TscanCode V2.9

用

户

手

册

修订记录

修订日期	修订版本	修改描述	作者
2016/12/19	V2.1.0	制定初稿	quark
2017/8/18	V2.8.0	修订	quark
2017/10/10	V2.9.0	增加代码扫描能力章节和工具配置说明章节	quark;ben

目录

一、 引言

1.1 编写目的

编写本文的目的在于说明如何使用 TscanCode GUI 工具，包含三个部分内容：

1. TscanCode 的静态代码扫描能力；
2. TscanCode 工具的使用说明；
3. TscanCode 工具的配置说明；

1.2 软件概述

TscanCode 是针对 C++/C#/Lua 代码的静态代码扫描解决方案，能精确发现 C++空指针、Unity 性能、Lua 手误等问题，提高代码质量，降低代码错误修复成本。另外，TscanCode 还支持 C#&Lua，C++&Lua 语言混合扫描，有效挖掘 Unity 项目跨语言交互问题。

TscanCode 单机工具无需构建复杂的编译环境，一键扫描，支持用户根据不同需求自定义配置检查项，有良好的扩展性和可维护性。

二、 静态代码扫描能力

2.1 检查项

TscanCode 工具支持 C++，C#，Lua 三种语言，总计 159 条扫描规则，按照语言和规则类型划分如下：

语言	类型	规则数	说明
C++	空指针错误	7	可能导致程序空指针解引用的错误

	越界错误	7	数组、缓冲区访问越界
	资源泄漏错误	8	内存或者资源（文件、管道等）泄漏错误
	运算错误	11	sizeof, ++等运算符导致的代码错误
	可疑错误	16	可疑的代码错误，比如 assert 中进行赋值操作
	逻辑错误	26	不符合正常代码逻辑的代码场景
	未初始化错误	10	变量、结构体、指针未初始化，然后使用
C#	空引用错误	6	可能导致空对象解引用的错误场景
	逻辑错误	22	不符合正常代码逻辑的代码场景
	Unity 特性检查	14	针对 Unity 项目性能、常见错误的检查项
Lua	未初始化	11	检查变量初始化相关问题
	语法错误	1	检查语法相关问题。目前只检查括号，if-end 匹配等有限的语法错误
	跨语言交互	3	检查 Lua 和 C++、C#交互相关的问题
	逻辑错误	17	除以上三种错误以外的其他问题，如函数参数匹配，变量重名，变量类型混用等

2.2 C++

典型空指针错误示例

空指针错误是 C++程序最容易遇到的程序错误，TscanCode 工具能够扫描各种明确的或者有极大风险出问题的空指针代码场景。

场景一，判空条件逻辑缺陷，导致空指针错误：

```
int Demo(STNullPointer* npSt)
{
    // if 条件表达式存在逻辑漏洞，&&应该换成||
    if (npSt == nullptr && npSt->m_node)
    {
        return nResult;
    }
    return 0;
}
```

场景二，指针判空范围覆盖不全：

```
void Demo(C* obj)
{
    if (obj != NULL) // obj 判空
    {
        obj->dosth();
    }
    // obj 解引用，此时对于 obj 的判空保护已经失效
    obj->dosth2();
}
```

越界&未初始化错误示例

下面是一个典型的存在越界风险的场景：

```
char buf[10];

void Demo(int i)
{
    // i 可能等于 10，下标保护条件存在漏洞
    if (i < 0 || i > 10)
        return;

    // i 等于 10 时，数组越界
    buf[i] = 'Q';
}
```

未初始化场景：

```
void Demo(int b)
{
    int a;
    if (b > 10)
    {
        a = 10;
    }

    // 缺少 else 分支，变量 a 可能不会初始化

    a++;
}
```

逻辑&可疑错误示例

代码作者意识到了 i * j 可能溢出，但是写法有问题：

```
void Demo(int i, int j)
{

    // 正确的写法应该是:long long t = (long long)i * j;

    long long ll = i * j;
}
```

场景二，多了一个分号：

```
void Demo(int iMax)
{

    // if 表达式后面的分号很有可能是多余的

    if (iMax > 0 && iMax != 3);
    {

    }
}
```

2.3 C#

典型空引用错误示例

类似于 C++空指针场景，C#语言中空对象解引用问题也经常会遇到：

```
class C
{
    public void Demo(A myA)
    {
        int a = 0;
        if (myA == null)
        {
            //判定了 myA 为空使用，可能条件写错了
            a = myA.a;
        }
    }
}
```

场景二：

```
class C
{
    public void Demo(A myA)
    {
        int a = 0;
        a = myA.a;
        //前面已经使用了 myA,要么检查是多余的，要么前面的使用存在风险
        if (myA == null)
        {
            return;
        }
    }
}
```

逻辑错误示例

场景一，格式化字符串中参数个数与实际传入的参数个数不一致：

```
class C
{
    public void Demo()
    {
        //格式化需要两个参数，实际只传入一个参数
        string str_cur = string.Format(
            "nothing{sss},string{120}", str);
    }
}
```

场景二，条件表达式存在逻辑缺陷：

```
class C
{
    public void Demo()
    {
        int nLogic = fRetVale();
        //条件恒为假
        if ((nLogic > 10) && (nLogic < 9))
        {
            Console.WriteLine("Error");
        }
    }
}
```

Unity 定制规则示例

场景一，MonoBehaviour 的子类原则上不应该实现构造函数：

```
class CS_UnsafeConstructor : MonoBehaviour
{
    //Unity 项目中，mono 子类不应该实现自己的构造函数
    public CS_UnsafeConstructor()
    {
        DoSomething();
    }
}
```

场景二，update 函数中使用 foreach 没产生额外的 GC，影响性能：

```
class ForEach : MonoBehaviour
{
    public Update()
    {
        //Unity 项目中，不应该在 Update 函数中使用 foreach，这回导致 GC
        foreach (var i in m_list)
        {
        }
    }
}
```

2.4 Lua

未初始化错误示例

lua 是一门动态语言，变量第一次使用即定义，这种特性导致程序中容易出现初始化相关的问题，这也是工具主要要扫描的问题。下面列举一些实际项目中出现过的案例。

未初始化示例 1，变量名拼写错误：

```
--前序代码省略
function Demo:onClose(eventType, tag)
    --很明显 enentType 拼写错误，成为一个新的变量
    self:onOption(enentType, #self.data)
    self:hide()
end
```

未初始化示例 2，变量应该初始化为空字符串或者应该添加 else 分支：

```
--前序代码省略
function Demo:setData(idx, data)
    self.chatBg.setFlipX(idx % 2 == 0)
    if data then
        local person = nil
        if data.type == 1 then
            person = "lover"
        elseif data.type == 2 then
            person = "friend"
        elseif data.type == 3 then
            person = "parent"
        end
        --person 初始值为nil，且缺少else分支

        self.ToLabel:setString("对他的"..person.."说")
    end
end
```

逻辑错误示例

逻辑错误示例 1，默认值赋值为 true 不应该用 or true：

```
--前序代码省略
function Demo:setAnim(state, loop, callback)
    state = state or 1
    --or true 存在缺陷，如果loop 赋值为false，loop 的值还是会变成true
    loop = loop or true
    callback = callback or 0
    --省略后续代码
end
```

逻辑错误示例 2，函数参数不可省略：

```
function CheckFull(idx)
    return _G['bag'][idx] >= 10
end

function Demo()
    --函数有一个参数，且参数不可省略
    CheckFull()
end
```

跨语言交互错误示例

C#访问了未定义的 lua 函数，C#代码：

```
using XLua;
namespace Assets.Scripts.Logic.Battle
{
    public static class BattleUtility
    {
        private static ProfileDelegate _profile = null;

        public static void Initialize(LuaEnv lua)
        {
            //ProfileReport 函数没有定义，直接访问
            _profile =
lua.Global.GetInPath<ProfileDelegate>("BattleUtility.ProfileReport");
        }

        public static void LuaProfileReport()
        {
            _profile();
        }
    }
}
```

BattleUtility.lua 文件的 Lua 代码：

```
function ReportVer()
    print('version:5.33')
end
```

语法错误示例

lua 的 elseif 和 C++的 else if 写法不一样：

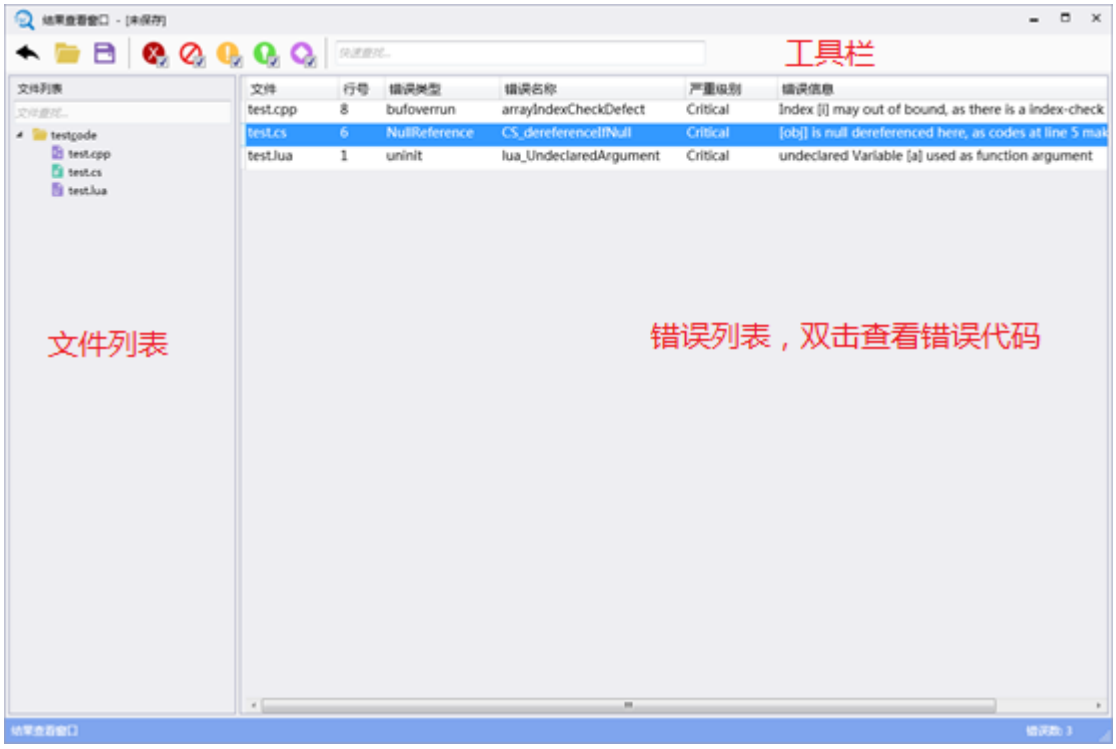
```
function Demo(idx)
    if idx > 10 then
        return 1

    --应该是elseif

    else if idx > 0 then
        return 0
    else
        return -1
    end
end
```

三、 工具使用说明

3.1 界面说明

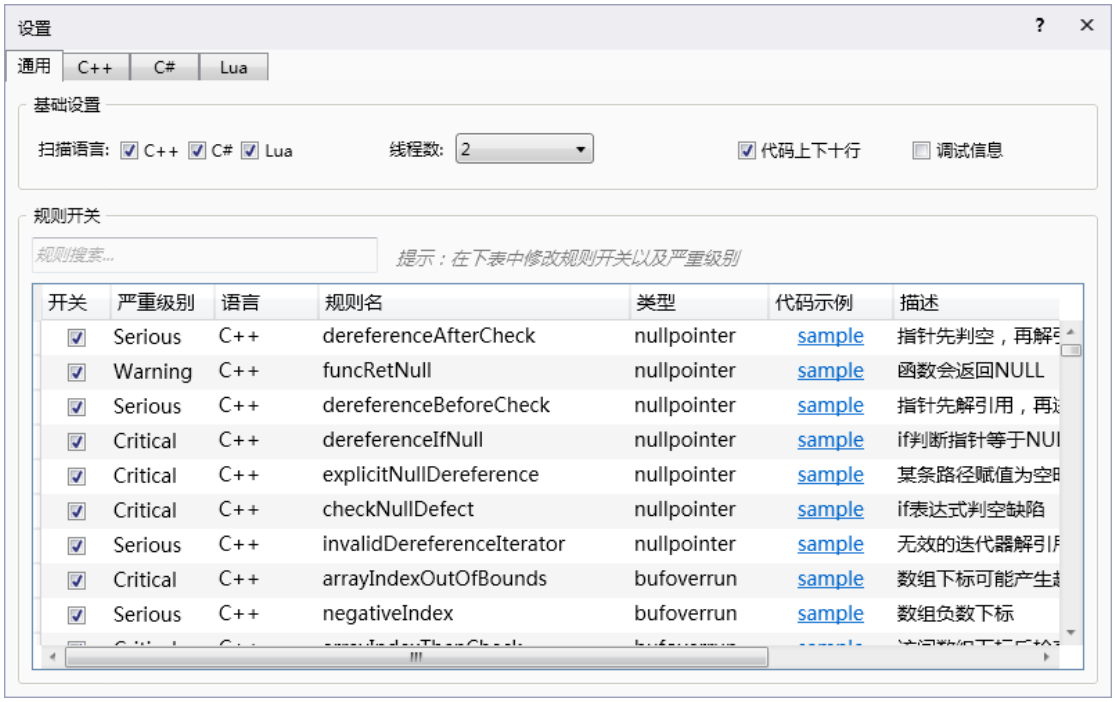
主界面

结果查看窗口



- 支持根据错误“严重程度”来筛选错误；
- 支持快速查找（错误类型、错误信息）；
- 支持按照报错文件进行筛选错误；
- 支持保存成 xml 文件和打开错误 xml 文件。

设置界面



- 扫描基础配置，包括扫描线程数量，代码上下十行，检查项开关，sample 查看等；
- C++，C#，Lua 语言特有配置。

3.2 操作流程

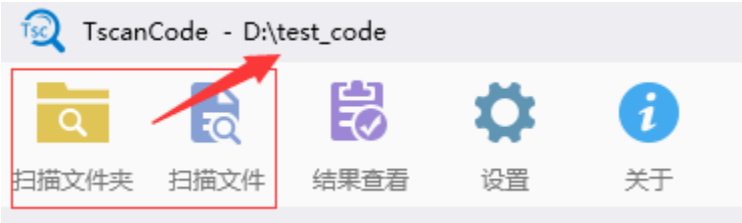
Step1 执行安装程序

下载并执行安装程序，根据安装向导完成安装。

Step2 修改扫描配置（可选）

TscanCode 工具针对各语言有一套默认配置，为了达到更好的扫描效果，您可以针对项目具体情况进行配置。

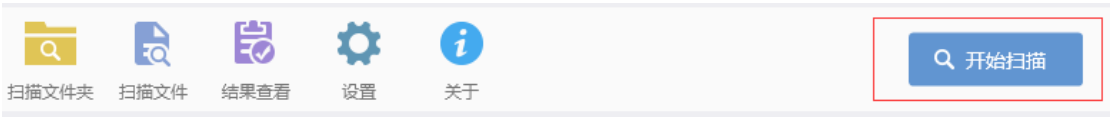
Step3 设置扫描文件夹/文件



Step4 启动/终止扫描

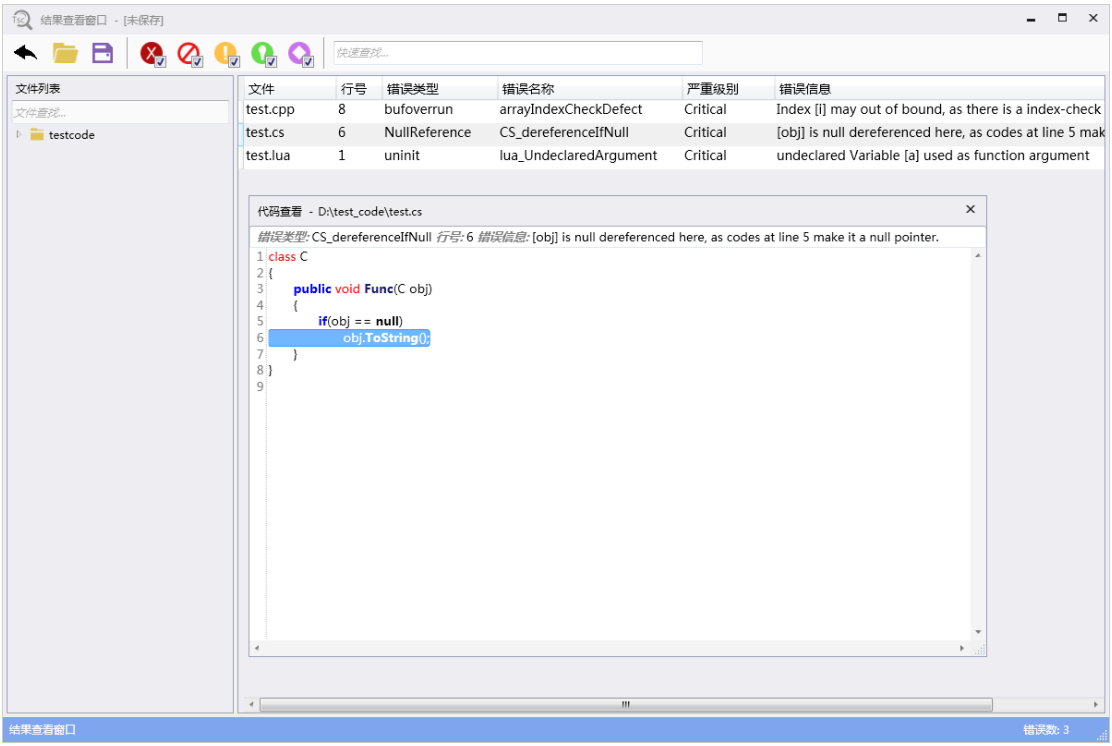
点击主界面工具栏右侧的“启动扫描”按钮，开启扫描。该按钮在扫描过程中会

切换为“停止扫描”，点击即可中断扫描过程。



Step5 结果查看、保存

扫描完成后，结果查看窗口会自动弹出，用户可以查看并保存错误列表，并进行代码修复。



四、工具配置说明

根据项目具体情况，您可以通过修改工具配置来达到更优的扫描结果。

4.1 公共配置

扫描语言

工具支持 C++、C#和 Lua 扫描，如果不需要扫描某种语言，可以取消勾选。如果需要检查跨语言问题，需要同时扫描三种语言。

线程数

线程数控制后台扫描进程使用的工作线程数，这个参数需要根据机器性能和代码规模来确定，一般不建议超过机器 cpu 核心数，或者不大于 4 个线程。

代码上下 10 行

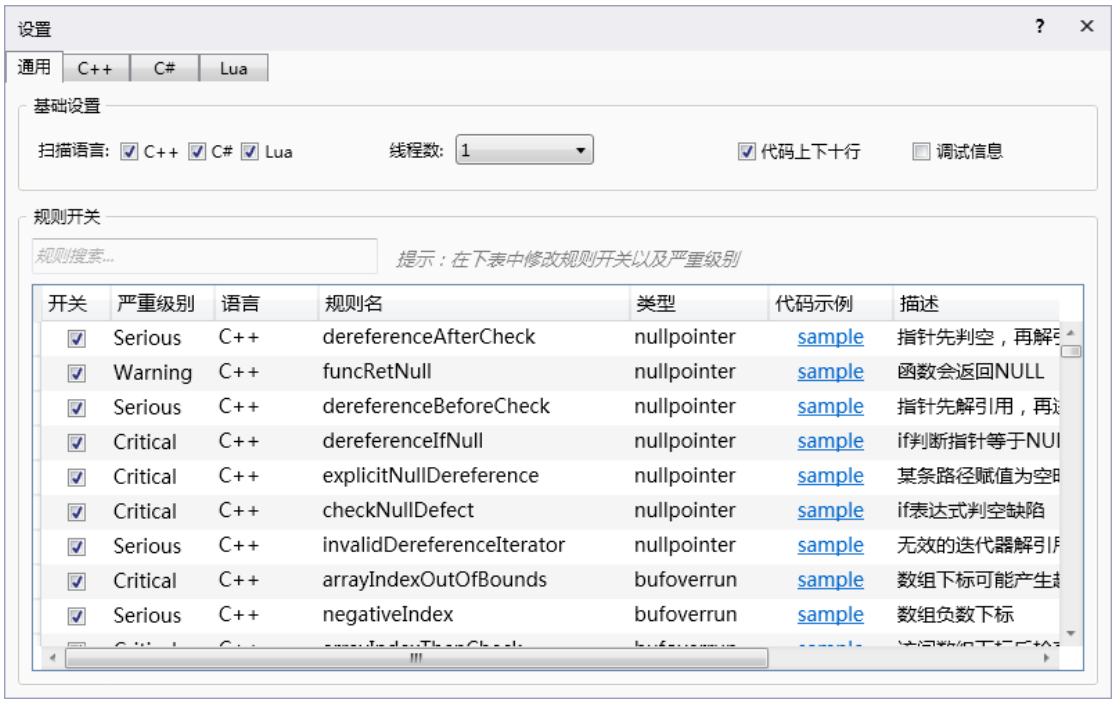
勾选后，保存的扫描结果中会包含报错所在代码行的上下 10 行代码。

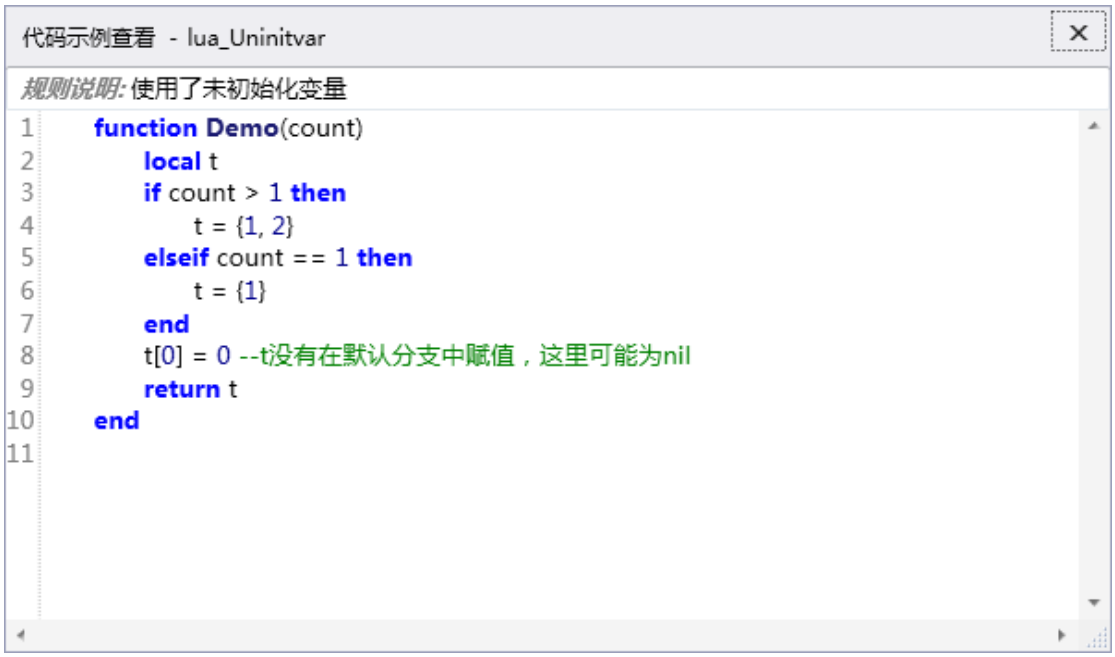
调试信息

当扫描工具出现异常后，用于辅助开发者进行问题定位，不建议勾选。

规则开关

扫描工具将错误按照规则分为很多类，每条规则对应一种具体的问题，如果不关注某一类问题，请关闭该条规则。规则的代码示例可以点击 sample 展示。





4.2 C++配置

屏蔽路径

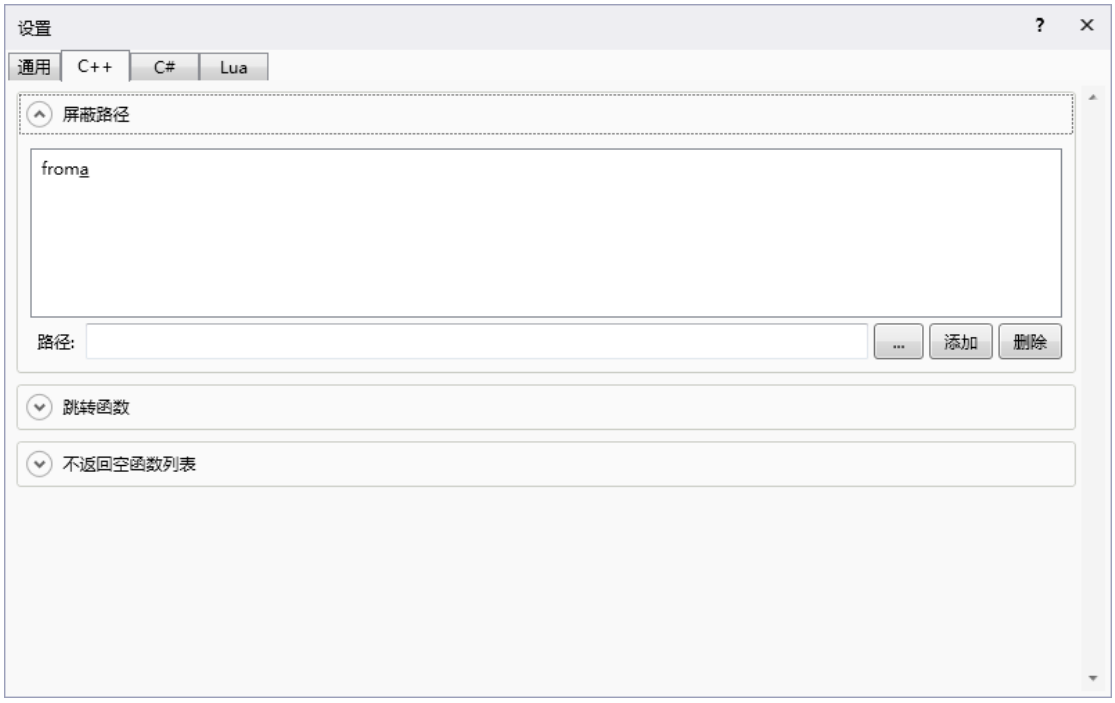
被屏蔽的文件路径包含的代码文件不会被扫描。这里通常需要配置一些第三方库目录。

跳转函数

项目中可能会使用类似于 assert 的宏或者函数来检查指针变量是否为 NULL，工具可能没有正确识别这个宏或者参数，导致误报。在这里配置该宏名称以及宏检查的参数位置，工具可以正确过滤被检查的指针变量。

不返回空函数列表

工具会检查函数返回值是否可能为空，某些情况下，工具的判定会返回空，但实际不会返回空，可以在此配置函数的名称，以忽略相关的报错。



4.3 C#配置

屏蔽路径

被屏蔽的文件路径包含的代码文件不会被扫描。这里通常需要配置一些第三方库目录。

跳转函数

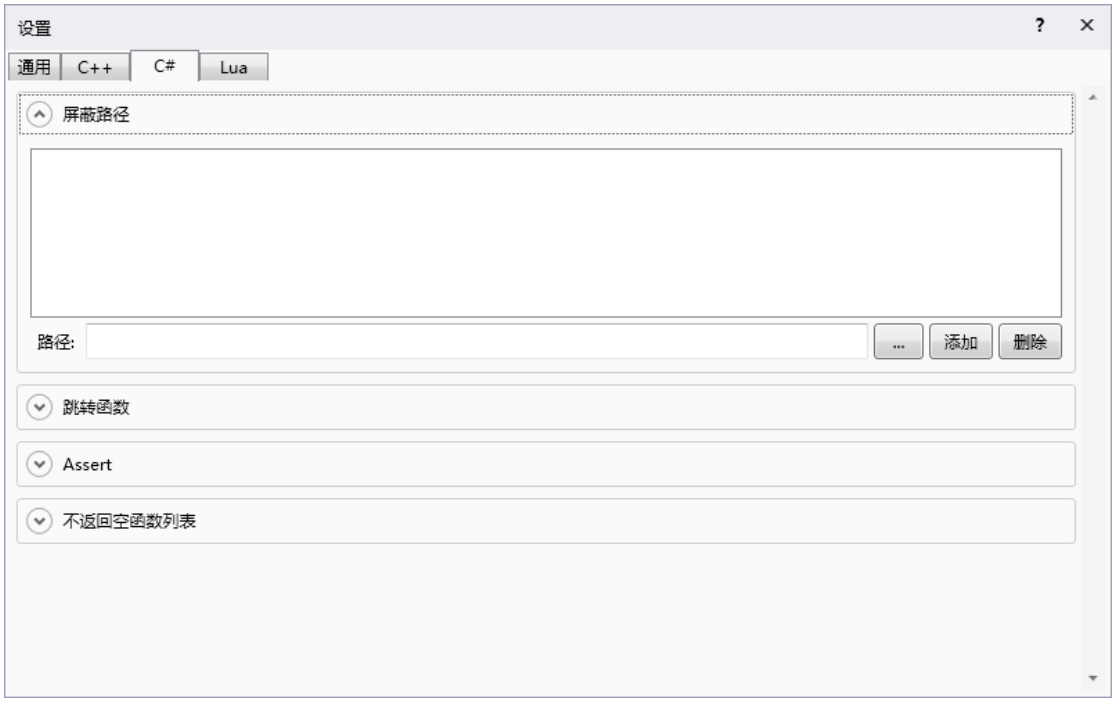
项目中可能会使用类似于 assert 的宏或者函数来检查对象是否为 null，工具可能没有正确识别这个宏或者参数，导致误报。在这里配置该宏名称以及宏检查的参数位置，工具可以正确过滤被检查的对象。

Assert

配置某些没有被正确识别、拥有类似 Assert 函数功能的函数名。

不返回空函数列表

工具会检查函数返回值是否可能为空，某些情况下，工具的判定会返回空，但实际不会返回空，可以在此配置函数的名称，以忽略相关的报错。



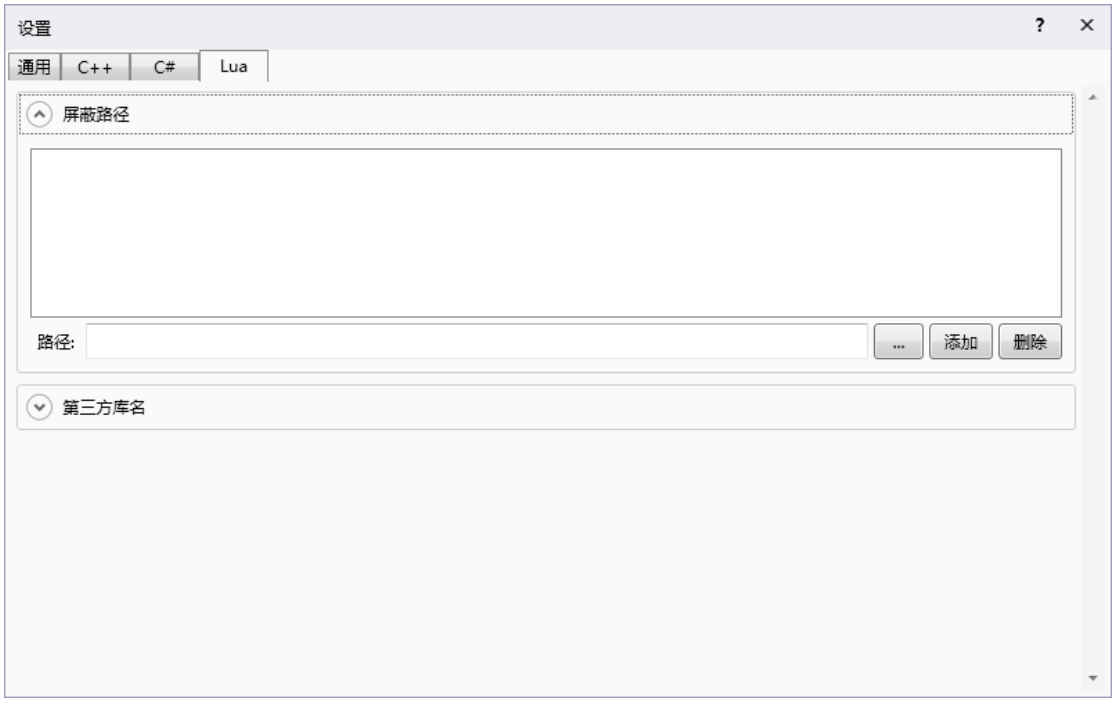
4.4 Lua 配置

屏蔽路径

被屏蔽的文件路径包含的代码文件不会被扫描。这里通常需要配置一些第三方库目录。

第三方库

Lua 会检查未定义的变量，如果是未知的第三方库，会导致误报。需要在这里配置第三方库名称，以屏蔽误报。工具已经添加了一些常见第三方库，包括 GDK，UnityEngine，ngx，winapi，wxLua 等。通过其他特殊方式定义的变量也可以在此添加屏蔽。


设置
通用
C++
C#
Lua
屏蔽路径
路径:
...
添加
删除
第三方库名