aardio 范例: 调用 .NET(C#) 之 lambda 函数

```
//aardio 调用 .NET(C#) 之 lambda 函数
import dotNet;
var compiler = dotNet.createCompiler("C#");
compiler.Source = /******
namespace CSharpLibrary
{
    public class Object
    {
        public delegate int TestDelegate(int a,int b);

        public TestDelegate GetDelegate()
        {
            //lambda 表达式: 类似JavaScript 的箭头函数，其实就是个匿名函数
            return (int a,int b)=>{
                return a+b;
            };

            /*
            C#3.0(.NET 3.0) 开始支持 lambda 表达式，注意 Win7自带.Net3.5，Win8,Win10,Win11 自带.Net 4.x
            其实 C#2.0(.NET 2.0) 就支持匿名 delegate，只是写法不一样，例如：
            */
            return delegate (int a,int b) { return a+b; };
        }

        /*
        C# 中函数只是代码执行体，委托里才存放了函数的值（或者说函数地址），
        而  lambda 函数将这两者合二为一，这更像 aardio 中的函数对象。
        */
        public TestDelegate method = (int a,int b)=>{
            return a+b;
        };

        public int SetDelegate(TestDelegate test){
            return test(2,3);
        }
    }
}
******/

/*
注意在 aardio 中编译，调用的是 CLR，而 CLR 只有 2.0 / 4.0 的区别，只支持这两个版本的语法。
例如安装了 .NET 3.5 但没有安装 .NET 4.x ，那么 CLR 2.0 下编译器不支持 var ，lambda 这些语法 (但是能运行编译后的 DLL)。
Win10 至少自带 .NET 4.6，Win7 在市场上已经接近消失，现在开发软件再处处考虑 Win7 兼容是不必要的。
*/

compiler.import("CSharpLibrary"); //编译 C# 代码并导入名字空间
var netObj = CSharpLibrary.Object();//创建 .NET 对象

//调用 C# 函数，可以在返回值里返回 .NET 委托
var func = netObj.GetDelegate();
var ret = func(1,2);//直接调用 .NET 委托。

import console;
console.log("调用 C# 委托的返回值",ret);

//调用 C# 对象成员字段里存储的 lambda 函数。
var ret = netObj["method"](22,33);
/*
注意 netObj.method(22,33) 会直接调用成员函数而不取函数值。
如果 netObj.method 不是 C# 函数而是 lambda 值，这样写会报错。
所以我们要写为 netObj["method"] 先取出 lambda 函数。
*/

//C# 委托参数可以指定 aardio 中的 lambda 函数
var ret = netObj.SetDelegate(
    lambda(a,b) a + b
)

//下面的写法与上面是等价的，lambda 就是匿名函数，只是写法上的区别。
var ret = netObj.SetDelegate(
    function(a,b) {
        return a + b;
    }
)
```

```
console.log(ret);
console.pause();
```

Markdown 格式