# Tbox

**简介**

TBOX是一个用c语言实现的跨平台开发库。

针对各个平台，封装了统一的接口，简化了各类开发过程中常用操作，使你在开发过程中，更加关注实际应用的开发，而不是把时间浪费在琐碎的接口兼容性上面，并且充分利用了各个平台独有的一些特性进行优化。

这个项目的目的，是为了使C开发更加的简单高效。

目前支持的平台有: Windows, Macosx, Linux, Android, iOS, \*BSD等等。

通过[xmake](https://github.com/xmake-io/xmake)支持各种编译模式：

* Release: 正式版编译，禁用调试信息、断言，各种检测机制，启用编译器优化
* Debug: 调试模式，默认启用详细调试信息、断言、内存越界检测、内存泄漏、锁竞争分析等检测机制
* Small: 最小化编译，默认禁用所有扩展模块，启用编译器最小化优化
* Micro: 针对嵌入式平台，仅仅编译tbox微内核，仅提供最基础的跨平台接口，生成库仅64K左右（内置轻量libc接口实现）

如果你想了解更多，请参考：[在线文档](https://docs.tboox.org/#/zh-cn/getting_started), [Github](https://github.com/tboox/tbox)以及[Gitee](https://gitee.com/tboox/tbox)

**特性**

**流库**

针对http、file、socket、data等流数据，实现统一接口进行读写，并且支持: 阻塞、非阻塞、异步 三种读写模式。 支持中间增加多层filter流进行流过滤，实现边读取，内部边进行解压、编码转换、加密等操作，极大的减少了内存使用。

主要提供以下模块：

* stream：通用非阻塞流，用于一般的单独io处理，同时支持协程以实现异步传输。
* transfer：流传输器，维护两路流的传输。
* static\_stream：针对静态数据buffer优化的静态流，用于轻量快速的数据解析。

**协程库**

* 快速高效的协程切换支持
* 提供跨平台支持，核心切换算法参考boost，并且对其进行重写和优化，目前支持架构：x86, x86\_64, arm, arm64, mips32
* 提供channel协程间数据通信支持，基于生产、消费者模型
* 提供信号量、协程锁支持
* socket、stream都模块原生支持协程，并且可在线程和协程间进行无缝切换
* 提供http、file等基于协程的简单服务器实例，只需几百行代码，就可以从socket开始写个高性能io服务器，代码逻辑比异步回调模式更加清晰
* 同时提供stackfull, stackless两种协程模式支持，stackless协程更加的轻量（每个协程只占用几十个bytes），切换更快（会牺牲部分易用性）
* 支持epoll, kqueue, poll, select 和 IOCP
* 在协程和poller中支持同时等待和调度socket，pipe io和process

**数据库**

* 统一并简化数据库操作接口，适配各种数据源，通过统一的url来自动连接打开支持的数据库，数据的枚举采用迭代器模型。
* 目前支持sqlite3以及mysql两种关系型数据库，也可自定义扩展使用其他关系型数据库。

**xml库**

* 针对xml提供DOM和SAX两种解析模式，SAX方式采用外部迭代模式，灵活性和性能更高，并且可以选择指定路径，进行解析。
* 解析过程完全基于stream，所以是高度流化的，可以实现边下载、边解压、边转码、边解析一条龙服务，使用较低的内存也可以解析大规模数据。
* 提供xml writer以支持对xml生成

**内存库**

* 参考linux内核内存管理机制的实现，并对其进行各种改造和优化，所实现的TBOX独有的一整套内存池管理架构。
* 调试模式下，可以轻松检测并定位内存泄露、内存越界溢出、内存重叠覆盖等常见内存问题，并对整体内存的使用进行了统计和简要分析。
* 针对大块数据、小块数据、字符串数据进行了充分的利用，避免了大量外部碎片和内部碎片的产生。分配操作进行了各种优化，96%的情况下，效率都是在O(1)。

**容器库**

* 提供哈希、链表、数组、队列、堆栈、最小最大堆等常用容器。
* 支持各种常用成员类型，在原有的容器期初上，其成员类型还可以完全自定义扩展。
* 所有容器都支持迭代器操作。
* 大部分容器都可以支持基于stream的序列化和反序列化操作。

**算法库**

* 提供各种排序算法：冒泡排序、堆排序、快速排序、插入排序。
* 提供各种查找算法：线性遍历、二分法搜索。
* 提供各种遍历、删除、统计算法。
* 以迭代器为接口，实现算法和容器的分离，类似stl，但是c实现的，更加轻量。

**网络库**

* 实现http客户端模块
* 实现cookies
* 实现dns解析与缓存
* 实现ssl(支持openssl, polarssl, mbedtls)
* 支持ipv4、ipv6
* 支持通过协程实现异步模式

**数学运算库**

* 提供各种精度的定点运算支持
* 提供随机数生成器

**libc库**

* libc的一个轻量级实现，完全跨平台，并且针对不同架构进行了优化。
* 支持大部分字符串、宽字符串操作。
* 扩展字符串、宽字符串的各种大小写不敏感操作接口
* 扩展memset\_u16、memset\_u32等接口，并对其进行高度优化，尤其适合图形渲染程序

**libm库**

* libm部分接口的一个轻量级实现，以及对常用系统接口的封装。（目前只实现了部分，之后有时间会完全实现掉）
* 扩展部分常用接口，增加对sqrt、log2等常用函数的整数版本计算，进行高度优化，不涉及浮点运算，适合嵌入式环境使用。

**object库**

* 轻量级类apple的CoreFoundation库，支持object、dictionary、array、string、number、date、data等常用对象，并且可以方便扩展自定义对象的序列化。
* 支持对xml、json、binary以及apple的plist(xplist/bplist)格式序列化和反序列化。 并且实现自有的binary序列化格式， 针对明文进行了简单的加密，在不影响性能的前提下，序列化后的大小比bplist节省30%。

**平台库**

* 提供file、directory、socket、thread、time等常用系统接口
* 提供atomic、atomic64接口
* 提供高精度、低精度定时器
* 提供高性能的线程池操作
* 提供event、mutex、semaphore、spinlock等事件、互斥、信号量、自旋锁操作
* 提供获取函数堆栈信息的接口，方便调试和错误定位
* 提供跨平台动态库加载接口（如果系统支持的话）
* 提供io轮询器，针对epoll, poll, select, kqueue进行跨平台封装
* 提供跨平台上下文切换接口，主要用于协程实现，切换效率非常高

**压缩库**

* 支持zlib/zlibraw/gzip的压缩与解压（需要第三方zlib库支持）。

**字符编码库**

* 支持utf8、utf16、gbk、gb2312、uc2、uc4 之间的互相转码，并且支持大小端格式。

**实用工具库**

* 实现base64/32编解码
* 实现crc32、adler32、md5、sha1等常用hash算法
* 实现日志输出、断言等辅助调试工具
* 实现url编解码
* 实现位操作相关接口，支持各种数据格式的解析，可以对8bits、16bits、32bits、64bits、float、double以及任意bits的字段进行解析操作，并且同时支持大端、小端和本地端模式，并针对部分操作进行了优化，像static\_stream、stream都有相关接口对其进行了封装，方便在流上进行快速数据解析。
* 实现swap16、swap32、swap64等位交换操作，并针对各个平台进行了优化。
* 实现一些高级的位处理接口，例如：位0的快速统计、前导0和前导1的快速位计数、后导01的快速位计数
* 实现单例模块，可以对静态对象、实例对象进行快速的单例封装，实现全局线程安全
* 实现option模块，对命令行参数进行解析，提供快速方便的命令行选项建立和解析操作，对于写终端程序还是很有帮助的

**正则表达式库**

* 支持匹配和替换操作
* 支持全局、多行、大小写不敏感等模式
* 使用pcre, pcre2和posix正则库

**一些使用tbox的项目：**

* [gbox](https://github.com/tboox/gbox)
* [vm86](https://github.com/tboox/vm86)
* [xmake](http://www.xmake.io/cn)
* [itrace](https://github.com/tboox/itrace)
* [更多项目](https://github.com/tboox/tbox/wiki/%E4%BD%BF%E7%94%A8tbox%E7%9A%84%E5%BC%80%E6%BA%90%E5%BA%93)

**使用 Xmake 编译**

请先安装: [xmake](https://github.com/xmake-io/xmake)

# 默认直接编译当前主机平台

$ cd ./tbox

$ xmake

# 编译mingw平台

$ cd ./tbox

$ xmake f -p mingw --sdk=/home/mingwsdk

$ xmake

# 编译iphoneos平台

$ cd ./tbox

$ xmake f -p iphoneos

$ xmake

# 编译android平台

$ cd ./tbox

$ xmake f -p android --ndk=xxxxx

$ xmake

# 交叉编译

$ cd ./tbox

$ xmake f -p linux --sdk=/home/sdk #--bin=/home/sdk/bin

$ xmake

**使用 xmake.sh 编译**

$ ./configure

$ make

**例子**

#include "tbox/tbox.h"

int main(int argc, char\*\* argv) {

if (!tb\_init(tb\_null, tb\_null)) return 0;

tb\_vector\_ref\_t vector = tb\_vector\_init(0, tb\_element\_str(tb\_true));

if (vector) {

tb\_vector\_insert\_tail(vector, "hello");

tb\_vector\_insert\_tail(vector, "tbox");

tb\_for\_all (tb\_char\_t const\*, cstr, vector) {

tb\_trace\_i("%s", cstr);

}

tb\_vector\_exit(vector);

}

tb\_exit();

return 0;

}

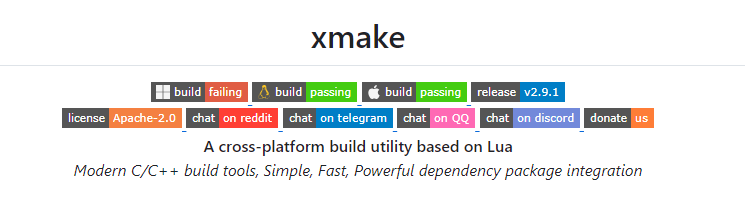
**技术支持**

你可以考虑赞助我们也获取技术支持服务，[[成为赞助商](https://docs.tboox.org/#/zh-cn/about/sponsor)]

**联系方式**

* 邮箱：[waruqi@gmail.com](mailto:waruqi@gmail.com)
* 主页：[TBOOX开源工程](https://tboox.org/cn)
* 社区：[Reddit论坛](https://www.reddit.com/r/tboox/)
* 聊天：[Telegram群组](https://t.me/tbooxorg), [Gitter聊天室](https://gitter.im/tboox/tboox?utm_source=badge&utm_medium=badge&utm_campaign=pr-badge&utm_content=badge)
* QQ群：343118190(满), 662147501
* 微信公众号：tboox-os

# Xmake



## 简介

Xmake 是一个基于 Lua 的轻量级跨平台构建工具。

它非常的轻量，没有任何依赖，因为它内置了 Lua 运行时。

它使用 xmake.lua 维护项目构建，相比 makefile/CMakeLists.txt，配置语法更加简洁直观，对新手非常友好，短时间内就能快速入门，能够让用户把更多的精力集中在实际的项目开发上。

我们能够使用它像 Make/Ninja 那样可以直接编译项目，也可以像 CMake/Meson 那样生成工程文件，另外它还有内置的包管理系统来帮助用户解决 C/C++ 依赖库的集成使用问题。

目前，Xmake 主要用于 C/C++ 项目的构建，但是同时也支持其他 native 语言的构建，可以实现跟 C/C++ 进行混合编译，同时编译速度也是非常的快，可以跟 Ninja 持平。

Xmake = Build backend + Project Generator + Package Manager + [Remote|Distributed] Build + Cache

尽管不是很准确，但我们还是可以把 Xmake 按下面的方式来理解：

Xmake ≈ Make/Ninja + CMake/Meson + Vcpkg/Conan + distcc + ccache/sccache

如果你想要了解更多，请参考：[在线文档](https://xmake.io/#/zh-cn/getting_started), [Github](https://github.com/xmake-io/xmake)以及[Gitee](https://gitee.com/tboox/xmake)，同时也欢迎加入我们的 [社区](https://xmake.io/#/zh-ch/about/contact).

## 课程

xmake 官方也推出了一些入门课程，带你一步步快速上手 xmake，课程列表如下：

* [Xmake 带你轻松构建 C/C++ 项目](https://xmake.io/#/zh-cn/about/course)

## 安装

#### 使用curl

curl -fsSL https://xmake.io/shget.text | bash

#### 使用wget

wget https://xmake.io/shget.text -O - | bash

#### 使用powershell

Invoke-Expression (Invoke-Webrequest 'https://xmake.io/psget.text' -UseBasicParsing).Content

#### 其他安装方式

如果不想使用脚本安装，也可以点击查看 [安装文档](https://xmake.io/#/zh-cn/guide/installation)，了解其他安装方法。

## 简单的工程描述

target("hello")

set\_kind("binary")

add\_files("src/\*.cpp")

## 包依赖描述

add\_requires("tbox 1.6.\*", "zlib", "libpng ~1.6")

官方的xmake包管理仓库: [xmake-repo](https://github.com/xmake-io/xmake-repo)

## 命令行使用

### 创建工程

$ xmake create hello

$ cd hello

### 构建工程

$ xmake

### 运行目标

$ xmake run console

### 调试程序

$ xmake run -d console

### 运行测试

$ xmake test

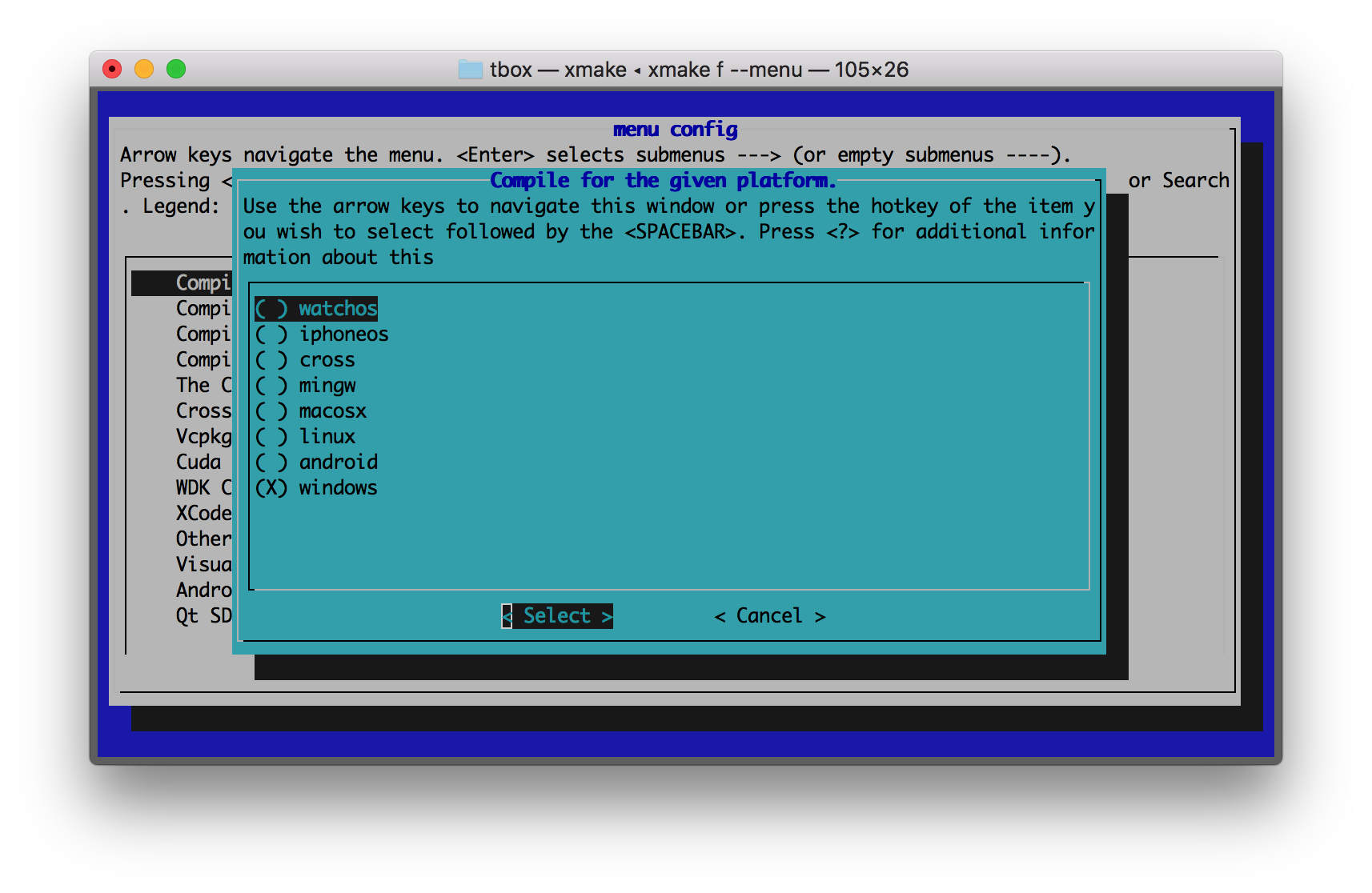
### 配置平台

$ xmake f -p [windows|linux|macosx|android|iphoneos ..] -a [x86|arm64 ..] -m [debug|release]

$ xmake

### 图形化菜单配置

$ xmake f --menu



## 跟ninja一样快的构建速度

测试工程: [xmake-core](https://github.com/xmake-io/xmake/tree/master/core)

### 多任务并行编译测试

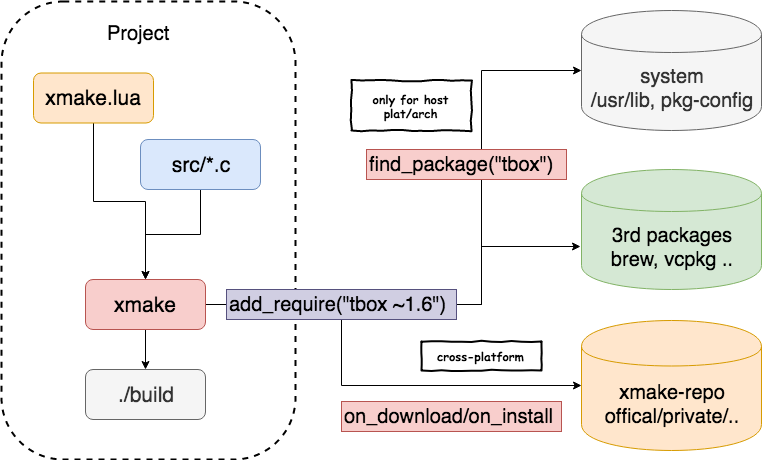
| **构建系统** | **Termux (8core/-j12)** | **构建系统** | **MacOS (8core/-j12)** |
| --- | --- | --- | --- |
| xmake | 24.890s | xmake | 12.264s |
| ninja | 25.682s | ninja | 11.327s |
| cmake(gen+make) | 5.416s+28.473s | cmake(gen+make) | 1.203s+14.030s |
| cmake(gen+ninja) | 4.458s+24.842s | cmake(gen+ninja) | 0.988s+11.644s |

### 单任务编译测试

| **构建系统** | **Termux (-j1)** | **构建系统** | **MacOS (-j1)** |
| --- | --- | --- | --- |
| xmake | 1m57.707s | xmake | 39.937s |
| ninja | 1m52.845s | ninja | 38.995s |
| cmake(gen+make) | 5.416s+2m10.539s | cmake(gen+make) | 1.203s+41.737s |
| cmake(gen+ninja) | 4.458s+1m54.868s | cmake(gen+ninja) | 0.988s+38.022s |

## 包依赖管理

### 架构和流程



### 支持的包管理仓库

* 官方自建仓库 [xmake-repo](https://github.com/xmake-io/xmake-repo) (tbox >1.6.1)
* 官方包管理器 [Xrepo](https://github.com/xmake-io/xrepo)
* [用户自建仓库](https://xmake.io/#/zh-cn/package/remote_package?id=%e4%bd%bf%e7%94%a8%e8%87%aa%e5%bb%ba%e7%a7%81%e6%9c%89%e5%8c%85%e4%bb%93%e5%ba%93)
* Conan (conan::openssl/1.1.1g)
* Conda (conda::libpng 1.3.67)
* Vcpkg (vcpkg::ffmpeg)
* Homebrew/Linuxbrew (brew::pcre2/libpcre2-8)
* Pacman on archlinux/msys2 (pacman::libcurl)
* Apt on ubuntu/debian (apt::zlib1g-dev)
* Clib (clib::clibs/bytes@0.0.4)
* Dub (dub::log 0.4.3)
* Portage on Gentoo/Linux (portage::libhandy)
* Nimble for nimlang (nimble::zip >1.3)
* Cargo for rust (cargo::base64 0.13.0)

### 包管理特性

* 官方仓库提供近 800+ 常用包，真正做到全平台一键下载集成编译
* 全平台包支持，支持交叉编译的依赖包集成
* 支持包虚拟环境管理和加载，xrepo env shell
* Windows 云端预编译包加速
* 支持自建包仓库，私有仓库部署
* 第三方包仓库支持，提供更加丰富的包源，例如：vcpkg, conan, conda 等等
* 支持自动拉取使用云端工具链
* 支持包依赖锁定

## 支持平台

* Windows (x86, x64, arm64)
* macOS (i386, x86\_64, arm64)
* Linux (i386, x86\_64, arm, arm64, riscv, mips, 390x, sh4 ...)
* \*BSD (i386, x86\_64)
* Android (x86, x86\_64, armeabi, armeabi-v7a, arm64-v8a)
* iOS (armv7, armv7s, arm64, i386, x86\_64)
* WatchOS (armv7k, i386)
* AppleTVOS (armv7, arm64, i386, x86\_64)
* AppleXROS (arm64, x86\_64)
* MSYS (i386, x86\_64)
* MinGW (i386, x86\_64, arm, arm64)
* Cygwin (i386, x86\_64)
* Wasm (wasm32, wasm64)
* Haiku (i386, x86\_64)
* Harmony (x86\_64, armeabi-v7a, arm64-v8a)
* Cross (cross-toolchains ..)

## 支持工具链

$ xmake show -l toolchains

xcode Xcode IDE

msvc Microsoft Visual C/C++ Compiler

clang-cl LLVM Clang C/C++ Compiler compatible with msvc

yasm The Yasm Modular Assembler

clang A C language family frontend for LLVM

go Go Programming Language Compiler

dlang D Programming Language Compiler (Auto)

dmd D Programming Language Compiler

ldc The LLVM-based D Compiler

gdc The GNU D Compiler (GDC)

gfortran GNU Fortran Programming Language Compiler

zig Zig Programming Language Compiler

sdcc Small Device C Compiler

cuda CUDA Toolkit (nvcc, nvc, nvc++, nvfortran)

ndk Android NDK

rust Rust Programming Language Compiler

swift Swift Programming Language Compiler

llvm A collection of modular and reusable compiler and toolchain technologies

cross Common cross compilation toolchain

nasm NASM Assembler

gcc GNU Compiler Collection

mingw Minimalist GNU for Windows

gnu-rm GNU Arm Embedded Toolchain

envs Environment variables toolchain

fasm Flat Assembler

tinycc Tiny C Compiler

emcc A toolchain for compiling to asm.js and WebAssembly

icc Intel C/C++ Compiler

ifort Intel Fortran Compiler

muslcc The musl-based cross-compilation toolchain

fpc Free Pascal Programming Language Compiler

wasi WASI-enabled WebAssembly C/C++ toolchain

nim Nim Programming Language Compiler

circle A new C++20 compiler

armcc ARM Compiler Version 5 of Keil MDK

armclang ARM Compiler Version 6 of Keil MDK

c51 Keil development tools for the 8051 Microcontroller Architecture

icx Intel LLVM C/C++ Compiler

dpcpp Intel LLVM C++ Compiler for data parallel programming model based on Khronos SYCL

masm32 The MASM32 SDK

iverilog Icarus Verilog

verilator Verilator open-source SystemVerilog simulator and lint system

cosmocc build-once run-anywhere

hdk Harmony SDK

## 支持语言

* C/C++
* Objc/Objc++
* Swift
* Assembly
* Golang
* Rust
* Dlang
* Fortran
* Cuda
* Zig
* Vala
* Pascal
* Nim
* Verilog
* FASM
* NASM
* YASM
* MASM32
* Cppfront

## 支持特性

* 语法简单易上手
* 快速安装，无任何依赖
* 全平台一键编译
* 支持交叉编译，智能分析交叉工具链信息
* 极速，多任务并行编译支持
* C++20 Module 支持
* 支持跨平台的 C/C++ 依赖包快速集成，内置包管理器
* 多语言混合编译支持
* 丰富的插件支持，提供各种工程生成器，例如：vs/makefile/cmakelists/compile\_commands 生成插件
* REPL 交互式执行支持
* 增量编译支持，头文件依赖自动分析
* 工具链的快速切换、定制化支持
* 丰富的扩展模块支持
* 远程编译支持
* 分布式编译支持
* 内置的本地和远程编译缓存支持

## 工程类型

* 静态库程序
* 动态库类型
* 控制台程序
* Cuda 程序
* Qt 应用程序
* WDK Windows 驱动程序
* WinSDK 应用程序
* MFC 应用程序
* iOS/MacOS 应用程序（支持.metal）
* Framework和Bundle程序（iOS/MacOS）
* SWIG/Pybind11 模块 (Lua, python, ...)
* Luarocks 模块
* Protobuf 程序
* Lex/yacc 程序
* C++20 模块
* Linux 内核驱动模块
* Keil MDK/C51 嵌入式程序
* Verilog 仿真程序

## 分布式编译和缓存

*  跨平台支持
*  支持 msvc, clang, gcc 和交叉编译工具链
*  支持构建 android, ios, linux, win, macOS 程序
*  除了编译工具链，无任何其他依赖
*  支持编译服务器负载均衡调度
*  支持大文件实时压缩传输 (lz4)
*  几乎零配置成本，无需共享文件系统，更加方便和安全

关于分布式编译和缓存，可以见下面的文档。

* [分布式编译](https://xmake.io/#/zh-cn/features/distcc_build)
* [编译缓存](https://xmake.io/#/zh-cn/features/build_cache)

## 远程编译

更多详情见：[远程编译](https://xmake.io/#/zh-cn/features/remote_build)

## 更多例子

#### Debug 和 Release 模式

add\_rules("mode.debug", "mode.release")

target("console")

set\_kind("binary")

add\_files("src/\*.c")

if is\_mode("debug") then

add\_defines("DEBUG")

end

#### 自定义脚本

target("test")

set\_kind("binary")

add\_files("src/\*.c")

after\_build(function (target)

print("hello: %s", target:name())

os.exec("echo %s", target:targetfile())

end)

#### 依赖包自动集成

下载和使用在 [xmake-repo](https://github.com/xmake-io/xmake-repo) 和第三方包仓库的依赖包：

add\_requires("tbox >1.6.1", "libuv master", "vcpkg::ffmpeg", "brew::pcre2/libpcre2-8")

add\_requires("conan::openssl/1.1.1g", {alias = "openssl", optional = true, debug = true})

target("test")

set\_kind("binary")

add\_files("src/\*.c")

add\_packages("tbox", "libuv", "vcpkg::ffmpeg", "brew::pcre2/libpcre2-8", "openssl")

另外，我们也可以使用 [xrepo](https://github.com/xmake-io/xrepo) 命令来快速安装依赖包。

#### Qt QuickApp 应用程序

target("test")

add\_rules("qt.quickapp")

add\_files("src/\*.cpp")

add\_files("src/qml.qrc")

#### Cuda 程序

target("test")

set\_kind("binary")

add\_files("src/\*.cu")

add\_cugencodes("native")

add\_cugencodes("compute\_35")

#### WDK/UMDF 驱动程序

target("echo")

add\_rules("wdk.driver", "wdk.env.umdf")

add\_files("driver/\*.c")

add\_files("driver/\*.inx")

add\_includedirs("exe")

target("app")

add\_rules("wdk.binary", "wdk.env.umdf")

add\_files("exe/\*.cpp")

更多WDK驱动程序例子(umdf/kmdf/wdm)，见：[WDK工程例子](https://xmake.io/#/zh-cn/guide/project_examples?id=wdk%e9%a9%b1%e5%8a%a8%e7%a8%8b%e5%ba%8f)

#### iOS/MacOS 应用程序

target("test")

add\_rules("xcode.application")

add\_files("src/\*.m", "src/\*\*.storyboard", "src/\*.xcassets")

add\_files("src/Info.plist")

#### Framework 和 Bundle 程序（iOS/MacOS）

target("test")

add\_rules("xcode.framework") -- 或者 xcode.bundle

add\_files("src/\*.m")

add\_files("src/Info.plist")

#### OpenMP 程序

add\_requires("libomp", {optional = true})

target("loop")

set\_kind("binary")

add\_files("src/\*.cpp")

add\_rules("c++.openmp")

add\_packages("libomp")

#### Zig 程序

target("test")

set\_kind("binary")

add\_files("src/main.zig")

### 自动拉取远程工具链

#### 拉取指定版本的 llvm 工具链

我们使用 llvm-10 中的 clang 来编译项目。

add\_requires("llvm 10.x", {alias = "llvm-10"})

target("test")

set\_kind("binary")

add\_files("src/\*.c)

set\_toolchains("llvm@llvm-10")

#### 拉取交叉编译工具链

我们也可以拉取指定的交叉编译工具链来编译项目。

add\_requires("muslcc")

target("test")

set\_kind("binary")

add\_files("src/\*.c)

set\_toolchains("@muslcc")

#### 拉取工具链并且集成对应工具链编译的依赖包

我们也可以使用指定的muslcc交叉编译工具链去编译和集成所有的依赖包。

add\_requires("muslcc")

add\_requires("zlib", "libogg", {system = false})

set\_toolchains("@muslcc")

target("test")

set\_kind("binary")

add\_files("src/\*.c")

add\_packages("zlib", "libogg")

## 插件

#### 生成IDE工程文件插件（makefile, vs2002 - vs2022, ...）

$ xmake project -k vsxmake -m "debug,release" # 新版vs工程生成插件（推荐）

$ xmake project -k vs -m "debug,release"

$ xmake project -k cmake

$ xmake project -k ninja

$ xmake project -k compile\_commands

#### 加载自定义lua脚本插件

$ xmake l ./test.lua

$ xmake l -c "print('hello xmake!')"

$ xmake l lib.detect.find\_tool gcc

$ xmake l

> print("hello xmake!")

> {1, 2, 3}

< {

1,

2,

3

}

更多内置插件见相关文档：[内置插件文档](https://xmake.io/#/zh-cn/plugin/builtin_plugins)

其他扩展插件，请到插件仓库进行下载安装: [xmake-plugins](https://github.com/xmake-io/xmake-plugins).

## IDE和编辑器插件

* [xmake-vscode](https://github.com/xmake-io/xmake-vscode)
* [xmake-sublime](https://github.com/xmake-io/xmake-sublime)
* [xmake-idea](https://github.com/xmake-io/xmake-idea)
* [xmake.vim](https://github.com/luzhlon/xmake.vim) (third-party, thanks [@luzhlon](https://github.com/luzhlon))
* [xmake-visualstudio](https://github.com/HelloWorld886/xmake-visualstudio) (third-party, thanks [@HelloWorld886](https://github.com/HelloWorld886))
* [xmake-qtcreator](https://github.com/Arthapz/xmake-project-manager) (third-party, thanks [@Arthapz](https://github.com/Arthapz))

### XMake Gradle插件 (JNI)

我们也可以在Gradle中使用[xmake-gradle](https://github.com/xmake-io/xmake-gradle)插件来集成编译JNI库

plugins {

id 'org.tboox.gradle-xmake-plugin' version '1.1.5'

}

android {

externalNativeBuild {

xmake {

path "jni/xmake.lua"

}

}

}

当gradle-xmake-plugin插件被应用生效后，xmakeBuild任务会自动注入到现有的assemble任务中去，自动执行jni库编译和集成。

$ ./gradlew app:assembleDebug

> Task :nativelib:xmakeConfigureForArm64

> Task :nativelib:xmakeBuildForArm64

>> xmake build

[ 50%]: ccache compiling.debug nativelib.cc

[ 75%]: linking.debug libnativelib.so

[100%]: build ok!

>> install artifacts to /Users/ruki/projects/personal/xmake-gradle/nativelib/libs/arm64-v8a

> Task :nativelib:xmakeConfigureForArmv7

> Task :nativelib:xmakeBuildForArmv7

>> xmake build

[ 50%]: ccache compiling.debug nativelib.cc

[ 75%]: linking.debug libnativelib.so

[100%]: build ok!

>> install artifacts to /Users/ruki/projects/personal/xmake-gradle/nativelib/libs/armeabi-v7a

> Task :nativelib:preBuild

> Task :nativelib:assemble

> Task :app:assembleDebug

## CI 集成

### GitHub Action

我们可以使用 [github-action-setup-xmake](https://github.com/xmake-io/github-action-setup-xmake) 在 Github Action 上实现跨平台安装集成 Xmake。

uses: xmake-io/github-action-setup-xmake@v1

with:

xmake-version: latest

## 谁在使用 Xmake?

请点击 [用户列表](https://xmake.io/#/zh-cn/about/who_is_using_xmake) 查看完整用户使用列表。

如果您在使用 xmake，也欢迎通过 PR 将信息提交至上面的列表，让更多的用户了解有多少用户在使用 xmake，也能让用户更加安心使用 xmake。

我们也会有更多的动力去持续投入，让 xmake 项目和社区更加繁荣。

## 联系方式

* 邮箱：[waruqi@gmail.com](mailto:waruqi@gmail.com)
* 主页：[xmake.io](https://xmake.io/#/zh-cn/)
* 社区
  + [Reddit论坛](https://www.reddit.com/r/xmake/)
  + [Telegram群组](https://t.me/tbooxorg)
  + [Discord聊天室](https://discord.gg/xmake)
  + QQ群：343118190, 662147501
* 源码：[Github](https://github.com/xmake-io/xmake), [Gitee](https://gitee.com/tboox/xmake)
* 微信公众号：tboox-os

## 感谢

感谢所有对xmake有所[贡献](https://github.com/xmake-io/xmake/blob/master/CONTRIBUTING.md)的人:

* [TitanSnow](https://github.com/TitanSnow): 提供xmake [logo](https://github.com/TitanSnow/ts-xmake-logo) 和安装脚本。
* [uael](https://github.com/uael): 提供语义版本跨平台c库 [sv](https://github.com/uael/sv)。
* [OpportunityLiu](https://github.com/OpportunityLiu): 改进cuda构建, tests框架和ci。
* [xq144](https://github.com/xq114): 改进 xrepo env shell，并贡献大量包到 [xmake-repo](https://github.com/xmake-io/xmake-repo) 仓库。