aardio 范例: 调试基础

```
//调试基础

/*
aardio 语言本身就是调试工具，不用断点就可以方便地输出对象与调试信息。
aardio 代码抛出的异常也自带调试信息。发布后的 EXE 脱离开发环境仍然可以方便地调试。

aardio 的 debug 库也提供了方便的、可编程的断点调试功能，
但是断点调试功能在 aardio 中用途不大，一般用户不必了解也不必学习。
如果你喜欢可视化断点调试，调用 debug 写一个并不难，不过这种调试其实笨拙低效，在 aardio 里没必要这样做。

请不要照搬 C++ , C # 的调试习惯，
如果写 C++ , 写 C # 可以像 aardio  这样方便地调试，
那是做梦都可以笑醒的事。
*/

import debug;
import debug.log;
import console;

//仅在开发环境中则自动打开控制台并输出调用栈，发布后仅向 stderr 输出信息（不含调用栈，不自动打开控制台）。
console.error("输出错误信息，仅在开发环境中输出调用栈")

console.logPause("右键点开控制台标题栏菜单 > 编辑 > 全选，然后按回车键可以复制");

//输出不定个数对象最简单的方法
console.log(123,"字符串",{});

//输出对象，对于表、数组、COM 对象会输出更详细的信息。
console.dump( {1;2;3;a={b="测试"}} );

//输出 table 对象的内容，基于 util.table.stringify
console.dumpTable({1,2,3})

//这个函数也可以输出变量,格式上有所区别
console.varDump({1;2;3;a={b="测试"}})

//下面的代码是转换为 JSON 格式输出到控制台,嵌套表看起来比较清楚
console.dumpJson({a={b="测试"}})

//暂停，按任意键清屏并继续。
console.more(1,true) ;
//console.pause(); //也可以用这个函数暂停

//下面演示写日志，对于复杂的程序，写日志是比较重要的。
//主线程启动时设置一次日志路径就可以,在多线程中自动生效
debug.log.setPath("/.config/log.txt");
debug.log.checkSize(0x20000); //设置日志大小

/*
输出错误调试信息到日志文件,
发布后会下面的代码会进行BASE64位编码后输出(并且不再显示于控制台) -> 可以使用 debug.log.dump() 解码为明文。
我们强烈建议大家在程序中使用日志记录异常信息。
*/
debug.log.print("错误信息");

/*
下面的代码用于隐藏错误信息,
*/
global.onError = function( err,over ){
    if(!over){
        import debug;
        var stack = debug.traceback(,"调用栈",3);//获取调用栈信息,此函数当前调用级别为1,调用onError的默认错误处理程序调用级别为2,真正发生错误的代码调用级别为3
        debug.log.write(stack,'\n',err)
    }

    if( _STUDIO_INVOKED ) return err; //在开发环境中仍然返回错误信息,不返回字符串类型的错误信息则不会显示
}


/*
下面的代码插入一个断点并打开控制台,
在控制台中可以使用aardio代码,调用debug支持库的函数进一步获取更多调试信息。

注意默认提供了两个快捷键
1、按't'键后回车执行代码 io.print( debug.traceback() )
2、按'c'键继续执行断点后的代码
*/
debug.debug();

/*
抛出异常,
第 2 个参数指定抛出异常的调用级别，1 为自身，2 为上层调用者，依次上推。

如果库函数报错，也可以临时增加代码检查异常参数,
并使用 error 函数向指定的调用级别抛出异常，以辅助定位错误的调用代码。
*/
console.pause();
```

Markdown 格式