AHK技巧及如何提升性能

请注意，以下所有涉及性能的讨论，都是在你处理海量文件或者进行海量运算时才需要用到的。
什么叫海量呢？我觉得至少几百万起算吧。
正常应用情况下，也就是99%的情况下，你应该考虑的都不是性能，而是代码易读易懂。
因为在正常情况下，是感知不到这些性能区别的。
而代码写得难懂的话，是会给别人或者在未来拾起自己代码时造成巨大的灾难！

假设有A,B两个文件.想知道A中的某行是否在B中存在,怎样最快?

代码：（全选）

方法1.逐行读取A,逐行读取B,并对比,直到一致或者B文件被读完.
不用说,膝盖也能想出这样慢到死.

代码：（全选）

方法2.逐行读取A,用FileRead一次性将B读到一个变量中,然后用IfInString来判断.
实践证明,速度很慢,并且比对不严谨,可能出现问题.

代码：（全选）

方法3.先逐行读取B,并将B每行的内容作为KEY存入一个对象中,VALUE设为1.直到完全将B转化为一个对象.
再逐行读取A,并以A中每行的内容为KEY,检查VALUE的值,显然,VALUE为1,这说明A的内容在B中存在.
实践证明,速度很快,是我能想到找到的最快速的办法.大概比方法2快几十倍吧.

```
obj_b:=[]
f:=FileOpen(文件b, "r")
while (!f.AtEOF)
    obj_b[f.ReadLine()]:=1

f_a:=FileOpen(文件a, "r")
while (!f_a.AtEOF)
```

顺便一说,AHK的对象的KEY实在是太伟大了,这货什么东西都能存进去.
变量名,还有规则长度限制什么的,比如变量名不能取为

代码：（全选）

```
`r`n123456+888`t`v
```

但是,对象的KEY可以,

代码：（全选）

```
ABC["`r`n123456+888`t`v"]
```

这就是合法的.
我用这个特性,解决了N多问题.

又想起一个,创建变量或者给变量赋值的时候
这样

代码：（全选）

```
a:=b:=c:=""
```

比这样

代码：（全选）

```
a:="",b:="",c:=""
```

快25%。而上面这样，又比下面这样

代码：（全选）

```
a:=""
b:=""
c:=""
```

快25%

当前,不创建变量更快.意思是,比如一个函数有返回值.
这样

代码：（全选）

```
msgbox,% abc()
```

比这样

代码：（全选）

```
a:=abc()
msgbox,%a%
```

快

关于函数返回值数量的问题
AHK函数返回值只能有1个,怎么绕过这个限制呢?
方法1.全局变量
方法2.byref参数
方法3.返回值是对象

比如

代码:（全选）

```
abc(a,b)
{
haha:=[]
haha["第一个返回值"]:=a
haha["第二个返回值"]:=b
return,haha
}

返回值:=abc(6,8)
msgbox,% 返回值["第一个返回值"]
msgbox,% 返回值["第二个返回值"]
```

在操作SQLITE数据库时
这样

代码:（全选）

```
DB.Exec("BEGIN TRANSACTION;")
    DB.Exec("我是要执行的SQL命令1;我是要执行的SQL命令2;")
    DB.Exec("COMMIT TRANSACTION;")
```

比这样

代码:（全选）

```
DB.Exec("BEGIN TRANSACTION;")
    DB.Exec("我是要执行的SQL命令1;")
    DB.Exec("我是要执行的SQL命令2;")
    DB.Exec("COMMIT TRANSACTION;")
```

快20%-30%,而上面这样,又比这样

代码:（全选）

```
DB.Exec("我是要执行的SQL命令1;")
DB.Exec("我是要执行的SQL命令2;")
```

快几十上百倍

还有一个重点，如果有很多时间值需要转换的话，比如要把int型的时间（相对1970-01-01 0:0:0到现在的秒数），转换成字符串型。
强烈建议让sqlite来转换，不要自己在ahk里面写函数转换，速度慢太多了。

没法直接发sql语句，会被安全策略屏蔽，所以给出sqlite中转换时间的相关函数。
最常用的就是下面这3个，都是把1970的int型时间转换为字符串型

代码:（全选）

```
date(1318238460,'unixepoch')
time(1318238460,'unixepoch')
datetime(1318238460,'unixepoch')
```

如果UPDATE比INSERT慢很多,可以尝试建索引。
不过创建索引后会导致insert变慢。当然，这些都是数据库的问题了，不在这里讨论。

ListView中
这样

代码:（全选）

```
GuiControl, -Redraw, MyListView     ;关闭控件重绘。待数据添加完成后再重绘，可大幅提高性能
Loop,100
    LV_Add("","")
GuiControl, +Redraw, MyListView     ;重启控件重绘。重启后将自动重绘一次控件
```

比这样

代码:（全选）

```
Loop,100
    LV_Add("","")
```

快

同时,根据我的实际测试,帮助中说的指定ListView的Count选项会提高性能,实测并不是这样的……
顺便一说,GUI要想性能高点,通用方式就是减少重绘次数.

关于下载
不管是要抓网页文字到变量,还是下载文件到硬盘,都用我的 URLDownloadTo 系列吧.
绝对绝对比自带的命令 URLDownloadToFile 稳定加强大.

关于字符串操作
尽量不用正则表达式,因为它慢而且越长越难以理解

尽量不用正则表达式,因为它慢而且越长越难以理解.

比如,要判断一串数字是不是IP地址.

123.152.100.1

正则表达式

代码:(全选)

```
(\d{1,2}|1\d\d|2[0-4]\d|25[0-5])\.(\d{1,2}|1\d\d|2[0-4]\d|25[0-5])\.(\d{1,2}|1\d\d|2[0-4]\d|25[0-5])\.(\d{1,2}|1\d\d|2[0-4]\d|25[0-5])\b
```

妈蛋,看着这串东西谁不想杀人?
想想自己作为一个人是怎么判断出这个IP合法不合法的,很自然的想到就是判断每段字符是否小于等于255.

代码:(全选)

```
IP:=StrSplit("123.152.100.1",".") ;以点为分隔符分割字符
Loop,4
    If (IP[A_Index]>255)        ;判断每段字符是否小于等于255
        MsgBox,WRONG
```

这就是最高效(性能高效,思考高效)的解决方式了.
帮助上面说,要在一个较大的字符串中搜索简单的子字符串, 请使用 InStr(), 因为它比 RegExMatch() 执行地更快.

假设一项功能,需要符合条件A或符合条件B才启动.
同时,假设条件A的符合概率是50%,条件B的符合概率是90%

这样

代码:(全选)

```
if  (条件B or 条件A)
  运行功能
```

比这样

代码:(全选)

```
if  (条件A or 条件B)
  运行功能
```

快

这样

代码:(全选)

```
abc:="haha.gaga"
MsgBox, % StrSplit(abc, ".")[2]
```

比这样

代码:(全选)

```
abc:="haha.gaga"
临时对象:=StrSplit(abc, ".")
MsgBox, % 临时对象[2]
```

快

真正的意义不是快,而是以前以为返回值是对象的情况下,必须用一个中间量接收后再调用,才发现可以直接省略那个中间量.

关于连接字符串。
这样

代码:(全选)

```
b:="我是字符串"
VarSetCapacity(a, 1048576, 0)
loop,100000
a.=b
```

比这样

代码:(全选)

```
b:="我是字符串"
loop,100000
a.=b
```

快非常多!
原因是后者变量a每次增加内容的时候，都需要程序自动的调整一次大小，而前者在最开始指定好了变量大小，因此调整过程中只用加入内容，不用重新调整大小。
另外，不用担心前者预先设置了一个大小，那是不是变量超过这个大小了程序就报错呢？不是的！！！超过这个预先指定的大小后，程序又会在需要的时候自行调整大小。

当你有一个文件有几十万行，你想读取到对象中时。
这样

代码:(全选)

```
obj:=[]
obj.SetCapacity(4000000)
while (!f.AtEOF)
    obj.Push(f.ReadLine())
```

比这样

代码:(全选)

```
obj:=[]
```

```
while (!f.AtEOF)
    obj.Push(f.ReadLine())
```
快非常多！
原理大概同预先指定变量大小类似吧，区别是这条性能优化方法从帮助到论坛都没看人提出来过。这是实测得来的经验（其实本文所有内容都是实测得来的经验！）

当你有一个变量想判断其和多个值是不是相等的时候。

这样

代码：（全选）
```
if var in a,b,c,d,e,f,g,h,i,j,k
```
比这样

代码：（全选）
```
if (var="a" or var="b" or var="c" or var="d" or var="e" or var="f" or var="g" or var="h" or var="i" or var="j" or var="k")
```
快

可是不幸的是，第一种语法在ahk v2版本中似乎不再支持了。

以下为速度优化设置3套餐，加在任何脚本最开始的地方，可以几倍的提升速度。

代码：（全选）
```
#NoEnv
SetBatchLines, -1
ListLines, Off
```
原理就自行翻帮助了，里面都有写。大概就是把ahk自身的一些延时设置为不延时。
以上3行设置在ahk v2版本中也是默认设置了！

这样

代码：（全选）
```
a>b ? 1 : 0
```
比这样

代码：（全选）
```
if (a>b)
    return, 1
else
    return, 0
```
快

原理不清楚，反正三元表达式就是比if else形式快，虽然他们是等价的。

64位版本比32位版本快一些。
这个也没什么需要说的了。

这样

代码：（全选）
```
a:=10
b:=20
MsgBox, % a+b
```
比这样

代码：（全选）
```
a:=10
b:=20
MsgBox, % fun(a,b)

fun(a,b)
{
    return, a+b
}
```
快，而上面这样，又比这样

代码：（全选）
```
a:=10
b:=20
函数名:="fun"
MsgBox, % %函数名%(a,b)

fun(a,b)
{
}
```
- 将数据存入Key，相当于为数据建立了**索引**，故查找快。
- **变量**作为语句解析，所以有语法（命名）限制；**Key**作为变量的值，无类型限制（数值、字符串、对象）（可以模拟select-case，见7#）。
- 貌似解释器按行解析语句，故，多行集成为**单行**更快。留意：语句"A,B,C"，V1、V2均从左向右计算，但作为**表达式返回值**，貌似V1返回C，V2返回A（总之顺序不同）。
- 创建**中间变量**加入了更多的步骤，在**可读性**与**性能**下平衡逻辑吧。
- [illegible]做参数（无需ByRef，甚至省去‘中间变量’Return）。

简单说[illegible]动态函数快。

补充：

- 自汐潮的测试，使用"?:"比if-else快很多。（用""(A,B)""可包含多语句，可替代if-else，可读性换性能）（可能与以上提及的写作一行有关）
- 灵活应用**可变参数**（区分，**可选参数**—带默认值；可变参数—参数项可有可无）。

来源： http://ahk8.com/thread-5060.html