aardio 范例: 拆分复杂字符串

```
//拆分复杂字符串
import console;

var str = /***

abc def 123
------------------------

dec xyz 456 中文
------------------------

***/

//不指定任何分隔符，按字符拆分，多字节的中文字符拆分为一个数组元素
var arr = string.split(str);

//指定多个单字节分隔符，连续的分隔符中间拆分为空字符串
var arr = string.split(str," -");

//将用尖括号指定连续字符组成的分隔串单位，string.split 不支持其他模式语法。
var arr = string.split(str,"<------------------------>");

//string.splitEx 提供更强的拆分功能，分隔符支持模式匹配语法
var arr = string.splitEx(str,"\-\-\-+");

//如果模式串以 ^ 开始，则分隔符的第一个捕获组添加到下个拆分结果的前面
var arr = string.splitEx(str,"^(\-\-\-+)");

//如果模式串以 $ 结束，则分隔符的第一个捕获组添加到上个拆分结果的后面
var arr = string.splitEx(str,"(\-\-\-+)$");

/*
string.lines 与 string.splitEx 拆分规则相同。
但 string.lines 返回的是迭代器，启动更快并按需拆分，不需等待全部拆分完成才能获取结果。
*/
for line in string.lines(str,"\-\-\-+"){
    //console.log(line);
}

//除了可以自定义行分隔符，还可以相同规则自定义列分隔符
for items in string.lines(str,"\-\-\-+","\s+"){
    //自定义列分隔符以后，每次循环返回的都是包含多个列的数组
    console.dumpTable(items);
}

/*
一些编程语言在以正则表达式拆分字符串时，将捕获组自动添加到拆分数组中。
这可能会导致不小心地误用，而且提取的捕获组与拆分结果被并列到拆分结果，可能需要二次处理。

aardio 的区别是：
- 使用更简更快的模式匹配
- 明确使用 ^ 或 $ 指定是否提取捕获组，避免误用，且可以灵活控制提取到拆分结果的头部还是尾部
- 可以同时进行行列多级拆分。

这个功能可以节省大量字符串处理代码。
在 aardio 代码中的 string.lines 使用频率也比较高。
*/

console.pause();
```

Markdown 格式