aardio 范例: 静态内存结构体

```
//静态内存结构体
//相关范例：调用其他语言 > C 语言 > 静态内存结构体
import console.int;
import raw.struct;

/*
创建静态内存结构体类型，也可以在第 2 个参数中指定内存指针
*/
var floatArray = raw.struct({
    float v[500]
})

/*
aardio 结构体在与原生 API 交互时动态分配内存指针。
但『静态内存结构体』可以分配固定不变的内存指针，调用原生 API 时不需要再动态分配内存。
在 aardio 中读写静态内存结构体的直接成员（不包含成员的成员）会更慢。

静态内存结构体可直接传入其他线程使用，传入其他线程后指针地址也不会变动。
静态内存结构体的生命周期也由创建该静态内存结构体的线程维护，其他调用线程不检查指针有效性。
多线程可同时读写静态内存结构体，如果有必要请自行调用 thread.lock 添加线程锁。
*/
thread.invokeAndWait(
    function(floatArray){
        /*
        注意创建floatArray的线程负责维护该对象的生命周期，
        使用floatArray的线程不会维护或检查内存指针是否有效。
        */
        floatArray.v = {
            //这种方式修改静态内存结构体，必须覆盖成员字段 floatArray.v[1] = 456 这样写是错的。
            456,2,3
        }

        /*
        如果该结构体的首个成员是数组，
        也可以直接使用索引读写该数组的成员，
        这时候是直接移动指针到索引指定的内存，速度更快
        */
        floatArray[2] = 789.1

        //也可以用 set 函数一次性更新结构体，这样避免多次与原生内存同步数据
        floatArray.set({
            v = {991,992,993}
        })
    },floatArray //可以直接传入线程使用
)

/*
不建议直接用 floatArray.v[1] 这种写法，这会导致不必要的内存同步操作。
我们可以调用 get 函数将结构体的值复制为普通结构体，这样读写就快了。
*/
var array = floatArray.get()

console.log("主线程获取静态内存结构体的值" ,array.v[2] );

/*
如果该结构体的首个成员是数组，
也可以直接使用索引读写该数组的成员，
这时候是直接移动指针到索引指定的内存，相对会好快一些。
*/
console.log("主线程获取静态内存结构体的值" ,floatArray[3] );
```

Markdown 格式