aardio 范例: 内存回收

```
//内存回收
/*
aardio 会自动回收内存，偶尔看到内存短时间小幅上升是正常的，
一段时间后就会自动释放（这样效率更高一些）。

有些不是由 aardio 分配的资源，可在析构函数中释放。
例如 inet.http 构造函数中调用 table.gc(this,"close") 绑定了 close 函数作为析构函数。

aardio 中所有需要析构的对象基本都提供了必须的析构函数。
那么有些对象，例如 web.rest 的源码里为什么没有看到析构函数呢？
web.rest 调用的是 inet.http ，既然 inet.http 有析构函数，web.rest 当然就不需要了。

那么，为什么大量创建 inet.http，web.rest 发现内存在上升呢？

1、对象额外分配的资源并不由 aardio 分配或管理。
对于某些较重的对象，可以提前释放，例如 inet.http 可以主动调用 close 函数，
COM 对象可选调用 com.Release() 释放（非必须，可以不这么做）。
aardio 仍然会在一段时间后自动释放这些对象（并不是立即、随时释放，有延迟）。

2、在循环内部大量创建 inet.http （还不关闭）是不必要的。
inet.http 本就共享会话，在循环外部创建一个 inet.http 对象代码更少，占用内存更少。

这些属于极端的特例，大多时候可以忽略。
不要因为遇到一次下雨，就每天出门都带伞。
*/

//正确写法一：
//---------------------------------------------------------
import inet.http;
var http = inet.http();
for(i=1;1000;1){
    //不必要在循环内部重复创建，
    //inet.http，web.rest 都是共享会话

    //http.get("http://www.example.com");
}
//http.close(); //一般没必要写

//正确写法二：
//---------------------------------------------------------
for(i=1;1000;1){
    //如果必须在循环内部重复创建大量消耗内存的对象
    var http = inet.http();

    //建议及时调用析构函数
    //不该省的代码就不要省，不然为什么要提供这个函数呢？
    http.close();
}

//绑定析构函数示例：
//---------------------------------------------------------
class http {
    //构造函数
    ctor( url ){

        //分配资源
        //仅演示分配，新手不建议用 raw.realloc 这种无保护的函数。
        this.ptr = ..raw.realloc(100);

        //绑定析构函数：参数 @2 可指定成员函数名字，也可以指定函数对象
        ..table.gc(this,"close");
    };
    //析构函数
    close = function(){

        //这里释放资源
        if(this.ptr){
            //仅演示分配，新手不建议用 raw.realloc 这种无保护的函数
            this.ptr = ..raw.realloc(0,this.ptr);
        }

        /*
        raw.realloc 是直接分配原生内存。
        新手很难理解或者用好这种函数（也没有必要用）。
        如果乱改很可能出现一些似是而非的误解。
```

```
        析构是一个特殊的过程，析构代码要干净，稳定，避免报错。
        当然，对于 aardio 这样自动管理内存的编程语言，
        需要自己写析构函数的时候很少。
        */
    };
}
```

Markdown 格式