# 精通cmake

1. 为什么用cmake

1.1 cmake的历史

1.2 为什么不使用Autoconf？

1.3 为什么不使用 JAM,qmake,SCons,或者ANT？

1.4 为什么不自己写脚本？

1.5 CMake能运行在什么平台上？

1. 开始

2.1 在你的电脑上(Unix和苹果系统，视窗系统)获取并安装CMake。

2.2 自己编译CMake

2.3 CMake基本用法和语法

2.4 Hello World

2.5 如何运行CMake？

使用Qt界面

使用ccmake界面

在命令行运行CMake

为CMake指定编译器

依赖检查

2.6 编辑CMakeLists文件

2.7 为CMake设定初始值

2.8 构建你的工程

1. 关键概念

3.1 主结构

3.2 目标

3.3 源文件

3.4 目录，生成器，测试和属性

3.5 变量和缓存实体

3.6 构建配置

1. 编写CMakeLists文件

4.1 CMake语法

4.2 基本命令

4.3 控制流

4.4 正则表达式

4.5 检查CMake版本

4.6 使用组件

使用SWIG组件

使用Qt组件

使用FLTK组件

4.7 策略

更新工程到新版本的CMake

4.8 链接程序库

通过某个目标的优化或Debug程序库

4.9 共享程序库和可加载组件

4.10共享程序库版本控制

4.11 安装文件

安装依赖共享库

4.12 高级命令

1. 系统检测

5.1 使用头文件和程序库

5.2 系统属性

5.3 查找包

5.4 内置组建查找

5.5 如何传递编译参数

5.6 如何配置头文件

5.7 生成CMake包配置文件

1. 定制命令及目标

6.1 可移植的定制命令

6.2 对目标使用add\_custom\_command

当一个可执行文件构建好以后如何拷贝？

6.3 使用add\_custom\_command创建文件

使用可执行文件构建源文件

6.4 添加定制目标

6.5 指定依赖和输出

6.6 当对于某个输出没有对应规则

单个命令产生多个输出

由不同命令产生相同输出

1. 转换现有构建系统到CMake

7.1 源文件目录结构

7.2 构建目录

7.3 使用CMake命令转换工程

7.4 转换UNIX Makefiles

7.5 转换基于Autoconf的工程

7.6 转换基于windows工程

1. 使用CMake交叉编译

8.1 工具链文件

查找外部程序库，程序，和其他文件

8.2 系统检测

使用编译检测

8.3 运行工程中的可执行文件

8.4 交叉编译Hello World

8.5 为微控制器交叉编译

8.6 对已存在工程交叉编译

8.7 对复杂工程交叉编译-VTK

8.8 提示及小技巧

1. 使用CPACK打包

9.1 CPack基础

当包含CPack.cmake时，发生了什么？

添加定制的CPack选项

使用CPack添加选项

9.2 CPack源码包

9.3 CPack安装命令

9.4 视窗系统的CPack安装器NSIS

CMake为NSIS使用CPack变量

在开始菜单创建视窗系统快捷方式

高级的NSIS CPack选项

设置文件扩展关联到NSIS

安装微软运行时库

CPack组件安装支持

9.5 Cygwin中的CPack安装

9.6 Mac OSX PackageMaker下的CPack

9.7 Mac OSX拖拽下的CPack

9.8 MAX OSX X11 Applications下的CPack

9.9 Debian下的CPack包

9.10 CPack RPM包

9.11 CPack文件

1. 使用CMake进行自动化测试

10.1 使用CMake，CTest和CDash测试

10.2 CMake如何帮助测试

10.3 额外的测试属性项

10.4 使用CTest测试

10.5 使用CTest驱动复杂测试

10.6 处理大量测试

10.7 产生测试主控面板

为工程添加测试主控面板

安装设置客户端

10.8 为工程定制主控面板

主控面板的子任务设置

过滤错误和警告

为主控面板添加注释

10.9 设置自动的主控面板客户端

设置持续的主控面板

CTest脚本中的变量

10.10 高级CTest脚本

传统CTest脚本中的限制

扩展的CTest脚本

10.11 设置主控面板服务器

CDash服务器

高级服务器管理

构建组

电子邮件

地址

图表

对构建添加注释

日志

测试时机

移动支持

备份CDash

升级CDash

CDash维护

10.12 子工程

使用ctest\_submit提交PARTS和FILES

将你的工程划分为若干子工程

1. 移植CMake工程到新平台和新语言

11.1 确定系统

11.2 设置语言支持

11.3 移植到新平台

11.4 添加新的语言

11.5 管理变量列表

生成标签变量

语言特定信息

11.6 编译器和平台的例子

Como编译器

Borland编译器

11.7 扩展CMake

创建加载的命令

使用加载的命令

1. 教程

12.1 基础（第一步）

添加版本号和配置头文件

12.2 添加程序库（第二步）

12.3 安装和测试（第三步）

12.4 添加系统内省（第四步）

12.5 添加生成的文件和生成器（第五步）

12.6 构建安装器（第六步）

12.7 添加主控面板的支持（第七步）

附件A 变量

改变行为的变量

描述系统的变量

关于语言的变量

控制构建的变量

提供信息的变量

附件B 命令行相关

CMake命令行选项

CMake生成器

CTest命令行选项

CPack命令行选项

CPack生成器

附件C列出文件命令

当前命令

兼容命令

附件D挑选的组件

CMake组件

附件E属性

全局属性项

目录属性项

目标属性项

测试属性项

源文件属性项

缓存属性

附件F策略

# 第一章

Why CMake?

为什么使用CMake？

If you have ever maintained the build and installation process for a software package, you will be interested in CMake. CMake is an open source build manager for software projects that allows developers to specify build parameters in a simple portable text ﬁle format. This ﬁle is then used by CMake to generate project ﬁles for native build tools including Integrated Development Environments such as Microsoft Visual Studio or Apple’s Xcode, as well as UNIX, Linux, NMake, and Borland style Makeﬁles. CMake handles the difﬁcult aspects of building software such as cross platform builds, system introspection, and user customized builds, in a simple manner that allows users to easily tailor builds for complex hardware and software systems.

如果你曾经维护过软件包的构建和安装过程，你就会对CMake感兴趣。CMake是一个开源的软件工程的构建管理工具，它允许开发者在一个可移植的文本格式的文件中指定构建参数。CMake使用这个文件生成使用开发环境中集成的本地构建工具的工程文件，比如微软的Visual Studio或者苹果的Xcode，在UNIX，Linux系统中，生成NMake，Borland风格的Makefile文件。CMake处理构建软件中复杂的部分比如（跨平台构建 （交叉编译）），系统内省，和定制构建，只用简单的操作就可以方便用户剪裁对应复杂硬件和软件系统的构建。

For any project, and especially cross platform projects, there is a need for a uniﬁed build system. Many projects today ship with both a UNIX Makeﬁle (or Makeﬁle.in) and a Microsoﬁ Visual Studio workspace. This requires that developers constantly try to keep both build systems up to date and consistent with each other. To target additional build systems such as Borland or Xcode requires even more custom copies of these ﬁles, creating an even bigger problem. This problem is compounded if you try to support optional components, such as including JPEG support if libjpeg is available on the system. CMake solves this by consolidating these different operations into one simple easy to understand ﬁle format.

对于任意工程，特别是跨平台工程，需要一个统一的构建系统。现在，许多工程都提供UNIX Makefile和微软Visual Studio构建环境。这就需要开发人员持续维护更新两套构建系统并使它们保持一致。如果要添加对应Borland或者Xcode的构建系统，就需要修改更多的文件，这将会产生更多的问题。如果要支持可选的组件，比如如果系统有libjpeg库，就包含JPEG，这将会使得问题变得更加复杂。CMake通过将这些不同的操作编写到一个易于理解的文件中解决这个问题。