**DB2篇**

**一、数据迁移工具**

这里介绍的工具或者命令用于在不同情况下的数据迁移，这些情况包括在同一数据库中的数据迁移，同一平台上不同数据库的数据迁移以及在不同平台之间的数据迁移。

Export

Importet

Datebase A File Datebase B

Load

Set Integrity

ASC、DEL

WSF、IXF

使用Export将数据库中表导出为文件，这里的文件格式有：ASC、DEL、WSF、IXF。ASC、DEL格式的文件为文本文件，可以用任何文本编辑器打开；WSF格式的文件可以将数据迁移到电子表格软件中，例如Excel；IXF格式的文件包括了数据表的数据描述语言（DDL）和里面的数据，使用IXF格式是非常方便的，利用它可以重建数据表， 而其他格式则没有办法这么做，但是导出的数据时二进制流式数据。

当数据导出到文件后，可以用Import将数据导入到数据库表中，如果使用ASC、DEL、WSF格式的文件作为中间文件，在它们导入之前数据表必须存在；而使用IXF格式的文件在导入前不需要存在相应的数据表；将数据导入到表的另外一种放法就是使用Load语句，Load语句比Import 更快，因为它不与DB2数据引擎交互而直接操作数据库页面。然而用Load方法导入数据不会检查数据的约束，也不会引发触发器。为了保证所导入数据的一致性，在使用Load之后通常会运行Set Integrity。

**1.1 导出（Export）工具**

db2 “export to employee.ixf of ixf select \* from employee where name=’zhang’ order by id asc fetch first 10 rows only”

将Employee表中姓氏是张的记录按id 顺序排序并提取前十条数据导出到employee.ixf文件中。

**1.2 导入（Export）工具**

db2 “import from employee.ixf of ixf replace\_create into employee\_copy”

将empoyee.ixf的所有数据导入到数据库表employee\_copy中。Replace\_create操作是Import工具提供的众多参数之一。这个参数表示如果employee\_copy数据表已经存在，则先清空数据表中的数据然后将ixf中的数据导入，如果employee\_copy数据表不存在，则会先建立该表，然后将数据导入。

db2 “import from employee.ixf of ixf insert into employee\_copy”

这里需要数据表employee\_copy存在并且结构要和备份数据来源的数据表一致。

**1.3 使用LOAD来导入**

db2 “load from employee.ixf of ixf replace into employee\_copy”

将ixf格式的employee.ixf文件里面的所有数据导入到数据表employee\_copy，replace是load工具所提供的众多选项之一，它表示将替换employee\_copy表中所有的数据。

执行完上面的命令后，该表进入check pending状态，这时你必须运行set integrity命令来检查数据的一致性。

db2 “set integrity for employee\_copy all immediate unchecked”

**1.4 db2move工具**

**1.4.1使用db2move命令导出数据库** 从另一数据库中导出初始化数据，首先进入导出的目录，如：D:\db,   
命令为：db2move dbname export –u username –p password。注：dbname 为原数据库名，username为用户名 password为密码。   
  
 恢复命令D:\db> db2move dbname import -u username -p password

**1.4.2 db2move实现跨数据库的移植**

由于db2数据库需要移植，使用db2 backup和db2 restore 发现会因为文件系统等原因出现很多问题，使用db2move发现简单，但效率稍低。使用方法如下：

1、在原始机建立一个目录test，cd进后：db2move dbname export –u user –p password   
 2、会生成很多文件，其中db2move.lst记录所有导出的文件，可以进行编辑从而决定导入哪些数据表。   
 3、在目标机建立一个目录test，cd进入：db2move dbname import –u user –p password

这样就实现了数据表结构和数据的移植，据说可以跨平台使用，但没有测试过，在AIX平台下测试成功。

注意若两个数据库使用了多个表空间的话需要先建立表空间在进行数据的导入，而且两者的数据库编码应保持一致。这里备份一下数据库编码的设置方法：   
 db2set db2codepage=1386   
 db2 terminate   
 注意一定要进行terminate，就是因为这个费了半天的时间。

**1.5 db2look**

db2look是DB2用于生成数据库DDL语句的一个工具,以便重新创建在数据库中定义的对象。

**1.5.1 命令以及语法**

命令：db2look -d DBname -a -e -p -i userID -w password -o d:\sample.sql   
 语法:   
 db2look -d DBname [-e] [-u Creator] [-z Schema] [-t Tname1 Tname2...TnameN] [-h] [-o Fname][-a][-m] [-c] [-r] [-l] [-x] [-xd] [-f] [-td x] [-noview] [-i userID] [-w password][-wrapper WrapperName][-server ServerName] [-nofed]  
 db2look -d DBname [-u Creator] [-s] [-g] [-a] [-t Tname1 Tname2...TnameN][-p] [-o Fname] [-i userID] [-w password]  
 db2look [-h]  
 参数说明：  
        -d: 数据库名：这必须指定  
        -e: 抽取复制数据库所需要的 DDL 文件  
             此选项将生成包含 DDL 语句的脚本  
             可以对另一个数据库运行此脚本以便重新创建数据库对象  
             此选项可以和 -m 选项一起使用  
        -u: 创建程序标识：若 -u 和 -a 都未指定，则将使用 ＄USER  
             若指定了 -a 选项，则将忽略 -u 选项  
        -z: 模式名：如果同时指定了 -z 和 -a，则将忽略 -z  
             联合部分的模式名被忽略  
         -t: 生成指定表的统计信息  
             可以指定的表的数目最多为 30  
        -h: 更详细的帮助消息  
        -o: 将输出重定向到给定的文件名  
            若未指定 -o 选项，则输出将转到 stdout  
        -a: 为所有创建程序生成统计信息  
             若指定了此选项，则将忽略 -u 选项  
        -m: 在模拟方式下运行 db2look 实用程序  
            此选项将生成包含 SQL UPDATE 语句的脚本  
            这些 SQL UPDATE 语句捕获所有统计信息  
            可以对另一个数据库运行此脚本以便复制初始的那一个  
            当指定了 -m 选项时，将忽略 -p、-g 和 -s 选项  
        -c: 不要生成模拟的 COMMIT 语句  
            除非指定了 -m 或 -e，否则将忽略此选项  
            将不生成 CONNECT 和 CONNECT RESET 语句  
       -r: 不要生成模拟的 RUNSTATS 语句  
            缺省值为 RUNSTATS。仅当指定了 -m 时，此选项才有效  
        -l: 生成数据库布局：数据库分区组、缓冲池和表空间。  
        -x: 如果指定了此选项，则 db2look 实用程序将生成授权 DDL  
            对于现有已授权特权，不包括对象的原始定义器  
      -xd: 如果指定了此选项，则 db2look 实用程序将生成授权 DDL  
            对于现有已授权特权，包括对象的原始定义器  
        -f: 抽取配置参数和环境变量  
            如果指定此选项，将忽略 -wrapper 和 -server 选项  
      -td: 将 x 指定为语句定界符（缺省定界符为分号（;））  
            应该与 -e 选项一起使用（如果触发器或者 SQL 例程存在的话）  
        -p: 使用简单文本格式  
        -s: 生成 postscript. 文件  
            此选项将为您生成 postscript. 文件  
            当设置了此选项时，将除去所有 latex 和 tmp ps 文件  
            所需的（非 IBM）软件：LaTeX 和 dvips  
            注意：文件 psfig.tex 必须在 LaTeX 输入路径中  
       -g: 使用图形来显示索引的页取装对  
            必须安装 Gnuplot，并且 <psfig.tex> 必须在您的 LaTeX 输入路径中  
            还将随 LaTeX 文件一起生成 <filename.ps> 文件  
       -i: 登录到数据库驻留的服务器时所使用的用户标识  
     -w: 登录到数据库驻留的服务器时所使用的密码  
   -noview: 不要生成 CREATE VIEW ddl 语句  
-wrapper: 为适用于此包装器的联合对象生成 DDL  
            生成的对象可能包含下列各项：  
            包装器、服务器、用户映射、别名、类型映射、  
            函数模板、函数映射和索引规范  
   -server: 为适用于此服务器的联合对象生成 DDL  
            生成的对象可能包含下列各项：  
            包装器、服务器、用户映射、别名、类型映射、  
            函数模板、函数映射和索引规范  
    -nofed: 不要生成 Federated DDL  
            如果指定此选项，将忽略 -wrapper 和 -server 选项  
LaTeX 排字: latex filename.tex 以获得 filename.dvi

**1.5.2应用示例**

1、db2look -d DEPARTMENT -u walid -e -o db2look.sql  
-- 这将生成由用户 WALID 创建的所有表和联合对象的 DDL 语句  
-- db2look 输出被发送到名为 db2look.sql 的文件中  
 2、db2look -d DEPARTMENT -z myscm1 -e -o db2look.sql  
-- 这将为模式名为 MYSCM1 的所有表生成 DDL 语句  
-- 还将生成 ＄USER 创建的所有联合对象的 DDL。  
-- db2look 输出被发送到名为 db2look.sql 的文件中  
 3、db2look -d DEPARTMENT -u walid -m -o db2look.sql  
-- 这将生成 UPDATE 语句以捕获关于用户 WALID 创建的表／别名的统计信息  
-- db2look 输出被发送到名为 db2look.sql 的文件中   
 4、db2look -d DEPARTMENT -u walid -e -wrapper W1 -o db2look.sql  
-- 这将生成由用户 WALID 创建的所有表的 DDL 语句  
-- 还将生成适用于包装器 W1 的用户 WALID 所创建所有联合对象的 DDL  
-- db2look 输出被发送到名为 db2look.sql 的文件中  
 5、db2look -d DEPARTMENT -u walid -e -server S1 -o db2look.sql  
-- 这将生成由用户 WALID 创建的所有表的 DDL 语句  
-- 还将生成适用于服务器 S1 的用户 WALID 所创建所有联合对象的 DDL  
-- db2look 输出被发送到名为 db2look.sql 的文件中  
 6、db2look -d newhis -l -e -o tbs.sql  
生成数据库newhis 下所有表的DDL，-l参数包括表空间等DDL数据。

     7、db2look -d DEPARTMENT -u walid -e -o db2look.sql   
-- 这将生成由用户 WALID 创建的所有表和联合对象的 DDL 语句  
-- db2look 输出被发送到名为 db2look.sql 的文件中   
 8、db2look -d DEPARTMENT -z myscm1 -e -o db2look.sql   
-- 这将为模式名为 MYSCM1 的所有表生成 DDL 语句  
-- 还将生成 $USER 创建的所有联合对象的 DDL。  
-- db2look 输出被发送到名为 db2look.sql 的文件中   
 9、db2look -d DEPARTMENT -u walid -m -o db2look.sql   
-- 这将生成 UPDATE 语句以捕获关于用户 WALID 创建的表／昵称的统计信息  
-- db2look 输出被发送到名为 db2look.sql 的文件中   
 10、db2look -d DEPARTMENT -u walid -e -wrapper W1 -o db2look.sql  
-- 这将生成由用户 WALID 创建的所有表的 DDL 语句  
-- 还将生成适用于包装器 W1 的用户 WALID 所创建所有联合对象的 DDL  
-- db2look 输出被发送到名为 db2look.sql 的文件中   
 11、db2look -d DEPARTMENT -u walid -e -server S1 -o db2look.sql   
-- 这将生成由用户 WALID 创建的所有表的 DDL 语句  
-- 还将生成适用于服务器 S1 的用户 WALID 所创建所有联合对象的 DDL  
-- db2look 输出被发送到名为 db2look.sql 的文件中

**二、提高 DB2 查询性能的常用方法**

**2.1 db2expln**

db2expln 是命令行下的解释工具，通过该命令可以获得文本形式的查询计划。

用法：

db2expln -d <数据库名> [ -u username ](可选可不选) { -q “db2 语句”} or { -f db2文件（里面是db2语句）} -g(可选可不选) -t(输出到终端) or -o<outfile>(输出到文件)

举例：

1、db2expln -d zpstst -q “select \*from zps\_ssdj\_w where jgm=’871010’” -g -t

2、db2expln -d zpstst -f file.sql -g -o outfile.txt

db2expln 将存取计划以文本形式输出，它只提供存取计划中主要的信息，并不包含每一个操作占用多少 CPU、I/O、占用 Buffer 的大小以及使用的[数据库](http://cpro.baidu.com/cpro/ui/uijs.php?adclass=0&app_id=0&c=news&cf=1001&ch=0&di=128&fv=18&is_app=0&jk=29dc4d1eaed5fb4a&k=%CA%FD%BE%DD%BF%E2&k0=%CA%FD%BE%DD%BF%E2&kdi0=0&luki=9&mcpm=0&n=10&p=baidu&q=mx163_cpr&rb=0&rs=1&seller_id=1&sid=4afbd5ae1e4ddc29&ssp2=1&stid=9&t=tpclicked3_hc&td=2004887&tu=u2004887&u=http://www.xker.com/page/e2010/0424/96153.html&urlid=0" \t "http://www.xker.com/page/e2010/0424/_blank)对象等信息，方便阅读。但是 db2expln 也会将各项有关存取计划的信息存入 Explain 表中，用户可以使用 db2exfmt 察看详细的格式化文本信息。

**2.2 db2advis**

db2advis 是 DB2 提供的另外一种非常有用的命令。通过该命令 DB2 可以根据优化器的配置以及机器性能给出提高查询性能的建议。这种建议主要集中于如何创建索引，这些索引可以降低多少[查询](http://cpro.baidu.com/cpro/ui/uijs.php?adclass=0&app_id=0&c=news&cf=1001&ch=0&di=128&fv=18&is_app=0&jk=29dc4d1eaed5fb4a&k=%B2%E9%D1%AF&k0=%B2%E9%D1%AF&kdi0=0&luki=5&mcpm=0&n=10&p=baidu&q=mx163_cpr&rb=0&rs=1&seller_id=1&sid=4afbd5ae1e4ddc29&ssp2=1&stid=9&t=tpclicked3_hc&td=2004887&tu=u2004887&u=http://www.xker.com/page/e2010/0424/96153.html&urlid=0" \t "http://www.xker.com/page/e2010/0424/_blank)代价，需要创建哪些表或者 Materialized Query Table(MQT) 等。

用法：

db2advis -d <db\_name> -a <user>/<password> -i <sql.file> -o <output>

举例：

db2advis -d zpstst -a zpstst/zpstst -i lab.sql -t 5

db2advis -d zpstst -i lab.sql -o output.txt

通过 -i 指定的 SQL 文件可以包含多个查询，但是查询必须以**分号**分隔。这与 db2expln 命令不同，db2expln 可以通过 -z 参数指定多个查询之间的分隔符。用户可以把某一个 workload 中所使用的所有查询写入 SQL 文件中，并在每个查询之前使用”--#SET FREQUENCY <num>”为其指定在这个 workload 中的执行频率。db2advis 会根据每个查询在这个 workload 的频率指数进行权衡来给出索引的创建建议，从而达到整个 workload 的性能最优。

**2.3** **db2batch**

前面介绍的工具和命令只提供了查询的估算代价，但有些时候估算代价和实际的执行时间并不是完全呈线形关系，有必要实际执行这些[查询](http://cpro.baidu.com/cpro/ui/uijs.php?adclass=0&app_id=0&c=news&cf=1001&ch=0&di=128&fv=18&is_app=0&jk=e555334991bdbf8d&k=%B2%E9%D1%AF&k0=%B2%E9%D1%AF&kdi0=0&luki=5&mcpm=0&n=10&p=baidu&q=mx163_cpr&rb=0&rs=1&seller_id=1&sid=8dbfbd91493355e5&ssp2=1&stid=9&t=tpclicked3_hc&td=2004887&tu=u2004887&u=http://www.xker.com/page/e2010/0424/96153.html&urlid=0" \t "http://www.xker.com/page/e2010/0424/_blank)。db2batch 就是这样一个 Benchmark 工具，它能够提供从准备到查询完成中各个阶段所花费地具体时间，CPU 时间，以及返回的记录。

用法：

db2batch -d <db\_name> -a <user>/<password> -i <time\_condition> -f <sql.file> -r <output>

-r result\_file 指定存放命令结果的输出文件

-i short|long|complete 指定从哪个方面测量所花费的时间。short 测量运行每条语句所花费的时间。long 测量运行每条语句所花费的时间，包括语句之间的开销。complete 测量运行每条语句所花费的时间，分别报告准备、执行和取数据的时间。

举例：

db2batch -d zpstst -a zpstst/zpstst -i complete -f lab.sql -r output.txt

运行结果范例：

\* Summary Table:  
Type      Number      Repetitions Total Time (s) Min Time (s)   ...  
--------- ----------- ----------- -------------- --------------   
Statement           1           1       0.052655       0.052655 ...  
Statement           2           1       0.004518       0.004518 ...  
...Max Time (s)   Arithmetic Mean Geometric Mean Row(s) Fetched Row(s) Output   
   -------------- --------------- -------------- -------------- -------------   
...      0.052655        0.052655       0.052655              5             5   
...      0.004518        0.004518       0.004518              8             5                                              
\* Total Entries:              2                 
\* Total Time:                 0.057173 seconds  
\* Minimum Time:               0.004518 seconds  
\* Maximum Time:               0.052655 seconds  
\* Arithmetic Mean Time:       0.028587 seconds  
\* Geometric Mean Time:        0.015424 seconds

-- db2batch.sql

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

--#SET PERF\_DETAIL 3

--#SET ROWS\_OUT 5

-- This query lists employees, the name of their department

-- and the number of activities to which they are assigned for

-- employees who are assigned to more than one activity less than

-- full-time.

--#COMMENT Query 1

select lastname, firstnme,

deptname, count(\*) as num\_act

from employee, department, emp\_act

where employee.workdept = department.deptno and

employee.empno = emp\_act.empno and

emp\_act.emptime < 1

group by lastname, firstnme, deptname

having count(\*) > 2;

--#SET PERF\_DETAIL 1

--#SET ROWS\_OUT 5

--#COMMENT Query 2

select lastname, firstnme,

deptname, count(\*) as num\_act

from employee, department, emp\_act

where employee.workdept = department.deptno and

employee.empno = emp\_act.empno and

emp\_act.emptime < 1

group by lastname, firstnme, deptname

having count(\*) <= 2;

1. **个人在工作中总结**

以下是个人在工作中有关DB2的总结，主要是一些DB2语句的应用以及注意事项。

**3.1 DB2的增删改查**

**3.1.1 插入数据篇**

1、插入单条记录方法一

db2 “insert into tabname (field1,field2,....) values (value1,value2,....)”

2、插入单条记录方法二

db2 “insert into tabname values (value1,value2,....)

注意：这里必须的value值必须是表的全部字段值

3、插入多条记录

db2 “insert into tabname (field1,field2,....) values (value1,value2,....),(value1’,value2’,....),....”

1. 将一个表记录插入另一张表

db2 “insert into tabname1 select \*from tabname2”

或者db2 “insert into tabname1 values ( select \* from tabname2 )”

注意：两个表结构要一致

1. 将一个表的某些字段插入另一张表

db2 “insert into tabname1(field1,field2,....) select field1,field2 from tabname2”

**3.1.2 删除数据篇**

1. 删除满足条件的记录方法一

db2 “delete from tabname where condition”

注意：这条语句性能不好，特别是删除大量数据的时候

1. 删除满足条件的记录方法二

db2 “delete from ( select \*from tabname where condition )”

注意：这条语句性能比1的要好一些，当删除整个表记录时将where去掉即可

1. 可以在不删除表的情况下删除表所有记录

db2 “delete from tabname”

注意：表结构、属性和索引依然是完整的

1. 删除表所有记录

先drop table，然后再create table，有点麻烦了

db2 “drop table tabname”

db2 “create table tabname (field1(length1),field2(length2),....)”

1. 删除数据引发的思考

db2 “alter table tabname activate not logged initially with empty table”

注意：这种方法更快、更简单的实现了删除表所有的数据

**3.1.3 修改数据篇**

1. 修改指定条件的记录

db2 “update tabname set field1=’value1’,field2=’value2’,.... where conditions”

**3.1.4 查找数据篇**

**3.1.4.1 普通的查询**

1. 查找特定条件的记录

db2 “select \* from tabname where condition”

1. 列出某些字段的值

db2 “select field1,field2,....from tabname where condition”

1. 列出表的所有记录

db2 “select \*from tabname”

1. 列出表的前n条记录

db2 “select \* from tabname fetch first n rows only”

1. 一个稍微复杂点的语句

db2 “select \* from tabname where condition order by field1,field2,....asc fetch first n rows only”

注意：asc表示按字段正序，一般默认是正序可省略，desc表示按字段反序

1. 列出不重复的记录

db2 “select distinct field1,field2,.... from tabname”

1. 显示出表的记录数

db2 “select count(\*) from tabname”

1. 从多个表中查找有交集的记录

db2 “select field1,field2 from tabname1,tabname2 where tabname1.field1= tabname2.field2”

9、列出表中字段field值在value1和value2之间记录

db2 “select \*from tabname where field between value1 and value2”

1. 列出表中所有的字段field值以value开头的记录

db2 “select \* from tabname where field like ‘value%’”

1. 列出表中所有的字段field值是value1,value2的记录

db2 “select \* from tabname where field in ( value1,value2 )”

**3.1.4.2 子查询**

db2 “select field1,field2,.... from tabname where field3=(select field3 from tabname where condition)”

现有两个表，用户表USER（用户ID userid,公司ID companyid,用户电话 telno）和公司表COMPANY（公司ID companyid,公司电话telno）。现在让你查询公司电话是8888888的用户有哪些？

1. 非相关子查询

db2 “select \* from user where companyid in ( select companyid from company where tleno=’8888888’)”

1. 相关子查询

db2 “select \* from user as u where exists ( select \* from company as c where telno=’8888888’ and u.company=c.company)”

总结：以上两条语句的作用是相同的，对比后发现，非相关子查询的子句（也就是括号内的语句）可以单独执行；相关子查询的子句依赖外部条件，不能单独执行。其实非相关子查询无论从性能还是可读性都优越于相关子查询。

但是在某些情况下，非相关子查询可能有实现不了的情况，下面再举一例，现在让你将用户号码更新为公司号码？？

db2 “update user as u set telno=( select telno from company as c where u.company=c.companyid)”

这里便是相关子查询的一个用处，能用非相关子查询的尽量用非相关子查询。

**3.1.4.3 多字段查询**

依然采用上面的USER表，现在让你查找用户电话是110，公司ID是1234的用户？

db2 “select \* from user where telno=’110’ and companyid=’1234’”

也可以这么写：

db2 “select \* from user where (telno,companyid) = (‘110’,’1234’)”

多字段查找：

db2 “select \* from user where (telno,companyid) in ( select telno,companyid from user where telno=’110’ and companyid=’1234’ )” 括号内为子查询语句

在更新表的时候也能用到

db2 “update user set (telno,companyid)=( select telno,companyid from user ) where telno=’110’ and companyid=’1234’” 括号内为相关子查询

**3.2 JOIN 多表连接**

有时为了得到完整的结果，我们需要从两个或更多的表中获取结果。我们就需要执行 join。

**3.2.1 JOIN分类**

下面列出了您可以使用的 JOIN 类型，以及它们之间的差异。

(inner 可省略)JOIN: 如果表中有至少一个匹配，则返回行

LEFT JOIN: 即使右表中没有匹配，也从左表返回所有的行

RIGHT JOIN: 即使左表中没有匹配，也从右表返回所有的行

FULL JOIN: 只要其中一个表中存在匹配，就返回行

CORSS JOIN：同JOIN

**3.2.2分类的概念**

内链接就是对表A和表B以元组为单位做一个笛卡尔积，记为表C，然后在C中挑选出满足符合on 语句后边的限制条件的条目。

左连接就是在内连接的基础上，将A中有但C中没有的元组也加上。由于C的列数比A的列数多，所以这新增的元组左边照搬a，右边为null。

右链接就是在内连接的基础上，将B中有但C中没有的元组也加上。由于C的列数比B的列数多，所以这新增的元组右边照搬B，左边为null。

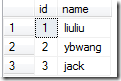
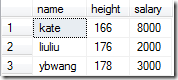
所谓的full join ，就是在join的结果d后边先加上left join 要加的项，再加上right join 要加的项

所谓的交叉连接(select \* from tableName1 cross join tableName2)和select \* from tableName1, tableName2是一样的，都是得到笛卡尔积。可以用select \* from table1,table2 where....模拟join操作。

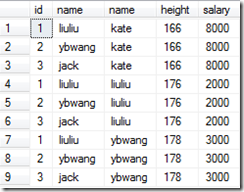
**3.2.3 举例**

现在有两个表A、B

A B

1、db2 “select \* from A cross join B”



1. db2 “select \* from A join B on A.name = B.name”

也可以这么写：

db2 “select E.id,E.name,D.name,D.height,D.salary from A E join B D on E.name = D.name”

\*\*\*温馨提示\*\*\*：在遇到表名很复杂写起来很麻烦而且在语句中频繁使用的情况下，可以考虑使用别名的方法来简化语句

就是对笛卡尔积，也即上边的cross join的结果进行进一步筛选，选出满足A.name=B.name的行。

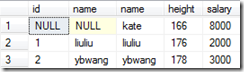
只有两行符合条件

IMG_256

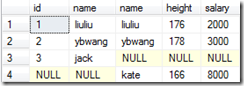
1. db2 “select \* from A left join B on A.name = B.name”



1. db2 “select \* from A right join B on A.name = B.name”



5、db2 “select \* from A full join B on A.name = B.name”



总结：在使用JOIN的时候特别容易出错，所以在需要多表连接时尽量使用不使用join，而使用将连接条件定义为where子句的查询，如：db2 “select \* from A,B where A.NO = B.NO”

使用这种方式也不是没有缺点，它很容易产生笛卡尔乘积，从而使结果倍增，即使你指定了正确的连接条件，如果条件不唯一的话也会产生笛卡尔乘积，如果这时使用分组统计，很可能产生不正确的结果，所以也要小心使用。

**3.2.4 外延知识**

要想正确使用Join语句，有2个知识点是必须掌握的。

1. SQL语句执行的顺序

FROM

JOIN ON

WHERE

GROUP BY

HAVING

SELECT

ORDER BY

FETCH FIRST

2、ON 和 WHERE 的区别：ON是用来定义连接条件的，WHERE用来过滤结果集

**3.3 VALUES**

Values是用来定义临时集合。

简单的有插入数据与Insert into 结合，可以插入一条也可以插入多条，甚至可以插入整个表。

这里介绍一种特殊的用法。

现有一张表USER，字段有：name姓名,department部门,birthday生日，其中department值有：1-市场部、2-管理部、3-研发部。

现在给你以下条件，让你查出姓名：

部门 生日

市场部 1949-10-1

管理部 1978-12-12

研发部 1982-11-23

方法一：新建一个表，把以上条件导入，然后再查询

db2 “Create table tmp ( department integer, birthday date )”

db2 “insert into tmp (department,birthday) values (1,’1949-1--1’),(2,’1978-12-12’),(3,’1982-11-23’)”

db2 “select \* from user where (department,birthday) in ( select department,birthday from tmp )”

缺点：麻烦

方法二：用values定义一个临时表即可

db2 “select \* from user where (department,birthday) in ( values (1,’1949-1--1’), (2,’1978-12-12’),(3,’1982-11-23’) )”

方法三：union 联合几条语句

db2 “select \* from user where department=1 and birthday=’1949-10-1’

union

select \* from user where department=2 and birthday=’1978-12-12’

union

select \* from user where department=3 and birthday=’1982-11-23’

**3.4 DB2语句的积累**

以下均是在工作中运用到的实例语句，仅列出一部分

1. Db2look -d zpstst -t zps\_ssdj\_w -e -o zps\_ssdj\_w.sql” 将数据库zpstst里表zps\_ssdj\_w的建表语句定向到sql脚本文件zps\_ssdj\_w.sql
2. Db2 “export to zps\_hdw.del of del select \*from zps\_hdw” 将表zps\_hdw数据导出到del文件zps\_hdw.del
3. Db2 “import from zps\_hdw.del of del insert into zps\_tmp” 将del文件zps\_hdw里数据导入到表zps\_tmp
4. Db2 -tvf zps\_zhang.sql 执行sql脚本文件(注意：文件里的sql语句要以分号结尾)
5. Db2 connect to zpstst 连接到本地数据库zpstst
6. Db2 describe table zps\_sys” 描述表zps\_sys的结构
7. Db2 list tablespaces show detail列出所有表空间详细信息,去掉show detail不是详细信息
8. Db2 backup db databasename 备份数据库databasename(数据库在使用的时候不能备份)
9. Db2 get dbm cfg 查看数据库参数
10. Db2 get db cfg databasename 查看当前数据库参数
11. Db2 drop table zps\_zhang\_employ” 彻底删除表zps\_zhang\_employ
12. Db2 list tables 列出当前数据包含的所有表名
13. Db2 “create database zhang” 创建数据库zhang
14. Db2 “drop database zhang” 删除数据库zhang
15. Db2 “create table zps\_zhang (field1(length),....)” 创建表zps\_zhang
16. Db2 “drop table tabname” 删除表
17. Db2 list active databases 列出活动的数据库和连接数(当前启用的数据库)
18. Db2 list database directory 列出所以数据库
19. Db2 connect reset 断开与当前数据库的连接
20. Db2start 启动数据库
21. Db2stop 停止数据库
22. Db2 start database dbname 启动数据库管理器
23. Db2 “describe indexes for table zps\_ssdj\_w show detail” 查看表zps\_ssdj\_w的索引信息
24. Db2 “create index I\_ZPS\_SSDJ\_W4 on zps\_ssdj\_w(field1,field2,....)” 在表zps\_ssdj\_w 中创建索引I\_ZPS\_SSDJ\_W4，可以是单索引，也可以是多列索引
25. Db2 “drop index I\_ZPS\_SSDJ\_W4” 删除索引I\_ZPS\_SSDJ\_W4，不需要指定表就可以删除，和sql有点区别
26. Db2expln -d zpstst -f lab.sql -g -t 查看lab.sql文件里db2语句的执行详细信息
27. Db2expln -d zpstst -f lab.sql -g -o lab.txt 将执行情况定向到lab.txt文件里
28. Db2advis -d zpstst -i lab.sql -o lab.exp 将提高查询性能的建议定向到lab.exp文件中，这里注意lab.sql语句要以；结尾

**3.5 隔离机制**

DB2公有四中隔离级别，由高到低分别是：

1. 重复读(RR,Repeatable Read) 在一个事务中进行查询时，不允许任何对这个查询表的数据修改；
2. 读稳定性(RS,Read Stability) 在一个事务中进行查询时，不允许读取其他事务update的数据，允许读取到其他事务提交的新增数据；

3、游标稳定性(CS,Cursor Stability) 在一个事务中进行查询时，允许读取提交前的数据，数据提交后，当前查询就可以读取到数据，update数据的时候并不锁表)--默认的隔离级别；

4、未落实的读(UR,Uncommited Read) 就是俗称“脏读“，在没有提交数据的时候能够读到更新的数据；

隔离级别越低，并发性越好，但是导致的并发性问题也越多。DB2可以在Session(会话级)、Connection(连接级)、Statement(语句级)设定隔离级别。一般，我们在执行select语句的时候，为了获得最大的并发性，可以容忍一些数据不一致，将隔离级别设置为最低级别UR，如下：

Db2 “select \* from tabname with ur”

**3.6 ALTER TABLE 语句**

ALTER TABLE 语句可以让数据库的设计者或设计人员在表创建以后修改

它的结构。该语句只可以用来增加或修改表中的某一列。

举例表如下：

create table emp (id int, name char(20),salary decimal(18,2)) 创建表emp

**3.6.1增加一列**

Db2 “alter table emp add sex char(1)” 增加sex这一列

**3.6.2更改字段类型**

1、Db2 “alter table emp alter sex set data type char(3)” 将sex这一列修改为char(3)

2、Db2 “alter table emp alter sex set data type char(2)” 将sex这列修改回char(2)

3、Db2 “insert into emp values (1,'smith',18.23,'22')” 向emp表插入一条记录

4、Db2 “alter table emp alter sex set data type char(3)” 在有记录的情况下，将sex这列修改为char(3),成功

5、Db2 “alter table emp alter sex set data type char(1)” 在有记录的情况下，将sex这列修改为char(1),报错：因为sex字段已有数据22至少两位来存储

6、Db2 “alter table emp alter sex set data type decimal(18,2)” 在有记录的情况下，将sex这列修改为decimal(18,2),成功(原来是'22'被转换成了22.00)

7、Db2 “alter table emp alter sex set data type char(2)” 将sex这列修改回char(2)失败---colunm sex that are not compatible with the existing colunm，SQLSTATE=42837。

8、Db2 “alter table emp alter sex set data type char(5)” 将sex这列修改为char(5)成功(原来的数字22.00被转换成了字符'22.00',5个字节)

9、delete from emp;  insert into  emp values (1,'smith',18.23,'dd') 删除原来的数据，并插入新的数据

10、Db2 “alter table emp alter sex set data type decimal(18,2)” 在有记录的情况下，将sex这列修改为decimal(18,2),失败---“指定的属性与现有列不兼容”('dd'和数字不兼容)

注意：执行上面的语句（执行几次就会报错了，再也执行不了了）会发生这种情况：表的状态对于操作失效，SQLSTATE=550019。

解读如下：

表状态：处于“REORG 暂挂”状态。  
导致的原因：在执行包含 REORG 建议的操作的 alter TABLE语句后，可能会发生这种情况。  
解决方案：使用 REORG TABLE 命令重组表

Db2 “REORG TABLE tablename”

总结：

(a)首先要保证目标类型和要当前表中所有的数据类型都要兼容，比如如果要更改char为decimal，那么原先的字段就不能出现字符，比如abcd等，只能出现数据型字符，比如1,2,3等。否则db2就报错，说“格式不兼容”。

(b)如果你要更改的只是size大小，而数据的类型不变，那么要保证字段是增加的，比如从char(5)到char(10),decimal(18,2)到decimal(19,2),这样是没问题的。

如果想缩减字段的size，那么就要保证所有表中的数据都小于你缩减后的值，否则也会报错。此时如果有需求必须要改变，那么就只能删除列，然后重建列了。当然重建表也可以。

**3.6.3删除列**

Db2 “alter table emp drop sex”

**3.6.4为字段添加默认值**

Db2 “alter table emp add sex char(1)”

Db2 “alter table emp alter sex set default 'm'”

注意：这个在公司电脑里运行是成功的，但是默认值‘m’并没有如期的插入

**3.6.5添加带默认值的字段**

Db2 “alter table emp add addr char(20) not null with default 'beijing'”

另外要注意的是，无论是添加，删除，还是更改字段的时候，为了保险起见，最好都运行一下reorg命令

Db2 reorg table table\_name

否则可能会造成其他操作时报错。

**3.6.6 NOT NULL和NULL转化**

1. 将 NOT NULL转化为NULL

Db2 “alter table tabname alter fileldname drop not null”

1. 将NULL转换为NOT NULL

Db2 “alter table tabname alter fieldname set not null”

**LINUX篇**

1. **Vim 基础命令**

以:和/开头的命令都有历史纪录，可以首先键入:或/然后按上下箭头来选择某个历史命令。

**1.1 启动vim**

在命令行窗口中输入以下命令即可：

vim ：直接启动vim

vim filename ：打开vim并创建名为filename的文件

**1.2 文件命令**

1、 vim file 打开单个文件

2、vim file1 file2 file3 ... 同时打开多个文件

3、:open file 在vim窗口中打开一个新文件

4、:split file 在新窗口中打开文件

5、:bn 切换到下一个文件

6、:bp 切换到上一个文件

7、:args 查看当前打开的文件列表，当前正在编辑的文件会用[]括起来。

8、Ctrl+g 查看当前打开的文件，并且还显示当前行数以及当前位置

**1.3 vim的模式**

1、正常模式（按Esc或Ctrl+[进入） 左下角显示文件名或为空

2、插入模式（按i键进入） 左下角显示--INSERT--

3、可视模式（按v键进入） 左下角显示--VISUAL--

**1.4 插入命令**

1、i 在当前位置生前插入

2、I 在当前行首插入

3、a 在当前位置后插入

4、A 在当前行尾插入

5、o 在当前行之后插入一行

6、O 在当前行之前插入一行

**1.5 查找命令**

1、/text　　查找text，按n健查找下一个，按N健查找前一个。

2、?text　　查找text，反向查找，按n健查找下一个，按N健查找前一个。

3、vim中有一些特殊字符在查找时需要转义　　.\*[]^%/?~$

4、:set ignorecase　　忽略大小写的查找

5、:set noignorecase　　不忽略大小写的查找

6、查找很长的词，如果一个词很长，键入麻烦，可以将光标移动到该词上，按\*或#键即可以该单词进行搜索，相当于/搜索。而#命令相当于?搜索。

7、:set hlsearch　　高亮搜索结果，所有结果都高亮显示，而不是只显示一个匹配。

8、:set nohlsearch　　关闭高亮搜索显示

9、:nohlsearch　　关闭当前的高亮显示，如果再次搜索或者按下n或N键，则会再次高亮。

10、:set incsearch　　逐步搜索模式，对当前键入的字符进行搜索而不必等待键入完成。

11、:set wrapscan　　重新搜索，在搜索到文件头或尾时，返回继续搜索，默认开启。

12、:set num 显示行号

13、:set nonum 不显示行号

**1.6 替换命令**

当前行：光标所在行

1、:ra 将当前字符替换为a，当期字符即光标所在字符。

2、:s/old/new/ 用new替换old，替换当前行的第一个匹配

3、:s/old/new/g 用new替换old，替换当前行的所有匹配

4、:%s/old/new/ 用new替换old，替换所有行的第一个匹配，等同于：g/new/s//old/

5、:%s/old/new/g 用new替换old，替换整个文件的所有匹配，等同于：g/new/s//old/g

6、:n,$s/old/new/ 用new替换old，替换第n行到最后一行中每一行的第一个匹配

7、:n,$s/old/new/g 用new替换old，替换第n行到最后一行的所有匹配

8、:.,$s/old/new/ 用new替换old，替换当前行到最后一行中每一行的第一个匹配

9、:.,$s/old/new/g 用new替换old，替换当前行到最后一行的所有匹配

10、m,ns/old/new/g 用于将从m行开始至n行结束的所有old替换为new

10、:10,20 s/^/ /g 在第10行知第20行每行前面加四个空格，用于缩进。

11、ddp 交换光标所在行和其下紧邻的一行。

**1.7 移动命令**

1、h 左移一个字符

2、l 右移一个字符，这个命令很少用，一般用w代替。

3、k 上移一个字符

4、j 下移一个字符

以上四个命令可以配合数字使用，比如20j就是向下移动20行，5h就是向左移动5个字符，在Vim中，很多命令都可以配合数字使用，比如删除10个字符10x，在当前位置后插入3个！，3a！<Esc>，这里的Esc是必须的，否则命令不生效。

5、w 向前移动一个单词（光标停在单词首部），如果已到行尾，则转至下一行行首。此命令快，可以代替l命令。

6、b 向后移动一个单词 2b 向后移动2个单词

7、e，同w，只不过是光标停在单词尾部

8、ge，同b，光标停在单词尾部。

9、^ 移动到本行第一个非空白字符上。

10、0（数字0）移动到本行第一个字符上，

11、<HOME> 移动到本行第一个字符。同0健。

12、$ 移动到行尾 3$ 移动到下面3行的行尾

gg 移动到文件头。 = [[

13、G（shift + g） 移动到文件尾。 = ]]

14、f（find）命令也可以用于移动，fx将找到光标后第一个为x的字符，3fd将找到第三个为d的字符。

15、F 同f，反向查找。

跳到指定行，冒号+行号，回车，比如跳到240行就是 :240回车。另一个方法是行号+G，比如230G跳到230行。

16、Ctrl + e 向下滚动一行

17、Ctrl + y 向上滚动一行

18、Ctrl + d 向下滚动半屏

19、Ctrl + u 向上滚动半屏

20、Ctrl + f 向下滚动一屏

21、Ctrl + b 向上滚动一屏

**1.8 撤销和重做**

1、u 撤销（Undo）

2、U 撤销对整行的操作

3、Ctrl + r 重做（Redo），即撤销的撤销。

**1.9 删除命令**

1、x 删除当前字符

2、nx 删除从当前光标所在位置开始向右的n个字符

3、X 删除当前字符的前一个字符。X=dh

4、nX 删除从光标所在位置之前的那个字符开始向左的n个字符

4、dl 删除当前字符， dl=x

5、dh 删除前一个字符

6、dd 删除当前行，在dd前加数字n，表示删除当前行 及其后n-1行内容

7、dj 删除上一行

8、dk 删除下一行

9、10d 删除当前行开始的10行。

1. D或d$ 删除从光标所在处开始到行尾的内容
2. d0 删除从光标前一个字符开始到行首的内容
3. dw 删除一个单词。若光标处在某个词的中间，则从光标所在位置开始删至词尾，痛dd一样，可在dw 前加数字n，表示删除n个指定的单词

12、kdgg 删除当前行之前所有行（不包括当前行）

13、jdG（jd shift + g） 删除当前行之后所有行（不包括当前行）

14、:1,10d 删除1-10行

15、:11,$d 删除11行及以后所有的行

16、:1,$d 删除所有行

17、J(shift + j)　　删除两行之间的空行，实际上是合并两行。

温馨提示：如果用户不小心误删了数据，不要紧，vim提供了恢复误操作的命令，并且乐意将恢复的内容意动，放在文本的任何地方。恢复命令用”np”，其中n为需要恢复的次数。例如使用dd命令删除了一行内容，然后使用”2p”命令，将光标放在你所期望的位置，则被删除的内容会被重新插入两遍。这样就实现了文本的任意移动操作。

**2.0 拷贝和粘贴**

1、yy 拷贝当前行

2、nyy 拷贝当前及其后的n-1行，比如2yy拷贝当前行及其下一行。

3、p 在当前光标后粘贴,如果之前使用了yy命令来复制一行，那么就在当前行的下一行粘贴。

4、shift+p 在当前行前粘贴

5、:1,10 co 20 将1-10行插入到第20行之后。

6、:1,$ co $ 将整个文件复制一份并添加到文件尾部。

正常模式下按v（逐字）或V（逐行）进入可视模式，然后用jklh命令移动即可选择某些行或字符，再按y即可复制

7、ddp交换当前行和其下一行

8、xp交换当前字符和其后一个字符

**2.1 剪切命令**

正常模式下按v（逐字）或V（逐行）进入可视模式，然后用jklh命令移动即可选择某些行或字符，再按d即可剪切

1、ndd 剪切当前行之后的n行。利用p命令可以对剪切的内容进行粘贴

2、:1,10d 将1-10行剪切。利用p命令可将剪切后的内容进行粘贴。

3、:1, 10 m 20 将第1-10行移动到第20行之后。

**2.2 退出命令**

1、:wq 保存并退出

2、ZZ 保存并退出

3、:q! 强制退出并忽略所有更改

4、:e! 放弃所有修改，并打开原来文件。

**2.3 窗口命令**

1、:split或new 打开一个新窗口，光标停在顶层的窗口上

2、:split file或:new file 用新窗口打开文件

注意：split打开的窗口都是横向的，使用vsplit可以纵向打开窗口。

3、Ctrl+ww 移动到下一个窗口

4、Ctrl+wj 移动到下方的窗口

5、Ctrl+wk 移动到上方的窗口

关闭窗口

6、:close 最后一个窗口不能使用此命令，可以防止意外退出vim。

7、:q 如果是最后一个被关闭的窗口，那么将退出vim。

8、ZZ 保存并退出。

关闭所有窗口，只保留当前窗口

9、:only

录制宏

按q键加任意字母开始录制，再按q键结束录制（这意味着vim中的宏不可嵌套），使用的时候@加宏名，比如qa。。。q录制名为a的宏，@a使用这个宏。

**2.4 执行shell命令**

1、:!command

2、:!ls 列出当前目录下文件

3、:!perl -c script.pl 检查perl脚本语法，可以不用退出vim，非常方便。

4、:!perl script.pl 执行perl脚本，可以不用退出vim，非常方便。

5、:suspend或Ctrl - Z 挂起vim，回到shell，按fg可以返回vim。

**2.5 注释命令**

perl程序中#开始的行为注释，所以要注释某些行，只需在行首加入#

1、3,5 s/^/#/g 注释第3-5行

2、3,5 s/^#//g 解除3-5行的注释

3、1,$ s/^/#/g 注释整个文档。

1. :%s/^/#/g 注释整个文档，此法更快。

**2.6 其他非编辑命令**

1、. 重复前一次命令

2、:set ruler?　　查看是否设置了ruler，在.vimrc中，使用set命令设制的选项都可以通过这个命令查看

3、:scriptnames　　查看vim脚本文件的位置，比如.vimrc文件，语法文件及plugin等。

4、:set list 显示非打印字符，如tab，空格，行尾等。如果tab无法显示，请确定用set lcs=tab:>-命令设置了.vimrc文件，并确保你的文件中的确有tab，如果开启了expendtab，那么tab将被扩展为空格。

5、:syntax 列出已经定义的语法项

6、:syntax clear 清除已定义的语法规则

7、:syntax case match 大小写敏感，int和Int将视为不同的语法元素

8、:syntax case ignore 大小写无关，int和Int将视为相同的语法元素，并使用同样的配色方案

1. **Linux常见命令总结**

以下所写内容均是针对自己在工作中常用到以及比较重要的命令，至于其他一些不常用以及简单的指令读者可以自行去看，这里不作介绍。如果你对Linux命令已经掌握了可以一笑弃之；不熟悉这些命令的初学者建议你先去看详细命令手册，再来阅读这部分内容，效果会更佳。

**2.1 查看文件命令--head和tail**

当需要查看一个文件的头部或者尾部时，head和tail命令可以非常方便的完成操作。Head命令用于查看一个文本文件的开头部分；而tail命令则用于显示文本文件的末尾几行。以下是常用的例子：

1、head file.txt 显示文件file.txt的前十行内容

2、head -n 20 file.txt 显示文件file.txt的前二十行内容

3、tail file.txt 显示文件file.txt的后十行内容

4、tail -n 20 file.txt 显示文件file.txt 的后二十行内容

5、tail -f file.txt 显示文件file.txt 的后十行内容，并在文件内容增加后，自动显示新增的文件内容

注意：我工作用的是IBM的AIX系统，在执行这两个指令时需要在数字n之前加上-不然会报错，读者可以根据自己的系统自行验证是否需要加

**2.2 查找文件命令--find**

find 命令是查找文件系统中的指定文件。

举例：

1. find . -type f -name “\*.log” -perm 0666 -print|xargs grep -i bin

查找当前目录下普通文件、文件名以.log结尾的、文件属主具有读、写权限、含有关键词bin（不区分大小写）的文件，并打印出所有满足条件的结果（结果中含有文件以及含有bin的行）

1. find . -mtime -90 -print

查找更改时间距今90天之内的文件，并打印所有满足条件的结果

3、find . -mtime +90 -print

查找更改时间距今在90天之前的文件，并打印所有满足条件的结果

4、find /etc -type f -name 'rc\*' -exec ls -l { } \；

查找/etc目录下文件名以rc开头的普通文件，并对查找结果执行ls -l命令

**2.3 文件内容查找命令--grep**

grep用于查找符合指定模式匹配的文件。

举例：

1. grep -in modify \*

从当前目录下查找含有modify（不区分大小写）的文件行

1. grep -c modify \*

从当前目录下查找含有modify的文件，只显示文件名还有对应含有nodify的行数

1. grep -r zhang /home/zpstst

在/home/zpstst目录及其子目录下的所有文件中查找zhang字符串

1. ps -ef | grep GGJY

查找进程中包含GGJY的所有进程信息

**2.4 用户和用户组命令**

命令有：groupadd、groupdel、useradd、userdel。

下面出现的命令均要在超级用户中执行。

举例：

1. groupadd test 添加用户组test

2、groupdel test 删除用户组test

3、useradd zhang 添加用户zhang

4、useradd -g jack rose 添加一个rose用户，把该用户归属于用户组jack

5、passwd jack 设置jack用户的密码

6、userdel zhang 删除用户zhang

提示：该用户默认的登录路径没有被删除，系统email邮箱也未被删除，需要手动删除

1. useradd -g grouptest -d /home/test -m -u 230

test添加一个test用户，把该用户归属于用户组groupname,d指定test的默认路径为/home/test,-m不存在路径则创建，-u指定用户号

8、userdel -r rose

删除rose用户,并删除rose用户在/home路径下的目录，/var/spool/mail下的email邮箱文件。当账户在使用时，会删除失败。

1. 将档案 file1.txt 的拥有者设为 users 群体的使用者 jessie :   
   chown jessie:users file1.txt   
   将目前目录下的所有档案与子目录的拥有者皆设为 users 群体的使用者 lamport :   
   chown -R lamport:users \*

温馨提示：在/etc/group目录下可以查看用户组以及用户的相关信息，

在/etc/passwd目录下可以查看用户密码的信息

在/home目录下可以查看用户名

在/var/spool/mail目录下也可以查到用户名

注意：在删除用户组时，一定要确认是否还有用户在该用户组下，否则会报错。

在删除用户时，一定要确保将上面提到的四个文件或者目录中的相关信息被删除。如果用

userdel username 这种方式仅仅能删除在/etc/passwd以及/etc/group文件下的数据，而/home以及/var/spool/mail目录下的数据并没有被删除，需要手动删除。想要一次性彻底删除秩序执行 userdel -r username就可以全部删除四个文件下username的数据。

**2.5 cut命令**

cut命令从文件的每一行剪切字节、字符和字段并将这些字节、字符和字段写至标准输出。

主要参数：

-b ：以字节为单位进行分割。这些字节位置将忽略多字节字符边界，除非也指定了 -n 标志。  
-c ：以字符为单位进行分割。  
-d ：自定义分隔符，默认为制表符。  
-f  ：与-d一起使用，指定显示哪个区域。  
-n ：取消分割多字节字符。仅和 -b 标志一起使用。如果字符的最后一个字节落在由 -b 标志的 List 参数指示的<br />范围之内，该字符将被写出；否则，该字符将被排除。

举例：

1. who|cut -b 1-3,5 或者who|cut 5,1-3提取who输出中的第1,3,5个字节

注意：cut命令如果使用了-b选项，那么执行此命令时，cut会先把-b后面所有的定位进行从小到大排序，然后再提取

1. who|cut -b -3 提取who输出的第一个字节到第三个字节
2. who|cut -b 3- 提取who输出的第三个字节到行尾
3. who|cut -b -3,3- 输出who的全部结果，第三个字节并不会重复输出
4. who|cut -c 1-3,5 提取who输出的第1,3,5个字符

注意：这里的-b和-c其实是不一样的参数，如果对含有中文的文本进行提取时两者的差异会显示出来，因为-b提取的是字节，而-c提取的是字符，用-b的提取中文的话容易出现乱码，可以使用-n选项，-n用于告诉cut不要将多字节字符拆开。

1. file.txt|cut -bn 1,2,3 （假设file.txt全是汉字）这时会输出file.txt的第一个汉字，因为此处覆盖了三个字节，而汉字是两个字节
2. file.txt|cut -bn 2 （假设file.txt全是汉字）什么都不会输出
3. cat /etc/passwd|head -n 5|cut -d : -f 1,3-5 提取passwd前五行，按：分隔符再提取1,3,5域
4. cat /etc/passwd|head -n 5|cut -d : -f -2 提取passwd前五行，按：分隔符再提取从1到2域

注意：如果文件里面的某些域是由若干个空格来间隔的，那么用cut就有点麻烦了，因为cut只擅长处理“以一个字符间隔”的文本内容

**2.6 sort命令**

sort将文件的每一行作为一个单位，相互比较，比较原则是从首字符向后，依次按ASCII码值进行比较，最后将他们按升序输出。

主要参数：

-c 测试文件是否已经被排序  complete

-k 指定排序的域   k1  k2nr 往后加命令即可

-m 合并两个已排序的文件   merge

-n 根据数字大小进行排序  number

-o [输出文件] 将输出写到指定的文件，相当于将输出重定向到指定文件  open

-r 将排序结果逆向    reverse

-t 改变域分隔符   eg：-t:   一般最新定义

-u 去除结果中的重复行     unique  -u

-f会将小写字母都转换为大写字母来进行比较，亦即忽略大小写

-c会检查文件是否已排好序，如果乱序，则输出第一个乱序的行的相关信息，最后返回1

-C会检查文件是否已排好序，如果乱序，不输出内容，仅返回1

-M会以月份来排序，比如JAN小于FEB等等

-b会忽略每一行前面的所有空白部分，从第一个可见字符开始比较。

举例：

1. sort ex.txt 按照每行首字符的ASCII值从小到大输出
2. sort -u ex.txt 按照每行首字符的ASCII值从小到大输出，相同的行只输出一行
3. sort -r ex.txt 按照每行首字符的ASCII值从大到小输出
4. sort -r number.txt -o number.txt 按照每行首字符的ASCII值从大到小输出并将结果重新写如源文件（覆盖之前的内容）注意：这里不能用>重定向，否则文件置空
5. sort -n number.txt 将文件按照数字排序
6. sort -n -k 2 -t : facebook.txt 使用冒号作为间隔符，并针对第二列来进行数值升序排序
7. sort -nc file.txt 检查文件是否已经按照数字排序
8. sort -m f1.txt f2.txt -o f3.txt 将f1.txt和f2.txt两个文件合并到f3.txt

注意：你有没有遇到过10比2小的情况。我反正遇到过。出现这种情况是由于排序程序将这些数字按字符来排序了，排序程序会先比较1和2，显然1小，所以就将10放在2前面喽。这也是sort的一贯作风。我们如果想改变这种现状，就要使用-n选项，来告诉sort，“要以数值来排序”！

1 准备素材

$ cat facebook.txt  
google 110 5000  
baidu 100 5000  
guge 50 3000  
sohu 100 4500

第一个域是公司名称，第二个域是公司人数，第三个域是员工平均工资。

2 我想让这个文件按公司的字母顺序排序，也就是按第一个域进行排序：（这个facebook.txt文件有三个域）

$ sort -t ‘ ‘ -k 1 facebook.txt  
baidu 100 5000  
google 110 5000  
guge 50 3000  
sohu 100 4500

看到了吧，就直接用-k 1设定就可以了。（其实此处并不严格，稍后你就会知道）

3 我想让facebook.txt按照公司人数排序

$ sort -n -t ‘ ‘ -k 2 facebook.txt  
guge 50 3000  
baidu 100 5000  
sohu 100 4500  
google 110 5000

此处出现了问题，那就是baidu和sohu的公司人数相同，都是100人，这个时候怎么办呢？按照默认规矩，是从第一个域开始进行升序排序，因此baidu排在了sohu前面。

4  我想让facebook.txt按照公司人数排序 ，人数相同的按照员工平均工资升序排序：

$ sort -n -t ‘ ‘ -k 2 -k 3 facebook.txt  
guge 50 3000  
sohu 100 4500  
baidu 100 5000  
google 110 5000

我们加了一个-k2 -k3就解决了问题。sort支持这种设定，就是说设定域排序的优先级，先以第2个域进行排序，如果相同，再以第3个域进行排序。（如果你愿意，可以一直这么写下去，设定很多个排序优先级）

5 我想让facebook.txt按照员工工资降序排序，如果员工人数相同的，则按照公司人数升序排序：（这个有点难度喽）

$ sort -n -t ‘ ‘ -k 3r -k 2 facebook.txt  
baidu 100 5000  
google 110 5000  
sohu 100 4500  
guge 50 3000

此处有使用了一些小技巧，在-k 3后面加上了一个小写字母r。r和-r选项的作用是一样的，就是表示逆序。因为sort默认是按照升序排序的，所以此处需要加上r表示第三个域（员工平均工资）是按照降序排序。此处你还可以加上n，就表示对这个域进行排序时，要按照数值大小进行排序，举个例子吧：

$ sort -t ‘ ‘ -k 3nr -k 2n facebook.txt  
baidu 100 5000  
google 110 5000  
sohu 100 4500  
guge 50 3000

我们去掉了最前面的-n选项，而是将它加入到了每一个-k选项中了。

6 -k选项的具体语法格式

要继续往下深入的话，就不得不来点理论知识。你需要了解-k选项的语法格式，如下：

[ FStart [ .CStart ] ] [ Modifier ] [ , [ FEnd [ .CEnd ] ][ Modifier ] ]

这个语法格式可以被其中的逗号（“，”）分为两大部分，Start部分和End部分。

先给你灌输一个思想，那就是“如果不设定End部分，那么就认为End被设定为行尾”。这个概念很重要的，但往往你不会重视它。

Start部分也由三部分组成，其中的Modifier部分就是我们之前说过的类似n和r的选项部分。我们重点说说Start部分的FStart和C.Start。

C.Start也是可以省略的，省略的话就表示从本域的开头部分开始。之前例子中的-k 2和-k 3就是省略了C.Start的例子喽。

FStart.CStart，其中FStart就是表示使用的域，而CStart则表示在FStart域中从第几个字符开始算“排序首字符”。

同理，在End部分中，你可以设定FEnd.CEnd，如果你省略.CEnd，则表示结尾到“域尾”，即本域的最后一个字符。或者，如果你将CEnd设定为0(零)，也是表示结尾到“域尾”。

7 突发奇想，从公司英文名称的第二个字母开始进行排序：

$ sort -t ‘ ‘ -k 1.2 facebook.txt  
baidu 100 5000  
sohu 100 4500  
google 110 5000  
guge 50 3000

看，我们使用了-k 1.2，这就表示对第一个域的第二个字符开始到本域的最后一个字符为止的字符串进行排序。你会发现baidu因为第二个字母是a而名列榜首。sohu和 google第二个字符都是o，但sohu的h在google的o前面，所以两者分别排在第二和第三。guge只能屈居第四了。

8 又突发奇想，，只针对公司英文名称的第二个字母进行排序，如果相同的按照员工工资进行降序排序：

$ sort -t ‘ ‘ -k 1.2,1.2 -k 3,3nr facebook.txt  
baidu 100 5000  
google 110 5000  
sohu 100 4500  
guge 50 3000

由于只对第二个字母进行排序，所以我们使用了-k 1.2,1.2的表示方式，表示我们“只”对第二个字母进行排序。（如果你问“我使用-k 1.2怎么不行？”，当然不行，因为你省略了End部分，这就意味着你将对从第二个字母起到本域最后一个字符为止的字符串进行排序）。对于员工工资进行排 序，我们也使用了-k 3,3，这是最准确的表述，表示我们“只”对本域进行排序，因为如果你省略了后面的3，就变成了我们“对第3个域开始到最后一个域位置的内容进行排序” 了。

9 在modifier部分还可以用到哪些选项？

可以用到b、d、f、i、n 或 r。

其中n和r你肯定已经很熟悉了。

b表示忽略本域的签到空白符号。

d表示对本域按照字典顺序排序（即，只考虑空白和字母）。

f表示对本域忽略大小写进行排序。

i表示忽略“不可打印字符”，只针对可打印字符进行排序。（有些ASCII就是不可打印字符，比如\a是报警，\b是退格，\n是换行，\r是回车等等）

10 思考思考关于-k和-u联合使用的例子：

$ cat facebook.txt  
google 110 5000  
baidu 100 5000  
guge 50 3000  
sohu 100 4500

这是最原始的facebook.txt文件。

$ sort -n -k 2 facebook.txt  
guge 50 3000  
baidu 100 5000  
sohu 100 4500  
google 110 5000

$ sort -n -k 2 -u facebook.txt  
guge 50 3000  
baidu 100 5000  
google 110 5000

当设定以公司员工域进行数值排序，然后加-u后，sohu一行就被删除了！原来-u只识别用-k设定的域，发现相同，就将后续相同的行都删除。

$ sort  -k 1 -u facebook.txt  
baidu 100 5000  
google 110 5000  
guge 50 3000  
sohu 100 4500

$ sort  -k 1.1,1.1 -u facebook.txt  
baidu 100 5000  
google 110 5000  
sohu 100 4500

这个例子也同理，开头字符是g的guge就没有幸免于难。

$ sort -n -k 2 -k 3 -u facebook.txt  
guge 50 3000  
sohu 100 4500  
baidu 100 5000  
google 110 5000

这里设置了两层排序优先级的情况下，使用-u就没有删除任何行。原来-u是会权衡所有-k选项，将都相同的才会删除，只要其中有一级不同都不会轻易删除的，你可以自己加一行sina 100 4500试试看。

11 最诡异的排序：

$ sort -n -k 2.2,3.1 facebook.txt  
guge 50 3000  
baidu 100 5000sohu 100 4500  
google 110 5000

以第二个域的第二个字符开始到第三个域的第一个字符结束的部分进行排序。

第一行，会提取0 3，第二行提取00 5，第三行提取00 4，第四行提取10 5。

又因为sort认为0小于00小于000小于0000….

因此0 3肯定是在第一个。10 5肯定是在最后一个。但为什么00 5却在00 4前面呢？（你可以自己做实验思考一下。）

答案揭晓：原来“跨域的设定是个假象”，sort只会比较第二个域的第二个字符到第二个域的最后一个字符的部分，而不会把第三个域的开头字符纳入比较范围。当发现00和00相同时，sort就会自动比较第一个域去了。当然baidu在sohu前面了。用一个范例即可证实：

$ sort -n -k 2.2,3.1 -k 1,1r facebook.txt  
guge 50 3000  
sohu 100 4500  
baidu 100 5000  
google 110 5000

12 有时候在sort命令后会看到+1 -2这些符号，这是什么东东？

关于这种语法，最新的sort是这么进行解释的：

On older systems, `sort’ supports an obsolete origin-zero syntax `+POS1 [-POS2]‘ for specifying sort keys.  POSIX 1003.1-2001 (\*note Standards conformance::) does not allow this; use `-k’ instead.

原来，这种古老的表示方式已经被淘汰了，以后可以理直气壮的鄙视使用这种表示方法的脚本喽！

（为了防止古老脚本的存在，在这再说一下这种表示方法，加号表示Start部分，减号表示End部分。最最重要的一点是，这种方式方法是从0开始计数的，以前所说的第一个域，在此被表示为第0个域。以前的第2个字符，在此表示为第1个字符。明白？）

**2.7 uniq命令**

uniq [ -c | -d | -u ] [ -f Fields ] [ -s Characters ] [ -Fields ] [ +Characters ] [ InFile [ OutFile ] ]

uniq命令用于去除文本文件中的重复行，这类似于sort命令的-u选项

sort -u命令时，只显示所有重复记录中的一条，不间隔的出现也可以

uniq命令去除的重复行必须是连续重复出现的行，中间不能夹杂任何其他文本行

uniq 命令删除文件中的重复行。 uniq 命令读取由 InFile 参数指定的标准输入或文件。该命令首先比较相邻的行，然后除去第二行和该行的后续副本。重复的行一定相邻。（在发出 uniq 命令之前，请使用 sort 命令使所有重复行相邻，如：sort file.txt|uniq -u。）最后，uniq 命令将最终单独的行写入标准输出或由 OutFile 参数指定的文件。InFile 和 OutFile 参数必须指定不同的文件。

输入文件必须是文本文件。文本文件是包含组织在一行或多行中的字符的文件。这些行的长度不能超出 2048 个字节（包含所有换行字符），并且其中不能包含空字符。

主要参数

-c 打印每行在文本中重复出现的次数   count

-d 只显示有重复的记录，每个重复记录只出现一次

-u 只显示没有重复的记录   unique

应用：文件经过处理后在它的输出文件中可能会出现重复的行。例如，使用cat命令将两个文件合并后，再使用sort命令进行排序，就可能出现重复行。这时可以使用uniq命令将这些重复行从输出文件中删除，只留下每条记录的唯一样本。

**2.8 cat命令**

cat命令的用途是连接文件或标准输入并打印。这个命令常用来显示文件内容，或者将几个文件连接起来显示，或者从标准输入读取内容并显示，它常与重定向符号配合使用。

cat主要有三大功能：

1.一次显示整个文件:cat filename

2.从键盘创建一个文件:cat > filename 只能创建新文件,不能编辑已有文件.

3.将几个文件合并为一个文件:cat file1 file2 > file

cat -nb file1 file2 > file3

将file1和file2的文件内容加上行号（空白行不加）之后将内容附加到file3中

注意：tac是反向输出文件内容，与cat正好相反

**2.9 join命令**

功能说明：

将两个文件中，指定栏位内容相同的行连接起来。

语法：join [-i][-a<1|2>][-e<string>][-o<格式>] [-t<字符>][-v<1|2>][-1<栏位>][-2<栏位>][--help] [--version][文件1][文件2]

补充说明：

找出两个文件中，指定栏位内容相同的行，并加以合并，再输出到标准输出设备。

参数：

-a<1|2>  除了显示原来的输出内容之外，还显示指令文件中没有相同栏位的行。

-e<字符串>  若[文件1]与[文件2]中找不到指定的栏位，则在输出中填入选项中的字符串。

-i或--igore-case  比较栏位内容时，忽略大小写的差异。

-o<格式>  按照指定的格式来显示结果。

-t<字符>  使用栏位的分隔字符。

-v<1或2>  跟-a相同，但是只显示文件中没有相同栏位的行。

-1<栏位>  连接[文件1]指定的栏位。

-2<栏位>  连接[文件2]指定的栏位。

--help  显示帮助。

--version  显示版本信息。

指定输出字段：

-o <FILENO.FIELDNO> ...

其中fileno=1表示第一个文件，fileno=2表示第二个文件，fieldno表示字段序号，从1开始编号。默认会全部输出，但关键字列只输出一次。

比如：-o 1.1 1.2 2.2 表示输出第一个文件的第一个字段、第二个字段，第二个文件的第二个字段。

使用示例

示例一 内连接（忽略不匹配的行）

不指定任何参数的情况下使用join命令，就相当于数据库中的内连接，关键字不匹配的行不会输出。

[root@rhel55 linux]# cat month\_cn.txt

1       一月

2       二月

13      十三月，故意的

[root@rhel55 linux]# cat month\_en.txt

1       January

2       February

14      MonthUnknown

注：注意两个文件的内容，中文版的多了十三月，英文版的多了14月，这纯粹是为了方便演示。

[root@rhel55 linux]# join month\_cn.txt month\_en.txt

1 一月 January

2 二月 February

[root@rhel55 linux]#

示例二 左连接（又称左外连接，显示左边所有记录）

显示左边文件中的所有记录，右边文件中没有匹配的显示空白。

[root@rhel55 linux]# join -a1 month\_cn.txt month\_en.txt

1 一月 January

2 二月 February

13 十三月，故意的

[root@rhel55 linux]#

示例三 右连接（又称右外连接，显示右边所有记录）

显示右边文件中的所有记录，左边文件中没有匹配的显示空白。

[root@rhel55 linux]# join -a2 month\_cn.txt month\_en.txt

1 一月 January

2 二月 February

14 MonthUnknown

[root@rhel55 linux]#

示例四 全连接（又称全外连接，显示左边和右边所有记录）

[root@rhel55 linux]# join -a1 -a2 month\_cn.txt month\_en.txt

1 一月 January

2 二月 February

13 十三月，故意的

14 MonthUnknown

[root@rhel55 linux]#

示例五 指定输出字段

比如参数 -o 1.1 表示只输出第一个文件的第一个字段。

[root@rhel55 linux]# join -o 1.1 month\_cn.txt month\_en.txt

1

2

[root@rhel55 linux]# join -o 1.1 2.2 month\_cn.txt month\_en.txt

1 January

2 February

[root@rhel55 linux]# join -o 1.1 2.2 1.2 month\_cn.txt month\_en.txt

1 January 一月

2 February 二月

[root@rhel55 linux]# join -o 1.1 2.2 1.2 1.3 month\_cn.txt month\_en.txt

字段1.3并不存在

1 January 一月

2 February 二月

**2.10 tr命令**

tr用来从标准输入中通过替换或删除操作进行字符转换。tr主要用于删除文件中控制字符或进行字符转换。使用tr时要转换两个字符串：字符串1用于查询，字符串2用于处理各种转换。tr刚执行时，字符串1中的字符被映射到字符串2中的字符，然后转换操作开始。

带有最常用选项的tr命令格式为：  
tr -c -d -s ["string1\_to\_translate\_from"] ["string2\_to\_translate\_to"] < input-file

这里：  
-c 用字符串1中字符集的补集替换此字符集，要求字符集为ASCII。  
-d 删除字符串1中所有输入字符。  
-s 删除所有重复出现字符序列，只保留第一个；即将重复出现字符串压缩为一个字符串。  
input-file是转换文件名。虽然可以使用其他格式输入，但这种格式最常用。

字符范围  
指定字符串1或字符串2的内容时，只能使用单字符或字符串范围或列表。  
[a-z] a-z内的字符组成的字符串。  
[A-Z] A-Z内的字符组成的字符串。  
[0-9] 数字串。  
\octal 一个三位的八进制数，对应有效的ASCII字符。  
[O\*n] 表示字符O重复出现指定次数n。因此[O\*2]匹配OO的字符串。  
tr中特定控制字符的不同表达方式  
速记符含义八进制方式  
\a Ctrl-G  铃声\007  
\b Ctrl-H  退格符\010  
\f Ctrl-L  走行换页\014  
\n Ctrl-J  新行\012  
\r Ctrl-M  回车\015  
\t Ctrl-I  tab键\011  
\v Ctrl-X  \030

举例

1、cat file | tr "abc" "xyz" > new\_file

注意：这里，凡是在file中出现的"a"字母，都替换成"x"字母，"b"字母替换为"y"字母，"c"字母替换为"z"字母。而不是将字符串"abc"替换为字符串"xyz"。

2、使用tr命令“统一”字母大小写

cat file | tr [a-z] [A-Z] > new\_file 小写-->大写  
cat file | tr [A-Z] [a-z] > new\_file 大写-->小写

注意：cat file| tr [:lower:] [:upper:] > new\_file 小写-->大写

cat file| tr [:upper:] [:lower:] > new\_file 大写-->小写

这两条同样也能实现大小写的转换

3、把文件中的数字0-9替换为a-j

cat file | tr [0-9] [a-j] > new\_file

4、删除文件file中出现的"Snail"字符

cat file | tr -d "Snail" > new\_file

注意：这里，凡是在file文件中出现的'S','n','a','i','l'字符都会被删除！而不是紧紧删除出现的"Snail”字符串。

5、删除文件file中出现的换行'\n'、制表'\t'字符

cat file | tr -d "\n\t" > new\_file

不可见字符都得用转义字符来表示的，这个都是统一的。

6、删除“连续着的”重复字母，只保留第一个

cat file | tr -s [a-zA-Z] > new\_file

7、删除空行

cat file | tr -s "\n" > new\_file

8、删除Windows文件“造成”的'^M'字符

cat file | tr -d "\r" > new\_file

或者

cat file | tr -s "\r" "\n" > new\_file

注意：这里-s后面是两个参数"\r"和"\n"，用后者替换前者

9、用空格符\040替换制表符\011

cat file | tr -s "\011" "\040" > new\_file

10、把路径变量中的冒号":"，替换成换行符"\n"

echo $PATH | tr -s ":" "\n"

**2.11 paste命令**

paste单词意思是粘贴。该命令主要用来将多个文件的内容合并，与cut命令完成的功能刚好相反。paste按行将不同文件行信息放在一行。缺省情况下， paste连接时，用空格或tab键分隔新行中不同文本，除非指定-d选项，它将成为域分隔符。

paste格式为:

paste -d -s -file1 file2

选项含义如下：

-d 指定不同于空格或tab键的域分隔符。例如用@分隔域，使用- d @。

-s 将文件中的内容合并成一行而不是按行粘贴。

- 使用标准输入。

举例：

1. paste -d ‘ ‘ file1.txt file2.txt > file3.txt (引号之间有空格)

将file1.txt和file2.txt每行按照空格分隔符来合并文件并重定向到file3.txt中

1. cat 11.txt|paste - 22.txt > 33.txt

将11.txt和22.txt合并并重定向到33.txt（分隔符好像是制表符）

1. paste -s 11.txt

将11.txt文件中的内容合并成一行显示

**2.12 split命令**

split命令：文件切割

有时候文件过大，导致不能正常使用，可以用split进行切割。

命令参数：

split [选项] [要切割的文件] [输出文件名前缀］

-a, --suffix-length=N 切割文件的后缀的位置，default 为2

-b, --bytes=SIZE 指定块的大小，可以指定size的单位，b表示512字节,k代表1K字节，m代表1M字节,还有G，T，P

-d, --numeric-suffixes 利用数字前缀，而不是字母前缀，默认是字母前缀。

-l, --lines=NUMBER 指定行数进行切割。

举例：

1. splite -l 1 file.txt ex

file文件每一行被切割为一个文件，文件名前缀为ex，如生成的文件为：exaa,axab,axac...

split -l 5 file.txt ex

每5行切割为一个文件

1. splite -b 1024 file ax

将文件file切割为每个大小为1024字节的文件，前缀为ax

1. split -b 1024 -a 3 file

将文件file切割为每个大小为1024字节的文件，指定文件后缀名个数为3个，默认文件后缀第一个字符为x开头

4、split -b 1024 -a 3 file cx

在3的基础上，指定每个文件以cx开头，cx后面有3个字符

**2.13 ftp拷贝远端文件**

ftp是一个客户机/服务器系统，用户通过一个支持FTP协议的客户机程序，连接到远程主机上的FTP服务器程序。用户通过客户机肠道内关系向服务器程序发出FTP命令，服务器程序执行用户发出的FTP命令，并且将执行结果返回给客户机。

在虚拟机终端下输入：ftp IP地址，提示输入用户名和密码即可登录进而进行文件的传输。

例如：ftp 158.222.65.180

提示输入用户名和密码，分别输入即可

注意：如果我不知道ftp命令的话，可以在进入ftp模式下，输入help即会显示ftp的全部指令，“help+指令名”显示指定指令的功能。想要将远端文件拷贝到用户指定目录下，可以先lcd一下，就是将主连接用户切换到主目录中，使用lcd +目录名就可以切换主连接用户的目录，这样可以将远端文件拷贝到任意目录下。

**2.14 su命令**

切换用户的命令。

举例：su - zpstst(假设当前用户为zpsdev，当前目录为public)

输入zpstst用户密码进入zpstst用户目录下

cp /home/zpsdev/1.txt ./ 或者cp /home/zpsdev/1.txt /home/zpstst/

将zpsdev用户家目录下的1.txt 拷贝到zpstst用户家目录下

注意：要移动zpsdev用户下的文件，则需要在zpsdev终端下以获取权限

**2.15 chown命令**

chown将指定文件的拥有者改为指定的用户或组，用户可以是用户名或者用户ID；组可以是组名或者组ID；文件是以空格分开的要改变权限的文件列表，支持通配符。系统管理员经常使用chown命令，在将文件拷贝到另一个用户的名录下之后，让用户拥有使用该文件的权限。

1. 命令格式：

chown [选项]... [所有者][:[组]] 文件...

2．命令功能：

通过chown改变文件的拥有者和群组。在更改文件的所有者或所属群组时，可以使用用户名称和用户识别码设置。普通用户不能将自己的文件改变成其他的拥有者。其操作权限一般为管理员。

3．命令参数：

必要参数:

　　-c 显示更改的部分的信息

　　-f 忽略错误信息

　　-v 显示详细的处理信息

-deference 作用于符号链接的指向，而不是链接文件本身

选择参数:

--reference=<文件或者目录>

--help 显示帮助信息

--version 显示版本信息

1. 使用实例
2. 将档案 file1.txt 的拥有者设为 users 群体的使用者 jessie :   
   chown jessie:users file1.txt   
   2、将目前目录下的所有档案与子目录的拥有者皆设为 users 群体的使用者 lamport   
   chown -R lamport:users \*
3. 将指定目录test下的所有档案与子目录的拥有者皆设为 users 群体的使用者 lamport

chown -R lamport:users test

4、将文件log2012.log的拥有者和群组都变成了root

chown root: log2012.log

5、将文件log2012.log的群组变成mail

chown :mail log2012.log

**2.16 chgrp命令**

在lunix系统里，文件或目录的权限的掌控以拥有者及所属群组来管理。可以使用chgrp指令取变更文件与目录所属群组，这种方式采用群组名称或群组识别码都可以。chgrp命令就是change group的缩写！要被改变的组名必须要在/etc/group文件内存在才行。

1．命令格式：

chgrp [选项] [组] [文件]

2．命令功能：

chgrp命令可采用群组名称或群组识别码的方式改变文件或目录的所属群组。使用权限是超级用户。

3．命令参数：

必要参数:

-c 当发生改变时输出调试信息

-f 不显示错误信息

-R 处理指定目录以及其子目录下的所有文件

-v 运行时显示详细的处理信息

--dereference 作用于符号链接的指向，而不是符号链接本身

--no-dereference 作用于符号链接本身

选择参数:

--reference=<文件或者目录>

--help 显示帮助信息

--version 显示版本信息

1. 使用实例

1、将文件log2012.log的群组改为bin

chgrp -v bin log2012.log

2、根据指定文件 log2012.log的群组属性来改变文件 log2013.log的群组属性

chgrp --reference=log2012.log log2013.log

1. 将指定目录test下的所有档案与子目录下的所有文件的群组属性改为bin

chgrp -R bin test6

1. 将指定目录test下的所有档案与子目录下的所有文件的群组属性改为群组识别码为100对应的组

chgrp -R 100 test6

注意：通过群组识别码改变文件群组属性，100为users群组的识别码，具体群组和群组识别码可以去/etc/group文件中查看。

**2.17 chomd命令**

chmod命令用于改变linux系统文件或目录的访问权限。用它控制文件或目录的访问权限。该命令有两种用法。一种是包含字母和操作符表达式的文字设定法；另一种是包含数字的数字设定法。

Linux系统中的每个文件和目录都有访问许可权限，用它来确定谁可以通过何种方式对文件和目录进行访问和操作。

　　文件或目录的访问权限分为只读，只写和可执行三种。以文件为例，只读权限表示只允许读其内容，而禁止对其做任何的更改操作。可执行权限表示允许将该文件作为一个程序执行。文件被创建时，文件所有者自动拥有对该文件的读、写和可执行权限，以便于对文件的阅读和修改。用户也可根据需要把访问权限设置为需要的任何组合。

　　有三种不同类型的用户可对文件或目录进行访问：文件所有者，同组用户、其他用户。所有者一般是文件的创建者。所有者可以允许同组用户有权访问文件，还可以将文件的访问权限赋予系统中的其他用户。在这种情况下，系统中每一位用户都能访问该用户拥有的文件或目录。

每一文件或目录的访问权限都有三组，每组用三位表示，分别为文件属主的读、写和执行权限；与属主同组的用户的读、写和执行权限；系统中其他用户的读、写和执行权限。当用ls -l命令显示文件或目录的详细信息时，最左边的一列为文件的访问权限。

1. 命令格式:

chmod [-cfvR] [--help] [--version] mode file

2. 命令功能：

用于改变文件或目录的访问权限，用它控制文件或目录的访问权限。

3. 命令参数：

必要参数：

-c 当发生改变时，报告处理信息

-f 错误信息不输出

-R 处理指定目录以及其子目录下的所有文件

-v 运行时显示详细处理信息

选择参数：

--reference=<目录或者文件> 设置成具有指定目录或者文件具有相同的权限

--version 显示版本信息

<权限范围>+<权限设置> 使权限范围内的目录或者文件具有指定的权限

<权限范围>-<权限设置> 删除权限范围的目录或者文件的指定权限

<权限范围>=<权限设置> 设置权限范围内的目录或者文件的权限为指定的值

权限范围：

u ：目录或者文件的当前的用户

g ：目录或者文件的当前的群组

o ：除了目录或者文件的当前用户或群组之外的用户或者群组

a ：所有的用户及群组

权限代号：

r ：读权限，用数字4表示

w ：写权限，用数字2表示

x ：执行权限，用数字1表示

- ：删除权限，用数字0表示

s ：特殊权限

该命令有两种用法。一种是包含字母和操作符表达式的文字设定法；另一种是包含数字的数字设定法。

　　1）. 文字设定法:

　　 chmod ［who］ ［+ | - | =］ ［mode］ 文件名

　　2）. 数字设定法

　　我们必须首先了解用数字表示的属性的含义：0表示没有权限，1表示可执行权限，2表示可写权限，4表示可读权限，然后将其相加。所以数字属性的格式应为3个从0到7的八进制数，其顺序是（u）（g）（o）。

　　例如，如果想让某个文件的属主有“读/写”二种权限，需要把4（可读）+2（可写）＝6（读/写）。

　　数字设定法的一般形式为：

　　 chmod ［mode］ 文件名

数字与字符对应关系如下：

r=4，w=2，x=1

若要rwx属性则4+2+1=7

若要rw-属性则4+2=6；

若要r-x属性则4+1=7。

4. 使用实例：

实例1：增加文件所有用户组可执行权限

命令：

chmod a+x log2012.log

输出：

[root@localhost test]# ls -al log2012.log

-rw-r--r-- 1 root root 302108 11-13 06:03 log2012.log

[root@localhost test]# chmod a+x log2012.log

[root@localhost test]# ls -al log2012.log

-rwxr-xr-x 1 root root 302108 11-13 06:03 log2012.log

[root@localhost test]#

说明：

　　即设定文件log2012.log的属性为：文件属主（u） 增加执行权限；与文件属主同组用户（g） 增加执行权限；其他用户（o） 增加执行权限。

实例2：同时修改不同用户权限

命令：

chmod ug+w,o-x log2012.log

输出：

[root@localhost test]# ls -al log2012.log

-rwxr-xr-x 1 root root 302108 11-13 06:03 log2012.log

[root@localhost test]# chmod ug+w,o-x log2012.log

[root@localhost test]# ls -al log2012.log

-rwxrwxr-- 1 root root 302108 11-13 06:03 log2012.log

说明：

　　即设定文件text的属性为：文件属主（u） 增加写权限;与文件属主同组用户（g） 增加写权限;其他用户（o） 删除执行权限

实例3：删除文件权限

命令：

chmod a-x log2012.log

输出：

[root@localhost test]# ls -al log2012.log

-rwxrwxr-- 1 root root 302108 11-13 06:03 log2012.log

[root@localhost test]# chmod a-x log2012.log

[root@localhost test]# ls -al log2012.log

-rw-rw-r-- 1 root root 302108 11-13 06:03 log2012.log

说明：

删除所有用户的可执行权限

实例4：使用“=”设置权限

命令：

chmod u=x log2012.log

输出：

[root@localhost test]# ls -al log2012.log

-rw-rw-r-- 1 root root 302108 11-13 06:03 log2012.log

[root@localhost test]# chmod u=x log2012.log

[root@localhost test]# ls -al log2012.log

---xrw-r-- 1 root root 302108 11-13 06:03 log2012.log

说明：

撤销原来所有的权限，然后使拥有者具有可读权限

实例5：对一个目录及其子目录所有文件添加权限

命令：

chmod -R u+x test4

输出：

[root@localhost test]# cd test4

[root@localhost test4]# ls -al

总计 312drwxrwxr-x 2 root root 4096 11-13 05:50 .

drwxr-xr-x 5 root root 4096 11-22 06:58 ..

-rw-r--r-- 1 root root 302108 11-12 22:54 log2012.log

-rw-r--r-- 1 root root 61 11-12 22:54 log2013.log

-rw-r--r-- 1 root root 0 11-12 22:54 log2014.log

[root@localhost test4]# cd ..

[root@localhost test]# chmod -R u+x test4

[root@localhost test]# cd test4

[root@localhost test4]# ls -al

总计 312drwxrwxr-x 2 root root 4096 11-13 05:50 .

drwxr-xr-x 5 root root 4096 11-22 06:58 ..

-rwxr--r-- 1 root root 302108 11-12 22:54 log2012.log

-rwxr--r-- 1 root root 61 11-12 22:54 log2013.log

-rwxr--r-- 1 root root 0 11-12 22:54 log2014.log

说明：

递归地给test4目录下所有文件和子目录的属主分配权限

其他一些实例：

1）.

命令：

chmod 751 file

说明：

给file的属主分配读、写、执行(7)的权限，给file的所在组分配读、执行(5)的权限，给其他用户分配执行(1)的权限

2）.

命令：

chmod u=rwx,g=rx,o=x file

说明：

上例的另一种形式

3）.

命令

chmod =r file

说明：

为所有用户分配读权限

3）.

命令：

chmod 444 file

说明：

同上例

4）.

命令：

chmod a-wx,a+r file

说明：

同上例

**2.18 用户和用户组管理时相关的文件**

Linux /etc/group文件与/etc/passwd和/etc/shadow文件都是有关于系统管理员对用户和用户组管理时相关的文件。linux /etc/group文件是有关于系统管理员对用户和用户组管理的文件,linux用户组的所有信息都存放在/etc/group文件中。具有某种共同特征的用户集合起来就是用户组（Group）。用户组（Group）配置文件主要有 /etc/group和/etc/gshadow，其中/etc/gshadow是/etc/group的加密信息文件。

将用户分组是Linux系统中对用户进行管理及控制访问权限的一种手段。每个用户都属于某个用户组；一个组中可以有多个用户，一个用户也可以属于不 同的组。当一个用户同时是多个组中的成员时，在/etc/passwd文件中记录的是用户所属的主组，也就是登录时所属的默认组，而其他组称为附加组。

用户组的所有信息都存放在/etc/group文件中。此文件的格式是由冒号(:)隔开若干个字段，这些字段具体如下：

组名:口令:组标识号:组内用户列表

组名是用户组的名称，由字母或数字构成。与/etc/passwd中的登录名一样，组名不应重复。

口令：

口令字段存放的是用户组加密后的口令字。一般Linux系统的用户组都没有口令，即这个字段一般为空，或者是\*。

组标识号：

组标识号与用户标识号类似，也是一个整数，被系统内部用来标识组。别称GID.

组内用户列表：

是属于这个组的所有用户的列表，不同用户之间用逗号(,)分隔。这个用户组可能是用户的主组，也可能是附加组。

**2.19 linux文件属性详解**

**L**inux 文件或目录的属性主要包括：文件或目录的节点、种类、权限模式、链接数量、所归属的用户和用户组、最近访问或修改的时间等内容。具体情况如下：

命令： ls -lih

输出：[root@localhost test]# ls -lih

总计 316K

2095120 lrwxrwxrwx 1 root root 11 11-22 06:58 linklog.log -> log2012.log

2095112 -rw-r--r-- 1 root root 296K 11-13 06:03 log2012.log

2095110 -rw-r--r-- 1 root root 61 11-13 06:03 log2013.log

2095107 -rw-r--r-- 1 root root 0 11-13 06:03 log2014.log

2095117 -rw-r--r-- 1 root root 0 11-13 06:06 log2015.log

2095118 -rw-r--r-- 1 root root 0 11-16 14:41 log2016.log

2095119 -rw-r--r-- 1 root root 0 11-16 14:43 log2017.log

2095113 drwxr-xr-x 6 root root 4.0K 10-27 01:58 scf

2095109 drwxrwxr-x 2 root root 4.0K 11-13 06:08 test3

2095131 drwxrwxr-x 2 root root 4.0K 11-13 05:50 test4

说明：

第一列：inode

第二列：文件种类和权限；

第三列： 硬链接个数；

第四列： 属主；

第五列：所归属的组；

第六列：文件或目录的大小；

第七列和第八列：最后访问或修改时间；

第九列：文件名或目录名

我们以log2012.log为例：

2095112 -rw-r--r-- 1 root root 296K 11-13 06:03 log2012.log

inode 的值是：2095112

文件类型：文件类型是-，表示这是一个普通文件；

文件权限：文件权限是rw-r--r-- ，表示文件属主可读、可写、不可执行，文件所归属的用户组不可写，可读，不可执行，其它用户不可写，可读，不可执行；

硬链接个数： log2012.log这个文件没有硬链接；因为数值是1，就是他本身；

文件属主：也就是这个文件归哪于哪个用户 ，它归于root，也就是第一个root；

文件属组：也就是说，对于这个文件，它归属于哪个用户组，在这里是root用户组；

文件大小：文件大小是296k个字节；

访问可修改时间 ：这里的时间是最后访问的时间，最后访问和文件被修改或创建的时间，有时并不是一致的；

当然文档的属性不仅仅包括这些，这些是我们最常用的一些属性。

关于inode：

inode 译成中文就是索引节点。每个存储设备或存储设备的分区（存储设备是硬盘、软盘、U盘等等）被格式化为文件系统后，应该有两部份，一部份是inode，另一部份是Block，Block是用来存储数据用的。而inode呢，就是用来存储这些数 据的信息，这些信息包括文件大小、属主、归属的用户组、读写权限等。inode为每个文件进行信息索引，所以就有了inode的数值。操作系统根据指令， 能通过inode值最快的找到相对应的文件。

做个比喻，比如一本书，存储设备或分区就相当于这本书，Block相当于书中的每一页，inode 就相当于这本书前面的目录，一本书有很多的内容，如果想查找某部份的内容，我们可以先查目录，通过目录能最快的找到我们想要看的内容。虽然不太恰当，但还是比较形象。

当我们用ls 查看某个目录或文件时，如果加上-i 参数，就可以看到inode节点了；比如我们前面所说的例子：

[root@localhost test]# ls -li log2012.log

2095112 -rw-r--r-- 1 root root 302108 11-13 06:03 log2012.log

log2012.log 的inode值是 2095112 ； 查看一个文件或目录的inode，要通过ls 命令的的 -i参数。

**三、shell编程**

**3.1 Shell编程中if的语法和常见判断用法**

if, for, while几乎是所有编程语言的关键字，在Shell编程中也不例外，其中if是使用频率最高的，由于Shell编程中不存在对象(Object)的概念，因此在if的比较判断中主要是对字符串、数字的值进行比较判断的：

**3.1.1 if的基本语法**

1. if与[之间要有空格

2. []与判断条件之间也必须有空格

3. ]与;之间不能有空格

**3.1.2 对字符串的判断**

1. if [ str1=str2 ];then fi #当两个字符串相同时返回真

2. if [ str1!=str2 ];then fi #当两个字符串不相等时返回真

3. if [ -n str1 ];then fi #当字符串的长度大于0时返回真 (判断变量是否有值)

4. if [ -z str1 ];then fi #当字符串的长度为0时返回真

5. if [ str1 ];then fi #当字符串的长度大于0时返回真（判断变量是否有值），同-n str1

**3.1.3 对数字的判断**

1. int1 -eq int2 #int1和int2相等

2. int1 -ne int2 #int1不相等int2

3. int1 -gt int2 #int1大于int2

4. int1 -ge int2 #int1大于等于int2

5. int1 -lt int2 #int1小于int2

6. int1 -le int2 #int1小于等于int2

**3.1.4 对文件属性的判断**

1. -r file #用户可读为真

2. -w file #用户可写为真

3. -x file #用户可执行为真

4. -f file #文件存在且为正规文件为真

5. -d file #如果是存在目录为真

6. -c file #文件存在且为字符设备文件

7. -b file #文件存在且为块设备文件

8. -s file #文件大小为非0为真，可以判断文件是否为空

9. -e file #如果文件存在为真

**3.1.5 逻辑判断**

1. -a #与

2. -o #或

3. ! #非

**3.1.6 算术运算**

1. 使用expr 外部处理式：r=`expr 4 + 5` 注意:4，+，5三者之间要有空格

2. 使用$(())：r=$((4+5))

3. 使用$[ ]：r=$[ 4+5 ] 注意：数值和[、]之间要有空格，有些不支持此功能

4. 使用let语句：let “var=$var+1” 注意:let 后有空格

**3.2 shell脚本中一些常见的特殊符号**

在编写Shell脚本时，我们需要会用到各种各样的特殊符号，通过这些特殊符号可以使我们编写的代码更加简洁和高效，这里给大家汇总下：

**3.2.1 {} 大括号**

用法一：通配符扩展

eg: ls my\_{finger,toe}s

这条命令相当于如下命令的组合：

ls my\_fingers my\_toes

eg: mkdir {userA,userB,userC}-{home,bin,data}

我们将得到 userA-home, userA-bin, userA-data, userB-home, userB-bin,userB-data,userC-home, userC-bin, userC-data，这几个目录。

用法二：可用于语句块的构造，语句之间用回车隔开。如果你想在某些使用单个语句的地方（比如在AND或OR列表中）使用多条语句，你可以把它们括在花括号{}中来构造一个语句块。

eg:

{

grep -v "$cdcatnum" $strack\_file > $temp\_file

cat $temp\_file > $strack\_file

echo

cat -n file1

} (注：以上大括号中的四句命令够成了一个语句块)

用法三：参数扩展

${name:-default} 使用一个默认值（一般是空值）来代替那些空的或者没有赋值的变量name；

${name:=default}使用指定值来代替空的或者没有赋值的变量name；

${name:?message}如果变量为空或者未赋值，那么就会显示出错误信息并中止脚本的执行同时返回退出码1。

${#name} 给出name的长度

${name%word} 从name的尾部开始删除与word匹配的最小部分，然后返回剩余部分

${name%%word} 从name的尾部开始删除与word匹配的最长部分，然后返回剩余部分

${name#word} 从name的头部开始删除与word匹配的最小部分，然后返回剩余部分

${name##word} 从name的头部开始删除与word匹配的最长部分，然后返回剩余部分

（注，name为变量名，word为要匹配的字符串）

用法三在处理字符串和未知变量时，是很有用的。

**3.2.2 [] 中括号：**

用法一：通配符扩展：

允许匹配方括号中任何一个单个字符

eg: ls /[eh][to][cm]\*

相当于执行 ls /etc /home（若有/eom目录，就相当于会执行ls /etc /home /eom）

注：在mkdir命令下不能扩展

用法二：用于条件判断符号：

[]符号可理解为指向test命令的一个软链接，所以其用法可完全参照test，将test位置替换为[便可。

eg: if [ "$?" != 0 ] 等价于 if test "$?" != 0

then echo "Executes error"

**3.2.3 `command` 反引号：`command`与$(command)的含义相同，都是返回当前执行命令的结果**

eg: #!/bin/sh

for file in $(ls f\*.sh);do

lpr $file

done

exit 0

该例实现了扩展f\*.sh给出所有匹配模式的文件的名字。

**3.2.4 'string' 单引号 和 "string" 双引号**

双引号：如果想在定义的变量中加入空格，就必须使用单引号或双引号，

单、双引号的区别在于双引号转义特殊字符而单引号不转义特殊字符

eg: $ heyyou=home

$ echo '$heyyou'

$ $heyyou （$没有转义）

eg: $ heyyou=home

$ echo "$heyyou"

$ home （很明显，$转义了输出了heyyou变量的值）

**3.2.5几种特殊参数变量的含义**

$#在shell它的作用是告诉你引用变量的总数量是多少

$$ 它的作用是告诉你shell脚本的进程号；

$\* 以一个单字符串显示所有的脚本传递的参数。等价于$1 $2 $3.......；

$@ 与$\*基本类似（参见序号7），但在数组赋值时有些不同；

$? 前一个命令的退出码；

$- 显示shell使用的当前选项；

$! 最后一个后台运行的进程ID号。

**3.2.6 $((...))语法**

对括号内的表达式求值。

eg:

#!/bin/sh

x=0

hile [ "$x" -ne 10 ];do

echo $x

x=$(($x+1))

done

exit 0

**3.2.7 shell中几种特殊的参数变量的引用**

$1、$2、$3……${10}、${11}、${12}…… ：表示脚本传入的的各个参数，注意当需表示两位数以后的参数时数字要用花括号括起。

$@ 列出所有的参数，各参数用空格隔开

$\*: 列出所有的参数，各参数用环境变量IFS的第一个字符隔开

**3.2.8 命令列表**

AND列表 statement1 && statement2 && statement3 && …:只有在前面所有的命令都执行成功的情况下才执行后一条命令

OR列表 statement1 || statement2 || statement3 || …:允许执行一系列命令直到有一条命令成功为止，其后所有命令将不再被执行

eg:#!/bin/sh

touch file\_one

rm -f file\_two

if [ -f file\_one ] && echo "hello" && [ -f file\_two ] && echo " there"

then

echo "in if"

else

echo "in else"

fi

exit 0

上例的输出为：

hello

in else

关于AND列表与OR列表，在逻辑判断中很使用，下面就举一个其最常用的例子：

[ condition ] && command for true || command for false:

当条件为真时，执行commandfor true ,当条件为假时，执行command for false

**3.2.9 :冒号内建空指令，返回值为0**

eg: $ :

$ echo $?

$ 0

while: (该语句结构可实现一个无限循环)

**3.2.10 ;分号在shell中功能**

担任"连续指令"功能的符号就是"分号"。

eg:cd ~/backup ; mkdir startup ; cp ~/.\* startup/.

**3.2.11 #井号在shell中的功能**

表示符号后面的是注解文字，不会被执行。

\* 匹配文件名中的任何字符，包括字符串；

？ 匹配文件名中的任何单个字符。

~ 代表使用者的 home 目录

**3.2.12 \ 倒斜线在shell中的功能**

放在指令前，有取消 aliases（别名） 的作用；放在特殊符号前，则该特殊符号的作用消失；放在指令的最末端，表示指令连接下一行（使得回车符无效，只起换行作用）

**3.2.13 !感叹号在shell中的功能**

通常它代表反逻辑的作用，譬如条件侦测中，用 != 来代表"不等于"

**3.2.14 \*\* 次方运算**

两个星号在运算时代表 "次方" 的意思

eg:let "sus=2\*\*3"

echo "sus = $sus"

$ sus = 8

1. 交换/etc/passwd文件最后一行的第一个单词和最后一个单词的位置

sed -nr ‘$s#(\w\*):(.\*):(.\*)$#\3:\2:\1#pg’ /etc/passwd 思考:\w？？？可能不具有移植性

sed -nr ‘$s/([^:]\*)(:.\*:)(.\*)/\3\2\1/pg’ /etc/passwd

1. 删除文件/etc/passwd 最后一行倒数第二个字符

sed -nr ‘$s/(.\*)(.)(.)/\1\3/pg’ /etc/passwd

**Python篇**

1. **Python3常用模块汇总**

1.1 os模块

[os模块](http://www.cnblogs.com/MnCu8261/p/5483657.html" \t "http://www.cnblogs.com/MnCu8261/p/_blank)提供了对目录或者文件的新建/删除/查看文件属性，还提供了对文件以及目录的路径操作。比如说：绝对路径，父目录……

os.sep可以取代操作系统特定的路径分隔符。windows下为 “\\”,Linux下为"/"

os.linesep字符串给出当前平台使用的行终止符。例如，Windows使用'\r\n'，Linux使用'\n'而Mac使用'\r'。

os.pathsep 输出用于分割文件路径的字符串，系统使用此字符来分割搜索路径（像PATH），例如POSIX上':',Windows上的';'

os.getcwd() 获取当前工作目录，即当前python脚本工作的目录路径

os.chdir("dirname") 改变当前脚本工作目录；相当于shell下cd

os.curdir 返回当前目录: ('.')

os.pardir 获取当前目录的父目录字符串名：('..')

os.mkdir('dirname') 生成单级目录；相当于shell中mkdir dirname

os.makedirs('dirname1/dirname2') 可生成多层递归目录

os.remove(file) 删除一个文件

os.removedirs('dirname1') 若目录为空，则删除，并递归到上一级目录，如若也为空，则删除，依此类推

os.rmdir('dirname') 删除单级空目录，若目录不为空则无法删除，报错；相当于shell中rmdir dirname

os.listdir('dirname') 列出指定目录下的所有文件和子目录，包括隐藏文件，并以列表方式打印

os.rename("oldname","newname") 重命名文件/目录,如果newname存在则替换出现错误

os.replace(src,dest) 重命名文件/目录，如果dest表示的是文件，存在则覆盖原来的文件，不报错；若是目录，存在则会报错

os.chmod(path, mode, \*, dir\_fd=None, follow\_symlinks=True)

Ex:os.chmod('C:\\my\_share\_file\\test.sh',755)

os.stat('path/filename') 获取文件/目录信息

os.utime(path,times) 修改时间属性 times是个元组,(atime,mtime),这两个时间数可以通过os.stat获取

os.walk(top[, topdown=True[, onerror=None[, followlinks=False]]])

1.top表示需要遍历的目录树的路径

2.topdown的默认值是”True”,表示首先返回目录树下的文件，然后在遍历目录树的子目录.Topdown的值为”False”时，

则表示先遍历目录树的子目录，返回子目录下的文件，最后返回根目录下的文件

3.onerror的默认值是”None”,表示忽略文件遍历时产生的错误.如果不为空，则提供一个自定义函数提示错误信息后继续遍历或抛出异常中止遍历

该函数返回一个元组，该元组有3个元素，这3个元素分别表示当前遍历的目录，当前遍历的目录列表，当前遍历的目录的文件列表

os.walk()举例：

>>> import os

>>> for root, dirs, files in os.walk("wd/chat", topdown=False):

... for name in files:

... print(os.path.join(root, name)) #打印文件绝对路径

... for name in dirs:

... print(os.path.join(root, name)) #打印目录绝对路径

os.name字符串指示你正在使用的平台。比如对于Windows，它是'nt'，而对于Linux/Unix用户，它是'posix'。

os.getenv()获取一个环境变量，如果没有返回none

os.putenv(key, value)设置一个环境变量值

os.environ[]获取环境变量的值 两者等价：os.environ[‘HOME’] <->os.getenv(‘HOME’)

os.system(command)函数用来运行shell命令。

os.popen("bash command") 运行shell命令，生成对象，可赋给变量，再用read读取

1.2 os.path模块

对文件路径的操作

os.path.split(p)函数返回一个路径的目录名和文件名。

os.path.splitext():分离文件名与扩展名

os.path.isfile()和os.path.isdir()函数分别检验给出的路径是一个文件还是目录。

os.path.isabs(path):判断一个路径是否是绝对路径

os.path.exists()函数用来检验给出的路径是否真地存在

os.path.getsize(name):获得文件大小，如果name是目录返回0L

os.path.getatime(path) 返回path所指向的文件或者目录的最后存取时间

os.path.getmtime(path) 返回path所指向的文件或者目录的最后修改时间

os.path.normpath(path):规范path字符串形式

>>> os.path.normpath('c://windows\\System32\\../Temp/')

'c:\\windows\\Temp'

os.path.join(path,name):连接目录与文件名或目录

os.path.basename(path):返回文件名

os.path.dirname(path) 返回path的目录。其实就是os.path.split(path)的第一个元素

os.path.abspath(path) 返回path规范化的绝对路径

>>> import os.path

>>> os.path.abspath('c.py')

'/root/py/c.py'

>>> os.path.abspath('../py/c.py')

'/root/py/c.py'

1.3 shutil模块

-- --High-level file operations  高级的文件操作模块

shutil.copyfileobj(fsrc, fdst[, length=16\*1024]) #copy文件内容到另一个文件，可以copy指定大小的内容

shutil.copyfile(src,dst)  #copy文件内容，是不是感觉上面的文件复制很麻烦？还需要自己手动用open函数打开文件，在这里就不需要了，事实上，copyfile调用了copyfileobj

shutil.copymode(src,dst)  #仅copy权限，不更改文件内容，组和用户

shutil.copystat(src,dst)   #复制所有的状态信息，包括权限，组，用户，时间等

shutil.copy(src,dst)  #复制文件的内容以及权限，先copyfile后copymode

shutil.copy2(src,dst)    #复制文件的内容以及文件的所有状态信息。先copyfile后copystat

类似于cp –p

shutil.copytree(src,dst,symlinks=False,ignore=None,copy\_function=copy2,ignore\_dangling\_symlinks=False)  #递归的复制文件内容及状态信息

shutil.rmtree(path, ignore\_errors=False, onerror=None)  #递归地删除文件

shutil.move(src, dst)  #说明：如果两个位置的文件系统是一样的话相当于是rename操作，只是改名如果是不在相同的文件系统的话就是做move操作了！

make\_archive(base\_name, format, root\_dir=None, base\_dir=None, verbose=0,dry\_run=0, owner=None, group=None, logger=None)  #压缩打包

base\_name：   压缩打包后的文件名或者路径名

format：          压缩或者打包格式    "zip", "tar", "bztar"or "gztar"

root\_dir :         将哪个目录或者文件打包（也就是源文件）

>>> shutil.make\_archive('tarball','gztar',root\_dir='copytree\_test')

[root@slyoyo python\_test]# ls -l

total 12

drwxr-xr-x. 3 root root 4096 May 14 19:36 copytree\_copy

drwxr-xr-x. 3 root root 4096 May 14 19:36 copytree\_test

-rw-r--r--. 1 root root 0 May 14 21:12 tarball.tar.gz

-rw-r--r--. 1 python python 79 May 14 05:17 test1

-rw-r--r--. 1 root root 0 May 14 19:10 test2

shutil.unpack\_archive(filename[, extract\_dir[, format]]) #解压归档文件，filename是归档的完整路径，extract\_dir是解压到的文件路径，format默认是根据归档的后缀来自动识别

1.4 collections模块

我们都知道，Python拥有一些内置的数据类型，比如str, int, list, tuple, dict等， collections模块在这些内置数据类型的基础上，提供了几个额外的数据类型：

（1）namedtuple(): 生成可以使用名字来访问元素内容的tuple子类   
（2）deque: 双端队列，可以快速的从另外一侧追加和推出对象   
（3）Counter: 计数器，主要用来计数   
（4）OrderedDict: 有序字典   
（5）defaultdict: 带有默认值的字典

**=>namedtuple(typename, field\_names, verbose=False, rename=False)**

比如我们用户拥有一个这样的数据结构，每一个对象是拥有三个元素的tuple。  
使用namedtuple方法就可以方便的通过tuple来生成可读性更高也更好用的数据结构

wdebsites = [

('Sohu', 'http://www.google.com/', '张朝阳'),

('Sina', 'http://www.sina.com.cn/', '王志东'),

('163', 'http://www.163.com/', '丁磊')

]

Website = namedtuple('Website', ['name', 'url', 'founder'])

for website in websites:

website = Website.\_make(website)

print (website)

Result:

Website(name='Sohu', url='http://www.google.com/', founder='张朝阳')

Website(name='Sina', url='http://www.sina.com.cn/', founder='王志东')

Website(name='163', url='http://www.163.com/', founder='丁磊')

**=>deque(iterable,maxlen)**

deque其实是double-ended queue的缩写，翻译过来就是双端队列，它最大的好处就是实现了从队列头部快速增加和取出对象: .popleft(), .appendleft()。原生的List也能从头部添加和去除对象，l.insert(0, v)、l.pop(0)。但是值得注意的是，list对象的这两种用法的时间复杂度是O(n)，也就是说随着元素数量的增加耗时呈线性上升。而使用deque对象则是O(1) 的复杂度，所以当你的代码有这样的需求的时候，一定要记得使用deque。

作为一个双端队列，deque还提供了一些其他的好用方法，比如 rotate等。

append(x):从队列末尾添加x

appendleft(x):从队列头部添加x

clear():清空队列

count(x):返回队列中x的数量

extend(iterable):在队列的尾部添加多个元素

extendleft(iterable):在队列的头部添加多个元素，反序插入（字母表相反顺序）

pop():将尾部一个元素移除

popleft():将头部一个元素移除

remove(x):将队列中第一次出现的x元素移除

reverse():将队列元素逆置

rotate(n):将队列尾部的n个元素添加到头部 rotate(1)等价于:d.appendleft(d.pop())

D.maxlen:返回队列的长度，如果是无界则返回None（定义的时候可以指定maxlen，否则默认是无界队列）

Example:

>>>d=collections.deque("gsgwqeasas")

>>> d

deque(['g', 's', 'g', 'w', 'q', 'e', 'a', 's', 'a', 's'])

>>> s=list(d)

>>> s

['w', 'g', 'g', 's', 'a', 's', 'a', 'e', 'q']

**=>Counter(iterable or mapping) #可迭代或者映射的对象**

计数器是一个非常常用的功能需求。

elements():返回一个重复元素的迭代器，重复次数和计数的次数一样多。元素以任意顺序返回。如果一个元素的计数小于1，elements()会忽略它。

most\_common([n]):返回n个计数器元素，若未指定n，则返回计数器的所有元素

update(iterable or mapping):增加计数器元素，元素可以来源于迭代对象或者一个Counter对象

Example:

>>> s

['w', 'g', 'g', 's', 'a', 's', 'a', 'e', 'q']

>>> c=collections.Counter(s)

>>> c

Counter({'g': 2, 'a': 2, 's': 2, 'e': 1, 'q': 1, 'w': 1})

>>> list(c)

['e', 'q', 'g', 'w', 'a', 's']

>>> dict(c)

{'a': 2, 'q': 1, 'g': 2, 'w': 1, 'e': 1, 's': 2}

>>> list(c.elements())

['e', 'q', 'g', 'g', 'w', 'a', 'a', 's', 's']

>>> c.most\_common(4)

[('g', 2), ('a', 2), ('s', 2), ('e', 1)]

>>> c.update('fds')

>>> c

Counter({'s': 3, 'g': 2, 'a': 2, 'e': 1, 'q': 1, 'w': 1, 'f': 1, 'd': 1})

>>> c.update(c)

>>> c

Counter({'s': 6, 'g': 4, 'a': 4, 'e': 2, 'q': 2, 'w': 2, 'f': 2, 'd': 2})

>>>

**=>OrderedDict(items)**

在Python中，dict这个数据结构由于hash的特性，是无序的，这在有的时候会给我们带来一些麻烦， 幸运的是，collections模块为我们提供了OrderedDict，当你要获得一个有序的字典对象时，用它就对了。

popitem(last=True):移除字典对象元素，last=True时“后进先出”，last=False时“先进先出”

move\_to\_end(key,last=True):将字典中一个已经存在的键移动到头部或者尾部。last=True时，移动到尾部，last=False时，移动到头部。

Example:

>>> d

{'w': 2, 'a': 4, 'f': 2, 'q': 2, 'g': 4, 'd': 2, 'e': 2, 's': 6}

>>> n=collections.OrderedDict(d)

>>> n

OrderedDict([('w', 2), ('a', 4), ('f', 2), ('q', 2), ('g', 4), ('d', 2), ('e', 2), ('s', 6)])

>>> n.popitem()

('s', 6)

>>> n

OrderedDict([('a', 4), ('f', 2), ('q', 2), ('g', 4), ('d', 2), ('e', 2)])

>>> n.move\_to\_end('f')

>>> n

OrderedDict([('a', 4), ('q', 2), ('g', 4), ('d', 2), ('e', 2), ('f', 2)])

**=>defaultdict**

我们都知道，在使用Python原生的数据结构dict的时候，如果用 d[key] 这样的方式访问， 当指定的key不存在时，是会抛出KeyError异常的。

但是，如果使用defaultdict，只要你传入一个默认的工厂方法，那么请求一个不存在的key时， 便会调用这个工厂方法使用其结果来作为这个key的默认值。

Example:

>>> s = [('yellow', 1), ('blue', 2), ('yellow', 3), ('blue', 4), ('red', 1)]

>>> d = defaultdict(list)

>>> for k, v in s:

d[k].append(v)

>>> list(d.items())

[('blue', [2, 4]), ('red', [1]), ('yellow', [1, 3])]

>>> m =dict()

>>> m["a"]

Traceback (most recent call last):

File"<stdin>", line1,in<module>

KeyError: "a"

>>>

>>> m =collections.defaultdict(int)

>>> m["a"]

0

>>> m["b"]

0

>>> m =collections.defaultdict(str)

>>> m["a"]

""

>>> m["b"]+="a"

>>> m["b"]

"a"

>>> m =collections.defaultdict(lambda:"[default value]")

>>> m["a"]

"[default value]"

>>> m["b"]

"[default value]"

**1.5 copy模块**

提供浅拷贝和深拷贝两种模式。

=>copy(x):返回x的浅拷贝

=>deepcopy(x):返回x的深拷贝

浅拷贝和深拷贝：

**浅拷贝复制不变对象，引用可变对象(如列表和字典)；**

**深拷贝复制可变和不可变对象。**

浅拷贝的几种方法：

(1)、使用切片[:]操作进行拷贝

(2)、使用工厂函数(如list/dir/set)等进行拷贝

(3)、copy.copy()

Example:

#深拷贝浅拷贝

import copy

person = [1,2,4,['a','b']]

b = person **#对象的赋值实际上是对象的引用**

c=person[:]

d=copy.copy(person)

e=copy.deepcopy(person)

person.append('hello')

b[3].insert(1,'bbb')

b.insert(1,'bbb')

c.insert(1,'ccc')

c[4].insert(1,'ccc')

d.insert(1,'ddd')

e.insert(1,'test')

print('原始数据person:\t',person)

print('赋值b:\t',b)

print('浅拷贝c:\t',c)

print('浅拷贝d:\t',d)

print('深拷贝e:\t',e)

原始数据person: [1, 'bbb', 2, 4, ['a', 'ccc', 'bbb', 'b'], 'hello']

赋值b: [1, 'bbb', 2, 4, ['a', 'ccc', 'bbb', 'b'], 'hello']

浅拷贝c: [1, 'ccc', 2, 4, ['a', 'ccc', 'bbb', 'b']]

浅拷贝d: [1, 'ddd', 2, 4, ['a', 'ccc', 'bbb', 'b']]

深拷贝e: [1, 'test', 2, 4, ['a', 'b']]

**1.6 subprocess模块**

**1.7 time模块**

Python中的时间模块。

1. 在Python中，通常有这几种方式来表示时间：1）时间戳 2）格式化的时间字符串 3）元组（struct\_time）共九个元素。
2. UTC（Coordinated Universal Time，世界协调时）亦即格林威治天文时间，世界标准时间。在中国为UTC+8。DST（Daylight Saving Time）即夏令时。

3、时间戳（timestamp）的方式：通常来说，时间戳表示的是从1970年1月1日00:00:00开始按秒计算的偏移量。我们运行“type(time.time())”，返回的是float类型。返回时间戳方式的函数主要有time()，clock()等。

4、元组（struct\_time）方式：struct\_time元组共有9个元素，返回struct\_time的函数主要有gmtime()，localtime()，strptime()。下面列出这种方式元组中的几个元素：

|  |  |  |
| --- | --- | --- |
|  | | |
| 索引（Index） | 属性（Attribute） | 值（Values） |
| 0 | tm\_year（年） | 比如2011 |
| 1 | tm\_mon（月） | 1 - 12 |
| 2 | tm\_mday（日） | 1 - 31 |
| 3 | tm\_hour（时） | 0 - 23 |
| 4 | tm\_min（分） | 0 - 59 |
| 5 | tm\_sec（秒） | 0 - 61 |
| 6 | tm\_wday（weekday） | 0 - 6（0表示周日） |
| 7 | tm\_yday（一年中的第几天） | 1 - 366 |
| 8 | tm\_isdst（是否是夏令时） | 默认为-1 |

1. **time.localtime([secs])**：将一个时间戳转换为当前时区的struct\_time。secs参数未提供，则以当前时间为准。

>>> time.localtime()  
time.struct\_time(tm\_year=2011, tm\_mon=5, tm\_mday=5, tm\_hour=14, tm\_min=14, tm\_sec=50, tm\_wday=3, tm\_yday=125, tm\_isdst=0)  
>>> time.localtime(1304575584.1361799)  
time.struct\_time(tm\_year=2011, tm\_mon=5, tm\_mday=5, tm\_hour=14, tm\_min=6, tm\_sec=24, tm\_wday=3, tm\_yday=125, tm\_isdst=0)

（2）**time.gmtime([secs])**：和localtime()方法类似，gmtime()方法是将一个时间戳转换为UTC时区（0时区）的struct\_time。

>>>time.gmtime()  
time.struct\_time(tm\_year=2011, tm\_mon=5, tm\_mday=5, tm\_hour=6, tm\_min=19, tm\_sec=48, tm\_wday=3, tm\_yday=125, tm\_isdst=0)

（3）**time.time()**：返回当前时间的时间戳。

>>> time.time()

1304575584.1361799

（4）**time.mktime(t)**：将一个struct\_time转化为时间戳。

>>> time.mktime(time.localtime())

1304576839.0

1. **time.sleep(secs)**：线程推迟指定的时间运行。单位为秒。

（6）**time.clock()**：这个需要注意，在不同的系统上含义不同。在UNIX系统上，它返回的是“进程时间”，它是用秒表示的浮点数（时间戳）。而在WINDOWS中，第一次调用，返回的是进程运行的实际时间。而第二次 之后的调用是自第一次调用以后到现在的运行时间。（实际上是以WIN32上QueryPerformanceCounter()为基础，它比毫秒表示更为精确）

import time

if \_\_name\_\_ == '\_\_main\_\_':

time.sleep(1)

print ("clock1:%s" % time.clock())

time.sleep(1)

print ("clock2:%s" % time.clock())

time.sleep(1)

print ("clock3:%s" % time.clock())

运行结果：

clock1:3.35238137808e-006

clock2:1.00004944763

clock3:2.00012040636

其中第一个clock()输出的是程序运行时间

1. 三个clock()输出的都是与第一个clock的时间间隔
2. （7）**time.asctime([t])**：把一个表示时间的元组或者struct\_time表示为这种形式：'Sun Jun 20 23:21:05 1993'。如果没有参数，将会将time.localtime()作为参数传入。

>>> time.asctime()

'Thu May 5 14:55:43 2011'

（8）**time.ctime([secs])**：把一个时间戳（按秒计算的浮点数）转化为time.asctime()的形式。如果参数未给或者为None的时候，将会默认time.time()为参数。它的作用相当于time.asctime(time.localtime(secs))。

>>> time.ctime()

'Thu May 5 14:58:09 2011'

>>> time.ctime(time.time())

'Thu May 5 14:58:39 2011'

>>> time.ctime(1304579615)

'Thu May 5 15:13:35 2011'

1. **time.strftime(format[, t])**： 把一个代表时间的元组或者struct\_time（如由time.localtime()和time.gmtime()返回）转化为格式化的时间字符串。 如果t未指定，将传入time.localtime()。如果元组中任何一个元素越界，ValueError的错误将会被抛出。

|  |  |  |
| --- | --- | --- |
| **格式** | **含义** | **备注** |
| %a | 本地（locale）简化星期名称 |  |
| %A | 本地完整星期名称 |  |
| %b | 本地简化月份名称 |  |
| %B | 本地完整月份名称 |  |
| %c | 本地相应的日期和时间表示 |  |
| %d | 一个月中的第几天（01 - 31） |  |
| %H | 一天中的第几个小时（24小时制，00 - 23） |  |
| %I | 第几个小时（12小时制，01 - 12） |  |
| %j | 一年中的第几天（001 - 366） |  |
| %m | 月份（01 - 12） |  |
| %M | 分钟数（00 - 59） |  |
| %p | 本地am或者pm的相应符 | 一 |
| %S | 秒（01 - 61） | 二 |
| %U | 一年中的星期数。（00 - 53星期天是一个星期的开始。）第一个星期天之前的所有天数都放在第0周。 | 三 |
| %w | 一个星期中的第几天（0 - 6，0是星期天） | 三 |
| %W | 和%U基本相同，不同的是%W以星期一为一个星期的开始。 |  |
| %x | 本地相应日期 |  |
| %X | 本地相应时间 |  |
| %y | 去掉世纪的年份（00 - 99） |  |
| %Y | 完整的年份 |  |
| %Z | 时区的名字（如果不存在为空字符） |  |
| %% | ‘%’字符 |  |

**备注：**

“%p”只有与“%I”配合使用才有效果。

文档中强调确实是0 - 61，而不是59，闰年秒占两秒（汗一个）。

当使用strptime()函数时，只有当在这年中的周数和天数被确定的时候%U和%W才会被计算。

举个例子：

>>> time.strftime("%Y-%m-%d %X", time.localtime())

'2011-05-05 16:37:06'

（10）**time.strptime(string[, format])**：把一个格式化时间字符串转化为struct\_time。实际上它和strftime()是逆操作。

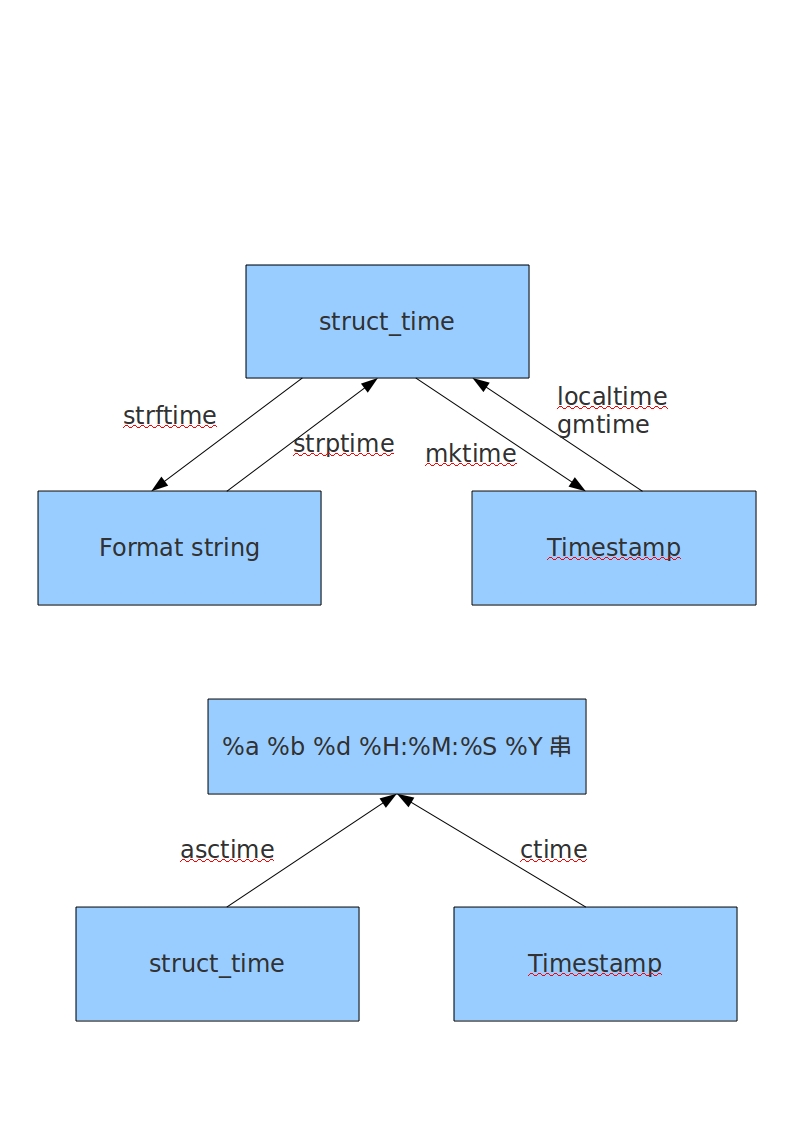
>>> time.strptime('2011-05-05 16:37:06', '%Y-%m-%d %X')

time.struct\_time(tm\_year=2011, tm\_mon=5, tm\_mday=5, tm\_hour=16, tm\_min=37, tm\_sec=6, tm\_wday=3, tm\_yday=125, tm\_isdst=-1)

在这个函数中，format默认为："%a %b %d %H:%M:%S %Y"。

最后，我们来对time模块进行一个总结。根据之前描述，在Python中共有三种表达方式：1）timestamp 2）tuple或者struct\_time 3）格式化字符串。

它们之间的转化如图所示：



**1.8 logging模块**

**1.9 datetime模块**

Python中的时间模块。time模块我在之前的文章已经有所介绍，它提供的接口与C标准库time.h基本一致。相比于time模块，datetime模块的接口则更直观、更容易调用。

datetime模块定义了两个常量：datetime.MINYEAR和datetime.MAXYEAR，分别表示datetime所能表示的最小、最大年份。其中，MINYEAR = 1，MAXYEAR = 9999。（对于偶等玩家，这个范围已经足够用矣~~）   
  
datetime模块定义了下面这几个类：   
  
•datetime.date：表示日期的类。常用的属性有year, month, day；   
•datetime.time：表示时间的类。常用的属性有hour, minute, second, microsecond；   
•datetime.datetime：表示日期时间。   
•datetime.timedelta：表示时间间隔，即两个时间点之间的长度。   
•datetime.tzinfo：与时区有关的相关信息。（这

里不详细充分讨论该类，感兴趣的童鞋可以参考python手册）   
注：上面这些类型的对象都是不可变（immutable）的

**1.10 xml模块**

**1.11 urllib模块**

**1.12 tkinter模块**

**1.13 http.client模块**

**1.14 random模块**

**1.15 functools模块**

**1.16 itertools模块**

**1.17 glob模块**

**二、python重要知识点**

2.1 迭代器

可以直接作用于for循环的数据类型有以下几种：

一类是集合数据类型，如list、tuple、dict、set、str等；

一类是generator，包括生成器和带yield的generator function。

这些可以直接作用于for循环的对象统称为可迭代对象：Iterable。

可以使用isinstance()判断一个对象是否是Iterable对象：

>>> from collections import Iterable

>>> isinstance([], Iterable)

True

>>> isinstance({}, Iterable)

True

>>>isinstance('abc', Iterable)

True

>>> isinstance((x for x in range(10)), Iterable)

True

>>> isinstance(100, Iterable)

False

而生成器不但可以作用于for循环，还可以被next()函数不断调用并返回下一个值，直到最后抛出StopIteration错误表示无法继续返回下一个值了。

可以被next()函数调用并不断返回下一个值的对象称为迭代器：Iterator。

可以使用isinstance()判断一个对象是否是Iterator对象：

>>> from collections import Iterator

>>> isinstance((x for x in range(10)), Iterator)

True

>>> isinstance([], Iterator)

False

>>> isinstance({}, Iterator)

False

>>> isinstance('abc', Iterator)

False

生成器都是Iterator对象，但list、dict、str虽然是Iterable，却不是Iterator。

把list、dict、str等Iterable变成Iterator可以使用iter()函数：

>>> isinstance(iter([]), Iterator)

True

>>> isinstance(iter('abc'), Iterator)

True

你可能会问，为什么list、dict、str等数据类型不是Iterator？

这是因为Python的Iterator对象表示的是一个数据流，Iterator对象可以被next()函数调用并不断返回下一个数据，直到没有数据时抛出StopIteration错误。可以把这个数据流看做是一个有序序列，但我们却不能提前知道序列的长度，只能不断通过next()函数实现按需计算下一个数据，所以Iterator的计算是惰性的，只有在需要返回下一个数据时它才会计算。

Iterator甚至可以表示一个无限大的数据流，例如全体自然数。而使用list是永远不可能存储全体自然数的。

总结：

凡是可作用于for循环的对象都是Iterable类型；

凡是可作用于next()函数的对象都是Iterator类型，它们表示一个惰性计算的序列；

集合数据类型如list、dict、str等是Iterable但不是Iterator，不过可以通过iter()函数获得一个Iterator对象。

Python的for循环本质上就是通过不断调用next()函数实现的，例如：

for x in [1, 2, 3, 4, 5]:

pass

实际上完全等价于：

# 首先获得Iterator对象:

it = iter([1, 2, 3, 4, 5])

# 循环:

while True:

try:

# 获得下一个值:

x = next(it)

except StopIteration:

# 遇到StopIteration就退出循环

break

2.2 生成器

通过列表生成式，我们可以直接创建一个列表。但是，受到内存限制，列表容量肯定是有限的。而且，创建一个包含100万个元素的列表，不仅占用很大的存储空间，如果我们仅仅需要访问前面几个元素，那后面绝大多数元素占用的空间都白白浪费了。

所以，如果列表元素可以按照某种算法推算出来，那我们是否可以在循环的过程中不断推算出后续的元素呢？这样就不必创建完整的list，从而节省大量的空间。在Python中，这种一边循环一边计算的机制，称为生成器：generator。

要创建一个generator，有很多种方法。第一种方法很简单，只要把一个列表生成式的[]改成()，就创建了一个generator：

>>> L = [x \* x for x in range(10)]

>>> L

[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

>>> g = (x \* x for x in range(10))

>>> g

<generator object <genexpr> at 0x1022ef630>

创建L和g的区别仅在于最外层的[]和()，L是一个list，而g是一个generator。

我们可以直接打印出list的每一个元素，但我们怎么打印出generator的每一个元素呢？

如果要一个一个打印出来，可以通过next()函数获得generator的下一个返回值：

>>> next(g)

0

>>> next(g)

1

......

......

>>> next(g)

81

>>> next(g)Traceback (most recent call last):

File "<stdin>", line 1, in <module>StopIteration

我们讲过，generator保存的是算法，每次调用next(g)，就计算出g的下一个元素的值，直到计算到最后一个元素，没有更多的元素时，抛出StopIteration的错误。

当然，上面这种不断调用next(g)实在是太变态了，正确的方法是使用for循环，因为generator也是可迭代对象：

>>> g = (x \* x for x in range(10))

>>> for n in g:

print(n)

0

1

4

....

....

81

所以，我们创建了一个generator后，基本上永远不会调用next()，而是通过for循环来迭代它，并且不需要关心StopIteration的错误。

generator非常强大。如果推算的算法比较复杂，用类似列表生成式的for循环无法实现的时候，还可以用函数来实现。

比如，著名的斐波拉契数列（Fibonacci），除第一个和第二个数外，任意一个数都可由前两个数相加得到：

1, 1, 2, 3, 5, 8, 13, 21, 34, ...

斐波拉契数列用列表生成式写不出来，但是，用函数把它打印出来却很容易：

def fib(max):

n, a, b = 0, 0, 1

while n < max:

print(b)

a, b = b, a + b

n = n + 1

return 'done'

注意，赋值语句：

a, b = b, a + b

相当于：

t = (b, a + b) # t是一个tuple:a = t[0];b = t[1]

但不必显式写出临时变量t就可以赋值。

上面的函数可以输出斐波那契数列的前N个数：

>>> fib(6)

1

1

2

3

5

8

'done'

仔细观察，可以看出，fib函数实际上是定义了斐波拉契数列的推算规则，可以从第一个元素开始，推算出后续任意的元素，这种逻辑其实非常类似generator。

也就是说，上面的函数和generator仅一步之遥。要把fib函数变成generator，只需要把print(b)改为yield b就可以了：

def fib(max):

n, a, b = 0, 0, 1

while n < max:

yield b

a, b = b, a + b

n = n + 1

return 'done'

这就是定义generator的另一种方法。如果一个函数定义中包含yield关键字，那么这个函数就不再是一个普通函数，而是一个generator：

>>> f = fib(6)

>>> f<generator object fib at 0x104feaaa0>

这里，最难理解的就是generator和函数的执行流程不一样。函数是顺序执行，遇到return语句或者最后一行函数语句就返回。而变成generator的函数，在每次调用next()的时候执行，遇到yield语句返回，再次执行时从上次返回的yield语句处继续执行。

举个简单的例子，定义一个generator，依次返回数字1，3，5：

def odd():

print('step 1')

yield 1

print('step 2')

yield(3)

print('step 3')

yield(5)

调用该generator时，首先要生成一个generator对象，然后用next()函数不断获得下一个返回值：

>>> o = odd()

>>> next(o)

step 11

>>> next(o)

step 23

>>> next(o)

step 35

>>> next(o)Traceback (most recent call last):

File "<stdin>", line 1, in <module>StopIteration

可以看到，odd不是普通函数，而是generator，在执行过程中，遇到yield就中断，下次又继续执行。执行3次yield后，已经没有yield可以执行了，所以，第4次调用next(o)就报错。

回到fib的例子，我们在循环过程中不断调用yield，就会不断中断。当然要给循环设置一个条件来退出循环，不然就会产生一个无限数列出来。

同样的，把函数改成generator后，我们基本上从来不会用next()来获取下一个返回值，而是直接使用for循环来迭代：

>>> for n in fib(6):

print(n)

1

1

2

3

5

8

但是用for循环调用generator时，发现拿不到generator的return语句的返回值。如果想要拿到返回值，必须捕获StopIteration错误，返回值包含在StopIteration的value中：

>>> g = fib(6)

>>> while True:

try:

x = next(g)

print('g:', x)

except StopIteration as e:

print('Generator return value:', e.value)

break

g: 1

g: 1

g: 2

g: 3

g: 5

g: 8

Generator return value: done

关于如何捕获错误，后面的错误处理还会详细讲解。

总结：

generator是非常强大的工具，在Python中，可以简单地把列表生成式改成generator，也可以通过函数实现复杂逻辑的generator。

要理解generator的工作原理，它是在for循环的过程中不断计算出下一个元素，并在适当的条件结束for循环。对于函数改成的generator来说，遇到return语句或者执行到函数体最后一行语句，就是结束generator的指令，for循环随之结束。

请注意区分普通函数和generator函数，普通函数调用直接返回结果：

>>> r = abs(6)

>>> r

6

generator函数的“调用”实际返回一个generator对象：

>>> g = fib(6)

>>> g<generator object fib at 0x1022ef948>

2.3排序sorted和sort

Python list内置sort()方法用来排序，也可以用python内置的全局sorted()方法来对可迭代的序列排序生成新的序列。

sorted(iterable,key=None,reverse=False)，返回新的列表，对所有可迭代的对象均有效

sort(key=None,reverse=False) 就地改变列表 reverse：True反序；False 正序

**Example1:**

>>>sorted([1,5,3,2,9])

[1,2,3,5,9]

>>>a=[5,3,2,1,4]

>>>a.sort()

>>>a

[1,2,3,4,5] #若用list.sort()则list本身将被修改

>>>sorted({1: 'D', 2: 'B', 3: 'B', 4: 'E', 5: 'A'})

[1,2,3,4,5] #sorted()对所有的可迭代序列都有效

在python2.4开始，list.sort()和sorted()增加key参数来指定一个函数，此函数在每个元素比较前被调用。

**Example2：**

>>>sorted("This is a test string from Andrew".split(), key=str.lower) #加了key，忽略大小写

['a', 'Andrew', 'from', 'is', 'string', 'test', 'This'] #key=len按照长度进行排序

>>>sorted("This is a test string from Andrew".split()) #未加key，默认大写在前，小写在后

['Andrew', 'This', 'a', 'from', 'is', 'string', 'test']

更多的情况是用复杂对象的某些值来对复杂对象进行排序。

**Example3:**

>>> student\_tuples = [('john', 'A', 15),('jane', 'B', 12),('dave', 'B', 10),]

>>> sorted(student\_tuples, key=lambda student: student[2])   # sort by age

[('dave', 'B', 10), ('jane', 'B', 12), ('john', 'A', 15)]

>>>student\_tuples.sort(key=lambda x: x[2])

[('dave', 'B', 10), ('jane', 'B', 12), ('john', 'A', 15)]

**Example4:**

>>>s=”Hello79351WorldMyNameIsMrFiona0352231964”

>>>''.join(sorted(s,key=lambda x: (x.isdigit(),x.isdigit() and int(x)%2==0,x.islower(),x.isupper(),x)))

'FHIMMNWaadeeilllmnooorrsy113335579902246'

大写在前，小写在后，数字放在最后并且奇数在偶数之前

>>>s={‘a’:10,’t’:5,’c’:2,’b’:12}

>>>sorted(s,key=lambda x:x[0])

[‘a’,’b’,’c’,’t’]

>>>s=[]

2.4 内置函数

**1、abx(x)**

返回一个数的绝对值。参数可以是普通的整数，长整数或者浮点数。如果参数是个复数，返回它的模

1. **all(iterable)**

如果iterable的所有元素为真（或者iterable为空）， 返回True。等同于:

def all(iterable):

for element in iterable:

if not element:

return False

return True

1. **any(iterable)**

如果iterable的任一元素为真，返回True。如果iterable为空，返回False。等同于：

def any(iterable):

for element in iterable:

if element:

return True

return False

1. **ascii(object)**

这个函数跟repr()函数一样，返回一个可打印的对象字符串方式表示。当遇到非ASCII码时，就会输出\x，\u或\U等字符来表示。与Python 2版本里的repr()是等效的函数。

**>>>print**(ascii(10), ascii(9000000), ascii('b\31'), ascii('0x\1000'))

10   9000000   'b\x19'   '0x@0'

1. **bin(x)**

将一个整数转化成一个二进制字符串。结果是一个合法的Python表达式。如果x不是一个Python int对象，它必须定义一个返回整数的\_\_index\_\_()方法

1. **bool([x])**

将一个值转化成布尔值，使用标准的真值测试例程。如果x为假或者没有参数，它返回False；否则它返回True。bool也是一个类，它是int的子类。bool不能被继承。它唯一的实例就是False和True

1. **bytearray([source[, encoding[, errors]]])**

返回一个新的字节数组。bytearray类型是一个可变的整数序列，整数范围为0 <= x < 256（即字节）。 它有可变序列的大多数方法，参见Mutable Sequence Types，同时它也有str类型的大多数方法，参见String Methods。

source参数可以以不同的方式来初始化数组，它是可选的：

如果是string，必须指明encoding（以及可选的errors）参数；bytearray()使用str.encode()将字符串转化为字节数组。

如果是integer，生成相应大小的数组，元素初始化为空字节。

如果是遵循buffer接口的对象，对象的只读buffer被用来初始化字节数组。

如果是iterable，它的元素必须是整数，其取值范围为0 <= x < 256，用以初始化字节数组。

如果没有参数，它创建一个大小为0的数组。

>>> c=bytearray('qwe1321','utf-8')

>>> c

bytearray(b'qwe1321')

>>> c[1]

119

**8、bytes([source[, encoding[, errors]]])**

返回一个新的字节数组。bytearray类型是一个不可变的整数序列，整数范围为0 <= x < 256（即字节）。其他类似于bytearray

1. **callable(object)**

如果object参数可调用，返回True；否则返回False。如果返回真，对其调用仍有可能失败；但是如果返回假，对object的调用总是失败。注意类是可调用的（对类调用返回一个新实例）；如果类实例有\_\_call\_\_()方法，则它们也是可调用的

1. **chr(i)**

返回一个单字符字符串，字符的ASCII码为整数i。例如，chr(97)返回字符串'a'。 它是ord()的逆运算。参数的取值范围为[0..255]的闭区间；如果i超出取值范围，抛出ValueError。参见unichr()

1. **classmethod(function)**

将function包装成类方法。

类方法接受类作为隐式的第一个参数，就像实例方法接受实例作为隐式的第一个参数一样。声明一个类方法，使用这样的惯例：

class C(object):

@classmethod

def f(cls, arg1, arg2, ...):

...

@classmethod是函数decorator（装饰器）参见Function definitions中的函数定义。

它即可以通过类来调用（如C.f()），也可以通过实例来调用（如C().f()）。除了实例的类，实例本身被忽略。如果在子类上调用类方法，子类对象被传递为隐式的第一个参数。

类方法不同于C++或Java中的静态方法。如果你希望静态方法，参见这节的staticmethod()。需要类方法更多的信息，参见The standard type hierarchy中标准类型层次部分的文档。

1. **compile(source, filename, mode[, flags[, dont\_inherit]])**
2. **complex([real[, imag]])**

创建一个复数，它的值为real + imag\*j；或者将一个字符串／数字转化成一个复数。如果第一个参数是个字符串，它将被解释成复数，同时函数不能有第二个参数。第二个参数不能是字符串。每个参数必须是数值类型（包括复数）。如果imag被忽略，它的默认值是0，这时该函数就像是int()，long()和float()这样的数值转换函数。如果两个参数都被忽略，返回0j。

注意：当从字符串转化成复数的时候，字符串中+或者-两边不能有空白。例如，complex('1+2j')是可行的，但complex('1 + 2j')会抛出ValueError异常。

1. **delattr(object, name)**

这个函数和setattr()有关。参数是一个对象和一个字符串。字符串必须是对象的某个属性的名字。只要对象允许，这个函数删除该名字对应的属性。例如，delattr(x, 'foobar')等同于del x.foobar

1. **dict(\*\*kwarg) 、dict(mapping, \*\*kwarg) 、dict(iterable, \*\*kwarg)**

返回一个新字典，从一个可选的位置参数和一个可能为空的关键字参数集合初始化。

如果没有给定位置参数，则创建一个空的字典。如果给定位置参数且它是一个映射对象，则创建一个与该映射对象具有相同的键-值对的一个字典。否则，定位参数必须是一个可迭代的对象。可迭代对象中的每个元素必须本身是具有恰好两个对象的可迭代对象。每个元素的第一个对象就成为新字典的键，第二个对象成为对应的值。如果键出现超过一次，该键的最后一个值将成为新字典中相应的值。

如果给出关键字参数，关键字参数和它们的值会被添加到从位置参数创建的字典。如果正在添加的键已存在，来自关键字参数的值将替换来自位置参数的值。

为了说明，下面所有的例子都返回与{"one": 1, "two": 2, "three": 3}相等的一个字典：

>>> a = dict(one=1, two=2, three=3)

>>> b = {'one': 1, 'two': 2, 'three': 3}

>>> c = dict(zip(['one', 'two', 'three'], [1, 2, 3]))

>>> d = dict([('two', 2), ('one', 1), ('three', 3)])

>>> e = dict({'three': 3, 'one': 1, 'two': 2})

>>> a == b == c == d == e

True

1. **dir([object])**

如果没有参数，返回当前本地作用域内的名字列表。如果有参数，尝试返回参数所指明对象的合法属性的列表。

如果对象有\_\_dir\_\_()方法，该方法被调用且必须返回一个属性列表。这允许实现了定制化的\_\_getattr\_\_()或者\_\_getattribute\_\_()函数的对象定制dir()报告对象属性的方式。

如果对象没有提供\_\_dir\_\_()，同时如果对象有定义\_\_dict\_\_属性，dir()会先尝试从\_\_dict\_\_属性中收集信息，然后是对象的类型对象。如果对象是自定义的话，返回列表有可能不完整或不准确\_\_getattr\_\_()。

对于不同类型的对象，默认的dir()行为也不同，因为它尝试产生相关的而不是完整的信息：

如果对象是模块对象，列表包含模块的属性名。

如果对象是类型或者类对象，列表包含类的属性名，及它的基类的属性名。

否则，列表包含对象的属性名，它的类的属性名和类的基类的属性名。

返回的列表按字母顺序排序。例如：

>>> import struct

>>> dir() # show the names in the module namespace

['\_\_builtins\_\_', '\_\_doc\_\_', '\_\_name\_\_', 'struct']

>>> dir(struct) # show the names in the struct module

['Struct', '\_\_builtins\_\_', '\_\_doc\_\_', '\_\_file\_\_', '\_\_name\_\_',

'\_\_package\_\_', '\_clearcache', 'calcsize', 'error', 'pack', 'pack\_into',

'unpack', 'unpack\_from']

>>> class Shape(object):

def \_\_dir\_\_(self):

return ['area', 'perimeter', 'location']

>>> s = Shape()

>>> dir(s)

['area', 'perimeter', 'location']

注意 因为dir()主要是为了在交互式环境下使用方便，它尝试提供有意义的名字的集合，而不是提供严格或一致定义的名字的集合，且在不同的版本中，具体的行为也有所变化。例如，如果参数是一个类，那么元类属性就不会出现在结果中。

1. **divmod(a, b)**

在长整数除法中，传入两个数字（非复数）作为参数，返回商和余数的二元组。对于混合的操作数类型，应用二元算术运算符的规则。对于普通整数或者长整数，结果等同于(a // b, a % b)。对于浮点数结果是(q, a % b)，q一般是math.floor(a / b)，但也可能比那小1。不管怎样，q \* b + a % b非常接近于a，如果a % b非0，它和b符号相同且0 <= abs(a % b) < abs(b)。

1. **enumerate(sequence, start=0)**

返回一个枚举对象。sequence必须是个序列，迭代器iterator，或者支持迭代的对象。enumerate()返回的迭代器的next()方法返回一个元组，它包含一个计数（从start开始，默认为0）和从sequence中迭代得到的值：

>>> seasons = ['Spring', 'Summer', 'Fall', 'Winter']

>>> list(enumerate(seasons))

[(0, 'Spring'), (1, 'Summer'), (2, 'Fall'), (3, 'Winter')]

>>> list(enumerate(seasons, start=1))

[(1, 'Spring'), (2, 'Summer'), (3, 'Fall'), (4, 'Winter')]

等同于：

def enumerate(sequence, start=0):

n = start

for elem in sequence:

yield n, elem

n += 1

1. **eval(expression[, globals[, locals]])**

参数是Unicode或者Latin-1编码的字符串，全局变量和局部变量可选。如果有全局变量，globals必须是个字典。如果有局部变量，locals可以是任何映射类型对象。

expression参数被当作Python表达式来解析并演算（技术上来说，是个条件列表），使用globals和locals字典作为全局和局部的命名空间。如果globals字典存在，且缺少‘\_\_builtins\_\_’，在expression被解析之前，当前的全局变量被拷贝进globals。这意味着一般来说expression能完全访问标准\_\_builtin\_\_模块，且受限的环境会传播。如果locals字典被忽略，默认是globals字典。如果都被忽略，表达式在eval()被调用的环境中执行。返回值是被演算的表达式的结果。语法错误报告成异常。例子：

>>> x = 1

>>> print eval('x+1')

2

该函数也能执行任意的代码对象（如compile()返回的结果）。 在这种情况下，传递代码对象而不是字符串。如果代码对象编译时mode参数为'exec'，eval()返回None。

提示：exec语句支持动态的语句执行。execfile()函数支持执行文件中的语句。globals()和locals()函数返回当前的全局变量和局部变量的字典，可以传递给eval()或者execfile()。

参见ast.literal\_eval()，该函数能安全演算只含字面量的表达式的字符串。

1. **exec(object[, globals[, locals]])**
2. **filter(function, iterable)**

构造一个列表，列表的元素来自于iterable，对于这些元素function返回真。iterable可以是个序列，支持迭代的容器，或者一个迭代器。如果iterable是个字符串或者元组，则结果也是字符串或者元组；否则结果总是列表。如果function是None，使用特性函数，即为假的iterable被移除。

注意，在function不为None的情况下，filter(function, iterable)等同于[item for item in iterable if function(item)]；否则等同于[item for item in iterable if item]（function为None）。

参见itertools.ifilter()和itertools.ifilterfalse()，以得到该函数的迭代器版本，以及该函数的变体（过滤function返回假的元素）。

1. **float([x])**

将字符串或者数字转化成浮点数。如果参数是字符串，它必须包含小数或者浮点数（可以有符号），周围可以有空白。参数也可以是[+|-]nan或者[+|-]inf。其它情况下，参数可以是原始／长整数或者浮点数，（以Python的浮点数精度）返回具有相同值的浮点数。如果没有参数，返回0.0。

注意 当传递字符串时，依赖于底层的C库，可以返回NaN（Not a Number，不是一个数字）和Infinity（无穷大）这样的值。该函数接受字符串nan（NaN），inf（正无穷大）和-inf（负无穷大）。对于NaN，不区分大小写和+/-号。总是用nan，inf或者-inf来表示NaN和Infinity。

1. **format(value[, format\_spec])**

将value转化成“格式化”的表现形式，格式由format\_spec控制。对format\_spec的解释依赖于value参数的类型，大多数内置类型有标准的格式化语法：Format Specification Mini-Language。

注意 format(value, format\_spec)仅仅调用value.\_\_format\_\_(format\_spec)。

1. **frozenset([iterable])**

返回一个新的frozenset对象，如果可选参数iterable存在，frozenset的元素来自于iterable。frozenset是个内置类。参见frozenset和Set Types — set, frozenset。

关于其它容器，参见set，list，tuple，和dict类，以及collections模块。

1. **getattr(object, name[, default])**

返回object的属性值。name必须是个字符串。如果字符串是对象某个属性的名字，则返回该属性的值。例如，getattr(x, 'foobar')等同于x.foobar。如果名字指明的属性不存在，且有default参数，default被返回；否则抛出AttributeError。

1. **globals()**

返回表示当前全局符号表的字典。它总是当前模块的字典（在函数或者方法中，它指定义的模块而不是调用的模块）

**27、hasattr(object, name)**

参数是一个对象和一个字符串。如果字符串是对象某个属性的名字，返回True；否则返回False。（实现方式为调用getattr(object, name)，看它是否抛出异常）。

**28、hash(object)**

返回对象的hash（哈希/散列）值（如果有的话）。hash值是整数。它被用于在字典查找时快速比较字典的键。相同的数值有相同的hash（尽管它们有不同的类型，比如1和1.0）

**29、help([object])**

调用内置的帮助系统。（这个函数主要用于交互式使用。）如果没有参数，在解释器的控制台启动交互式帮助系统。如果参数是个字符串，该字符串被当作模块名，函数名，类名，方法名，关键字或者文档主题而被查询，在控制台上打印帮助页面。如果参数是其它某种对象，生成关于对象的帮助页面。

这个函数经由site模块加入内置的命名空间。

**30、hex(x)**

将任意大小的整数转化成以“0x”打头的小写的十六进制字符串，例如：

>>> hex(255)

'0xff'

>>> hex(-42)

'-0x2a'

>>> hex(1L)

'0x1L'

如果x不是Python的int或者long对象，它必须定义\_\_index\_\_()方法以返回一个整数。

参见int()，它将十六进制字符串转化成一个整数。

注意 使用float.hex()方法得到浮点数的十六进制字符串表示。

**31、id(object)**

返回对象的“标识”。这是一个整数（或长整数），保证在对象的生命期内唯一且不变。生命期不重叠的两个对象可以有相同的id()值。

CPython实现细节： 这是对象的内存地址。

**32、input([prompt])**

该函数不会捕获用户错误。如果输入语法不合法，将抛出SyntaxError。如果执行中有错误，将抛出其它异常。

如果有装载readline，input()将会用它来提供复杂的行编辑和历史功能。

建议使用raw\_input()函数来得到用户的一般输入。

**33、int(x=0)、int(x, base=10)**

将数字或字符串x转化成一个整数，如果没有参数则返回0。如果x是个数字，它可以是普通整数／长整数，或者浮点数。如果x是浮点数，则向0截断。如果参数超出了整数的范围，则返回长整数对象。

如果x不是个数字，或者存在base参数，则x必须是个表示以base为基数的integer literal（整数字面量）的字符串或者Unicode对象。字面量的前面可以有+或者-（中间不能有空格），周围可以有空白。以n为基数的字面量包含数字0到n-1，用a到z（或者A到Z）来表示10到35。默认的base是10。允许的值为0和2-36。二进制，八进制和十六进制的字面量前面可以有0b/0B，0o/0O/0，或者0x/0X，就像代码中的整数字面量一样。基数0表示严格按整数字面量来解释字符串，所以实际的基数为2，8，10或者16。

整数类型描述于Numeric Types — int, float, long, complex。

**34、isinstance(object, classinfo)**

如果参数object 是参数classinfo 的一个实例；或者是其一个（直接的、间接的或者virtual）子类的实例，返回真。如果classinfo是类型对象（新式类）而object是该类型对象；或者是其子类（直接的，间接的，或者virtual），返回真。如果object不是给定类型的类实例或者对象，该函数总是返回假。如果classinfo既不是类对象，也不是类型对象，它可以是类／类型对象的元组，或者递归包含这样的元组（不接受其它的序列类型）。如果classinfo不是类，类型，类／类型的元组，抛出TypeError异常。

**35、issubclass(class, classinfo)**

如果class是classinfo的子类（直接的，间接的，或者virtual) ，返回真。一个类被认为是它自己的子类。classinfo可以是类对象的元组，这时classinfo中的每个类对象都会被检查。其它情况下，抛出TypeError异常。

1. **iter(object[, sentinel])**

返回一个iterator对象。根据有无第二个参数，对第一个参数的解释相差很大。如果没有第二个参数，o必须是个集合对象，要么支持迭代协议（即 \_\_iter\_\_()方法），要么支持序列协议（即\_\_getitem\_\_()方法，整数参数从0开始）。 如果这些协议都不支持，抛出TypeError。如果有第二个参数sentinel，o必须是个可调用对象。这种情况下返回的迭代，每当调用其next()方法时，将会调用o（不带参数）；如果返回值等于sentinel，抛出StopIteration，否则返回该值。

第二种形式的iter()的一个有用的应用就是读一个文件的行，直到读到特定行。下面的例子读一个文件，直到readline()方法返回一个空字符串：

with open('mydata.txt') as fp:

for line in iter(fp.readline, ''):

process\_line(line)

**37、len(s)**

返回对象的长度（元素的个数）。参数可以是序列（如字符串，字节，元组，列表或者范围）或者集合（如字典，集合或者固定集合）。

**38、list([iterable])**

返回一个列表，其元素来自于iterable（保持相同的值和顺序）。iterable可以是个序列，支持迭代的容器，或者迭代器对象。如果iterable已经是个列表，返回其拷贝，类似于iterable[:]。例如，list('abc')返回['a', 'b', 'c']，list( (1, 2, 3) )返回[1, 2, 3]。如果没有参数，返回一个新的空的列表，[]。

list是可变序列类型，见文档Sequence Types — str, unicode, list, tuple, bytearray, buffer, xrange。关于其它容器参见内置dict，set，和tuple类，以及collections模块。

**39、locals()**

更新并返回表示当前局部符号表的字典。当locals在函数块中而不是类块中被调用时，locals()返回自由变量。

注意：不应该修改该字典的内容；所做的改变不一定会影响到解释器所用的局部和自由变量的值。

**40、map(function, iterable, ...)**

将function应用于iterable的每一个元素，返回结果的列表。如果有额外的iterable参数，并行的从这些参数中取元素，并调用function。如果一个参数比另外的要短，将以None扩展该参数元素。如果function是None使用特性函数；如果有多个参数，map()返回一元组列表，元组包含从各个参数中取得的对应的元素（某种变换操作）。iterable参数可以是序列或者任意可迭代对象；结果总是列表。

1. **max(iterable, \*[, key, default]) 、max(arg1, arg2, \*args[, key])**

返回可迭代的对象中的最大的元素，或者返回2个或多个参数中的最大的参数。

如果有一个位置参数，iterable必须是个非空的可迭代对象（如非空字符串，元组或者列表）。返回可迭代对象中最大的元素。如果有2个或更多的位置参数，返回最大位置参数。

可选的key参数指明了有一个参数的排序函数，如list.sort()中使用的排序函数。如果iterable为空，则返回default，如果iterable为空且default未设置的话则引发ValueError错误；如果有key参数，它必须是关键字参数（例如，max(a,b,c,key=func)

1. **memoryview(obj)**
2. **min(iterable, \*[, key, default]) 、min(arg1, arg2, \*args[, key])**

返回可迭代的对象中的最小的元素，或者返回2个或多个参数中的最小的参数。

如果有一个位置参数，iterable必须是个非空的可迭代对象（如非空字符串，元组或者列表）。返回可迭代对象中最小的元素。如果有2个或更多的位置参数，返回最小位置参数。

可选的key参数指明了有一个参数的排序函数，如list.sort()中使用的排序函数。如果iterable为空，则返回default，如果iterable为空且default未设置的话则引发ValueError错误；如果有key参数，它必须是关键字参数（例如，max(a,b,c,key=func)

**44、next(iterator[, default])**

通过调用iterator的next()方法，得到它的下一个元素。如果有default参数，在迭代器迭代完所有元素之后返回该参数；否则抛出StopIteration。

**45、object()**

返回一个新的无特征的对象。object是所有新式类的基类。它有对所有新式类的实例通用的方法。

**46、oct(x)**

将一个（任意尺寸）整数转化成一个八进制字符串。结果是一个合法的Python表达式。

1. **open(file, mode='r', buffering=-1, encoding=None, errors=None, newline=None, closefd=True, opener=None)**

**48、ord(c)**

给定一个长度为一的字符串，如果参数是unicode对象，则返回表示字符的代码点的整数；如果参数是八位字符串，返回字节值。例如，ord('a')返回整数97，ord(u'\u2020')返回8224。它是八位字符串chr()的反函数，也是unicode对象unichr()的反函数。如果参数是unicode且构建Python时添加了UCS2 Unicode支持，那么字符的码点必须在[0..65535]的闭区间；如果字符串的长度为2，则抛出TypeError。

**49、pow(x, y[, z])**

返回x 的 y次幂; 如果 z 提供的时候，, 返回 x 的 y 次幂，然后对 z 取模。(这样比 pow(x, y) % z) 更高效。两个参数的形式 pow(x, y) 与使用 操作符: x\*\*y 是等价的。

参数必须是数字类型的。由于操作数是混合类型的，二进制计算的原因需要一些强制的规定。对于整型和长整型的操作数，计算结果和操作数（强制后的）是相同的类型。除非第二个 参数是负数。在这种情况下, 所有的参数都会被转化成浮点型，并且会返回一个浮点的结果。例如, 10\*\*2 返回 100, 但 10\*\*-2 返回0.01. (这个新特性被加入在Python2.2中在Python2.1和之前的版本中,如果两个参数是整型，并且第二个参数为负数的情况下，会抛出一个异常。）如果第二个参数为负数，那么第三个参数必须省略。如果提供参数 z , x and y 必须为整数，而且y要是非负整数。(这个限制是在Python2.2加入的。Python2.1以及之前的版本, 三个参数都是浮点型的pow() 版本，返回的结果依赖 平台 对于浮点数的取整情况。)

**50、print(\*objects, sep=' ', end='\n', file=sys.stdout, flush=False)**

以sep分割，end的值结尾，将对象打印到文件流中。sep, end 和 file，如果提供这三个参数的话，必须以键值的形式。

所有非键值形式提供的参数，都被转化为字符串，就像用str()转化那样。然后写到 文件流中，以 sep 分割， end 结尾。sep and end 都必须是字符串形式的; 也可以留成 None, 这样会使用默认值。如果没有打印 对象, print() 只打印一个 结束符号 end.

file 参数一定要是含有 write(string)方法的对象 ; 如果该参数为空，或为None, 默认使用sys.stdout 作为输出。是否输出缓冲通常是由文件确定的，但如果刷新关键字flush参数是真的，流被强制刷新。

**52、property(fget=None, fset=None, fdel=None, doc=None)**

**53、range(start, stop[, step])**

这是一个创建包含数列的列表的通用函数。它最常用于for循环。参数必须为普通的整数。如果step参数省略，则默认为1。如果start参数省略，则默认为0。该函数的完整形式返回一个整数列表[start, start + step, start + 2 \* step, ...]。如果step为正，则最后一个元素start + i \* step最大且小于stop；如果step为负，则最后一个元素start + i \* step最小且大于stop。step必须不能为零（否则会引发ValueError）。

**54、repr(object)**

返回某个对象可打印形式的字符串。它与字符串转换式（反引号）产生的值相同。有时候能够把这个操作作为一个普通的函数访问非常有用。For many types, this function makes an attempt to return a string that would yield an object with the same value when passed to eval(), otherwise the representation is a string enclosed in angle brackets that contains the name of the type of the object together with additional information often including the name and address of the object. 类可以通过定义\_\_repr\_\_()方法控制该函数对其实例的返回。

**55、reversed(seq)**

返回一个反向迭代器。seq必须是一个具有\_\_reversed\_\_() 方法或支持序列协议的对象（整数参数从0开始的\_\_len\_\_()方法和\_\_getitem\_\_() 方法）。

**56、round(number[, ndigits])**

返回一个浮点型 近似值，保留小数点后 ndigits 位。如果省略ndigits，它默认为零。结果是一个浮点数。该值将被圆整为距离10的负ndigits次方的最近的某个倍数; 如果有两个倍数离的一样近,结果取离0较远的（所以，例如，round(0.5)是1.0 且round(-0.5)是-1.0）。

注意：浮点数round()的行为可能让人惊讶，例如round(2.675, 2)给出的是2.67 而不是期望的2.68。这不是一个错误：大部分十进制小数不能用浮点数精确表示，它是因为这样的一个事实的结果。更多信息，请参阅Floating Point Arithmetic: Issues and Limitations。

set([iterable])

返回一个新的set 对象，其元素可以从可选的iterable获得。set是一个内建的类。 关于该类的文档，请参阅set和集合类型 — set, frozenset。

关于其它容器请参阅内建的frozenset、list、tuple和dict类，还有collections模块。

**57、setattr(object, name, value)**

getattr()的相反操作。参数是一个对象、一个字符串和任何一个值。字符串可以是一个已存在属性的名字也可以是一个新属性的名字。该函数将值赋值给属性，只要对象允许。例如，setattr(x, 'foobar', 123)等同于x.foobar = 123。

**58、slice(start, stop[, step])**

返回一个slice对象，表示由索引range(start, stop, step)指出的集合。start和step参数默认为None。切片对象具有只读属性start、stop和step，它们仅仅返回参数的值（或者它们的默认值）。它们没有其他显式的函数；它是它们用于Numerical Python和其它第三方扩展。在使用扩展的索引语法时同样会生成切片对象。例如：a[start:stop:step]或a[start:stop, i]。返回迭代器的另外一个版本可以参阅itertools.islice()。

**59、sorted(iterable[, key][, reverse])**

依据iterable中的元素返回一个新的列表。

key和reverse与list.sort()方法的参数含义相同（在可变的序列类型一节描述）。

key指定一个带有一个参数的函数，它用于从每个列表元素选择一个比较的关键字：key=str.lower。默认值是None（直接比较元素）。

reverse是一个布尔值。如果设置为True，那么列表元素以反向比较排序。

通常情况下，key和reverse转换处理比指定一个等同的cmp函数要快得多。这是因为cmp为每个元素调用多次但是key和reverse只会触摸每个元素一次。使用functools.cmp\_to\_key()来转换旧式的cmp函数为key函数。

**60、staticmethod(function)**

返回function的一个静态方法。

静态方法不接受隐式的第一个参数（也就是实例名称self）。要声明静态方法，请使用下面的习惯方式：

class C(object):

@staticmethod

def f(arg1, arg2, ...):

...

@staticmethod形式是一个函数装饰器 – 细节请参阅函数定义中函数定义的描述。

它既可以在类上调用（例如C.f()）也可以在实例上调用（例如C().f()）。除了它的类型，实例其他的内容都被忽略。

**61、str(object=b'', encoding='utf-8', errors='strict')**

返回一个字符串，包含对象的友好的可打印表示形式。对于字符串，它返回字符串本身。与repr(object)的区别是str(object)不会永远试图返回一个eval()可接受的字符串；它的目标是返回一个可打印的字符串。如果没有给出参数，则返回空字符串''。

**62、sum(iterable[, start])**

将start以及iterable的元素从左向右相加并返回总和。start默认为0。iterable的元素通常是数字，start值不允许是一个字符串。

对于某些使用场景，有比sum()更好的选择。连接字符串序列的首选和快速的方式是调用''.join(sequence)。如要相加扩展精度的浮点数，请参阅math.fsum()。若要连接一系列的可迭代量，可以考虑使用itertools.chain()。

**63、super(type[, object-or-type])**

返回一个代理对象，这个对象指派方法给一个父类或者同类. 这对进入类中被覆盖的继承方法非常有用。搜索顺序和 getattr() 一样。而它自己的 类型 则被忽略。

type的\_\_mro\_\_属性罗列了用getattr()和super()来搜索排序的解决方法。它可以随着继承关系的变化，动态的更新。

如果没有第二个参数，则返回未绑定的super对象 If the second argument is an object, isinstance(obj, type) must be true. If the second argument is a type, issubclass(type2, type) must be true (this is useful for classmethods)。

**64、tuple([iterable])**

返回一个元组，其元素及顺序与iterable的元素相同。iterable可以是一个序列、支持迭代操作的容器或迭代器对象。如果iterable已经是一个元组，它将被原样返回。例如，tuple('abc')返回('a', 'b', 'c')，tuple([1, 2, 3])返回(1, 2, 3)。如果没有给出参数，则返回一个空的元组()。

tuple是一个不可变序列类型，其文档在序列类型 — str, unicode, list, tuple, bytearray, buffer, xrange。关于其它容器，请参阅内建的dict、list和set类以及collections模块。

**65、type(name, bases, dict)**

只有一个参数时，返回object的类型。返回值是一个类型对象。建议使用内建函数isinstance()测试一个对象的类型。

带有三个参数时，返回一个新的类型对象。它本质上是class语句的动态形式。name字符串是类的名字且将成为\_\_name\_\_属性；bases元组逐条列举基类并成为\_\_bases\_\_属性；dict字典是包含类体定义的命名空间并成为\_\_dict\_\_属性。例如，下面的两条语句创建完全相同的type对象：

>>> class X(object):

... a = 1

...

>>> X = type('X', (object,), dict(a=1))

**66、vars([object])**

返回模块、类、实例或其它任何具有\_\_dict\_\_属性的对象的\_\_dict\_\_属性。

模块和实例这样的对象具有可更新的\_\_dict\_\_属性；然而，其它对象可能对它们的\_\_dict\_\_属性具有写限制（例如，新式类使用dictproxy 来防止直接的字典更新）。

如果不带参数，则vars()的行为类似locals()。注意，局部字典只用于读取因为对局部字典的更新被忽略。

**67、zip(\*iterables)**

该函数返回一个以元组为元素的列表，其中第 i 个元组包含每个参数序列的第 i 个元素。返回的列表长度被截断为最短的参数序列的长度。当多个参数都具有相同的长度时，zip()类似于带有一个初始参数为None的map()。只有一个序列参数时，它返回一个1元组的列表。没有参数时，它返回一个空的列表。

可以保证迭代按从左向右的计算顺序。这使得使用zip(\*[iter(s)]\*n)来将一系列数据分类归并为长度为n的组成为习惯用法。

zip() 与 \* 操作符一起可以用来 unzip 一个列表：

>>> x = [1, 2, 3]

>>> y = [4, 5, 6]

>>> zipped = zip(x, y)

>>> zipped

[(1, 4), (2, 5), (3, 6)]

>>> x2, y2 = zip(\*zipped)

>>> x == list(x2) and y == list(y2)

True

**68、\_\_import\_\_(name, globals=None, locals=None, fromlist=(), level=0)**

2.5 匿名函数lambda

lambda argument1,argument2,...argumentN :expression using arguments

1、lambda是一个表达式，而不是一个语句。

因为这一点，lambda可以出现在python语法不允许def出现的地方---例如，在一个列表常量中或者函数调用的参数中，此外，作为一个表达式，lambda返回一个值一个值（一个新的函数），可以选择性地值给一个变量名。相反，def语句总是得在头部将一个新的函数赋值给一个变量名，而不是将这个函数作为结果返回。

2、lambda 的主体是一个单个的表达式，而不是一个代码块。

lambda是一个为编写简单的函数设计的，而def用来处理更大的任务。

Example:

>>>f=lambda x,y,z: x+y+z

>>>f(2,3,4)

9

>>>x=(lambda a=”fee”,b=”fie”,c=”foe”: a+b+c)

>>>x(“wee”)

‘weefiefoe’

通常用lambda来编写跳转表，如下：

>>>L = [lambda x: x\*\*2,

lambda x: x\*\*3,

lambda x: x\*\*4 ]

>>>for f in L:

print(f(2))

4

8

16

>>>print(L[0](3))

9

嵌套的lambda，如下：

>>>def action(x):

return (lambda y: x+y)

>>>act=action(99)

>>>act(2)

101

>>>action = (lambda x: (lambda y: x+y))

>>>act = action(99)

>>>act(2)

101

>>>((lambda x: (lambda y: x+y))(99))(2)

101

2.6 map函数

map(function, sequence[, sequence, ...]) -> iterator

通过定义可以看到，这个函数的第一个参数是一个函数，剩下的参数是一个或多个序列，返回值是一个迭代器。

function可以理解为是一个一对一或多对一函数，map的作用是以参数序列中的每一个元素调用function函数，返回包含每次function函数返回值的迭代器。

返回可迭代对象，需要list调用来显示所有结果。

>>> list(map(lambda x:x+2, [1, 2, 3]))

[3, 4, 5]

>>>list(map(pow,[1,2,3],[2,3,4]))

[1,8,81]

2.7 filter函数

filter函数会对指定序列执行过滤操作。

filter函数的定义：

filter(function or None, sequence) ->iterator

filter函数会对序列参数sequence中的每个元素调用function函数，最后返回的结果包含调用结果为True的元素。

返回可迭代对象，需要list调用来显示所有结果。

>>>list(filter((lambda x: x>0),range(-5,5)))

[1,2,3,4]

>>>list(filter(None,range(-5,5)))

[-5, -4, -3, -2, -1, 1, 2, 3, 4]

若function为None，则会返回包含非空元素的迭代器。

2.8 reduce函数

reduce函数，reduce函数会对参数序列中元素进行累积。

reduce函数的定义：

functools.reduce(function, iterable[, initializer]) #python3中reduce是在functools模块中

function参数是一个有两个参数的函数，reduce依次从iterable中取一个元素，和上一次调用function的结果做参数再次调用function。

第一次调用function时，如果提供initial参数，会以iterable中的第一个元素和initial作为参数调用function，否则会以iterable中的前两个元素做参数调用function。

等价于：

def reduce(function, iterable, initializer=None):

it = iter(iterable)

if initializer is None:

value = next(it)

else:

value = initializer

for element in it:

value = function(value, element)

return value

>>> functools.reduce(lambda x, y:x+y, [1,2,3,4])

10

>>> functools.reduce(lambda x, y:x+y, [1,2,3,4], 10)

20

>>> functools.reduce(lambda x, y:x\*y, [1,2,3,4])

24

如果没有initial参数，这么算：（（（1+2）+3）+4）

如果有initial参数，这么算： （（（（10+1）+2）+3）+4）

注意：function函数不能为None，function必须是有2个参数的函数。

2.9 zip函数

其中sorted()和zip()返回一个序列(列表)对象，reversed()、enumerate()返回一个迭代器(类似序列)

定义：zip([seql, ...])接受一系列可迭代对象作为参数，将对象中对应的元素打包成一个个tuple（元组），然后返回由这些tuples组成的list（列表）。若传入参数的长度不等，则返回list的长度和参数中长度最短的对象相同。

>>> list(zip([1,23,3],[213,45,2])) #两个列表长度一致

[(1, 213), (23, 45), (3, 2)]

>>> list(zip([1,23,3],[213,45,2,34,54])) #两个列表长度不一致，以短的为准

[(1, 213), (23, 45), (3, 2)]

zip一些应用：

>>> [ [ i for i in range(3\*n+1,3\*n+4) ] for n in range(3) ]

[[1, 2, 3], [4, 5, 6], [7, 8, 9]]

1. 二维矩阵变换（矩阵的行列互换）

>>>a = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

>>>[ [row[col] for row in a] for col in range(len(a[0]))]

[[1, 4, 7], [2, 5, 8], [3, 6, 9]]

>>>list(zip(\*a))

[[1, 4, 7], [2, 5, 8], [3, 6, 9]]

>>> map(list,zip(\*a))

[[1, 4, 7], [2, 5, 8], [3, 6, 9]]

1. \*操作符与zip函数配合可以实现与zip相反的功能,即将合并的序列拆成多个tuple

>>>>x=[1,2,3],y=['a','b','c']

>>>>zip(\*zip(x,y))

[(1,2,3),('a','b','c')]

1. 使用zip合并相邻的列表项

>>> a = [1, 2, 3, 4, 5, 6]

>>> list(zip(\*([iter(a)] \* 2)))

[(1, 2), (3, 4), (5, 6)]

>>> group\_adjacent = lambda a, k: zip(\*([iter(a)] \* k))

>>> list(group\_adjacent(a, 3))

[(1, 2, 3), (4, 5, 6)]

>>> list(group\_adjacent(a, 2))

[(1, 2), (3, 4), (5, 6)]

>>> list(group\_adjacent(a, 1))

[(1,), (2,), (3,), (4,), (5,), (6,)]

>>> list(zip(a[::2], a[1::2]))

[(1, 2), (3, 4), (5, 6)]

>>> list(zip(a[::3], a[1::3], a[2::3]))

[(1, 2, 3), (4, 5, 6)]

>>> group\_adjacent = lambda a, k: zip(\*(a[i::k] for i in range(k)))

>>> list(group\_adjacent(a, 3))

[(1, 2, 3), (4, 5, 6)]

>>> list(group\_adjacent(a, 2))

[(1, 2), (3, 4), (5, 6)]

>>> list(group\_adjacent(a, 1))

[(1,), (2,), (3,), (4,), (5,), (6,)]

1. 使用zip和iterators生成滑动窗口 (n -grams)

>>> from itertools import islice

>>> def n\_grams(a, n):

z = (islice(a, i, None) for i in range(n))

return zip(\*z)

>>> a = [1, 2, 3, 4, 5, 6]

>>> list(n\_grams(a, 3))

[(1, 2, 3), (2, 3, 4), (3, 4, 5), (4, 5, 6)]

>>> list(n\_grams(a, 2))

[(1, 2), (2, 3), (3, 4), (4, 5), (5, 6)]

>>>list(n\_grams(a, 4))

[(1, 2, 3, 4), (2, 3, 4, 5), (3, 4, 5, 6)]

5、使用zip反转字典

>>> m = {'a': 1, 'b': 2, 'c': 3, 'd': 4}

>>> list(m.items())

[('a', 1), ('c', 3), ('b', 2), ('d', 4)]

>>> list(zip(m.values(), m.keys()))

[(1, 'a'), (3, 'c'), (2, 'b'), (4, 'd')]

>>>dict(zip(m.values(), m.keys()))

{1: 'a', 2: 'b', 3: 'c', 4: 'd'}

2.10 变量输出问题

>>> print('n =',1232)

n = 1232

>>> print('f = ',456.789)

f = 456.789

>>> print(r"s1= 'Hello,world'")

s1= 'Hello,world'

>>> print(r''''s2='Hello,\' Adam\" ''')

's2='Hello,\' Adam\"

>>> print(r'''s2='Hello,\' Adam\" ''')

s2='Hello,\' Adam\"

>>> print(r"""s3 = r'Hello," Bart" '""")

s3 = r'Hello," Bart" '

>>> print(r"""s3 = r' Hello," Bart" '""")

s3 = r' Hello," Bart" '

>>> print(r"""s4=r" ' Hello,\nLisa!" ' """)

s4=r" ' Hello,\nLisa!"

注意：也可以用转义符’\’来闪现，上面的最外层r可以去掉，结果一致

2.11 列表

列表里的元素可以是相同类型，也可以包含各种类型，比如列表里嵌套另一个列表

list的索引是也是从0开始，但也可以从后访问，L1[-1] 表示L1中的最后一个元素；对列表可以进行切片，切片的操作类似于对函数的调用，返回值一个新的列表，切片 L1[ x : y : z ] 是半开闭区间（z通常不用写），如L1[1:3] 返回的是一个从 L1[1] 开始到 L1[2] 结束的列表，不包含L1[3]；x 不写表示从头开始，y 不写表示直到列表结束，z 用于表示步长， 默认是1， 可以认为是在这个区间里每 z 个元素取一个（取第一个），可以是负数，表示从后到前遍历。

>>> L1 = [1,2,3,4,5,6]

>>> L1[1:3]

[2, 3]

>>> L1[:3]

[1, 2, 3]

>>> L1[1:]

[2, 3, 4, 5, 6]

>>> L1[-3:-1]

[4, 5]

>>> L2 = L1[:]

>>> L2

[1, 2, 3, 4, 5, 6]

>>> L1[::2]

[1, 3, 5]

>>> L1[::-1]

[6, 5, 4, 3, 2, 1]

列表可以做加法，做乘法，字符串也可以看做一个字符的列表

>>> L1 = [1,2]

>>> L2 = [3,4]

>>> L1 + L2

[1, 2, 3, 4]

>>> 5 \* L1

[1, 2, 1, 2, 1, 2, 1, 2, 1, 2]

in语句，判断一个对象是否在一个字符串/列表/元组里

not 语句表示对后面的否定

len 可以检测字符串/列表/元祖/字典的元素个数

max 可以返回最大元素，min 返回最小元素

>>> L1

[1, 2, 3, 4, 2]

>>> 3 in L1

True

>>> 5 in L1

False

>>> 3 not in L1

False

>>> 5 not in L1

True

>>> len(L1)

5

>>> max(L1)

4

>>> min(L1)

1

>>> #赋值

>>> L1[1] = 5

>>> L1

[1, 5, 3, 4, 2]

>>> #删除

>>> del L1[1]

>>> L1

[1, 3, 4, 2]

>>> #分片赋值

>>> L1[2:] = [6,7,8]

>>> L1

[1, 3, 6, 7, 8]

>>> L1[1:3] = []

>>> L1

[1, 7, 8]

**list 的函数方法如下：**

append(x)追加到链尾

clear()清空列表元素

copy()列表的浅拷贝

extend(L)追加一个列表，等价于+=

insert(i,x)在位置i插入x，其余元素向后推，如果i大于列表长度，就在最后添加，如果i小于0，就在最开始添加

remove(x)删除第一个值为x的元素，如果不存在会抛出异常

reverse()反转序列

pop([i])删除位置 i 的元素并将它返回，默认可以不写 i ，删除最后一个元素，不存在会出错。pop 会做两件事:删除 list 的最后一个元素,然后返回删除元素的值。

index(x)返回x在列表中第一次出现的位置，不存在则抛出异常

count(x)返回x出现的次数

sort 将原列表排序，返回None，有两个可选参数，key 和 reverse，默认为升序排列

>>> L1

[8, 7, 2, 1]

>>> L1.sort()

>>> L1

[1, 2, 7, 8]

>>> L1.sort(reverse = True)

>>> L1

[8, 7, 2, 1]

>>> L1 = ['a','ccc','abcd','bc','cd','abc']

>>> L1.sort(key = len)

>>> L1

['a', 'bc', 'cd', 'ccc', 'abc', 'abcd']

len(List)返回List的长度

del list[i]删除列表list中指定的第i+1个变量

切片

切片指的是抽取序列的一部分，其形式为：list[start:end:step]。其抽取的规则是：一般默认的步长为1，但也可自定义。

**python中对list去重的多种方法**

今天遇到一个问题，在同事随意的提示下，用了 itertools.groupby 这个函数。不过这个东西最终还是没用上。问题就是对一个list中的id进行去重，去重之后要保证顺序不变。

最简单的思路就是：

ids = [1,2,3,3,4,2,3,4,5,6,1]

news\_ids = []

for id in ids:

    if id not in news\_ids:

        news\_ids.append(id)

print news\_ids

这样也可行，但是看起来不够爽。

另外一个解决方案就是用set：

ids = [1,4,3,3,4,2,3,4,5,6,1]

ids = list(set(ids))

这样的结果是没有保持原来的顺序

按照索引再次排序

最后通过这种方式解决：

ids = [1,4,3,3,4,2,3,4,5,6,1]

news\_ids = list(set(ids))

news\_ids.sort(key=ids.index)

**技巧：这里排序可以选择以另一个列表的值的先后作为排序规则，但是news\_ids中的元素在ids也要存在，不然会报错！**

使用itertools.grouby

文章一开始就提到itertools.grouby, 如果不考虑列表顺序的话可用这个：

ids = [1,4,3,3,4,2,3,4,5,6,1]

ids.sort()

it = itertools.groupby(ids)

for k, g in it:

    print k

关于itertools.groupby的原理可以看这里：http://docs.python.org/2/library/itertools.html#itertools.groupby

用reduce

In [5]: ids = [1,4,3,3,4,2,3,4,5,6,1]

In [6]: func = lambda x,y:x if y in x else x + [y]

In [7]: reduce(func, [[], ] + ids)

Out[7]: [1, 4, 3, 2, 5, 6]

上面是我在ipython中运行的代码，其中的 lambda x,y:x if y in x else x + [y] 等价于 lambda x,y: y in x and x or x+[y]

思路其实就是先把ids变为[[], 1,4,3,......] ,然后在利用reduce的特性。

本文实例总结了Python实现list反转的方法。分享给大家供大家参考。具体实现方法如下：

下面有几个不同实现的函数

import math

def resv(li):

    new = []

    if li:

        cnt = len(li)

        for i in range(cnt):

            new.append(li[cnt-i-1])

    return new

def resv2(li):

    li.reverse()

    return li

def resv3(li):

    hcnt = int(math.floor(len(li)/2))

    tmp = 0

    for i in range(hcnt):

        tmp = li[i]

        li[i] = li[-(i+1)]

        li[-(i+1)] = tmp

    return li

li = [1, 2, 3, 4, 5]

print resv(li)

ps: resv2() 方法会改变原来list的排序，其它则不会