# Preface 序言

重写了 q for mortals，虽然目录相似，但是很少的内容留下来了。

作者人生格言：You can only be young once but you can be immature your entire life.

Arthur Whitney’s brilliant invention of q and kdb+，然后一堆感谢

Arthur Whitney开发了q 编程语言，和他的数据库 kdb+，被Kx System发布，初步版本 善于表达 快速和效率

APL

处理大量的时间序列数据，存储过程语言

一些lisp在q的基因里，q的基础数据是list 基础数据，list，字典，数学映射

## 0.1 Philosophy

熟练的q开发员的思维方式和传动的编程语言。。。不同。后面叫做traditional programming，为了正确的思维，列出一些为了新手

列出一些传统的数据库编程的问题

在内存中的表现形式，例如，收集对象，必须被映射为不同的表现形式，表 为了持久化，要花相当大的努力进行对象-关系的一致正确性。

对象必须被映射为另一个关系，用来传输，通常二进制和xml格式展开为链，

数据操作，查询，分类和聚合，在一个大的数据集最好再存储过程 数据库服务器，苛求做数值计算最好再数据库外的应用服务器上做。

数据的转成GUI显示最好再隔离层显示，html5 javascript 浏览器中

太多的传统编程的格式为了关系正确，需要多行的代码排列资源并且同步不同的关系。这些对q/kdb+太容易了

解释性语言，q是解释性语言，不是编译语言，运行时，数据和函数生存在内存空间，开发迭代周期会很快，因为所有的运行时信息需要被测试和debug必须是立刻获得在工作空间中。Q编程存储和执行在简单的文本叫做scripts，可以动态的需要程序

Types q是动态语言，类型检查不严格，每个类型都匹配当前的数值，同时遇到数学符号自动转型

表达式顺序 q是从左到右，表达式是从右到左，函数在他的右边，没有操作符优先，没有括号，标点符号影响减少。

空值和无穷大值 在古典的sql中，空值null表示缺失数据对一个任何类型的并且不占用空间，在q中，null值占用同样的空间对于非null值，数值型类型同样含有最大值，null值和最大值同样可以参加到算法的运算中。

**Integrated I/O** io 函数句柄 窗户对于外面的世界，句柄一旦创建，传值就被写了

表向导的，放弃对象，和传统编程相比，你不会发现类，对象，继承，虚函数，q有table作为一级实体，缺少对象一开始不明显

有序链表，以为古典的sql是代数集合，无副本无序集合，行列的顺序没有定义，导致创建时间序列非常麻烦和慢，在q中，数据结构基于有序链表，所以时间序列依赖的顺序就被创建了，此外，简单的链表发生临近的仓库，所以加工大数据很快

列调整 sql表被推荐为行分布通过仓库

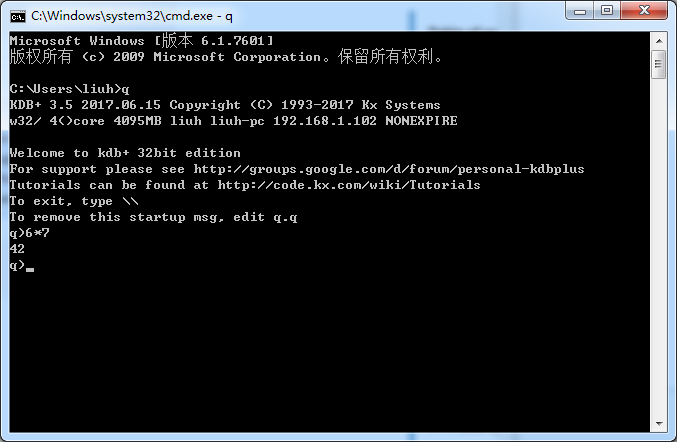
Kdb+可以被认为是内存数据库，自从能用q直接进行数据操作，没有单独分离的存储过程语言，kdb+包含序列化q列链表，写入数据文件，映射到内存。

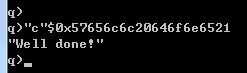
数学运算的专业进修

为了去理解q，很重要的是有一个清晰的把握对基础概念和数学函数的术语。事实上，q的基础结构都能被解释为函数映射，下面的是为不会数学准备的

Q 震惊 和 敬畏

本章母体是快速教程q的某些重点，不期望第一次全部跟上，但是希望被q的能量和经济感动





变量

在q，和大多数语言一样接受可变状态，一个变量被恰当的赋值。

Q赋值没有滥用=，不同于其他语言，在q中=的意思是测试相等

声明变量并且对其赋值使用简单一步使用操作符：

变量名必须是要给字符为本字符，可以有alpha数字和下划线

起名字推荐格式

选择一个足够长的确保起名的目的可见，但不要太长。

为函数用动词，为数据用名词

尽量不要用下划线，用也不能在最后一个字符用

空格符

一般，q许可但是不需要再扫左幅分隔符括号周围有空格。

使用 show a:42 就能将a打印出来

前下划线/表明注释的开始

至少在需要注释的时候，使用/前面加一个空格

动态类型在结合可变变量很灵活但很危险

求值顺序

从右到左的顺序

B:1+a:42

42先复制给a，然后再加上1，复制给b

我们读程序可以清楚的从左往右读

数据类型101

Q3.0，基础 integer的类型是64位的

小数值是float，也就是其他编程中的double 8byte

除法使用%，因为/被用于注释，除法结果多为float

布尔值尾部显示b

两个常用的时间是日期和时间戳

2000.01.01为0

最基本的q数据结构是list，从左到右的一个有序集合，每个元素之间的分隔符为；

列表list可以包含不同类型的元素，但是最好避开混合类型，运行速度慢

不同于函数是编程语言，list

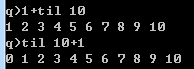
同一种类型的list，simple list，q采用了简化的存储和显示，括号和分号都被舍弃，例如下面 以空格为分隔符

简单的booleans类型的list，并列显示没用空格，在尾部加上b

Symbols的显示也没有空格

之后解释一些基础操作符来创建和操作列表，最基本的是til，接受一个非零的0，返回一个从0到n的list，不包括n。

我们获得从1开始的，就要用1加上til 10，注意是从左往后的计算表达式，尽量的使用向量运算



Join操作符使用的是， 1 2 3,4 5 将右边的join到左边上

提取操作符使用的是#，正的是取前面的，负的意思是取后面的

使用#通常结果为list

特别的事 使用0#返回空list，类型是第一个元素的乐行

这是用来创造空制定类型空list的方法

#前面截取的数大于后面list的个数，重新再截取，直到取到够数位置

特别的是，如果用#截取一个数，则会一直循环这个数，也是简单的方法创建重复元素list制定长度。

5#42

42 42 42 42 42

同样list可以被赋值为变量

L:10 20 30

所有内置的q操作符都是函数，唯一不同是以下

这些内置的是些使用k或c编写的，优化过得

所有的q函数被用作中缀，操作符-是用作前缀使用

函数在q中符合lambda表达式，或者匿名函数在其他语言中，这就表示函数是一级值，类似于long或float值，只在分配给一个变量的时候获得值

概念上讲，函数是讲输入变成输出的一些列操作。Q不是纯粹的功能性的

函数定义使用{}为界限，在开括号{之后使用[]定义形参，形参之间使用;作为分隔符，之后是操作，也是用；分隔符

A和b都是局部的变量都是应用程序持续时间内创建的。

X,y和z是可以省略的

函数定义和函数应用的精髓，但不完全，对于q，大多数是函数式编程语言，调用函数的时候可以不需要[]，直接使用空格隔开即可

因为q是向量语言，大多数内置操作符都是工作在列表之外的，用q来说，函数都是原子性的，意思是递归复杂的数据结构运行他们的操作，将一个原子添加到一个列表中，或将一个列表添加到一个原子，或两个长度相同的列表

同样可以用在判断相等和比较操作符上

函数式编程语言，累加

0 +/ 1 2 3 4 5

0 {x+y}/ 1 2 3 4 5

(+/) 1 2 3 4 5

|和& 一个返回两个数最大的书，一个返回两个数最小的数

42|98 42&98

(|/) 20 10 40 30

(&/) 20 10 40 30

有些 / 太常用有自己的名字

sum 1+til 10 / this is +/

prd 1+til 10 / this is \*/ -- note missing "o"

max 20 10 40 30 / this is |/

min 20 10 40 30 / this is &/

反斜杠\运算的，不仅仅是最终结果，迭代的每一步都会看到

反斜杠也有自己的名字

sums 1+til 10 / this is +\

prds 1+til 10 / this is \*\ / note missing 'o'

maxs 20 10 40 30 / this is |\

mins 20 10 40 30 / this is &\

斐波纳契

定义迭代

基础元素，初始化序列有列表1 1两个值

迭代步骤，当前值大小等于最近前两个值得和

注意要从右往左读，

Overload重载的方法 / 前面表示迭代次数

到底出事元素还是迭代次数？

求根 牛顿迭代法

柯里化

{[c; xn] xn-((xn\*xn)-c)%2\*xn}[2.0;]/[1.0]

{[c; xn] xn-((xn\*xn)-c)%2\*xn}[3.0;]/[1.0]

{[p; c; xn] xn-(((\*/)p#xn)-c)%p\*(\*/)(p-1)#xn}[2; 3.0;]/[1.0]

{[p; c; xn] xn-(((\*/)p#xn)-c)%p\*(\*/)(p-1)#xn}[5; 7.0;]/[1.0]

Deltas list的逐步差异值，第一个值是 初始化list的第一个值

q)deltas 1 2 3 4 5

q)deltas 10 15 20

\: 将独立步骤一起完成

2&sums[buys]

6&sums[buys]

9&sums[buys]

11&sums[buys]

sums[sells] &\: sums[buys]

deltas each deltas sums[sells] &\: sums[buys]

在列表之后，q中的第二个基础数据结构是dictionary，kv值。字典使用两个长度一样的list用操作符！构建的，左边是k，右边是value，字典也是一级值，可以被赋值

Flip用来转置

Dictionary用flip转置之后的显示像一个table

经过转置的dictionary叫table，也是一级类别在q中

在table中，选中的字典内容叫做记录，相当于sql table中的行数据

([] c1:10 20 30; c2:1.1 2.2 3.3)

方括号必须存在，用来区分是表还是list

冒号不是赋值，使用来分割列值得

列名不是symbols，虽然在下面转成了symbols

有多种方法去操作表，首先可以把表当做列字典，使用字典的操作它，q-sql

基本的q-sql操作是select模板，不是从右到左，而是模仿sql中的select，唯一一点不同是sql中的select是行基础，select表现在向量操作在列list中

如果保持行思维会很苦。

创建新表

最简单格式是select，查询所有书籍，不需要使用\* select和from必须同时出现

By用来聚合，类似于sql中的group by

先进之处在于by可以用作计算之后的列

Update用来修改列，符号：用来制定对应的列

Update是列向量的操作，不是rowbyrow

使用xasc来升序排序

`c2 xasc t

使用？生成假随机数 10？20 10个随机数，0-20的大小，包括0不包括20

10?100.0

10?`aapl`ibm

加权平均价

Xbar的左边是组宽度，右边值是数值的个数

更原始的文本类型为char，单一ascii码用双引号引用

“a”是原子但是和symbol `a不同

“string”长度为6 `string长度为1

`:path/filename

基础数据类型，原子

所有的数据最后的构建与原子

基本数据类型 boolean short int并不是保留字符

整型数据

三种integer data

在3.0之后的版本 8byte的long是基本integer数据类型

可选的正负号，没有小数点，可选的尾部标记j

J是多余的

版本的问题，j和i 4byte的int和8byte的long

2byte的short尾部是h

4byte的int尾部是i

算数类型提升是自动的在算数操纵中

同类的list操作中，小的类型被提升

浮点数

单精度和双精度，单精度能被支持，但是双精度更普遍

Float在q中是双精度，在其他语言中是double，尾部可选的f，至少15位小数

科学计数法，e代表10的次方，加号和0是可选的

Real 是单精度的4byte大小，尾部指示符使用e，至少6为小数

如果real用科学计数法，要写两个e

Q 控制台 只显示七位，尽管不显示，但是精度会保存

使用\p能够改变现实个数 16之内，\p 0则显示17位，最后一位不靠谱

二进制数据

二进制数据被表示为bit或者byte值

布尔值使用一个byte来存储一个bit，尾部标识符使用b，没有关键字true或者false

0b 1b

二进制数据含蓄的提升为无符号integer的值当参加计算和比较时

布尔值参加运算，可以消除掉条件判断

Byte值用一个byte大小保存无符号16bit的值

前置使用0x字符，后面使用2个十六进制的数字，字母大小写都可以，小写常见

Guid是q3.0引进，16byte大小的二进制值，受时间和空间的影响，基本唯一

1?0Ng

2?0Ng

-1?ONg

-2?ONg

正值使用的是相同的初始化种子，而负值使用的是随机种子，正值可再生

文本数据类型

在q中有两种原子文本类型，比起传统的编程语言，和sql中的char和varchar更相似

Char保持ascii码，8bit的编码，占用1byte大小，和sql中的char一样，单独的一个字符使用双引号包裹

当表示引号的一些字符的时候，使用转义字符。

“\”” “\\” “\n” “\r” “\t”

和C一样，可是在双一号之间不写字符，使用一个反斜杠和三个8进制数字表示

“\142”

Symbol

文本的一个原子属性，由引言领导的`，读作back tick 在q语言中

Symbol在q中用于名字，所有名字都是symbol，但不是所有symbol都是名字

Symbol更想sql中的varchar，可以包含任意数量的字符，但也是原子属性的，

“q”和`kdb都是原子实体。一个symbol是不可约的，表示其中包含独立的字符是不能直接拿到

Symbol不是string，list of char是string，symbol和string是堂兄弟，着重强调symbol不是char的聚合，`a和”a”是不同的

`a~"a"

`$"A symbol with blanks and `"

时间数据类型

Q处理时间序列和有关的data使用始终如一和有效的方式，扩展与基础sql中的日期和时间类型的有利于时间的操作，

日期date

日期存储在一个4byte的integer中，表示yyyy.mm.dd

2015.01.01

2000.01.01=0

2000.01.02=1

1999.12.31=-1

现实世界中，月和日都是从1开始的，前置0是必须的，缺失报错2015.01.01

强制转成int

`int$2000.02.01

时间类型

有两种时间类型，根据需要进行选择，如果毫秒足够，使用time类型，凌晨开始毫秒为单位，32bit的integer，表示为hh:mm:ss.uuu

12:34:56.789

12:00:00.000=12\*60\*60\*1000

前置的0是必须的，对于所有时间值，可以毫米count转成int

`int$12:00:00.000

如果好表不够用，需要舒勇timespan时间戳类型，纳秒类型的，long integer

表示为0Dhh:mm:ss.nnnnnnnnn，前置的0D是可选的

12:34:56.123456789

12:34:56.123456 / microseconds become nanos

`long$12:34:56.123456789

日期时间类型

第一种不推荐是用，以防遇到旧代码

Datatime，data和time的组合，中间是用T符号隔开

2000.01.01T12:00:00.000

2000.01.02T12:00:00.000=1.5

`float$2000.01.02T12:00:00.000

抽取日期和时间

`date$2000.01.02T12:00:00.000

`time$2000.01.02T12:00:00.000

不要用datetime做为key，因为下面的float值是模糊的，会有意想不到的结果

推荐使用的类型是timestamp，是词汇的结合日期和时间，之间的分隔符为D

使用long表示的纳秒，千禧年2000为0，之前的日期为负

2014.11.22D17:43:40.123456789

`long$2014.11.22D17:43:40.123456789

从timestamp中抽取日期和时间timespan

`date$2014.11.22D17:43:40.123456789

`timespan$2014.11.22D17:43:40.123456789

使用timestamp而不是datetime用作key值或者join，或者分割日期和时间列

月份month

月份类型是一个保存在32bit有符号integer数并且格式为yyyy.mm，尾部的标识符为m，count开始是从千禧年开始，之前为负值

2015.11m

2001.01m=12

如果尾部不加m，则是float，常见的错误

2014.11 / this is a float!

月份转成int

`int$2015.01m

Data type的月的第一天，等慕month类型

q)2015.07m=2015.07.01

分钟minute

Minute类型是32bit有符号integer并且表示为hh:mm，从凌晨开始计值

12:30

12:00=12\*60

Minute type转成int

`int$12:00

Minute type和对应的时间相等 time 和 timestamp

12:00=12:00:00.000

12:00=12:00:00.000000000

Second 秒钟类型

秒钟类型也是32bit的integer，表示为hh:mm:ss，从凌晨开始计数

23:59:59

23:59:59=-1+24\*60\*60

计数的标准不一样，和time和timespan的值都不一样

`int$12:34:56

`int$12:34:56.000

`long$12:34:56.000000000

Second和time和timespan的值是不同的，但是在同一时刻在q中用等号=做对比是相同的，同一位置上的不同单位，类似于100cm和1m

成分的点表示法

Dt.year dt.mm dt.dd ti.hh ti.mm ti.ss

这种点写法不适用运行再内部的fuctions中

因此推荐避免这种点标识符，而是使用cast转换，通常试用时间的抽取和转换，为了个人需要，也可以抽取高阶成分

`dd$dt

`mm$dt

`dd$dt

`month$dt

抽取毫秒和纳秒的时候，使用mod截取尾部

(`int$12:34:56.789) mod 1000

(`long$12:34:56.123456789) mod 1000000000

算数最大值和空值

Float正无穷大，负无穷小，null float

0w -0w 0n

long

0W -0W 0N

在q中，数值除法的结果通常为float

在数学中，正值除以0为无穷大，赋值除以0为无穷小

数学中0除以0没有定义，因此在q中，0n表示NaN，即未定义的float，

Q的理念中，所有的算术表达式都会产生一个值，而不是运行时异常，一次除以0生成一个特殊的float而不是抛出异常，

42<0W

-0W<42

最大值和最小值通常和二进制bit模式有关

0N MIN\_INT -9223372036854775808

-0W MIN\_INT+1 -9223372036854775807

0W MAX\_INT +9223372036854775807

因此得到的排序是

0N < -0W < normal integer < 0W

9223372036854775806+1

-0W-1

-0W+1

Q不捕捉溢出，所以会发生正无穷加一等于空值的一个回路

0W+1

0W+2

0W+3

Nulls值

空值的概念常常表示确实数据，这是q与其他编程语言和sql不同的地方

Q的情况，没有引用和指针，因此未分配实体的概念不存在，大多数类型都有雨正常值不同的类型，并且占用相同的数据存储空间，有些类型不指定一个明显的空值，因为没有可用的位模式，例如boolean，byte和char，底层的位模式都是有意义的

二进制没有空值null，每一位模式都是有意义的值

数字型和时间型的null值

数值型和时间型都有他们自己设计的null值，和sql情况相似，可以区分数据为null和数值为0

Q的优点在与，null值和其他值在表达式中相似，必须在类型检查的情况下使用null值

文本null值

Symbol的null值为`

Char的null值为” “，作为二进制数据，无法判断是确实char的空值还是空白符，这在练习中不是严谨的要求，注意应用程序中不要使用

“”，这个值不是null char 而是一个空char 的list

不必记住所有的null值，可以使用null来判断，不是使用=判断

Null 42

Null `

Null “ “

Null “”

列表list

所有的数据结构在q中最终都是由list构造的，dictionary是一段list，table是一个特殊的dictionary，keyed table是一对table，因此重点是彻底的掌握list

所有的list都是平等的，但是有些list更高于其他，list里面的元素都是相同的，叫做simple list，在数学中叫做vector向量，最佳的存储，效率最高

Q中的list和其他语言的操作都是相似的，但是存储和计算的过程是不一样的

他们不存储在封面下的单一链接列表中，simple list占用连续的存储空间，一遍一次的是连续空间的指针

在尾部添加元素，而不是在前面 cons ing

可以直接通过索引值得到对应的元素

一般的list可以保持不同状态的元素，不用重新改变其状态

非常有益的想法是，可以把q中的list看做动态可分配的数组

List是有序的集合，集合中的元素是原子或者其他的list，重点关注list中包含统一原子属性

List的定义和赋值

用括号表示，用分号相隔开

空格在list中是可选的

(1; 1.1; `1)

(1;2;3)

("a";"b";"c";"d")

(`Life;`the;`Universe;`and;`Everything)

(-10.0; 3.1415e; 1b; `abc; "z")

((1; 2; 3); (4; 5))

((1; 2; 3); (`1; "2"; 3); 4.4)

List从左至有，提供一个固有的顺序，list 1;2和2;1是不同的，sql是基于set的，没有固有的顺序，这样会导致语言上的错误在查询q table和查询sql table时，固有的顺序使时间序列更自然更快在q语言中，在sql中就很慢

List可以被赋值给变量

L1:(1;2;3)

L2:("z";"a";"p";"h";"o";"d")

L3:((1; 2; 3); (`1; "2"; 3); 4.4)

数list中元素个数的方法是count

count (1; 2; 3)

count L1

第一次遇到q的function或是operator，count返回一个long值等于list元素的个数

Q3中list的个数最大值是2^64-1 q2是2百万

显然一个atom的count值为1，尽管atom不是一个list

Coun 42

Count `zaphod

其他有用的操作符是first和last，分别返回第一个和最后一个list中的元素

First (1;2;3)

Last (1;2;3)

Simple list

List中的元素类型都是一致的，就叫simple list，和数学中的vector一致，这样的列表在q中是专门处理的，他们有简单的符号，有更少的存储空间和计算更快比起一般列表

Simple integer list

可以省略 括号，可以将分号分割符换成空格，list身份一致验证用~

(100;200;300)

100 200 300

100 200 300~(100;200;300)

同样的符号，就在结尾处价上尾部标识符，适用于short int

(1h;2h;3h)

(100i;200i;300i)

内部的时候每一个都要添加，显示仅仅为了简单

Simple floating point list

Simple float 和 real list 和integral 一样

(123.4567;9876.543;99.0)

123.4566 9876.543 99

Q控制台抑制了小数点后面的0的输出，但是值并不是integer，这是计数显示效率对于float的显示，如果所有的小数点后都是零，则在尾部标识符中添加f

1.0 2.0 3.0

1 2 3f~1.0 2.0 3.0

如果在float list 中添加integer，q会假设认为省去了尾部的小数点，生成的是float list 而不是general list

Simple binary list

二进制的list，boolean和byte数据并列显示，中间没有空格，同时尾部标识符加上b

(0b;1b;0b;1b;1b)

0x标识符将会在最前面显示

(0x20;0xa1;0xff)

0x20a1ff~(0x20;0xa1;0xff)

Guid list

3?0Ng

Simple symbol list

Simple symbol list并列显示，中间没有分隔符

(`Life;`the;`Universe;`and;`Everything)

如果在两个 symbol atoms元素中加上空格则会产生错误

`bad `news

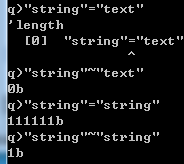
Simple char list和strings

(“s”;”t”;”r”;”i”;”n”;”g”)

“string”

一个simple list of char通常被叫做string，通常没有原子性的string在大多数语言中，她是char list，不是原子，不能判断两个长度不同的string是否相等，但是可以进行身份唯一验证，~，可以用作任何两个实体

也就是说只有atom属性的值可以进行=，相等的判断，其他的只能进行唯一性判断使用~



List of Temporal data

由于是真实的integer，所以简单的由空格分隔符隔开

特别的是混合的时间类型list表现与数值类型的list不同

12:34 01:02:03

01:02:03 12:34

为了强制对混合的类型转成强制类型，加上尾部标识符

01:02:03 12:34 11:59:59.999u

空列表和单例列表

List中有一个或者空，值得提别讨论，尤其q的表达和显示不是那么直观

The general empty list

没有元素的list是有用的，一对括号不包含任何东西，除了空格之外，表示一般的空列表，控制台没有显示

()

如果旭阳强制显示空list，使用q实用的 -3!,改操作把实体entity强转成string，名字叫wizard of Oz，因为揭露的背后的内幕

L:()

L

-3!L

单元素列表 Singleton Lists

单元素列表是列表中只有一个元素，一个元素和包含一个元素的列表是不一样的

单例的语法给q的计数带来问题，我们想仅仅写42就表示单元素列表42，但是不能，问题在于我们使用括号来声明list，也用来在表达式中进行数学计算的聚合，后面这个原因到值，他是一个计算顺序，

不幸运，没有办法创建的单例。单例列表使用函数 enlist，创建一个元素的list，为了避免被混用，所以使用了具有攻击性的词

Enlist 42

控制台的输出使用可k语言显示enlist的方式，逗号，，我们不能再q中用的

单例的char list 不能用”a”，这是单个字符atom，使用enlist

Enlist “a”

单例中的这个元素，可以是q的其他实体

Enlist 1 2 3

Enlist (10 20 30; `a`b`c)

这次没有显示逗号，，确实让人困惑

索引indexing

一个列表list序列从左到右的是他们的位置，索引，从0开始，因此长度为n的list的索引值是0到n-1

获取值使用[i]

(100; 200; 300)[0]

100 200 300[0]

L:100 200 300

L[0]

L[1]

L[3] / index out of bounds returns null value

L[2]

Indexed assignment

可以使用索引对其元素进行复制

L:1 2 3

L[1]:42

当往simple list中使用index赋值其他元素的时候，报错，窄类型不会提升类型，所以index对simple list赋值的时候，类型准确匹配

这个有点用惯了动态类型语言的人来说很奇怪

Indexing的值域

为index索引提供无效的是会产生错误

L:(-10.0;3.1415e;1b;`abc;”z”)

L[1.0]

作为对比，如果index越界了，不会产生错误，而是给你一个空值，表示确实数据，

L1:1 2 3 4

L1[4]

L2:1.1 2.2 3.3

L2[-1]

L3:(`1;2;3.3)

Simple list的index越界，返回type的null值

一般list越界返回第一元素类型的null值

空index 和null元素

[]中省略了index则显示全部

L:10 20 30 40

L[]

省略了索引和，索引是空列表不一样，索引是空列表返回空列表

使用-3!强转显示

-3!L[()]

使用::明确的表示是省略的列表，显示全部内容,::表示 nil元素

L[::]

L:(::; 1 ; 2 ; 3)

type L

-3!L[0]

这是一种方法避免不好的惊讶当q自动转换成simple type，简单的方式是重新定义那个不符合类型的元素在列表中

L:(1;2;3;`a)

L[3]:4

L

L[3]:`a

当重新复制之后，`a就不能被赋值会原来的位置，一种方法是放置::作为list的警戒

L:(::,1;2;3;`a)

L[4]:4

L[4]:`a

缺点是

List with Expressions

任何有效的q表达式都能出现在list集合中

A:42

B:43

(a;b)

L1:1 2 3

L2:40 50

(L1;L2)

(count L1;sum L2)

如果有变量，不能有简短的书写形式 a b 会报错

Combining Lists

Joining with

Join的操作符是逗号 , ，结果是一个新的list，把左边的list添加到右边list的结尾，join接受原子在其参数中

1 2 3,4 5

1,2 3 4

1 2 3,4

如果参数不是同一种类型，结果为一般list

为了是一个实体变成list，使用(),x或者x,()，而不是enlist x

Merging with ^

另一个方法去合并两个长度向东的list使用的是^，结果是右边的list对应位置的元素替换左边list的元素，除非右边的元素为null

L1:10 0N 30

L2:100 200 0N

L1^L2

List as Maps

到目前位置，我们认为list是数据，即元素的静态集合。我们也可以认为是数学的mapping映射关系，被他们的索引indexing，尤其是值域是integeer，上域是list元素，map映射关系input的值为i，output的值为L[i]

i | – > L[i]

101 102 103 104

(`a; 123.45; 1b)

(1 2; 3 4)

强两个例子中的上域为atom，最后的为list

List可以视为一个map

索引越界，返回null，可以认为映射隐式扩展到所有整数域到null的输出

Nesting嵌套

List中的元素还是list

深度

嵌套list的级数叫做depth，atom的depth是0，simplelist是1

L:(1;2;(100;200))

count L

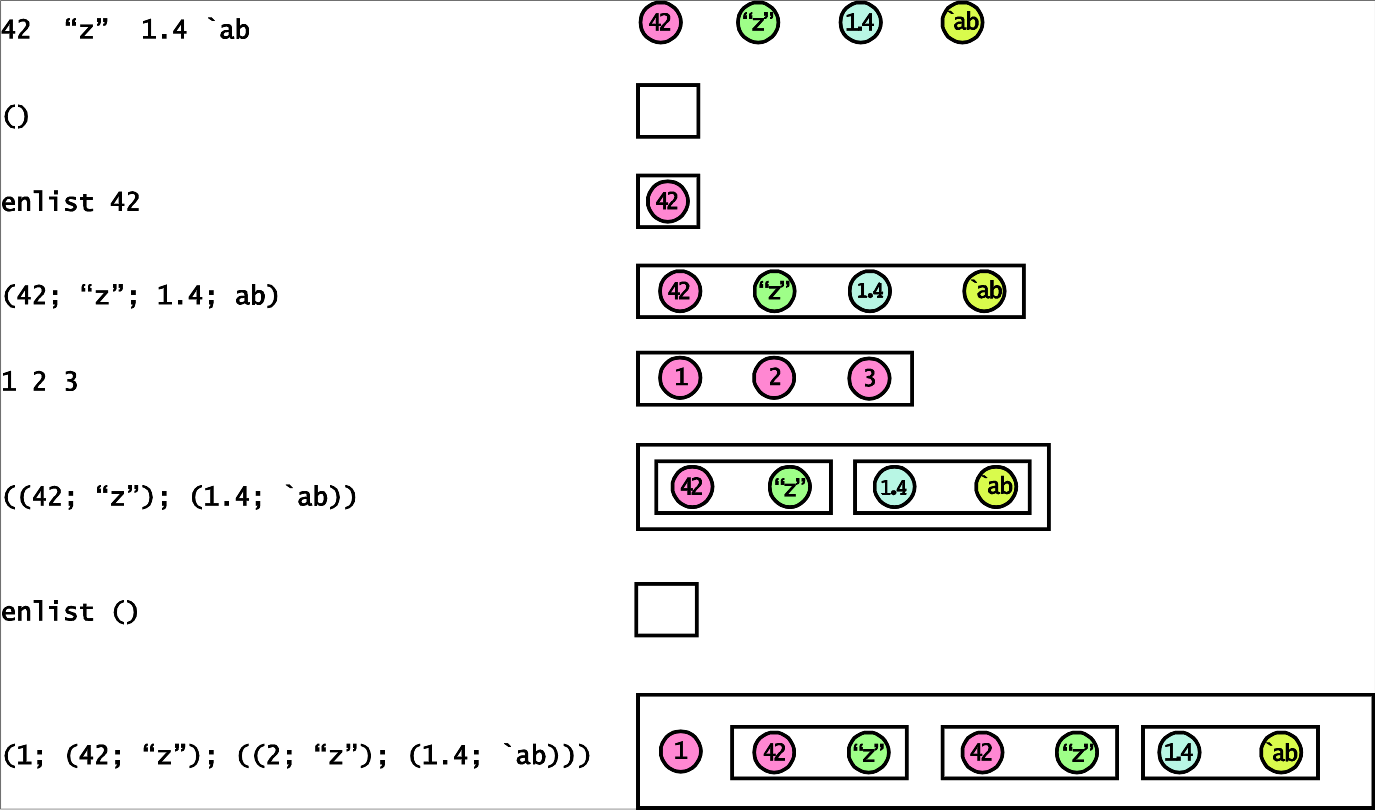
L:(1;2;100 200)

count L

L[0]

L[1]

L[2]



L2:((1;2;3);(`ab;`c))

count L2

L3:((1; 2h; 3j); ("a"; `bc`de); 1.23)

count L3

L3[1]

count L3[1]

L4:enlist 1 2 3 4

count L4

count L4[0]

m:((11; 12; 13; 14); (21; 22; 23; 24); (31; 32; 33; 34))

M

m[0]

m[1]

m[1][0]

迭代索引，迭代高维list索引

Iterated indexing and indexing at depth

L:(1; (100; 200; (1000; 2000; 3000; 4000)))

L[0]

L[1]

L[1][2]

L[1][2][0]

语法糖 indexing at depth

L[1;2;0]

不要把分号和逗号搞混了

嵌套list的赋值需要使用indexing at depth，多个[][]会报错

L:(1; (100; 200; (1000 2000 3000 4000)))

L[1; 2; 0]: 999

L

L[1][2][0]:42

错误的原因是中间检索是短暂的，他们不是可寻址的实体

Matrix

m:((11; 12; 13; 14); (21; 22; 23; 24); (31; 32; 33; 34))

M

m[0][0]

m[0; 0]

m[0; 1]

m[1; 2]

能够创建根据行或列，来展开list，重新定义形状，使用操作符#和制动0N在#操作符的左边

2 0N#til 10

0N 3#til 10

List的索引

作为向量语言，q更喜欢解决文艺使用list，

作为向量语言，q的力量

L:100 200 300 400

L[0]

L[2]

可用通过一个list来检索多个元做

L[0 2]

索引可以有其他顺序，同时可可以重复，相当于重新取回，输出的形状依赖于输入

L[3 2 0 1]

L[0 2 0]

01101011b[0 2 4]

"beeblebrox"[0 7 8]

索引值可以是一个变量

L

I:0 2

L[I]

索引值可以是一个nested list，同时生成的对应形状的list

L:100 200 300 400

L[(0 1;2 3)]

同过list的索引进行赋值

L:100 200 300 400

L[0]:100

L

L[1 2 3]:2000 3000 4000

L

赋值顺序，从左到右

L[3 2 1]:999 888 777

因此，index list 中重复的，最后的是是最右边的

L:100 200 300 400

L[0 1 0 3]:1000 2000 3000 4000

L

可以对多个index赋值同一个值

L:100 200 300 400

L[1 3]:999

L

并列Juxtaposition

函数式编程语言中，去掉方括号，并列显示list的index，分隔符使用空格

L:100 200 300 400

L[0]

L 0

L[2 1]

L 2 1

I:2 1

L I

L ::

我们 并不需要任何臭架子

那种方式根据你个人喜好，

查找Find ?

查找操作符，被重载的，二元操作符是问号 ?，他显示的是，左边list，第一次出现右边值得index

1001 1002 1003?1002

Find与map相反，通过条目反转找到索引

如果find的值不在list中，结果返回的integer等于count of the list

1001 1002 1003?1004

不在列表的值，在列表的末尾，可以在列表的末尾添加该值

Find的值可以从atom扩展到list，依次查询list中的atom

1001 1002 1003?1003 1001

Elided Indices被取消的indices

嵌套list的index at depth

m:(1 2 3 4; 100 200 300 400; 1000 2000 3000 4000)

m

检索矩阵的一行或一列，可以用缺失的index实现

m[1;]

m[;3]

m[1;]~m[1]

两种写法是一样的，推荐第一种写法，避免引发歧义

L:((1 2 3;4 5 6 7);(`a`b`c`d;`z`y`x`;`0`1`2);("now";"is";"the"))

L

L[;1;]

L[;;2]

L[0 2;;0 1]

直角list和矩阵

L:(1 2 3; (10 20; 100 200; 1000 2000))

Rectangular list 能被操作符flip调整顺序，行和列根据对角线旋转，对数据进行物理转置，生成新的存储，复制原来的数据

L:(1 2 3; 10 20 30; 100 200 300)

L

flip L

L:(1 2 3; 10 20 30; 100 200 300)

M:flip L

L[1;2]

M[2;1]

正规的矩阵定义

矩阵是矩形列表的特殊显示，0维矩阵是一个标量，一维矩阵时一个simple list，即向量，矩阵的count是长度和维度，一些函数式编程有tuple元组，固定维度的向量，q没有

Q中，向量没有必须是数值型的

v1:1 2 3 / vector of integers

v2:98.60 99.72 100.34 101.93 / float vector

v3:`so`long`and`thanks`for`all`the`fish / symbol vector

N>1，定义矩阵的维度为n，是一个n-1维度的矩阵的list，一次二维矩阵是一个vector的list，所有都有同样的size，如果矩阵中的原子值都是有同样的类型，这个类型为矩阵

二维和三维矩阵

可以通过缺失index，获得第i行 m[i;]

第j列m[;j]

第i行第j列的值是m[i;j]

按照惯例，我们认为矩阵时行的集合，将每个向量作为一列，二维数组作为列的集合，a table是列的集合，是逻辑上不是物理上转换为简单索引，约束和计算在q-sql操纵中是在table的列上，所以很快，尤其当列是simplelists的时候，尤其当simple 时间序列能被表示为两个平行list，一个是保持时间值，一个是保持关联值，检索和操作列用vector很快，在有序数量级对比同样的操作在rdbms上的无序数据

在q中高维矩阵发生的频率低于其祖先，为了完美 2\*3\*2的矩阵，每一个都是3\*2的矩阵

mm:((1 2;3 4;5 6);(10 20;30 40;50 60))

Mm

mm[0]

mm[1]

矩阵的灵活性flexibility

虽然在q中没有独立的构造矩阵，

Index可以扩展

m:(1 2; 10 20; 100 200; 1000 2000)

m 0 2

有用的list操作符

出现频率高的list操作，

Til

Til取一个非负整数n，返回一个从0开始，连续n个自然数list，非常有用的构造一个整数序列集合

til 10

1+til 10

2\*til 10

1+2\*til 10

-5+4\*til 3

如果你的代办中包含构造格式 函数 each til count，你几乎是写了loopy，不是vector，vbq

Distinct

单一去重函数 distinct 返回唯一的元素在list参数中，为了第一次出现

distinct 1 2 3 2 3 4 6 4 3 5 6

Where

基础格式 where 返回 是1b的boolean list 的序列值list，报告哪里是1

where 101010b

这对于一个操作是用来识别位置信息是很重要的

L:10 20 30 40 50

L[where L>20]:42

L[where L>20]:42

L

单参数的函数group，接受一个list，返回一个dictionary，显示各自的值，出现在对应的位置indices上

group "i miss mississippi"

Operators 运算符

Operator function verb

操作符是内部函数，被嵌入到标记中，我们可以写的和内部函数有两大区别

我们写的函数的名字必须是字母数字，而内部函数是可以用象征性符号的

我们写的函数必须要用前缀，而内部的可以用前缀和中缀表达式

函数在q中用方括号[]包含参数，使用;分割多重参数，单一的输入x写成f[x]，可以忽略单一参数的方括号[]，f x，应用于二阶的函数g，参数为x和y，写成g[x;y]

一个原子函数递归的作用于数据结构上，例如，将其作用于list和将其作用于list中的每一个元素是一样的

加法，加号写在两个数之间

2+3

在q中，也是这种方式写，但是需要从右往左读，add 3 to 2

2+3

在q中二元函数，写成中缀表达式通常叫做verb，起名的原因是认为操作数为主，函数作用其上，

原始的操作符 是内部函数，包括基础的算法，关系和比较运算符，一些被表示为单一的ascii码符号，如+ - < 其他的是组合符号如 <= >= <> 其他的还有名字not和neg

操作符和verb同样可以当做普通的函数使用，如+用作二元函数，接受连个参数，返回一个结果，sum[a;b]

+[2;3]

=[2;3]

甚至可以使用二进制动词组合中缀和函数表示，

(2+)[3]

(2+)3

扩展原子方法

原子属性函数的基本原理特征是他们可以自动扩展到list的各个元素中，当然，结合两个list，他们必须长度相等

neg 1 2 3

1 2 3+10 20 30

1 2 3+10 20 30 40

请求同样应用于嵌套list中，

neg (1 2 3; 4 5)

(1 2 3; 4 5)+(100 200 300; 400 500)

原子操作符的另一个基本属性，是隐式的将atom参数匹配到list上

100+1 2 3

1 2 3+100

原子扩展同样接受嵌套list

100+(1 2 3; 4 5)

(1 2 3; 4 5)+100

操作符优先

Operator precedence

There is none

f(g(x))，x先进行g，后对g(x)进行f，从右到左，

记住从右到左的运算，没有运算符有限这个概念

2\*3+4

(2\*3)+4

4+2\*3

(2+3)\*3+4

有些情况下，括号是不可避免的

如果一个算子的左操作符是一个表达式，一定需要括号，否则运算符将绑定到最右边的元素上

这些在左边的括号鞋业不需要再传统的编程语言中，在q中取消是错误的，不要过分为了安全加括号，多思考5秒，最终将是第二天性

没有运算符优先级的原理

运算符优先超级复杂，一些语言允许用户自己写二阶函数编程操作符，q不行，

匹配 ~

非原子属性的二元操作符，匹配，~，接受任意的两个q实体，返回，当完全一致的时候返回1b，否则返回0b，为了连个实体去匹配，他们的形状一样，类型一样，值一样，但是他们可以占用分来的存储地址，在q中，克隆被认为是完全相同的

与其他传统语言不通的是，同一性，有指针，和对象，必须是同一块内存，自己写方法判断是否深度复制

对于匹配的两个操作数的形状和类型没有任何要求

42~40+2

42~42h

42f~42.0

42~`42

`42~"42"

4 2~2 4

42~(4 2;(1 0))

(4 2)~(4;2\*1)

(())~enlist ()

(())这个是二阶的空list，单例的list是写不出来的，必须使用enlist来创建

Enlist ()这个是单例的list，里面的是一个空list，所以这两个是不相等的

(1; 2 3 4)~(1; (2; 3; 4))

(1 2;3 4)~(1;2 3 4)

42~(42)

1. 常被误以为是单例list，这是不正确的

相等和关系运算符

新的向量编程者惊讶的认为，关系运算符是原子函数，返回布尔值，关系运算符不需要类型匹配，但是需要能兼容

相等操作符=不同于match操作符~，相等操作符的两个操作数是原子属性，意识是测试两个原子而不是实体，所有的原子都是数值，时间，字符，能兼容相互使用相等操作符，symbol仅仅兼容symbol

相等判断=，判断的是两个atom在数值上是否一致，类型可能不同

42=42i

42=42.0

42=0x42

42="\*"

字符\*的ascii码转成integer值是42

对于时间类型，比较的是日历和时钟上的点，而不是潜在的计数

2000.01.01=2000.01.01D00:00:00.000000000

2015.01.01<2015.02m

12:00:00=12:00:00.000

一个symbol和一个character是不能比较的，symbol只能和symbol做相等比较，否则会报错

`a="a"

不等于的符号是<>

42<>0x42

不等于，也可是使用 not =

Not 42=98.6

当比较floats，q使用非零值得乘法公差，使floating小数值算法给出合理的结果，10^-14

r:1%3

R

2=r+r+r+r+r+r

非0 not

单元操作符 not ,和传统的编程语言不一样，返回的是布尔值，作用域是数值，时间和字符类型，不能用于symbols，not操作符一般用于翻转true和false的bit值

not 0b

not 1b

not 0b

not 1b

not 42

not 0

not 0xff

not 98.6

对于char值，not返回0b，除了底层表示0的值

not "\*"

not " "

not "\000"

对于时间类型，底层表示的数值是0的

not 2000.01.01

not 2014.01.01

not 2000.01.01T00:00:00.000000000

not 2000.01m

not 00:00:00

not 12:00:00.000000000

顺序 >,<=,>,>=

Atomic,可比较的类型，数值和字符之间可以相互比较，symbols只能和symbols相互比较

4<42

4h>0x2a

-1.4142<99i

同相等一样，时间类型的比较是日历和时钟点，而不是底层的数值

2000.01.01<2000.01.01D00:00:00.000000001

2015.01.01<2015.02m

12:00:01>12:00:00.000

对于char，比较底层ascii的数值

"A"<"Z"

"a"<"Z"

"A"<"O"

"?"<"?"

显示256的 ascii码的代码

16 16#"c"$til 256

Symbols的比较基于字典顺序

`a<`b

`abc<`aba

Atom的关系操作符，可以扩站到simple list中， 返回simple boolean lsit

2 1 3=1 2 3

10 20 30<=30 20 10

2=1 2 3

"zaphod"="Arthur"

`a`b`a`d=`ad`a`b

基础的算术符 + - \* %

算术运算符作用于原子值，二元和一元参数，可以作用于所有数值和时间类型

大部分都是相似了，除了除法%，因为/用作注释

2+3

a:6

b:7

b-a

a\*b

4%2

除法的结果总是float

重点是运算的时候，从右到左，没有优先级

6\*3+4

类型提升，对于运算符来说，遵循以下两个规则

二进制值提升为int

操作的结果类型，是能容纳连个操作数的最窄类型

1b+1b

42\*1b

5i\*0x2a

上溢出和下溢出不被困在算术运算中，在integer类型

9223372036854775806+4

2\*5223372036854775800

-9223372036854775806-4

如果表达式中出现floating-point类型，结果是float

6+7.0

1.0+1b

6.0\*7.0e

算术运算符的符号是二元的

赋值使用neg

42

-42

a:42

-a / error

neg a

算术运算符，同样提升作用在list

1.0+10 20 30

10 20 30%1 2 3

100 200 300+1b

1+(100 200;1000 2000)

最大值maximum | and 最小值Minimum &

二阶原子操作符，符合同样的type提升，和兼容性规则，和算术运算符一致，

能用在所有底层是数值型的类型，不能用在symbols和GUID

最大值 | 返回两个值得最大值，二进制是or的意思，最小值 & 返回两个值得最小值，二进制的意思是and

42|43

98.6&101.9

0b|1b

1b&0b

42|0x2b

"a"|"z"

`a|`z / error

可以在列表list使用这两个操作符

2|0 1 2 3 4

11010101b&01100101b

"zaphod"|"arthur"

为了可读性，用作二进制数据的时候，| 可以写作 or， & 可以写作 and

1b or 0b

1b and 0b

42 or 43

修改符 :

一个重载的符号 : assigns in place

C中的表达式 c+=2

Q中的amend a variable in place 的符号是 +:

x:42

x+:1

x

这里的+没有什么特别之处，可以使用其他的运算符代替

a:43

a-:1

a

a&:21

a

尽管在语义上有歧义，一个q变量可以被修改，即使他们有被预定义分配值

X

x+:42

X

Amend with lists

修改位置的能力，可以扩展到list和索引上

L:100 200 300 400

L[1]+:99

L[1 3]-:1

L1:(1 2 3; 10 20 30)

L1[;2]+:100

一个非常有用的习语是 ,: 用来添加list的元素，加到尾部

L:1 2 3

L,:4

L,:100 200

Amend会自动根据类型，提升到类型进行combined。但是 ,: 必须完全的进行类型匹配，否则会报错

指数基础，sqrt,exp,log,xexp,xlog

Sqrt

返回数值类型值得平方分，为float类型，如果没定义平方根，则返回null，也就是0n

sqrt 2

sqrt 42.4

sqrt 1b

sqrt -2

Exp，数值型的值，返回float，结果为e的n次方

exp 1

exp 4.2

exp -12i

不要被另一个e 表示10的n次方科学计数法搞混

1e10

log

数值型，返回float，自然数e为底，求log，如果没定义返回null，0n

log 1

log 42.0

log .0001

log -1

Xexp

二元，数值型，返回float，返回左值得右值次方，如果没定义，返回null

右值为分数的时候，开倒数的次方，不能为0

2 xexp 5

-2 xexp .5

Exp是一个一元版本，如果加上一个x，xexp就会变成二元版本，x表示待赋值

Xlog

二元，数值型，返回float，返回以左值为底，右值的 log值，如果未定义，返回null

2 xlog 32

2 xlog -1

更多的数值型基础

整数除法div和求模运算mod

Div

二元，数值型，左边是被除数，右边是除数，除数需要为正值，得到的是float则返回小于该值得整数，如果除数是非正值，则返回null

实践证明，除数可以为负值

7 div 2

7 div 2.5

-7 div 2

-7 div 2.5

7 div -2

3 4 5 div 2

7 div 2 3 4

3 4 5 div 2 3 4

Mod

求模运算，对左值进行求右值的模，右值为正数，

右值为非正，返回null

实践右值非正也是可以的

7 mod 2

7 mod 2.5

-7 mod 2

-7 mod 2.5

7 mod -2

3 4 5 mod 2

7 mod 2 3 4

3 4 5 mod 2 3 4

Sign

操作符signum一元的，可以对数值，和时间类型作为input，返回int值，表示正负号，当1i代表正值，当-1i代表，当0i表示零

signum 42

signum -42.0

signum 1b

signum 0

时间类型的值，处理对应的底层值

signum 1999.12.31

signum 12:00:00.000000000

倒数 reciprocal

求倒数操作符reciprocal，一元，返回1除以input，1除以正无穷为0

reciprocal 0.02380952

reciprocal 0.0

reciprocal -0.0

Floor 和 ceiling

Floor值，一元的，接受integer和float类型的数，返回小于等于该值得最大整数，类型为long

floor 4.2

floor 4

floor -4.2

可以用来截短小数值得右边其他值

x:4.242

0.01\*floor 100\*x

类似floor，操作符ceiling返回大于该值得最小整数

ceiling 4.2

ceiling 4

ceiling -4.2

Floor和ceiling的不接受short值，会报错 floor 4h 报错

绝对值 absolute value abs

绝对值操作符abs 一元的，接受integer和floating，作用域，输出类型和input的类型一直，但是如果是二进制类型，则返回int

在时间类型上的操作符

时间类型也是integral值，基于基础时间点，性能和操作是相似的，

将时间值转成integer count

`int$1999.12.31

`int$2013.01m

`int$12:00:00.123

`long$12:00:00.123456789

在q中没有时区概念

时间类型比较

比较时间点，时间点相同，但是底层的integer值不同

2000.01.02=2000.01.02D00:00:00.000000000

`int$2000.01.02

`long$2000.01.02D02:00:00.000000000

不同的类型比较，在同一单位，强制转成最细的单位

`timestamp$2001.01.02

为了比较不同类型的值，q转成最细的值，然后对底层的integer值进行比价

Temporal Arithmetic

时间类型的算法

包含时间和数值类型的值得函数式通常是有意义的，时间值和数值值比较的是他们之间的偏移量

2000.01.01=0

12:00:00=12\*60\*60

1999.12.31<0

在时间值中添加一个整体值，因为他只是被添加到底层的偏移量

2014.12.31+1

2015.01.01+til 31 / all days in January

12:00:00+1

在一个时间值，加上另一个时间值，

12:00:00+01:00:00

一个重要的加法是一个timespan加上一个date等于timestamp，时间这个工作，底层时间一些计算

2015.01.01+12:00:00.000000000

两个时间值得差异，在于其计数值之间的差

2001.01.01-2000.01.01

2015.06m-2015.01m

一个类型的两个值之间的差值是基本偏移量的差值，表示为同类型，即。作为一个跨度。

2015.01.01D00:00:00.000000000-2014.01.01D00:00:00.000000000

12:00:00-11:00:00

12:00-11:00

日期之间的差值是计数

时间之间的差值是，时间的跨度 as a span

对无穷值和null值得操作符

无穷值和null值都是有值表示的

下面的表达式在其他语言中可能报错

(0%0)=0%0

使用not对无穷值和null值，返回的是0b

not 0W

not -0w

not 0N

使用neg，无穷值改变符号，null值还是null值

neg 0W

neg -0w

neg 0N

Null<负无穷<正常值<正无穷

不同类型的Null值得关系比较否都是0b

不同类型的正无穷的比较

Short<int <long<real<float

不同类型的负无穷比较

-float<-real<-long<-int<-short

42<0W

-0w<42.0

-0w<1901.01.01

-0w<0w

0W<0w

-0w<0W

-10000000<0N

0N<42i

0n<-0w

Symbol的null值`小于其他的symbol值

`a<`

同样的|和&满足比较法则

42|0W

-42&0N

0w|0n

-0w&0n

0n|0N

Float的空值，和long的空值不能比较，但是float的null值，大于long的null值

0Wi&0W

Int的正无穷大被提升为long值，于对应的long的正无穷大，进行二进制位数的and操作

别名Alias ::

因为q是严格的，表达式通常会在解释权遇到之后，立刻评估的，特别的，赋值与右边表达式要求在分配结果之前对表达式进行评估

别名是一个表达式的变量，不是表达式的结果，而是表达式本身，因此别名提供了一个方法，延迟表达式的评估

别名的评估是lazy的，懒加载，意思是在需要的时候才发生，更准确的是，变量被引用的时候，强制评估，这是表达式是否需要重新评估

第一次引用的时候，或者在最近一次引用后，改变了变量的值，评估表达式所有变量的当前值，结果被保存下来，并且return，保存结果的过程叫做memoized

如果引用的变量没有改变过值，memoized的值直接返回

别名变量依赖于其他变量的任何置

通过双引号::

创建别名

::用于函数定义的外部，左边值为别名变量，右边是是表达式，当别名被引用的时候，表达式进行评估

a:42

b::a

c:a

a:43

b

c

w::(x\*x)+y\*y

x:3

y:4

w

y:5

w

一个很使用而且什么的符号 0N! 能够强制在控制台输出数据

一种侵入式在控制台输出

w::(0N!x\*x)+y\*y

x:3

y:4

w::(0N!x\*x)+y\*y

x:3

y:4

w

w

y:6

w

别名和函数

函数也是延迟计算

fu:{(x\*x)+y\*y}

fu[3;4]

类似函数和别名的不同，有两点，

在调用函数的时候，同时提供参数，进行评估。在别名中，变量可以随时赋值，但是只有在调用别名的时候才进行评估，

函数没有memoze他的结果，每次都进行计算，甚至参数不变

依赖

别名变量依赖表达式连接的实体，w依赖于x和y，可以使用.z.b来显示系统中的依赖关系list，可以用\b

w::(x\*x)+y\*y

.z.b

每个key 与他依赖的实体关联

可以创建别名在依赖于其他别名的表达式，有一个依赖联，lazy reference，都是懒加载

这样的递归引用，导致了层次关系，不仅依赖于自己的表达式，而且还依赖于其他的表达式的变量，可以创建复杂的依赖图

也可以用它创建不可维护的系统

有可能造成回环loop

别名通常用于通过指定查询作为表达式提供数据库视图

别名常用语，提供数据库视图，通过制定的查询表达式

表和视图之间的依赖关系

t:([]c1:`a`b`c`a;c2:20 15 10 20;c3:99.5 99.45 99.42 99.4)

v::select sym:c1,px:c3 from t where c1=`a

v

update c3:42.0 from `t where c1=`a

字典Dictionaries

字典是由key list和value list显示定义的映射连接关系，连个list必须含有相同的个数，并且key list 必须是唯一的集合，一般的list也可以创建字典，多说的dictionaries涉及simple list 作为key

可以理解为key-value对，但是底层存储的是一对list

又叫association，domain list of keys co-domain list of values

创建使用 ! 读作 bang 将两个连接起来

Keys!values

所有的dictionary都有类型 99h

10 20 30!1.1 2.2 3.3

`a`b`c!100 200 300

Key和value都不需要是simple list ，大师key需要满足奇怪的事情在操作中

(`Arthur`Dent; `Zaphod`Beeblebrox; `Ford`Prefect)! 100 42 150

1001 1002 1003!(`Arthur`Dent; `Zaphod`Beeblebrox; `Ford`Prefect)

可以使用key和value，分解dictionary的key和value，count计数

d:`a`b`c!100 200 300

key d

value d

count d

虽然q中不强制key的唯一性，历史遗留问题，dictionary为每一个输入input提供唯一的输出output，在查找中只实现第一个key关系

当你知道key的值是唯一的，可以使用 `u# 将dictionary变成hash table，在查找的时候会加速，比起default linear lookup

(`u#`a`b`c)!10 20 30

Key和value list的顺序使有意义的，像列表的顺序一样重要，顺序不同的dictionary的唯一性是不同的

(`a`b`c!10 20 30)~`a`c`b!10 30 20

空和单例 dictionary

创建空dictionary通过空key和空value list

()!()

typed empty dictionary using typed empty key and value lists.

就是讲空key和value强转

(`symbol$())!`float$()

Singleton dictionary

连个单例list创建单例字典

(enlist `x)!enlist 42

`x!42这个方法不对，是链接列的枚举值

Lookup

为了查找对应输入的输出值，可以使用look up key值，lookup使用同样的标记list 的indexing，方括号[]或者并列

d:`a`b`c!10 20 30

d[`a]

d `b

如果查询不到，则返回null值，类型为dictionary初始值的类型

和list一样，查找的index可以扩展到一个simple list

d[`a`c]

ks:`a`c

d ks

反向查找使用 ?

查找操作符? 返回list中第一次出现该值的index

10 20 30 10 40?10

本质上是转换超找的位置，转换k-2-v的查找。查找value第一次出现右值时的key值

d:`a`b`c`a!10 20 30 10

如果没有找到，则返回空值，空值的类型为key list的初始值类型

Dictionary vs list

可以通过显示位置的检索，创建出一个很像list的dictionary

L:10 20 30

d:0 1 2!10 20 30

L 0

d 0

L 1 2

d 1 2

d~L

这两个是不同的，主要的不同是dictionary可以被扩展通过赋值，而list不行

例子显示了dictionary紧凑而有效的稀疏list

d1:0 100 500000!10 20 30

d2:0 99 1000000!100 200 300

d1+d2

不唯一的key和value

不唯一的key是可以容忍的，但是lookup只显示第一个

ddup:`a`b`a`c!10 20 30 20

ddup[`a]

Reverse反向查找lookup也只限制第一个出现的key

ddup?30

ddup?20

可以通过where来查找给定value值，所有key的映射

Where 20=ddup

不是simple list作为key和values

Key和value的list也不需要第统一类型的原子，可以为嵌套list

d:(`a`b; `c`d`e; enlist `f)!10 20 30

d `f

d?20

d:`a`b`c!(10 20; 30 40 50; enlist 60)

d `b

d?30 40 50

d?enlist 60

预先警告，不要使用不规则的的key value list，项目不能一致 吃的lookup和reverse lookup混乱

dwhackey:(1 2; 3 4 5; 6; 7 8)!10 20 30 40 / atom 6 is whack

dwhackey 1 2

dwhackey 6

dwhackval:10 20 30 40!(1 2; 3 4 5; 6; 7 8) / atom 6 is whack

dwhackval?3 4 5

dwhackval?6

观察到，任何一个方向上的查找，在不同形状 单例list 第一项失败

用在字典上的操作符

Amend and upsert

和list相同，可以通过key来修改值

d:`a`b`c!10 20 30

d[`b]:42

和list不用，dictionary可以延伸赋值，对一个不存在的key进行赋值

d:`a`b`c!10 20 30

d[`x]:42

测验更贴近的能力，通过key修改延伸 赋值，

D[k]:v

当k在d的key之中就是update，当k不在d的key中，就是insert

d:`a`b`c!10 20 30

d[`b]:42 / update

d[`x]:100 / insert

这个update和insert的行为叫做upsert，因为tables和keyed tables是dictionary，upsert涉及到kdb+

提取子字典

Extracting a Sub-Dictionary

字典查找，一个key或者一个key的list，返回对应的values值，能够提取key和value值，通过重载的操作符 #，左边的参数是 a list of keys，右边的参数是dictionary数据源，结果是一个子字典通过制定的key，结果受限制于制定的key

d:`a`b`c!10 20 30

d `a`c

a`c#d

(enlist `c)#d

注意的是，#左边的操作参数是一个list，如果抽取一个值，是单例list

如果有重复的key，则抽取第一个

ddup:`a`b`a`c!10 20 30 20

`a`c#ddup

操作符同样应用于 non-simple keys

dns:(`arthur`dent; `ford`prefect; `zaphod`beebelbrox)!100 150 42

(`arthur`dent; `zaphod`beebelbrox)#dns

删除实体

Removing entries

#的助手操作符是\_，能够删除k-v对，句法是相似的，左边是一个list of keys，右边是一个dictionary，返回一个字典，是删除特定key知道的字典

在使用 \_ 的时候，左边必须要有空格

连起来写的话，就不是操作符了

d:`a`b`c!10 20 30

`a`c \_ d

(enlist `b) \_ d

注意，一个key的所有出现都被删除了，意图删除一个不存在的key是没有效果的

d:`a`b`c`a!10 20 30 40

`a`c \_ d

`x`a \_ d

删除所有dictionary中的实体，返回一个empty dictionary，类型为之前的类型，控制台不会显示，除非使用 -3! 函数强制显示

耳机吃操作符 cut 和 \_ 是一样的

`a`c cut d

另一个重载的 \_ 的方法是，左边参数是dictionary，右边是一个key，单独删除这个key，不常使用，只删除第一个出现的

d \_ `b

在dictionary上的基础操作符

因为字典是一个map，能够被其他map组成，获得一个dictionary，，kv复杂连接，新map，将函数运用于字典，有效的运用于值list，一元函数非常简单

就是一元参数的函数和操作符，运用在字典上，会自动递归运用在values list上

d:`a`b`c!10 20 30

neg d

2\*d

d=20

f:{x\*x}

当连个字典的作用域是一直的，表现的二元操作符也是直接的，做值和值的操作通过key，例如两个dictionary相加，通过key对应，对应的value值相加

d1:`a`b`c!1 2 3

d2:`a`b`c!10 20 30

d1+d2

如果发生作用域不一样，首先是，结果的字典的与是参数域的结合，即key list 的结合，对其中的交点，key值相同的，进行值和值之间的计算

问题是，不同的key，什么都不操作，没有任何事情可做，简单的携带值在结果中，本质，使用标识符元素进行操作，以代替丢失的值

Join ,

Join的操作符是 , ，合并两个dictionary，当心 互斥的key，有些值会被省略掉

d1+d2

d2:`x`y!40 50

相同的key的时候，因为q是从右到左的运算，所以操作符右边的值被保留下来，左边的dictionary的值被省略掉了

d1:`a`b`c!10 20 30

d2:`a`b`c!100 200 300

Lo and behold，操作符右边的值把左边的替换了

相当于每个k-v值，被upserted 到左边的dictionary

d1:`a`b`c!10 20 30

d1[`c]:300

d1[`d]:400

因此 join 不满足交换了，顺序很重要

d1:`a`b`c!10 20 30

d2:`c`d!300 400

d1,d2

d2,d1

合并coalesce ^

合并操作符 , 使用upsert 将两个dictionaries合并，不同的是，右值为空，则使用左值

d1:`a`b`c!10 0N 30

d2:`b`c`d!200 0N 400

d1^d2

算术和关系运算符

在 两个dictionary对应的key上进行计算

d1:`a`b`c!10 20 30

d2:`b`c`d!200 300 400

d1+d2

判断相等和比较操作符，在相同key上进行操作，在不想交的key中，另一个dictionary没有的key对应的是补充null值，如果此时的值是null，连个null值是相等的，会返回1b

(`a`b`c!10 20 30)=`b`c`d!20 300 400

(`a`b`c!0N 20 30)=`b`c`d!20 300 0N

(`a`b`c!10 20 30)<`b`c`d!20 300 400

Column Dictionaries

列字典，是基础为了table

Definition and terminology

一个非常有用的dictionary 是映射简单symbols list 对一个矩形list

`c1`c2!(`a`b`c; 10 20 30)

每个symbol是一个名字，values list相当于list作为vector的列 ，可以叫这个list为column dictionary，一个普通的 列字典的基本形式

c1...cn!(v1;...;vn)

Ci和vi有相同的长度，

通常vi是simple list

travelers:`name`iq!(`Dent`Beeblebrox`Prefect;42 98 126)

travelers

可以得到name的列

travelers[`name]

因为column是list，同样可以使用index

travelers[`name][1]

travelers[`iq][2]

一般，遇见重复的index，我们可以把index合并写成深度

travelers[`name; 1]

travelers[`iq; 2]

因此可以认为是二维实体

General case

一个普通列字典的定义

travelers[`iq; 2]

列cj的第i个

dc[cj][i]

dc[cj;i]

深度索引的时候，省略第一个值，表示所有的column列都取，当在第二个参数的时候制定一个值时，是一个数据的切片

travelers[; 2]

列字典可以看做一个二维实体，第一个维度通过key symbol list 列名检索，第二个维度可以通过位置检索

制定一个名称检索相应的列，然后可以按位置索引

只制定一个列索引可以得到一个切片字典，该字典将映射到该索引的相关值

这个数据结构有潜力但是有一个问题，索引是倒退的，也就是做在第一个槽检索。通常在二维数据结构中，列有两个槽

Column dictionary with a single column

Key的list 和value的list 要有同样的长度

为了得到 single column ，需要使用 enlist 获得 single key 和 single column list

dc1:(enlist `c)!enlist 10 20 30

不是有效字典，更不用说single column `c!1 2 3

转置 flipping a column dictionary

不是你可以在dictionary上做的，在嵌套矩阵中使用 flip转置

L:(10 20 30; 100 200 300)

flip L

对flip的到的结果，在index 相反的槽操作得到同样的值，就是行列号调换

L:(10 20 30; 100 200 300)

M:flip L

L[0;0]

L[0;1]

L[0;2]

M[0;0]

M[1;0]

M[2;0]

当列字典被认为是一个二维实体时，它是矩形的，因为它的列列表都有相同的计数。控制台显示显示这个

`c1`c2!(`a`b`c; 10 20 30)

因此对其进行转置是有意义的

dc:`c1`c2!(`a`b`c; 10 20 30)

t:flip dc

t

我们期望在转置过后的实体上，能够进行深度索引

dc[`c1; 0]

dc[`c1; 1]

dc[`c1; 2]

t[0; `c1]

t[1; `c1]

t[2; `c1]

列现在在第二个位置上进行索引

dc[`c1;]

t[;`c1]

dc[; 0]

t[0;]

dc[`c2; 1]

t[1; `c2]

忘记t的来源，对t进行隔离检索

t[;`c1]

t[0;]

t[1; `c2]

第一个槽中的索引部分，可以忽略分号之后的的elided的indices

t[0]

t[1]

t[2]

总结A column dictionary 的转置

这个一个二维数据结构，它使用第一个槽中的整数索引，第二个槽中的列名符号symbol

值在第一个槽中制定一个整数，返回该索引中的部分字典

在第二个槽中制定一个列名，检索该列

指定一个整数和一个列名，来剪碎该列中该索引的字段

可以看做是部分字典的列表

经过两次转置，会得到原来的值

转置一个转置过的列字典，得到原来的列字典

不同于转置矩形list，转置a column dictionary 不会物理的重排数据

已转置的列字典已列顺序存储

相反，结果是一个合理的调整，意味着深度索引的槽是颠倒的

Functions

内置的function

函数说明

Q中的函数，符合数学中的算法映射关系，一个函数，包含一些类，待计算的表达式，可选的输入参数和返回值，调用函数的过程，评估表达式，代入参数，结果应该有一个，是函数的输出值

因为q可以定义全局变量，q不是一个纯粹的函数式编程语言，一个数学函数无法达到自身身体吱哇，并产生副作用，减少代码可维护

Fucntion definition

函数的定义是一个{}其中包含表达式，对比静态语言，参数类型和返回值类型可以不用确定，深圳他们不用申请，函数名也是可选的

函数最后一个表达式为返回值，可选的空格，在参数和表达式之间为了可读性

{[x] x\*x}

用括号括起来的方法来调用函数，里面的数值对应的函数内部的参数，里面的表达式倍计算，返回结果output值

{[x] x\*x}[3]

函数是一级值，是想long和float一样的数据，尤其函数被复制给一个变量，变量名可以代替使用到函数的地方

f:{[x] x\*x}

F[3]

Function notation and terminology

{[p1;...;pn] e1; ...; em}

P1,..,pn为参数，e为表达式，涉及到参数，表达一步步算法，表达式们的顺序是从左到右，内部单个表达式是从右到左

为了可读行，空格，逗号，分号，方括号，参数

分号的意思是分割，不是终止，不要在最后一个表达式后添加分号，除非有意控制输出为空

事实上，分号;计算左边值，并且放弃左边值，然后计算右边值，返回该值仔细思考，这就是分号该做的

a:2\*3;6\*7

a

意思是分号左边计算2\*3赋值给a，然后打印输出6\*7

输入参数，隐式或者显示叫做函数的阶

大多数是一元函数，或者二元函数

没有参数的函数

F:{[] ...}

调用函数使用 f[]，说明函数返回常数值或者调用外部资源

{[] 42} / pure function returns constant 42

{[] a\*a} / impure function: references global a

函数的最大参数个数是8，多了出错，可以构造一个list或者dictionary

函数的输出output是最后一个表达式，容易调试线性，debug

使用 : 将直接返回冒号后面的表达式，

{[] a\*a} / impure function: references global a

{[x] a:1; :x\*x; b:3} / dead code

Q的函数重载不仅仅是一元和二元个数，而是anything else，类型 符号 排序

调用函数 function application

调用函数与定义正式函数，函数之后紧跟着方括号，分号分割值，调用函数是的表达式倍计算

{[x] x\*x}[3]

f:{[x] x\*x}

f[4]

{[x;y] x+y}[3;4]

g:{[x;y] x+y}

g[3;4]

函数的调用是严格的，意思是参数中的表达式，在给形参之前进行计算

f:{[x] x\*x}

f[0N!3+1]

Valence也叫做rank秩，调用函数传递参数过多，返回 `rank 错误

{[x] x\*x}[3;4]

调用一个无参数函数，传递的函数都会被忽略

const42:{[] 42}

const42[]

const42[98.6]

无返回函数，在函数体的最后一个表达式后面加上分号;

fvoid:{[x] `a set x;}

fvoid 42

事实上，通常返回到额是nil ::

-3!fvoid 42

调用一元函数有两种方式，可以省略掉方括号，而是用空格隔开函数名和参数

{[x] 2\*x} 42

f:{[x] x\*x}

f 5

这种规则叫做并列，juxtaposition ，函数式编程常见

当函数分配到一个全局变量时，可以用名字来表示，用变量的symbol名字来代替变量本身

f:{x\*x}

f[5]

`f[5]

`f 5

.my.name.space.f:{2\*x}

`.my.name.space.f[5]

这个可行是因为，全局变量被实现成，symbolic-name和function对在字典中，实际上引用变量时找其名字，因此，名称的应用，不适合局部变量和q系统变量

不要混淆应用函数的名字，和按名字传递参数，

隐式参数

像数学一样，经常是用到参数x，y和z，可以省略他们和方括号，三个隐式参数x，y和z能自动定义和得到在函数的表达式中，当他们没有显示的声明是，

因此两个规范造成了等效的输入输出

{[x] x\*x}

{x\*x}

{[x;y] x+y}

{x+y}

当使用隐式参数，x总是被第一个实参代替，y第二个，z第三个

只要有y必须输入2个参数，只有有z，必须输入三个参数

g:{x+z} / likely meant x+y; requires 3 args in call

g[1;2] / still waiting for 3rd arg – i.e., a projection

g[1;2;3] / 2nd arg is required but ignored

对于前三个参数，只是用[x;y;z]这一种格式，顺序和值也是

evil:{[y;z;x] ...} / don't do this!

匿名函数和lambda表达式

很多传统的编程语言包含函数名定义，函数没有名字叫匿名函数，函数式编程所有的函数都是匿名函数，并且叫lambda表达式，所有的q函数是匿名函数，知道被复制到一个变量，顺便，lambda这个词是在早期的工作中是一个排版错误，其实是lambda calculus

一个你们函数很有用，可以当做一个宏来使用

f{[...] ...; {...}[...]; ...}

可以认为把他抽出来容易理解

匿名函数的另一个用处死动态匹配的基本概念，在集合中放置函数，list，dictionary，table，然后动态选择，而不是用一个内部名字

powers:({1}; {x}; {x\*x}; {x\*x\*x})

selected:2

powers[selected]

The identity function ::

当::被用作函数调用的使用，他是 identity function 返回输入作为输出

无实体的identity function能使用并列的写法，必须用括号括起来

::[42]

::[`a`b`c]

:: 42 / error

(::) 42

函数是数据

Q数据实体，我们遇到的有atoms各种类型的，list，dictionary，函数式编程新手，惊讶function也是data，函数可以传递，像long和float一样

可以在list中

(1; 98.6; {x\*x})

f:{x\*x}

(f; neg)

(f; neg)[0]

(f; neg)[1; 5]

函数可以被用作输入和输出被其他函数使用

apply:{x y}

sq:{x\*x}

apply[sq; 5]

Apply接受一个函数x和一个值y，两者并列显示，y作为input调用x

函数操作调用其他函数叫做高阶函数，高阶函数在函数式编程里是很强的概念，

导数和不定积分是初等微积分的高阶函数

按名字调用 call-by-name

普通的函数调用在q中叫做call-by-value，也就是在函数调用的开始，参数被还原为值，在这个过程中，复制是为了原始值不被打扰，当输入数据的很大，这个复制是被禁止的，在这种情况下，还有另一种调用方法，叫做call-by

-name,通过传递全局变量的名称而不是他们的值

这里没用到，函数实现只是期望一个变量名的符号并且适当的处理他，使用call-by-name的函数的例子是内置函数，返回的值是全局变量的名字传递

a:42

get `a

另一个是set，二元函数，在全局赋值中起到实际作用，期望一个全局变量的名称和被分配的值

`a set 43

a

任何内置函数的call-by-name的函数调用结果是symbol 被传递input，这使得call-by-name函数能够被链接起来，组成

返回值和错误的信息

不要讲回调值和错误信息混淆，前者以一个后沟开始，后者以一个tick开始

局部变量和全局变量

定义局部变量和全局变量

变量被复制，使用:，在函数体内，叫做局部变量，

a

局部变量的最大个数，24个

一些注意点

局部变量的存在仅仅反正在调用时，不存在静态局部变量的概念，它在调用之间维持其值

局部变量在其定义范围之外是不可见的

一个局部变量不能用作一个参数，在q中函数使用的call-by-name的方式

局部变量在同一范围内定义的局部函数的主题呢不可见

因此，q没有词汇范围

证明变量没有 范围属性，f定义局部变量a，和局部函数helper，在函数式编程语言中，a在其定义范围内可以访问的，在f中，a可以在其局部函数helper中得到

f:{[p1] a:42; helper:{[p2] a\*p2}; helper p1}

在q中是不行的

f 5

相反的，你必须多声明一个参数在局部函数中，然后白局部变量当做参数传递过去，投影提供了一种相对简单的方法

f:{[p1] a:42; helper:{[a; p2] a\*p2}[a;]; helper p1}

f 5

在函数体外赋值得变量时全局变量，全局变量在哪都是可见的

b:7

f:{b\*x}

f[6]

全局变量最大值

在一个函数用一用的全局变量最大值是32，封装，创建list和dictionary

在函数中对全局变量进行复制

尽管函数是编程纯粹主义者畏缩，函数式编程减少副作用的原因，你可以在函数体内对一个全局变量复制，有些q编程者错误的使用双冒号::，不是创建局部变量，而是用全局变量

b:6

f:{b::7; x\*b}

f[6]

b

双冒号::的用法类似于其他格式的amend-in-place,例如+:或者,:，而不是复制运算符本身:，这说明他有一些不直观的特性，并不是全局作业操作符

如果有一个本地变量与想复制的全局变量同名，使用双冒号,则对本地变量复制，而不是全局变量

b:6

f:{b:42; b::x; b}

f[98]

b

全局变量赋值

我们推荐不是用::进行全局变量赋值，反而推荐是用set对真正的全局变量复制，不会复制到局部变量

Set 使用 call-by-name

a:42

f:{a:98.6; `a set x}

f 43

映射 Projection

映射的意思是在函数的调用中仅仅指定一些参数，返回需要其他参数的函数，这个应该叫做柯里化

函数映射

在某些情况，多价的函数中，一部分函数的参数是固定的，而其余的参数是不同的，一种情况是日期是最后一个参数，在许多日期中调用该函数，其他位置的参数是不变的

这种情况下，可以指定和固定的参数，将其他参数留给以后，这种情况较做projecting函数剩余参数，在函数调用的时候，一部分被提供，一部分被省略

例如，二元函数，提供第一个值，被映射到第二个参数的函数

add:{x+y}

add[42;]

可以看到，q将映射剩余参数的部分调用函数，映射后的函数是合法的一元函数

函数是一级公民，可以对projectiong的函数赋值变量，

g:add[42;]

[3]

Projection有关的是柯里化，在函数式编程里

add3:{x+y+z}

add3[2][3][4]

推荐的写法，

在柯里化的时候不要省略尾部的分号;，会对阅读代码产生困难

F是二元的，和f是三元的都能运行，无法确定

mystery:f[2]

编程柯里化的时候的方括号是不可省略的，但是在柯里化后的一元函数的方括号可以省略

{x+y}[42;][3]

{x+y}[42;] 3

Projection之后的函数是不变的，尽管原来的函数变了，也不变

f:{x-y}

g:f[42;]

g

g[6]

f:{x+y}

g

g[6]

操作符映射

当使用中缀的格式时，q的操作符可以被映射到左边的参数，这需要括号

(7\*) 6

由于q中重载括号，操作符不能和右边的参数映射，-42仅仅是原子-42

，而不是映射

但是操作符也是函数，可以是用前缀的格式

-[;42] 98

空格可以不用，

(7\*)6

-[;42]98

Multiple Projections

大于2价的函数有多种映射方式，3个参数，可以映射第一个和第三个，结果是一个需要输入第二个参数的一元函数

{x+y+z}[1;;3]

{x+y+z}[1;;3] 2

可以通过两步

{x+y+z}[1;;]

{x+y+z}[1;;][2;3]

{x+y+z}[1;;][;3]

{x+y+z}[1;;][;3] 2

{x+y+z}[;;3][1;]

{x+y+z}[;;3][1;] 2

Everything is a map

探索更深的关系，在q的数据结构和函数之间，

毫无疑问注意到列表的索引符号，字典查找和调用单值函数是相同的

L:0 1 4 9 16 25 36

f:{x\*x}

L[2]

L 2

f[2]

f 2

这不是巧合，list是一个map，定义具体位置通过indexing，dictionary是一个map定义为k-v查找，函数是一个map，定义输入到输出的一些列算术表达式，所有都是map，理解起来需要点时间

List的索引，key和参数

我们可以用索引遍历list，可以通过key list遍历lookup其中的value值，可以通过fanction，使参数到输出的映射

L:10 20 30 40 50

L[2 5]

I:2 5

L I

d:`a`b`c!10 20 30

ks:`a`c

d ks

f[2 5]

f I

都是一个map

深度索引值

深度索引和多价函数求值，嵌套list是list

a:(1 2 3; 100 200)

a

a[1;0]

更深层次的嵌套是一个位置的映射

同样，复杂值的dictionary可以视为一个多元的map

d:`a`b!(1 2 3; 100 200)

d[`a;1]

第一个参数定位到key，第二个参数定位到value list的位置

映射和索引省略

注意到，函数的projection和list的index省略的符号是相同的，

m:(10 20 30; 100 200 300)

m

m[1;2]

L:m[;2]

L

L[1]

数组越界

其他语言数组越界会产生错误，但是因为是map的缘故，所以index的值域问题，如果越界是缺失值，而不是错误

可以理解为没有对应的映射关系

10 20 30 40[100]

`a`b`c[-1]

(1.1; 1; `1)[3]

字典也是类似的

d:`a`b`c!10 20 30

d[`x]

Atomic Functions

描述q的特性为向量语言，大部分内置操作符都是原子性的，原子性函数的一些特性，调用原子性函数的特点，递归到函数的参数的数据结构中，直到他的原子，然后应用于此，没有显示的循环和流程控制构造

一元属性函数和map

原子行为最容易看到的一个单值函数

一元函数的结果的shape和参数的shape总是一样的，输出依照输入

一个特殊的情况是，把原子属性函数运用到list，就和运用到list中的每一个元素一样，被称为 atomic functions extend automatically to list，类似的在传统语言中的foreach和函数式语言的高阶函数map中实现了

二元的原子属性函数和zip

Find ? 查询在左边参数中，查询右边的吃出现的index索引号，如果没有则返回左边list的count

10 20 30?10

10 20 30?10 20 30 40 50

(enlist 10)?10

10 20?10

10 20 30 40 50?10

算术，比较和关系运算符都是二元的原子属性操作，有4中应用场景

* atom with atom
* atom with list
* list with atom
* list with list

在最后一种情况，两个list必须有相同的长度

1+10

1+10 20 30

1 2 3+10

1 2 3+10 20 30

创建原子属性函数

有保证的方法是在你的函数中组装内置原子属性函数

一元

f:{(x\*x)+(2\*x)-1}

f 0

f til 10

二元

pyth:{sqrt (x\*x)+y\*y}

pyth[1; 1]

pyth[1; 1 2 3]

pyth[1 2 3; 1 2 3]

完成这个行为，不适用原子属性，时间问题

Adverbs

Adverbs是高阶的函数，定义函数行为，修改list，命名来源于，q中operator操作符命名为verb，精通adverb是一个人的技巧

并不是所有的都是原子属性，仍希望运用到所有的数据结构中，而不需要使用loopy代码

一元，

函数会自动运行到list中的每个值

neg (1 2 3; 4 5)

奖励表集合到atom中的函数不能是原子的，聚合操作符，toplevel of a nested list

count 10 20 30

count (10 20 30; 40 50)

想要count 两个list的各自的个数，使用each这个adverb，也是一元函数，接受每个list，而不是整个list，要用空格

count each (10 20 30; 40 50)

Each的特性是高阶函数，可以使用下面格式，each接收一个count函数，变成柯里化，再接收嵌套list

each[count] (10 20 30; 40 50)

类似 foreach 和map

可以使用each至少一层，多层的嵌套循环使用多层

count (1 2 3; 10 20 30)

count each (1 2 3; 10 20 30)

(count each) each (1 2 3; 10 20 30)

最后一个是1 1 1；1 1 1

可以将each用于任何的一元函数，尽管有些是多余的

reverse "live"

reverse ("life"; "the"; "universe"; "and"; "everything")

reverse each ("life"; "the"; "universe"; "and"; "everything")

neg each (1 2 3; 4 5)

将一个n长度的vector转成n\*1的矩阵，可以使用enlist each，但是使用flip enlist更快

Each-both

原子属性操作符，自动了将两个同长度的list，分队操作

1 2 3+10 20 30

二元join操作符,，对应参数是完整的

"abc","de"

如果我们想机智的合并两个类似

("abc"; "uv"),'("de"; "xyz")

Adverb each-both ‘ 定义了二元操作符，接受承兑的list，之间没有空格，并且不会产生错误，在其他语言中，each-both ‘ 又叫 zip，zip很好理解，成对去运算

("abc"; "uv"),'("de"; "xyz")

当操作数为原子的时候，没有任何作用

3,'4

Each-both 有通常的二元原子性函数的特性，例如左右两边shape相同，否则会报长度错误

("abc"; "uv"),'("de"; "xyz"; "uhoh")

同样可以自动扩展匹配list

1,'10 20 30

1 2 3,'10

2#'("abcde"; "fgh"; "ijklm")

最后一个用each也行

也可以用作前缀

,'[("abc"; "uv"); ("de"; "xyz")]

两个简单列表工作正常，但是一般列表就不是那样了，会损失掉单例list的属性

一个可信赖的方法是使用flip

flip (L1; L2)

,’很好的例子用早table中，

t1:([] c1:1 2 3)

t2:([] c2:`a`b`c)

t1,'t2

Each-left \:

原子操作符将一个原子扩展到右边的操作数，以在匹配到左边的列表。

1 2 3+10

Each-left adverb \: 二元属性函数连个参数，对左边的每个参数进行操作，

("abc"; "de"; enlist "f") ,\: ">"

Each-right /:

原子操作符，左边的参数 变成list，匹配右边参数

10+1 2 3

Each-right adverb /: 例如将左边的string添加到右边的string list上

“</” ,/: (“abc”;”de”;enlist “f”)

"</",/:("abc"; "de"; enlist "f"),\:">"

Cross product

笛卡尔积，两个list，左边的每个，分别对应右边的每个，是用 each-right和each-left组成，如果不希望格外的嵌套，可以用raze消除

1 2 3,/:\:10 20

raze 1 2 3,/:\:10 20

内置的操作符 cross，同样的作用

1 2 3 cross 10 20

可以通过调整顺序达到效果

raze 1 2 3,\:/:10 20

累计操作符 斜杠 /

Over adverb / 是一个高阶函数，q中提供递归操作，积累的作用

如果需要1加到10，需要控制流，初始化值，确定循环，函数式编程，简单的声明累加的开始值，开始累加，不用计数，不用测试，不用循环，what to do而不是how to do it

为了可读性，加了空格

0 +/ 1 2 3 4 5 6 7 8 9 10

复合操作符 + 和 /

空格操作符不能再 函数和/之间，因为/还有注释的作用

初始化的值，在操作符的左边，开始依次加操作符的右边的值，在覆盖下，每一项都逐步加到累加器上，在列表中一种，直至达到终点，最后的累加值返回，

addi:{0N!(x;y); x+y}

0 addi/ 1 2 3 4 5

显然，作为高阶矩阵函数/ 接收一个二元函数，返回一个二元函数，两个函数是匀称的，左操作是是累加器初始值，右操作数是累加列表

在函数式编程语言中，over叫做reduce，

通常情况，初始值是不必须的，在总和的例子，可以将第一项作为累加值，进行累加，需要用括号包围函数，同时省略初始累加值

(+/) 1 2 3 4 5 6 7 8 9 10

另一种方式思考

(1; 2; 3; 4; 5; 6; 7; 8; 9; 10)

+/的作用是有效的用+替换分号

该替换的格式

括号是必须的

修改过后的函数是一元的

这个结构通常k 而不是q，不要告诉人你写k

一元和二元的格式都可以

+/[0; 1 2 3 4 5 6 7 8 9 10]

+/[1 2 3 4 5 6 7 8 9 10]

这个基础格式很轻，可以用在其他常用的聚合操作符上

(\*/) 1 2 3 4 5 6 7 8 9 10 / product

(|/) 7 8 4 3 10 2 1 9 5 6 / maximum

(&/) 7 8 4 3 10 2 1 9 5 6 / minimum

累加，累乘，最大，最小 prd 累乘

空列表，最大最小

max[L1,L2] = max[L1]|max[L2]

(|/) `long$()

(|/) `float$()

|这种操作，边缘情况，保留了类型的一致性

(&/) `long$()

(+/) `long$()

(\*/) `long$()

(,/) ()

通过,/连接list，有效的消除了顶层嵌套，如果只有一层嵌套，flatten list a plain list ，raze起到同样的效果

(,/)((1 2 3; 4 5); (100 200; 300 400 500))

raze ((1 2 3; 4 5); (100 200; 300 400 500))

可以使用自己的function

f:{2\*x+y}

100 f/ 1 2 3 4 5 6 7 8 9 10

(f/) 1 2 3 4 5 6 7 8 9 10

More Over Iteration

下一种递归方式，声明loopy代码，在改变本的 / 中，操作符左侧是自然数表示运行的次数，而右侧是迭代的初始值，中缀和前缀的表示方式

计算 fibonacci值

fib:{x,sum -2#x}

10 fib/ 1 1

fib/[10; 1 1]

另一个版本的/运算递归直到收敛，或者直到循环结束，牛顿迭代，求根公式，用一个值逼近求导，近似值是，斜率乘以左边点和右边点，这个函数的提供的值是极小的，以为导数定义为切线的斜率，斜率的极限，为了求2的个，发现0值{-2+x\*x}

f:{-2+x\*x}

secant:{[f;x;e] (f[x+e]-f x-e)%2\*e}

{x-f[x]%secant[f; x; 1e-6]}/[1.5]

Q怎么结局卷积，每一步的结果斗鱼前面的步骤相比较，如果两者在等式差值内

10的-14次方，算法就被认为是收敛的，递归结束，否则继续

在数学中，可以产生无限循环的牛顿迭代公式，

 x3 – 2x + 2  初始化为0，causes the algorithm to go into a cycle.

newtcycle:{[xn] xn-((xn\*xn\*xn)+(-2\*xn)+2)%-2+3\*xn\*xn}

newtcycle/[0.0]

Q如何结束迭代，在每个步骤中，结果与初始值进行比较，如果匹配，就会出现一个循环，并停止递归，否则继续

如果值相同，类型不同，没有检测到循环，循环继续，提供非初始值导致无限循环，如果输入一下内容，将不得不终止控制台回话

newtcycle/[0] / oops

最终重载的/等价于命令式编程中的while，提供了一种声明方式来指定迭代测试

fib:{x,sum -2#x}

fib/[{1000>last x}; 1 1]

反斜杠 \

和/是相似的，中间产物都生成，而不是只有最后一个

0+\1 2 3 4 5 6 7 8 9 10

(\*\)1 2 3 4 5 6 7 8 9 10

(|\)7 8 4 3 10 2 1 9 5 6

(&\)7 8 4 3 10 2 1 9 5 6

100 f\1 2 3 4 5 6 7 8 9 10

(f\)1 2 3 4 5 6 7 8 9 10

可以观看中间产物

newtsqrt[1.0]

\P

newtsqrt[1.0]

Each-previous ‘:

‘: 对列表的每个项目执行一个与他的前任相关的操作

在遍历列表中，当前项是左操作数，前一个项是右操作数

因为列表的初始项没有一个前身，我们必须在操作符左边提供一个前身，一个选择是列表的初始值，计算价格列表的增量

100 -': 100 99 101 102 101

作为其他的 adverbs，有一元的格式 each-previous，返回参数的初始项

(-':)100 99 101 102 101

deltas 100 99 101 102 101

(%':)100 99 101 102 101

ratios 100 99 101 102 101

原因是，最初的前身被隐式转化为二元函数

sums deltas 100 99 101 102 101

deltas sums 100 99 101 102 101

当你寻找绝对值差异的时候

deltas 100 99 101 102 101

和容易创建自己的版本实现这种情况

deltas 100 99 101 102 101

一个有用的习语 用到each-previous 使用 ~ 判断项是否一致，通常用作他们是否匹配，differ同样的效果

(~':) 1 1 1 2 2 3 4 5 5 5 6 6

not (~':) 1 1 1 2 2 3 4 5 5 5 6 6

differ 1 1 1 2 2 3 4 5 5 5 6 6

Differ的第一个值是1b，where定位范围，使用cut分割

L:1 1 1 2 2 3 4 5 5 5 6 6

where differ L

(where differ L) cut L

在q中做些事情

runs:(where differ L) cut L / store runs

ct:count each runs / store count of each run

runs where ct=max ct / find the runs of maximum length

找到截取出来的list，个数最多的组合

runs where ct=max ct:count each runs:(where differ L) cut L

用比较和合理对的初始值

L:9 8 7 11 10 12 13

(where -0W>':L) cut L

(where 0W<':L) cut L

General appliction