开始数据库的物理连接完全由系统自动实施控制。

表7-1 作业初始化参数

参 数	默 认 值	范 围	说 明
JOB_QUEUE_PROCESSES	0	0～10(在Oracle8中为0～36)	可同时运行的进程数目
JOB_QUEUE_INTERVAL	60	1～3600s	两个进程间唤醒的间隔。进程按指定的时间进行睡眠

7.1.2 运行作业

运行作业的方法有两种，一种是将作业提交给作业队列，另一种是强制作业立即运行。当作业被提交给作业队列时，SNP就在该作业的启动时刻运行该作业。如果指定了作业的运行间隔的话，该作业将自动地周期运行。如果作业是立即启动的，该作业就运行一次。

1. 提交：SUBMIT

使用下面的SUBMIT过程可以将作业提交到作业队列中。该过程的的语法是：

节选自在线代码TempInsert.sql

```
PROCEDURE SUBMIT( job OUT BINARY_INTEGER,
                  what IN VARCHAR2,
                  next_date IN DATE DEFAULT SYSDATE,
                  interval IN VARCHAR2 DEFAULT NULL,
                  no_parse IN BOOLEAN DEFAULT FALSE);
```

下面说明了SUBMIT语句中使用的参数：

参 数	类 型	说 明
job	BINARY_INTGER	作业号。创建作业时，作业将被赋予一个作业号。只要该作业存在，其作业号将保持不变。作业号在实例的范围内是唯一的
what	VARCHAR2	组成作业的PL/SQL代码。通常，该代码是存储过程的调用
next_date	DATE	作业下一次运行的日期
interval	VARCHAR2	计算作业再次运行时间的函数。该函数的值必须是一个时间值或为空 NULL
no_parse	BOOLEAN	如果该参数为真的话，作业将在其第一次运行时才进行语法分析。如果该参数为假值的话（默认值），则作业在提交时就对其进行语法分析。如果作业引用的数据库对象不存在但又必须提交该作业时，可以将该参数设置为真。（被引用的对象在运行时必须存在。）

Oracle 8i 及更高版本

Oracle8i允许作业运行在 Oracle Parallel Server(OPS)环境下的指定实例中。DBMS_JOB. SUBMIT提供的两个参数——instance和force以实现上述功能。这两个参数的意义7.1.3节的“实例仿射性”中讨论。

例如，假设我们用下面的程序来创建过程 TempInsert:

节选自在线代码TempInsert.sql

```
CREATE SEQUENCE temp_seq
  START WITH 1
  INCREMENT BY 1;

CREATE OR REPLACE PROCEDURE TempInsert AS
BEGIN
  INSERT INTO temp_table (num_col, char_col)
    VALUES (temp_seq.nextval,
            TO_CHAR(SYSDATE, 'DD-MON-YYYY HH24:MI:SS')); COMMIT;
END TempInsert;
```

我们可以使用下面的SQL *Plus脚本指定过程TempInsert每90秒钟运行一次：

节选自在线代码TempInsert.sql

```
SQL> VARIABLE v_JobNum NUMBER
SQL> BEGIN
  2    DBMS_JOB.SUBMIT(:v_JobNum, 'TempInsert;', SYSDATE,
  3                    'sysdate + (90/(24*60*60))');
  4    COMMIT;
  5 END;
  6 /
PL/SQL procedure successfully completed.

SQL> print v_JobNum
 V_JOBNUM
---------
2
```

注意 要启动作业运行必须提交包括调用 DBMS_JOB.SUBMIT语句的事务。其原因是SUBMIT语句通过在数据字典表中插入一行来记录作业信息， SNP进程将查询该表以决定是否有作业需要运行。在这种情况下， INSERT和SELECT语句是由不同的会话实现的，所以必须提交事务。

如果把初始化参数JOB_QUEUE_PROCESSES设置为本0，我们仍然可以发布SUBMIT命令而不会出错。但在这种情况下，作业无法运行。这样一来，为了观察过程 TempInsert对表temp_table的插入操作，我们至少要把 JOB_QUEUE_PROCESSES的值设置为 1。如果JOB_QUEUE_INTERVAL是其默认值（60s），则作业可能不会在90s后开始运行。在运行该作业所需时间之后的90s内，该作业将被标记为就绪运行状态。然而，该作业的真正运行时间是在SNP进程唤醒后才可能开始，（唤醒SNP需要60s时间）。因此，在该作业再次运行前，有可能要等待长达150s。例如，下面是作业运行三次后表Temp_tbale的输出样本：

```
SQL> SELECT * FROM temp_table;
  NUM_COL CHAR_COL
--------- -------------------------
        1 25-APR-1999 18:18:59
        2 25-APR-1999 18:21:02
        3 25-APR-1999 18:23:05
```

在上例中，在作业第一次和第二次运行期间，以及第二次和第三次运行之间的间隔都是123s。

注意 上面SUBMIT调用将启动过程TempInsert周期性地运行直到数据库关闭，或该作业被REMOVE调用删除为止。7.1.3节中的“删除作业”将介绍REMOVE语句。

作业号 当一个作业初次被提交时，该作业就被赋予一个作业号，该作业号是由序列SYS.JOBEQ生成的。一旦给作业赋予了作业号，该作业号将在作业被删除或再次提交前保持不变。

警告 可以象处理数据库对象那样对作业进行导入或导出操作，这种操作不会变更作业的作业号。如果企图导入一个其作业号已经存在的作业的话，系统将给出错误信息，并将禁止将该作业导入。在这种情况下，只须再次提交该作业，就可以生成新的作业号。我们也可以使用过程USER_EXPORT来导出作业。

作业定义 参数what指定了作业的代码。通常，作业由存储过程组成，因此参数what应是调用存储过程的字符串。我们将在本节的后面介绍该字符串的完整格式。参数what调用的过程可以带有任何数量的参数。但所有参数都必须是IN类型参数，这是由于该过程没有实参来接收OUT或IN OUT形参的值。该规则的唯一例外是特殊标识符next_date和broken(将在后面介绍)。

警告 一旦提交了作业，该作业将由后台的SNP进程控制运行。为了了解运行结果，一定要在作业过程末尾写上提交COMMIT代码。如果作业不发布COMMIT命令，当运行该作业的会话结束时，则事务将自动地重新开始。

如表7-2所示，在作业定义中有三个可以合法使用的标识符。其中，参数job是IN类型的参数，因此作业只能读该参数的值。参数next_date和broken都是IN OUT参数，所以，该作业可以对它们进行修改。

表7-2 作业控制标识符

标 示 符	类 型	说 明
job	BINARY_INTEGER	计算当前作业的作业号
next_date	DATE	计算作业下一次运行的日期。如果作业将该参数设置为空的话，则该作业将被从队列中删除
broken	BOOLEAN	计算作业的状态，如果作业被中断，则为真值，否则为假值。如果作业自身将该参数设置为真的话，该作业将被标记为执行中断，但不会从队列中删除

假设我们对TempInsert修改如下：

节选自在线代码TempInsert.sql

```
CREATE OR REPLACE PROCEDURE TempInsert
  (p_NextDate IN OUT DATE) AS
  v_SeqNum   NUMBER;
  v_StartNum NUMBER;
  v_SQLErr VARCHAR2(60);
BEGIN
```

```
  SELECT temp_seq.NEXTVAL
    INTO v_SeqNum
    FROM dual;

  -- See if this is the first time we're called.
BEGIN
  SELECT num_col
    INTO v_StartNum
    FROM temp_table
    WHERE char_col = 'TempInsert Start';

  -- We've been called before, so insert a new value.
  INSERT INTO temp_table (num_col, char_col)
    VALUES (v_SeqNum,
            TO_CHAR(SYSDATE, 'DD-MON-YYYY HH24:MI:SS'));

EXCEPTION
  WHEN NO_DATA_FOUND THEN
    -- First time we're called. First clear out the table.
    DELETE FROM temp_table;

    -- And now insert.
    INSERT INTO temp_table (num_col, char_col)
      VALUES (v_SeqNum, 'TempInsert Start');
END;

-- If we've been called more than 5 times, exit.
IF v_SeqNum - V_StartNum > 5 THEN
   p_NextDate := NULL;
   INSERT INTO temp_table (num_col, char_col)
     VALUES (v_SeqNum, 'TempInsert End');

  END IF;

  COMMIT;
EXCEPTION
  WHEN OTHERS THEN
   -- Record the error in temp_table.
   v_SQLErr := SUBSTR(SQLERRM, 1, 60);
   INSERT INTO temp_table (num_col, char_col)
     VALUES (temp_seq.NEXTVAL, v_SQLErr);
   -- Exit the job.
   p_NextDate := NULL;
   COMMIT;
END TempInsert;
```

提交如下：

节选自在线代码TempInsert1.sql

```
BEGIN
```

```
  DBMS_JOB.SUBMIT(:v_JobNum, 'TempInsert(next_date);', SYSDATE,
                  'SYSDATE + (60/(24*60*60))');
  COMMIT;
END;
```

现在，该作业将每 60秒运行一次，并在被调用五次之后自动地将自己从作业队列中删除（通过把P_NextDate改置为Null）。该作业的输出样本如下：

```
SQL> SELECT * FROM temp_table ORDER BY num_col;
NUM_COL CHAR_COL
--------- ------------------------------------
        1 TempInsert Start
        2 25-APR-1999 18:45:37
        3 25-APR-1999 18:46:38
        4 25-APR-1999 18:47:40
        5 25-APR-1999 18:48:41
        6 25-APR-1999 18:49:43
        7 25-APR-1999 18:50:44
        7 TempInsert End
```

注意 在运行该上面的例子之前，一定要使用 DBMS_JOB.REMOVE来删除前面调用的TempInsert作业。

通过参数next_date(与参数broken一起或单独使用)的返回值，作业可以把自己从作业队列中删除。

参数what是一个VARCHAR2类型的字符串。这样做的结果使得用于调用作业过程中使用的任何字符都必须用两个单引号括起来，而过程调用也必须以分号结束。例如，如果我们使用下面的声明创建一个过程：

```
CREATE OR REPLACE PROCEDURE Test(p_InParameter In VARCHAR2);
```

这时，我们可以使用下面的what字符串来提交该过程：

```
'Test("This is  a character string");'
```

运行间隔 参数next_date指定了作业在提交后的运行时间，该参数的值是在该作业第一次运行时计算的。就在该作业运行之前，对由参数 interval给定的函数进行求值。如果该作业执行成功，则由interval返回的结果就变成了新的 next_date参数（假设该作业没有说明 next_date参数）。如果作业执行成功并且对 interval的求值结果为空，则该业就被从作业队列中删除。参数 interval给出的表达式是按字符串传递的，但其结果应是日期值。下面是对某些常用的表达式和它们的作用的说明：

间 隔 值	执 行 结 果
'SYSDATE+7'	从上次执行后的整一个星期。如果作业在星期二初次提交，则下一次运行将在下一个星期二。如果第二次运行失败的话，而在星期三获得成功，则后续的运行将在星期三开始
'NEXT_DAY(TRUNC (SYSDATE),"FRIDAY")+12/24'	每星期五中午运行。请在字符串内使用两个单引号把 FRIDAY括起来
SYSDATE+1/24	每小时

2. 运行命令RUN

过程DBMS_JOB.RUN可以立即启动作业运行。其语法如下：

```
RUN（job IN BIANRY_INTEGER）;
```

其中，作业必须是通过调用 SUBMIT已创建的作业。该命令不管指定作业的当前状况如何，立即在当前进程下运行该作业。需要提醒的是， SNP后台进程与该作业无关。

警告 DBMS_JOB.RUN将会初始化当前会话包，这样做的原因是为了给作业提供一个连续的环境来运行，就象SNP进程对作业的处理一样。

7.1.3 其他的DBMS_JOB子程序

下面，我们来讨论过程 DBMS_JOB中其他子程序的功能。表 7-3对这些子程序做了说明。在DBMS_JOB的包头中有一个额外的程序 —— ISUBMIT。该程序是由其他过程使用指定的作业号进入作业的接口程序。

表7-3 DBMS_JOB子程序说明

子 程 序 名	功 能 说 明
SUBMIT	把一个新的作业提交给队列，由 SNP进程控制运行
ISUBMIT	由其他过程使用指定的作业号进入作业的接口程序。程序员不能在其程序代码中直接使用该程序，只能使用 SUBMIT
RUN	强迫指定的作业运行在当前进程中
REMOVE	从作业队列中删除一个作业
BROKEN	标记作业中断或没有中断
CHANGE	变更作业中任何可配置的字段
WHAT	变更what字段
NEXT_DATE	变更next_date字段
INTERVAL	变更interval字段
INSTANCE	在Oracle8i及更高版本下，变更 instance字段
USER_EXPORT	返回需要重新创建作业调用的文本
CHECK_PRIVS	确认给定的作业号可访问

1. 删除作业

我们在本章前几节中已经看到，通过把参数 next_date设置为空，作业可以把自己从作业队列中删除。我们也可以使用下面的过程显式地把作业从队列中删除：

REMOVE（job IN BINARY_INTEGER）;

上面唯一的参数是作业号。如果该作业的 next_date的值为空的话（该作业所设置的值或参数间隔的值为空），则该作业在其结束运行后将被删除。如果调用删除过程时，该作业正在运行期间，则该作业将在其运行结束后从队列中删除。

2. 中断作业

对于运行失败的作业， Oracle系统将自动重新运行该作业。该作业将在其第一次运行失败一分钟后再次运行。如果上述努力再次失败，下一次运行将在两分钟后开始。每次运行的间隔将逐次加倍，从四分钟，到八分种，一直持续下去。如果重试间隔超过了该作业的执行间隔，

则使用执行间隔。一旦该作业失败了16次，它就被标识为中断的作业。中断的作业将不能再次运行。

我们可以使用RUN命令来运行中断的作业。如果调用成功的话，则该作业的失败记数器将重置为0，并且将该作业的中断标志取消。过程BROKEN也可以用来修改作业的状态。该过程的语法如下：

```
BROKEN( job IN BINARY_INTEGER,
        broken IN BOOLEAN,
        next_date IN DATE DEFAULT SYSDATE);
```

该过程参数的说明如下：

参　数	类　型	说　明
job	BIANARY_INTEGER	作业的状态将要发生变更的作业号
broken	BOOLEAN	作业的新状态。如果该值为真，则该作业就被标识为中断的作业。如果为假值，则该作业没有被中断并将在由next_date指定的时间到来时再次运行。
next_date	DATE	作业下次运行的日期。其默认值是SYSDATE

3. 变更作业

在作业被提交后，该作业的参数也可以变更。实现这种功能的过程的语法如下：

```
PROCEDURE CHANGE( job IN BINARY_INTEGER,
                  what IN VARCHAR2 DEFAULT NULL,
                  next_date IN DATE DEFAULT NULL,
                  interval IN VARCHAR2 DEFAULT NULL);

PROCEDURE WHAT( job IN BINARY_INTEGER,
                what IN VARCHAR2);

PROCEDURE NEXT_DATE( job IN BINARY_INTEGER,
                     next_date IN DATE);

PROCEDURE INTERVAL( job IN BINARY_INTEGER,
                    interval IN VARCHAR2);
```

过程CHANGE用来一次变更一个以上作业的属性，过程WHAT、NEXT_DATE、INTERVAL则用来改变与它们相关参数标识的属性。

以上所有参数的作用都与过程SUBMIT中参数的功能一样。如果我们使用CHANGE或WHAT来修改what参数，则当前环境就将变为适应该作业的新的执行环境。有关作业环境的进一步介绍，请参阅7.1.5节。

4. 实例仿射性

Oracle 8i及更高版本

当我们使用OPS（Oracle Parallel Server）时，我们可以指定一个实例来运行给定的作业。这就叫做实例仿射性（Instance Affinity）。实例可以由过程INSTANCE指定，其语法是：

```
PROCEDURE INSTANCE( job IN BINARY_INTEGER,
                    instance IN BINARY_INTEGER,
                    force IN BOOLEAN DEFAULT FALSE);
```

其中，instance是运行作业的实例号。如果实例是DBMS_JOB.ANY_INSTANCE(0)，则作业的仿射性将变更并且任何可用的实例都可以运行该作业，此时与参数 force的值无关。如果instance是正值并且参数force为假值，则作业仿射性只有在指定的实例处于运行状态才被变更。如果指定的实例没有运行，或该实例不合法，这时 Oracle8i将返回错误：“ORA-23428：与实例号字符串相关联的作业非法。”

其他DBMS_JOB子程序也支持实例仿射性。例如，如下所示的过程 SUBMIT就带有参数instance和force：

```
PROCEDURE SUBMIT( job OUT BINARY_INTEGER,
                  what IN VARCHAR2,
                  next_date IN DATE DEFAULT SYSDATE,
                  interval IN VARCHAR2 DEFAULT NULL,
                  no_parse IN BOOLEAN DEFAULT FALSE,
                  instance IN BINARY_INTEGER DEFAULT ANY_INSTANCE,
                  force IN BOOLEAN DEFAULT FALSE);
```

参数instance和force的含义和使用方法都与过程DBMS_JOB.INSTANCE的同类参数一样。

过程CHANGE在Oracle8i下得到了增强：

```
PROCEDURE CHANGE( job IN BINARY_INTEGER,
                  what IN VARCHAR2 DEFAULT NULL,
                  next_date IN DATE DEFAULT NULL,
                  interval IN VARCHAR2 DEFAULT NULL,
                  instance IN BINARY_INTEGER DEFAULT NULL,
                  force IN BOOLEAN DEFAULT FALSE);
```

其中参数instance和force与上面的参数一样。

最后，DBMS_JOB.RUN也带有force参数：

```
PROCEDURE RUN( job IN BINARY_INTEGER,
               force IN BOOLEAN DEFAULT FALSE);
```

如果参数force为真，则该作业仅可以在从指定实例内部调用 DBMS_JOB.RUN的情况下运行。

5. 导出作业

过程USER_EXPORT返回重新创建给定作业所需的文本：

```
PROCEDURE USER_EXPORT( job in BINARY_INTEGER,
                       mycall IN OUT VARCHAR2);
```

例如，如果我们通过TempInsert的第二版提交的作业调用过程USER_EXPORT的话，该过程将返回下面的文本：

节选自在线代码jobExport.sql

```
SQL> DECLARE
  2    v_JobText VARCHAR2(2000);
  3 BEGIN
  4    DBMS_JOB.USER_EXPORT(:v_JobNum, v_JobText);
  5    DBMS_OUTPUT.PUT_LINE(v_JobText);
  6 END;
```

```
  7 /
dbms_job.isubmit(job=>10,what=>'TempInsert(next_date);',next
_date=>to_date('1999-03-29:00:07:37','YYYY-MM-DD:HH24:MI:SS'
),interval=>'sysdate + (5/(24*60*60))',no_parse=>TRUE);
```

该作业必须在当前作业队列中，否则过程 USER_EXPORT将返回错误：“ ORA-23421：作业号job_num不在作业队列中 ”。

提示 USER_EXPORT返回对DBMS_JOB.ISUBMIT的调用，而不是 DBMS_JOB.SUBMIT的调用。如果要重新创建具有相同作业号的作业的话，我们可以使用ISUBMIT。然而，这种方法不太常用。我们推荐使用过程 SUBMIT。该命令将重新提交作业并生成一个新的作业号。 USER_EXPORT是由导出工具程序自动调用并生成后面重新导入时所使用的代码。

6. 检查作业的权限

过程CHECK_PRIVS有两个功能：其一是校验给定的作业号是否存在，其二是锁定数据字典中的相应的行以便程序对其进行修改。该过程的语法定义如下：

```
PROCEDURE CHECK_PRIVS(job IN BINARY_INTEGER);
```

其中，job是作业号，如果该作业号不存在，或该作业号没有被当前用户提交，则 Oracle将提示错误信息：“ ORA-23421：作业号job_num不在作业队列中 ”。

7.1.4 在数据库视图中观察作业

Oracle有几个数据字典视图专门用来记录作业信息。其中视图 dba_jobs和user_jobs返回作业的what、next_date、interval等有关信息。除此之外，这些视图还提供运行环境的信息。视图dba_jobs_running描述了当前运行的作业。有关这些视图的的介绍，请看本书附录 C的内容。

7.1.5 作业运行环境

当我们把作业提交给队列时，当前的环境就被记录下来。记录信息中包括如 NLS_DATE_FORMAT之类的NLS参数的设置。这些在作业创建时记录下来的信息将在该作业运行时使用。如果我们使用CHANGE或WHAT过程修改what的属性时，上述设置将随之改变。

注意 作业可以通过发布DBMS_SQL 包的ALTER SESSION命令（或Oracle8i中的本地动态SQL命令）来修改其运行环境。如果运行环境发生了变更，它将仅影响当前作业的运行，而不会对将来的运行有影响。包 DBMS_SQL和本地动态SQL的内容在本书的第8章中介绍。

作业将运行在其自己提交器所属的权限组之下，不允许任何角色运行。如果我们需要在作业中允许角色执行，我们可以使用动态 SQL来发布SET ROLE命令。当然，如果该作业是由存储过程组成的话，则所有的角色都将被禁止。

作业仅可以由提交器（ Submitter）控制运行。作业的执行权限与作业没有任何关联；包自身的EXECUTE权限是唯一必要的数据库权限。

7.2 文件输入输出

PL/SQL 2.3 及更高版本

如上所述，PL/SQL语言并没有把输入、输出功能配置在该语言内部， PL/SQL下的输入输出是通过所提供的包所支持的功能实现的。 PL/SQL输出到屏幕的功能是通过本书第3章介绍的包DBMS_OUTPUT 所提供的功能实现的。 PL/SQL2.3版通过包UTL_FILE的功能把文件I/O扩充到了文本文件。在该版本下无法借助其 UTL_FILE包来把输出直接转换为二进制文件。

Oracle 8 及更高版本

Oracle8允许通过使用类型 BFILE来读入二进制文件 ,这里BFILE是特殊形式的外部LOB。本书的第15，16将专门讨论BFILE类型以及LOB的其他类型。然而，即使使用Oracle8i，包UTL_FILE也不能用来处理二进制文件。

本节将描述UTL_FILE的工作原理。本章结尾处将提供三个完整的案例来介绍该包的使用方法。

7.2.1 安全

客户端PL/SQL有一个类似于UTL_FILE的包叫做TEXT_IO的包。值得注意的是，客户端要处理的安全问题要比服务器端要多，使用客户端包 TEXT_IO首次创建的文件在其操作系统权限的限制下，可以放置在客户端的任何地方。 PL/SQL或数据库本身没有与其关联的用于客户端 I/O操作的权限。我们将在本节讨论 UTL_FILE的安全实现。

1. 数据库安全

数据库服务器需要更可靠的安全机制来保证系统安全运行。 Oracle数据库通过限制包UTL_FILE可以访问目录的范围来实现安全运行。系统允许访问的目录由数据库初始化文件中的参数UTL_FILE_DIR说明。每个可以访问的目录都在初始化文件中用下面的参数行指定：

```
UTL_FILE_DIR = diretory_name
```

参数diretory_name所指定的目录在很大程度上与操作系统有关。如果操作系统对大小写敏感，则diretroy_name也区分大小写字母。例如，下面的目录入口在 Unix系统下是合法的（假设所指定的目录是存在的）：

```
UTL_FILE_DIR=/tmp
UTL_FILE_DIR=/home/oracle/output_files
```

为了能够使用 UTL_FILE来访问文件，目录名和文件名都要作为单独的参数传递给函数FOPEN。该函数将接收的目录名与可访问的文件清单进行比较。如果清单中有该文件，则允许对给文件进行操作，其具体操作方式还与下一节将介绍的操作系统的安全约束有关。如果函数FOPEN所接收的文件不可访问，该函数就返回错误。除此之外，可访问目录的子目录一般也是不可访问的，除非该子目录也显式地出现在可访问目录表中。假设上面的目录是可访问的，表 7-4对合法与非法的目录/文件名给予了说明。

注意 即使操作系统对大小写不敏感，在指定目录与可访问目录之间的比较也是大小写

敏感的。

如果初始化文件中有下面一行：

```
UTL_FILE_DIR = *
```

则数据库访问权限将被禁止。该命令将使所有的目录都可由 UTL_FILE访问。

表7-4 合法与非法文件的说明

目 录 名	文 件 名	说 明
/tmp	myfile.out	合法
/home/oracle/output_files	studengts.list	合法
/tmp/1995	january.results	非法，子目录/tmp/1995不能访问
/home/oralce	output_files/classes.list	非法，子目录名不能作为文件名的一部分
/TMP	myfile.out	非法，大写字母不对

警告 执行关闭数据库访问权限的操作必须十分小心。 Oracle并不推荐用户在产生式系统中使用该选择项，其原因是该选择项的使用可能会限制操作系统的访问权限。除此之外，也不要使用‘ .’来代表要访问目录的当前目录部分（ Unix使用该符号表示当前目录），目录的指定一定要使用目录的全名。

2. 操作系统的安全性

由UTL_FILE执行的文件操作是作为 Oracle用户实现的。（Oracle用户是用于运行数据库所需文件的拥有者，同时它也是组成数据库实例的进程的拥有者。）这样一来，Oracle用户就必须具有操作系统读写所有可访问文件的权限。如果 Oracle用户没有访问权限，则任何对该目录的访问将被操作系统禁止。

由UTL_FILE创建的任何文件将归属于 Oracle用户所有，这些文件在创建时已具有操作系统为Oracle用户配置的权限。如果有其他用户要在 UTL_FILE之外访问这些文件的话，就需要操作系统变更这些文件的访问权限。

警告 禁止对所有可访问目录的写操作也是一种安全措施。在这种情况下，写操作的权限只赋予Oracle用户。

7.2.2 UTL_FILE引发的异常

如果UTL_FILE中的过程或函数遇到了错误，它们将引发异常。这些可能发生的异常在表 7-5中给予了说明，这些异常中有八个在 UTL_FILE中有定义，其余的两个是予定义的异常（NO_DATA_FOUND和VALUE_ERROR）。UTL_FILE异常可以按名字或由异常处理程序OTHERS来捕捉处理。予定义的异常还可以由 SQLCODE的值（1403或6502）标识。

7.2.3 打开和关闭文件

UTL_FILE中的所有操作都使用文件句柄实现。所谓文件句柄就是 PL/SQL中用来标识文件的值，它类似于DBMS_SQL中使用的游标ID。所有的文件句柄都是UTL_FILE.FILE_TYPE类型。

文件句柄由FOPEN返回并作为IN类型参数传递给UTL_FILE中其他子程序使用。

1. FOPEN

FOPNE可以打开用于输入或输出的文件。在任何时候，给定的文件每次只能打开用于输入或用于输出。打开的文件不能同时用于输入和输出操作。 FOPEN的语法如下：

```
FUNCTION FOPEN( location IN VARCHAR2,
                filename IN VARCHAR2,
                open_mode IN VARCHAR2)
  RETURN FILE_TYPE;
```

要打开文件的目录路径必须是已经存在的，否则 FOPEN不会创建新的目录。如果打开方式为‘ w ’，则FOPEN将覆盖现有的文件。FOPEN的返回值和参数的说明如下所示：

参 数	类 型	说 明
location	VARCHAR2	文件位于的目录路径。如果该目录与可访问目录表中的目录不匹配，则将引发异常UTL_FILE.INVALID_PATH。
filename	VARCHAR2	要打开的文件名。如果open_mode为‘ w ’，则将现存文件覆盖。
open_mode	VARCHAR2	打开方式。其合法值有：‘ r ’读文本，‘ w ’写文本，‘ a ’ 追加文本。该参数对大小写不敏感。如果指定了‘ a ’方式但该文件并不存在，则就按‘ w ’方式创建新文件
return value	UTL_FILE.FILE_TYPE	后续函数使用的文件句柄

表7-5 UTL_FILE引发的异常

异 常	引发条件	引发函数
INVALID_PATH	非法或不能访问的目录或文件名	FOPEN
INVALID_MODE	非法打开模式	FOPEN
INVALID_FILEHANDLE	文件句柄指示的文件没有打开	FCLOSE，GET_LINE,PUT, PUT_LINE,NEW LINE,PUTF,FFLUSH
INVALID_OPERATION	文件不能按要求打开，该异常与操作系统的权限有关。该异常也可能在企图对以读方式打开的进行写操作时，或企图对以写方式打开的文件进行读操作时引发	GET_LINE,PUT, PUT_LINE,NEW_LINE, PUTF,FFLUSH
INVALID_MAXLINESIZE	所指定的最大行数太大或太小。该异常只在Oracle8.0.5及更高的版本下引发	FOPEN
READ_ERROR	在读操作期间出现的操作系统错误	GET_LINE
WEITE_ERROR	在写操作期间出现的操作系统错误	PUT,PUT_LINE,PUTF,NEW_LINE,FFLUSH, FCLOSE,FCLOSE_ALL
INTERNAL_ERROR	未说明的内部错误	所有函数
NO_DATA_FOUND	读操作中遇到了文件结束符	GET_LINE
VALUE_ERROR	输入文件大于GET_LINE中指定的缓冲区	GET_LINE

FOPEN可以引发下列异常之一：

• UTL_FILE.INVALID_PATH

• UTL_FILE.INVALID_MODE
• UTL_FILE.INVALID_OPERATION
• UTL_FILE.INTERNAL_ERROR

2. 使用参数max_linesize的FOPEN

在Oracle8.0.5及更高版本下，UTL_FILE提供了FOPEN的另一个重载版本：

```
FUNCTION FOPEN( location IN VARCHAR2,
                filename IN VARCHAR2,
                open_mode IN VARCHAR2,
                max_linesize IN BINARY_INTEGER)
   RETURN FILE_TYPE;
```

其中，location,filename,open_mode参数的意义与FOPEN的第一版相同。参数max_linesize用来说明文件的最大行数，其范围从1-32767.如果没有使用该参数，则最大行数为1024。如果该参数的值小于1或大于32767，则引发UTL_FILE.INVALID_MAXLINESIZE异常。

3. FCLOSE

当文件的读写操作结束后，要使用FCLOSE将该文件关闭。关闭文件将释放UTL_FILE用来操作该文件所占用的资源。FCLOSE的语法如下：

```
PROCEDURE FCLOSE (file_handle IN OUT FILE_TYPE) ;
```

FCLOSE的唯一参数是文件句柄file_handle。在关闭该文件之前，任何等待写入文件的未决变更都将实现。如果在写入过程中出现了错误，则将引发UTL_FILE.WRITE_ERROR异常。如果文件句柄没有指向合法打开的文件，将引发UTL_FILE.INVALID_FILEHANDLE异常。

4. IS_OPEN

该逻辑值函数在指定的文件处于打开的状态下返回TRUE，反之返回FALSE。IS_OPEN的定义如下：

```
FUNCTION IS_OPEN(file_handle IN FILE_TYPE)
  RETURN BOOLEAN;
```

即使IS_OPEN返回TRUE，在文件处于打开状态下也存在着操作系统出错的可能。

5. FCLOSE_ALL

该过程将关闭所有打开的文件。该功能可用来清理错误句柄。该过程的定义如下：

```
PROCEDURE FCLOSE_ALL;
```

该过程没有参数。在文件关闭前，文件的所有未决状态都将被写入文件。由于可能执行写操作，所以如果在写操作中出现错误时，该过程可能会引发UTL_FILE.WRITE_ERROR异常。

警告 FCLOSE_ALL将关闭文件并释放UTL_FILE占用的资源。然而，该过程并不对文件做关闭标志。这样一来，在执行FCLOSE_ALL后，IS_OPEN仍将返回TRUE。除此之外，执行FCLOSE_ALL之后的任何读或写操作都将失败。

7.2.4 文件输出

有五个过程可用来把数据输出到文件中。它们分别是：PUT、PUT_LINE、NEW_LINE、

PUTF和FFLUSH。其中PUT、PUT_LINE、NEW_LINE的作用非常类似于我们在第3章中讨论过的包DBMS_OUTPUT中的对应参数。输出记录的最大容量是1023个字节（除非使用FOPEN说明新的容量值）。该记录中包括了一个表示新行的字符。

1. 过程PUT

PUT将把指定的字符串输出到指定文件中。该文件在执行PUT操作前应处于打开状态。PUT的定义如下：

```
PROCEDURE PUT ( file_handle IN FILE_TYPE,
                buffer IN VARHCAR2) ;
```

PUT将不在该文件中追加新行字符，如果需要的话，则必须使用PUT_LINE或NEW_LINE在行中加入行结束符。如果在写入操作时出现了操作系统错误，将引发UTL_FILE.WRITE_ERROR异常。PUT参数的说明如下：

参　数	类　型	说　明
file_handle	UTL_FILE.FILE_TYPE	由FOPEN返回的文件句柄。如果该句柄是非法的，则将引发UTL_FILE.INVALID_FILEHANDLE异常
buffer	VARCHAR2	等待写入文件中的文本字符串。如果没有使用‘w’或‘a’方式来打开文件，就引发UTL_FILE.INVALID_OPERATION

2. 过程NEW_LINE

NEW_LINE将把一个或多个行结束符写入到指定的文件中。该过程的语法定义如下：

```
PROCEDURE NEW_LINE(file_handle IN FILE_TYPE,
                   lines IN NATURAL:=1);
```

行结束符与操作系统有关，不同的操作系统可能使用不同的行结束符。如果在写入操作时出现了操作系统错误，则引发UTL_FILE.WRITE_ERROR异常。参数NEW_LINE的说明如下：

参　数	类　型	说　明
file_handle	UTL_FILE.FILE_TYPE	由FOPEN返回的文件句柄。如果该句柄不合法，将引发UTL_FILE.INVALID_FILEHANDLE异常
lines	NATURAL	输出的行结束符个数。其默认值是1,即输出一个新行。如果文件没有用‘w’或‘a’方式打开，系统将引发UTL_FILE.INVALID_OPERATION异常

3. PUT_LINE

PUT_LINE把指定的字符串输出到指定的文件中，被写入的文件必须以写操作方式打开，在字符串写入该文件后，系统在行尾加上系统定义的行结束符。PUT_LINE的定义如下：

```
PROCEDURE PUT_LINE(file_handle IN FILE_TYPE,
                   buffer IN VARCHAR2);
```

PUT_LINE的参数的含义如下表所示。调用PUT_LINE等价与调用PUT后再调用NEW_LINE来写入行结束符。如果在写入期间发生了操作系统错误，则引发UTL_FILE.WRITE_ERROR异常。

参 数	类 型	说 明
file_handle	UTL_FILE.FILE_TYPE	由FOPEN返回的文件句柄。如果该句柄不合法，将引发 UTL_FILE.INVALID_FILEHANDLE异常
buffer	VARCHAR2	写入到文件中的文本字符串。如果写入文件没有以‘ w ’或‘ a ’模式打开，则引发UTL_FILE.INVALID_OPERATION异常

4. 过程PUTF

PUTF的功能与PUT类似，但该过程允许对输出字符串进行格式。 PUTF实现了C语言printf()的部分功能，其语法也类似于C语言的printf()语句。PUTF的语法如下：

```
PROCEDURE PUTF( file_handle IN FILE_TYPE,
                format IN VARCHAR2,
                arg1 IN VARCHAR2 DEFAULT NULL,
                arg2 IN VARCHAR2 DEFAULT NULL,
                arg3 IN VARCHAR2 DEFAULT NULL,
                arg4 IN VARCHAR2 DEFAULT NULL,
                arg5 IN VARCHAR2 DEFAULT NULL);
```

请注意，参数arg1-arg5具有默认值，因此这几个参数可以省略。格式字符串 format除了带有正常的文本外，还有两个特殊字符‘ %s ’和‘ \n ’。在格式字符串中出现‘ %s ’的地方都将用一个上述可选参数替代。格式字符串中出现‘ \n ’的地方都将用一个新的行结束符替代。上述参数的详细用法在下面案例后面的表中给予说明。对于 PUT和PUT_LINE，如果在写入过程中出现了操作系统错误，则引发UTL_FILE.WRITE_ERROR异常。

例如，如果我们运行下面的块：

```
DECLARE
  v_OutputFile UTL_FILE.FILE_TYPE;
  v_Name VARCHAR2(10) := 'Scott';
BEGIN
  v_OutputFile := UTL_FILE.FOPEN(...);
  UTL_FILE.PUTF(v_OutputFile,
    'Hi there!\nMy name is %s, and I am a %s major.\n',
   v_Name, 'Computer Science');
  FCLOSE(v_OutputFile);
END;
```

则输出文件将带有下面内容：

```
Hi There!
My name is Scott, and I am a Computer Science major.
```

参 数	类 型	说 明
file_handle	UTL_FILE.FILE_TYPE	FOPEN返回的文件句柄。如果该句柄不合法，则引发 UTL_FILE.INVALID_FILEHANDLE异常
format	VARCHAR2	格式字符串包括常规文本和两个特殊的格式字符‘ %s ’和‘ \n ’。如果文件没有以‘ w ’或‘ a ’的模式打开，则引发 UTL_FILE. INVALID_OPERATION异常
arg1-agr5	VARCHAR2	五个可选的参数。每个参数将被对应‘ %s ’的格式字符代替。如果‘ %s ’字符多于参数的话，就使用空字符串来代替格式字符

5. 过程FFLUSH

使用PUT，PUT_LINE,PUTF，或NEW_LINE输出的数据通常是存储在缓冲区中的。当该缓冲区装满后，该缓冲区中的字符将被送往输出文件。 FFLUSH强行把缓冲区中的字符立即写入指定文件。需要提醒的是， FFLUSH只是把缓冲区中以 NEW_LINE字符结尾的行写入文件中。任何最后执行PUT操作放入缓冲区的字符都将在缓冲区中保留。该过程的语法是：

```
PROCEDURE FFLUSH (file_handle IN FILE_TYPE) ;
```

FFLUSH可以引发下列异常：

- UTL_FILE.INVALID_FILEHANDLE
- UTL_FILE.INVALID_OPERATION
- UTL_FILE.WRITE_ERROR

7.2.5 文件输入

GET_LINE是用来从文件中执行读入操作的，该过程每次从指定的文件中读入一行文本并到将该文本串送入缓冲区，新行字符不包括在返回字符串中。下面是该过程的定义：

```
PROCEDURE GET_LINE(file_handle IN FILE_TYPE,
                   buffer OUT VARCHAR2);
```

当从指定文件中读入最后一行时，异常 NO_DATA_FOUND将会引发。如果该行不适宜进入作为实参提供的缓冲区，就将引发异常 VALUE_ERROR。读入空行将返回一个空字符串 NULL。如果在读入期间，操作系统系统出现了错误，则引发 UTL_FILE.READ_ERROR异常。输入行的最大长度是1022字节。（除非使用FOPEN的参数max_linesize指定最大行数。）下面是这些参数的说明：

参　数	类　型	说　明
file_handle	UTL_FILE.FILE_TYPE	FOPEN返回的文件句柄。如果该句柄不合法，则将引发UTL_FILE.INVALID_FILEHANDLE异常
buffer	VARCHAR2	将一行写入的缓冲区。如果文件没有以读‘r’的模式打开，则引发UTL_FILE.INVALID_OPERATION异常

7.2.6 文件操作案例

本节将分析三个使用UTL_FILE的程序案例。第一个案例是我们在第 3章介绍的包DEBUG的新版本。第二个案例从一个文件中将学生信息读入并装入到表中。第三个程序是打印成绩单。

1. 包DEBUG

UTL_FILE的第一种用法是实现调试程序包。由于 DBMS_OUTPUT只能在块运行结束后才能打印执行结果（我们在第 3章讨论过该问题），而UTL_FILE可以提供更快捷的输出。下面是用UTL_FILE实现的DEBUG包：

节选自在线代码Debug.sql

```
CREATE OR REPLACE PACKAGE Debug AS
  /* Global variables to hold the name of the debugging file and
```

```
      directory. */
    v_DebugDir VARCHAR2(50) := '/tmp';
    v_DebugFile VARCHAR2(20) := 'debug.out';
    /* Call Debug to output a line consisting of:
       p_Description: p_Value
       to the debugging file. */
    PROCEDURE Debug(p_Description IN VARCHAR2,
                    p_Value IN VARCHAR2);

    /* Closes the debugging file first, then calls FileOpen to
    set the packaged variables and open the file with the new
    parameters. */
    PROCEDURE Reset(p_NewFile IN VARCHAR2 := v_DebugFile,
                    p_NewDir IN VARCHAR2 := v_DebugDir);

    /* Sets the packaged variables to p_NewFile and p_NewDir, and
       opens the debugging file. */
  PROCEDURE FileOpen(p_NewFile IN VARCHAR2 := v_DebugFile,
                     p_NewDir IN VARCHAR2 := v_DebugDir);

    /* Closes the debugging file. */
      PROCEDURE FileClose;
    END Debug;

  CREATE OR REPLACE PACKAGE BODY Debug AS
    v_DebugHandle UTL_FILE.FILE_TYPE;

    PROCEDURE Debug(p_Description IN VARCHAR2,
                    p_Value IN VARCHAR2) IS
    BEGIN
       IF NOT UTL_FILE.IS_OPEN(v_DebugHandle) THEN
          FileOpen;
    END IF;
       /* Output the info, and flush the file. */
       UTL_FILE.PUTF(v_DebugHandle, '%s: %s\n',
                     p_Description, p_Value);
    UTL_FILE.FFLUSH(v_DebugHandle);
  EXCEPTION
    WHEN UTL_FILE.INVALID_OPERATION THEN
      RAISE_APPLICATION_ERROR(-20102,
                              'Debug: Invalid Operation');
    WHEN UTL_FILE.INVALID_FILEHANDLE THEN
      RAISE_APPLICATION_ERROR(-20103,
                              'Debug: Invalid File Handle');
    WHEN UTL_FILE.WRITE_ERROR THEN
      RAISE_APPLICATION_ERROR(-20104,
                              'Debug: Write Error');
    WHEN UTL_FILE.INTERNAL_ERROR THEN
```

```
    RAISE_APPLICATION_ERROR(-20104,
                            'Debug: Internal Error');
  END Debug;

    PROCEDURE Reset(p_NewFile IN VARCHAR2 := v_DebugFile,
                    p_NewDir IN VARCHAR2 := v_DebugDir) IS
  BEGIN

     /* Make sure the file is closed first. */
  IF UTL_FILE.IS_OPEN(v_DebugHandle) THEN
    FileClose;
  END IF;

  FileOpen(p_NewFile,p_NewDir);

END Reset;

PROCEDURE FileOpen(p_NewFile IN VARCHAR2 := v_DebugFile,
                   p_NewDir IN VARCHAR2 := v_DebugDir) IS
BEGIN
     /* Open the file for writing. */
  v_DebugHandle := UTL_FILE.FOPEN(p_NewDir, p_NewFile, 'w');
  /* Set the packaged variables to the values just passed in. */
  v_DebugFile := p_NewFile;
  v_DebugDir := p_NewDir;
EXCEPTION
  WHEN UTL_FILE.INVALID_PATH THEN
    RAISE_APPLICATION_ERROR(-20100, 'Open: Invalid Path');
  WHEN UTL_FILE.INVALID_MODE THEN
    RAISE_APPLICATION_ERROR(-20101, 'Open: Invalid Mode');
  WHEN UTL_FILE.INVALID_OPERATION THEN
    RAISE_APPLICATION_ERROR(-20101, 'Open: Invalid Operation');
  WHEN UTL_FILE.INTERNAL_ERROR THEN
    RAISE_APPLICATION_ERROR(-20101, 'Open: Internal Error');
END FileOpen;

PROCEDURE FileClose IS
BEGIN
  UTL_FILE.FCLOSE(v_DebugHandle);
EXCEPTION
  WHEN UTL_FILE.INVALID_FILEHANDLE THEN
    RAISE_APPLICATION_ERROR(-20300,
                            'Close: Invalid File Handle');
  WHEN UTL_FILE.WRITE_ERROR THEN
    RAISE_APPLICATION_ERROR(-20301,
                            'Close: Write Error');
  WHEN UTL_FILE.INTERNAL_ERROR THEN
    RAISE_APPLICATION_ERROR(-20302,
```

```
                                'Close: Internal Error');
  END FileClose;
END Debug;
```

使用该DEBUG包的方法是很直观的。该包的变量 v_DebugDir和v_DebugFile指定了输出文件的位置和名称。如果我们先调用 Debug.Debug，则该文件打开时将带有上述的值。为了修改这些值，我们可以调用 Debug.FileOpen来复位该包的变量并重新打开该文件。或者调用 Debug.Reset首先将该文件关闭，然后再执行 FileOpen的操作。Debug.FileClose将在操作结束时关闭该文件。例如，如果我们执行下面的块：

节选自在线代码CallDebug.sql

```
BEGIN
  Debug.Debug('Scott', 'First call');
  Debug.Reset('debug2.out', '/tmp');
  Debug.Debug('Scott', 'Second call');
  Debug.Debug('Scott', 'Third call');
  Debug.FileClose;
END;
```

接着文件/tmp/debug.out中将包括下面的内容：

```
Scott: First call
```

文件/tmp/debug2.out将包括下面的内容：

```
Scott:Second call
Scott:Third call
```

提示 请注意程序中各种例程的异常处理程序。这些异常处理程序识别引发的错误类型以及引发该错误的过程。这种方法是值得我们在使用 UTL_FILE时借鉴。

2. 加载学生数据程序

过程LoadStudents将根据其接收的文件内容来对表进行插入操作。该文件是用逗号分界的，也就是说，该文件的每一行都是一个记录，该行中的逗号用来分割字段。这是文本文件的常用格式。下面是该过程的代码：

节选自在线代码LoadStudents.sql

```
CREATE OR REPLACE PROCEDURE LoadStudents (
  /* Loads the students table by reading a comma-delimited file.
     The file should have lines that look like

     first_name,last_name,major

     The student ID is generated from student_sequence.
     The total number of rows inserted is returned by
   p_TotalInserted. */
p_FileDir IN VARCHAR2,
p_FileName IN VARCHAR2,
p_TotalInserted IN OUT NUMBER) AS
```

```
v_FileHandle UTL_FILE.FILE_TYPE;
v_NewLine VARCHAR2(100); -- Input line
v_FirstName students.first_name%TYPE;
v_LastName students.last_name%TYPE;
v_Major students.major%TYPE;
/* Positions of commas within input line. */
v_FirstComma NUMBER;
v_SecondComma NUMBER;

BEGIN
  -- Open the specified file for reading.
  v_FileHandle := UTL_FILE.FOPEN(p_FileDir, p_FileName, 'r');

  -- Initialize the output number of students.
  p_TotalInserted := 0;

  -- Loop over the file, reading in each line. GET_LINE will
  -- raise NO_DATA_FOUND when it is done, so we use that as the
  -- exit condition for the loop.
  LOOP
    BEGIN
      UTL_FILE.GET_LINE(v_FileHandle, v_NewLine);
    EXCEPTION
      WHEN NO_DATA_FOUND THEN
        EXIT;
  END;

  -- Each field in the input record is delimited by commas. We
  -- need to find the locations of the two commas in the line
  -- and use these locations to get the fields from v_NewLine.
  -- Use INSTR to find the locations of the commas.
  v_FirstComma := INSTR(v_NewLine, ',', 1, 1);
  v_SecondComma := INSTR(v_NewLine, ',', 1, 2);

  -- Now we can use SUBSTR to extract the fields.
  v_FirstName := SUBSTR(v_NewLine, 1, v_FirstComma - 1);
  v_LastName := SUBSTR(v_NewLine, v_FirstComma + 1,
                       v_SecondComma - v_FirstComma - 1);
  v_Major := SUBSTR(v_NewLine, v_SecondComma + 1);

  -- Insert the new record into students.
INSERT INTO students (ID, first_name, last_name, major)
  VALUES (student_sequence.nextval, v_FirstName,
```

过程LoadStudents使用的一个输入文件的内容如下：

节选自在线代码students.sql

```
            v_LastName, v_Major);
    p_TotalInserted := p_TotalInserted + 1;
```

```
  END LOOP;
  -- Close the file.
  UTL_FILE.FCLOSE(v_FileHandle);

  COMMIT;
EXCEPTION
  -- Handle the UTL_FILE exceptions meaningfully, and make sure
  -- that the file is properly closed.
  WHEN UTL_FILE.INVALID_OPERATION THEN
    UTL_FILE.FCLOSE(v_FileHandle);
    RAISE_APPLICATION_ERROR(-20051,
                            'LoadStudents: Invalid Operation');
  WHEN UTL_FILE.INVALID_FILEHANDLE THEN
    UTL_FILE.FCLOSE(v_FileHandle);
    RAISE_APPLICATION_ERROR(-20052,
                            'LoadStudents: Invalid File Handle');
WHEN UTL_FILE.READ_ERROR THEN
  UTL_FILE.FCLOSE(v_FileHandle);
  RAISE_APPLICATION_ERROR(-20053,
                          'LoadStudents: Read Error');
  WHEN UTL_FILE.INVALID_PATH THEN
    UTL_FILE.FCLOSE(v_FileHandle);
    RAISE_APPLICATION_ERROR(-20054,
                            'LoadStudents: Invalid Path');
  WHEN UTL_FILE.INVALID_MODE THEN
    UTL_FILE.FCLOSE(v_FileHandle);
    RAISE_APPLICATION_ERROR(-20055,
                            'LoadStudents: Invalid Mode');
  WHEN UTL_FILE.INTERNAL_ERROR THEN
    UTL_FILE.FCLOSE(v_FileHandle);
    RAISE_APPLICATION_ERROR(-20056,
                          'LoadStudents: Internal Error');
  WHEN VALUE_ERROR THEN
    UTL_FILE.FCLOSE(v_FileHandle);
    RAISE_APPLICATION_ERROR(-20057,
                            'LoadStudents: Value Error');
  WHEN OTHERS THEN
    UTL_FILE.FCLOSE(v_FileHandle);
  RAISE;
END LoadStudents;
```

过程LoadStudents使用的一个输入文件的内容如下

```
Scott,Smith,Computer Science
Margaret,Mason,History
Joanne,Junebug,Computer Science
Manish,Murgratroid,Economics
Patrick,Poll,History
Timothy,Taller,History
```

```
Barbara,Blues,Economics
David,Dinsmore,Music
Ester,Elegant,Nutrition
Rose,Riznit,Music
Rita,Razmataz,Nutrition
Shay,Shariatpanahy,Computer Science
```

注意 过程LoadStudents使用student_sequence序列来确认学生的ID。如果该序列在表students被载入后进行了重建的话，该过程将可能返回一个已经在表 students中的值。在这种情况下，表students的主约束键将会非法。

3. **打印成绩单程序**

该案例介绍了如何使用UTL_FILE**来打印学生的成绩单。下面是过程** CalculateGPA**的代码：**

节选自在线代码CalculateGPA.sql

```
CREATE OR REPLACE PROCEDURE CalculateGPA (
/* Returns the grade point average for the student identified
by p_StudentID in p_GPA. */
p_StudentID IN students.ID%TYPE,
p_GPA OUT NUMBER) AS

CURSOR c_ClassDetails IS
SELECT classes.num_credits, rs.grade
FROM classes, registered_students rs
WHERE classes.department = rs.department
AND classes.course = rs.course
AND rs.student_id = p_StudentID;
  v_NumericGrade NUMBER;
  v_TotalCredits NUMBER := 0;
  v_TotalGrade NUMBER := 0;

BEGIN
  FOR v_ClassRecord in c_ClassDetails LOOP
    -- Determine the numeric value for the grade.
    SELECT DECODE(v_ClassRecord.grade, 'A', 4,
                                       'B', 3,
                                       'C', 2,
                                       'D', 1,
                                       'E', 0)
    INTO v_NumericGrade
    FROM dual;

  v_TotalCredits := v_TotalCredits + v_ClassRecord.num_credits;
  v_TotalGrade := v_TotalGrade +
                  (v_ClassRecord.num_credits * v_NumericGrade);
  END LOOP;

  p_GPA := v_TotalGrade / v_TotalCredits;
```

```
END CalculateGPA;
```

PrintTranscript **用以下代码创建**

节选自在线代码printTranscript.sql

```
CREATE OR REPLACE PROCEDURE PrintTranscript (
  /* Outputs a transcript to the indicated file for the indicated
     student. The transcript will consist of the classes for which
     the student is currently registered and the grade received
     for each class. At the end of the transcript, the student's
     GPA is output. */
  p_StudentID IN students.ID%TYPE,
  p_FileDir IN VARCHAR2,
  p_FileName IN VARCHAR2) AS

  v_StudentGPA NUMBER;
  v_StudentRecord students%ROWTYPE;
  v_FileHandle UTL_FILE.FILE_TYPE;
  v_NumCredits NUMBER;

  CURSOR c_CurrentClasses IS
    SELECT *
      FROM registered_students
      WHERE student_id = p_StudentID;

BEGIN
-- Open the output file in append mode.
v_FileHandle := UTL_FILE.FOPEN(p_FileDir, p_FileName, 'a');

SELECT *
  INTO v_StudentRecord
  FROM students
  WHERE ID = p_StudentID;

-- Output header information. This consists of the current
-- date and time, and information about this student.

UTL_FILE.PUTF(v_FileHandle, 'Student ID: %s\n',
  v_StudentRecord.ID);
UTL_FILE.PUTF(v_FileHandle, 'Student Name: %s %s\n',
  v_StudentRecord.first_name, v_StudentRecord.last_name);
UTL_FILE.PUTF(v_FileHandle, 'Major: %s\n',
  v_StudentRecord.major);
UTL_FILE.PUTF(v_FileHandle, 'Transcript Printed on: %s\n\n\n',
  TO_CHAR(SYSDATE, 'Mon DD,YYYY HH24:MI:SS'));

UTL_FILE.PUT_LINE(v_FileHandle, 'Class Credits Grade');
UTL_FILE.PUT_LINE(v_FileHandle, '------- ------- -----');
FOR v_ClassesRecord in c_CurrentClasses LOOP
```

```
  -- Determine the number of credits for this class.
  SELECT num_credits
    INTO v_NumCredits
    FROM classes
    WHERE course = v_ClassesRecord.course
    AND department = v_ClassesRecord.department;

  -- Output the info for this class.
  UTL_FILE.PUTF(v_FileHandle, '%s %s %s\n',
    RPAD(v_ClassesRecord.department || ' ' ||
         v_ClassesRecord.course, 7),
    LPAD(v_NumCredits, 7),
    LPAD(v_ClassesRecord.grade, 5));
  END LOOP;

  -- Determine the GPA.
  CalculateGPA(p_StudentID, v_StudentGPA);

  -- Output the GPA.
  UTL_FILE.PUTF(v_FileHandle, '\n\nCurrent GPA: %s\n',
    TO_CHAR(v_StudentGPA, '9.99'));

  -- Close the file.
  UTL_FILE.FCLOSE(v_FileHandle);
EXCEPTION
  -- Handle the UTL_FILE exceptions meaningfully, and make sure
  -- that the file is properly closed.
  WHEN UTL_FILE.INVALID_OPERATION THEN
    UTL_FILE.FCLOSE(v_FileHandle);
    RAISE_APPLICATION_ERROR(-20061,
                            'PrintTranscript: Invalid Operation');
  WHEN UTL_FILE.INVALID_FILEHANDLE THEN
    UTL_FILE.FCLOSE(v_FileHandle);
  RAISE_APPLICATION_ERROR(-20062,
                          'PrintTranscript: Invalid File Handle');
  WHEN UTL_FILE.WRITE_ERROR THEN
  UTL_FILE.FCLOSE(v_FileHandle);
  RAISE_APPLICATION_ERROR(-20063,
                          'PrintTranscript: Write Error');
  WHEN UTL_FILE.INVALID_MODE THEN
  UTL_FILE.FCLOSE(v_FileHandle);
  RAISE_APPLICATION_ERROR(-20064,
                           'PrintTranscript: Invalid Mode');
  WHEN UTL_FILE.INTERNAL_ERROR THEN
  UTL_FILE.FCLOSE(v_FileHandle);
RAISE_APPLICATION_ERROR(-20065,
                        'PrintTranscript: Internal Error');
END PrintTranscript;
```

假设我们给定表registered_students**（由过程**relTables.sql**创建的）的内容如下：**

```
SQL> select * from registered_students;
STUDENT_ID DEP COURSE G
---------- --- --------- -
         10000 CS 102 A
         10002 CS 102 B
         10003 CS 102 C
         10000 HIS 101 A
         10001 HIS 101 B
         10002 HIS 101 B
         10003 HIS 101 A
         10004 HIS 101 C
         10005 HIS 101 C
         10006 HIS 101 E
         10007 HIS 101 B
         10008 HIS 101 A
         10009 HIS 101 D
         10010 HIS 101 A
         10008 NUT 307 A
         10010 NUT 307 A
         10009 MUS 410 B
         10006 MUS 410 E
         10011 MUS 410 B
         10000 MUS 410 B
20 rows selected.
```

如果我们调用过程PrintTranscript**来打印学生**ID**号为**10000**和**10009**的成绩单，我们将得到下面两个输出文件：**

```
Student ID: 10000
Student Name: Scott Smith
Major: Computer Science
Transcript Printed on: Apr 26,1999 22:24:07
Class Credits Grade
------- ------- -----
CS 102     4     A
HIS 101    4     A
MUS 410    3     B

Current GPA: 3.73

Student ID: 10009
Student Name: Rose Riznit
Major: Music
Transcript Printed on: Apr 26,1999 22:24:31

Class Credits Grade
------- ------- -----
```

```
HIS 101    4     D
MUS 410    3     B

Current GPA: 1.86
```

7.3 小结

我们在本章分析了两个实用程序包：一个是 DBMS_JOB，另一个是UTL_FILE。数据库作业允许过程由数据库在预先定义的时间自动启动运行。包 UTL_FILE在确保服务器安全的前提下为PL/SQL语言增加了文件输入输出的功能。这些实用程序为 PL/SQL语言提供了新的功能。