# 编译器工具调研和介绍

——2053171白钰

总页数：10

参考文献：

1. <https://blog.csdn.net/kai8wei/article/details/44774175（编译器的工作原理详解）>
2. <https://blog.csdn.net/fuhanghang/article/details/124116080（GCC详解）>
3. <https://blog.csdn.net/m0_37698164/article/details/127795370（gcc与g++区别）>
4. <https://blog.csdn.net/python2021_/article/details/124142864（11款最佳python编译器）>
5. <https://blog.csdn.net/shuirenbushijun/article/details/107835006（brython）>
6. <https://blog.csdn.net/qq_45503700/article/details/108331349(cpython)>
7. <https://blog.csdn.net/weixin_45370502/article/details/124561540（java编译器）>
8. <https://blog.csdn.net/vip0809/article/details/116570961(openjdk)>

目录

[编译器工具调研和介绍 1](#_Toc4302)

[一． c/c++编译器 2](#_Toc14941)

[1. GCC 2](#_Toc16032)

[1.1. GCC,gcc与g++的区别 3](#_Toc17720)

[二． Python编译器 4](#_Toc29885)

[1. Brython 4](#_Toc25587)

[2. CPython 6](#_Toc1522)

[三． Java编译器 7](#_Toc22085)

[1.整体介绍 7](#_Toc29383)

[3. Openjdk 8](#_Toc9086)

在编译原理这门课程中，我们学习了编译器的工作原理，即：

1. 词法分析(Lexical Analysis)。  
   (2)语法分析 (Syntax Analysis)。

(3)语义分析 (Semantic Analysis)

(4)中间代码生成。

(5)优化。

(6)代码生成。

在这些步骤进行完之后，再交由链接器把外部模块链接上去，之后程序执行。在之前的项目开发经验中，我们已经接触到了许多编译器，其中性能有好有坏，下面是对于各种编译器的调研和介绍。

## c/c++编译器

### GCC

GCC是一套由GNU开发的编程语言编译器，在Linux系统下可以用它调用其他不同的工具进行**预处理、编译、汇编、链接**这样的工作，其执行效率比一般编译器高20%~30%。由于它是GNU项目之一，是开源的软件，我们可以直接从网上免费地下载安装它。

编译的过程如下：

**1.预处理阶段：**

GCC在第一个阶段会调用**预处理器cpp**来对C源程序进行[预处理](https://so.csdn.net/so/search?q=%E9%A2%84%E5%A4%84%E7%90%86&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/fuhanghang/article/details/_blank)，所谓的预处理就是**解释源程序当中的所有的预处理指令**，那些诸如#include、#define、#if等以井号#开头的语句就是预处理指令，**预处理指令实际上并不是语言本身的组成部分，而是为了更好地组织程序所使用的一些"预先处理的"工作，这些工作用一种叫做预处理指令的语句来描述，然后用预处理器来解释，**这些工作包括我们熟悉的诸如文件包含、宏定义、条件编译等等。

如：

gcc hello.c -o hello.i -E // -o 执行输出的文件名

加上一个编译选项 -E 就可以使得 GCC 在进行完第一阶段的预处理之后停下来，生成一个默认后缀名为.i 的**文本文件**。

**2.编辑阶段：**

经过预处理之后生成的.i 文件依然是一个文本文件，不能被处理器直接解释，我们需要进一步的翻译。接下来的编译阶段是四个阶段中最为复杂的阶段，它包括**词法和语法的分析，最终生成对应硬件平台的汇编语言**（不同的处理器有不同的汇编格式），具体生成什么平台的汇编文件取决于所采用的[编译器](https://so.csdn.net/so/search?q=%E7%BC%96%E8%AF%91%E5%99%A8&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/fuhanghang/article/details/_blank)，如果用的是 GCC，那么将会生成 x86 格式的汇编文件，如果用的是针对 ARM 平台的交叉编译器，那么将会生成 ARM 格式的汇编文件。

gcc hello.i -o hello.s -S

再加上一个编译选项 -S 就可以使得 gcc 在进行完第一和第二阶段之后停下来，生成一个默认后缀名为.s 的文本文件。打开此文件看一看，你会发现这是一个**符合 x86 汇编语言的源程序文件**。

**3.汇编阶段**

接下来的步骤相对而言比较简单，编译器 gcc 将会调用汇编器 as 将汇编源程序翻译成为**可重定位文件**。汇编指令跟处理器直接运行的二进制指令流之间基本是一一对应的关系，该阶段只需要将.s 文件里面的汇编翻译成指令即可。

gcc hello.s -o hello.o -c

只要在编译的时候加上一个编译选项-c，则会生成一个扩展名为.o 的文件，这个文件是一个 ELF 格式的可重定位(relocatable)文件。程序中的所有的全局符号尚未定位，所谓的全局符号，就是指函数和全局变量，函数和全局变量默认情况下是可以被外部文件引用的。

由于定义和调用可以出现在**不同的文件当中**，因此他们在编译的过程中需要**确定其入口地址**，比如 a.c 文件里面定义了一个函数 func( )，b.c 文件里面调用了该函数，那么在完成第三阶段汇编之后，b.o 文件里面的函数 func( )的地址将是 0，显然这是不能运行的，必须要找到 a.c 文件里面函数 func( )的确切的入口地址，然后将 b.c 中的“全局符号”func重新定位为这个地址，程序才能正确运行。

**4.链接阶段**

如前面所述，经过汇编之后的可重定位文件不能直接运行，因为还有两个很重要的工作没完成，**首先是重定位，其次是合并相同权限的段。**关于重定位的问题，上面已经给出了简单的描述。一般地，我们编译一个程序通常都需要链接系统的标准 C 库、gcc 内置库等基本库文件。因为 Linux 下任何一个程序编译都需要用到这些基本库的全局符号。

gcc hello.o -o hello -lc -lgcc

标准 C 库 和 gcc 内置库是如此的基本，因此-lc 和-lgcc 是默认的，可以省略。

#### GCC,gcc与g++的区别

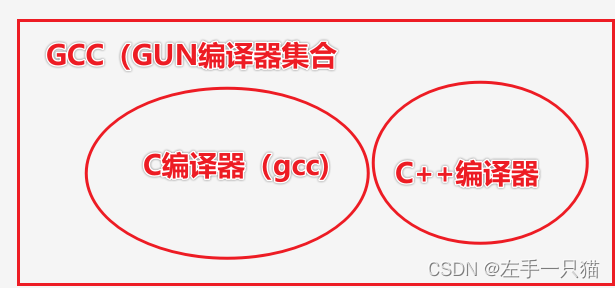
一些名词解释如下：

GCC:GNU Compiler Collection(GUN 编译器集合)，它可以编译C、C++、JAV、Fortran、Pascal、Object-C等语言。

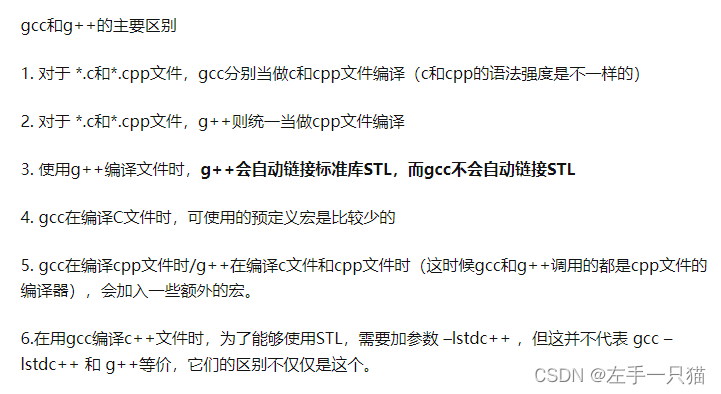
gcc: GCC中的GUN C Compiler（C 编译器）

g++: GCC中的GUN C++ Compiler（C++编译器）

三者的关系如图所示：



其中：gcc与g++的主要区别如下：



## Python编译器

### Brython

**Brython的目标是用Python取代Javascript，作为Web浏览器的脚本语言。那么**它作为Python的客户端Web编程工具，和JS有什么区别呢？

1.可轻易地在页面中内嵌Python终端进行测试

2.运行速度接近于CPyhon

3.写法方便，社区强大，可进行[敏捷开发](https://so.csdn.net/so/search?q=%E6%95%8F%E6%8D%B7%E5%BC%80%E5%8F%91&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/shuirenbushijun/article/details/_blank)，相同的功能，用Python写起来可能会比JS快。

4.和JS一样，你不用安装任何东西就可以开始编写

**一个简单的例子：**

**<html>**

**<head>**

**<script src="/brython.js"></script>**

**</head>**

**<body onload="brython()">**

**<script type="text/python">**

**from browser import document, alert**

**# 给按钮绑定事件‘click’到方法‘echo’上**

**def echo(ev):**

**alert(document["zone"].value)**

**document["mybutton"].bind("click", echo)**

**</script>**

**<input id="zone">**

**<button id="mybutton">click !</button>**

**</body>**

**</html>**

为了处理Python脚本，必须包含 brython.js并brython()在页面加载时运行该函数（使用标记的onload属性<BODY>）。在开发阶段，可以将参数传递给brython()函数：1将错误消息显示到Web浏览器控制台，2也可以显示与错误一起显示的Javascript代码。

如果Python程序很大，另一个选择是将它写在一个单独的文件中，并使用脚本标记的src属性加载它：

<html>

<head>

<script src =“/ brython.js”> </ script>

</head>

<body onload =“brython（）”>

<script type =“text / python”src =“test.py”> </ script>

<input id =“zone”autocomplete =“off”>

<button id =“mybutton”> click！</ button>

</body>

</html>

请注意，在这种情况下，Python脚本将通过Ajax调用加载：它必须与HTML页面位于同一个域中。

在上面两个代码示例中，当我们单击按钮时，onclick事件会调用并运行echo()在Python脚本中定义的函数。此函数通过其id（区域）获取INPUT元素的值。这是通过以下语法实现的document["zone"]：document在模块浏览器中定义，是表示当前在浏览器中显示的文档的对象。它的行为类似于字典，其键是DOM元素的ID。因此，在我们的示例中， document["zone"]是一个映射到INPUT元素的对象; 该值 属性保存，有趣的是，对象的值。

在Brython中，输出可以通过各种方式完成，包括使用函数alert()（也在浏览器中定义），该函数显示弹出窗口，文本作为参数传递。

### CPython

CPython 是默认的且使用最广泛的 Python 编译器。它是用 C 语言编写的，并使用 GIL（全局解释器锁），这使得并发 CPython 进程之间的通信很困难。CPython 中的编译步骤包括：解码、令牌化、解析、抽象语法树和编译。

当我们从Python官方网站下载并安装好Python 3.5后，我们就直接获得了一个官方版本的解释器：CPython。这个解释器是用C语言开发的，所以叫CPython。在命令行下运行python就是启动CPython解释器。CPython是使用最广的Python解释器。教程的所有代码也都在CPython下执行。之所以使用CPython这个词，是因为Python还有一些其它的实现，比如Jython，就是Java版的Python，还有烧脑的PyPy，使用Python再把Python实现了一遍。

例：

创建pyx文件：

hello.pyx

import time

t0 = time.time()

for i in range(100000):

pass

print("time is {}".format(time.time()-t0))

setup.py文件

from setuptools import setup

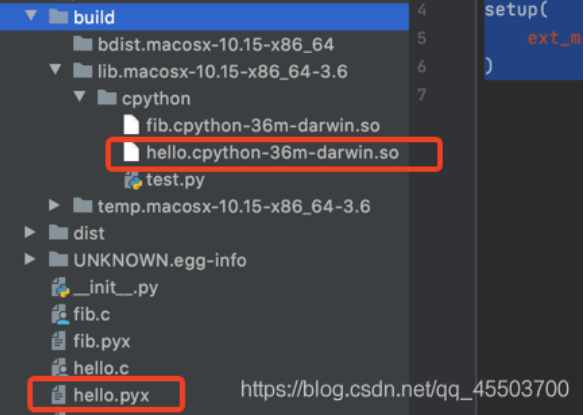
from Cython.Build import cythonize

setup(

ext\_modules=cythonize("hello.pyx")

)

命令行输入python setup.py build\_ext --inplace对hello.pyx文件进行编译将会生成hello.的so动态库，我们调用此so动态库即可。如图所示：



test.py结果显示：

cpython 输出： time is 0.002833843231201172

我们用python同样输出： time is 0.005338907241821289

由此可见，使用cpython 速度提高了一半左右，要是将大量的计算方入编译，速度会有较高的提升。

## Java编译器

### 1.整体介绍

Java编译器分为2种：

A.解析器 (bytecode intepreter)

B.JIT (just in time compiler)

解释执行：解释执行是采用匹配执行解释器（解释器是个黑盒，通常也有编译器的组成部分）内部已经编译好的机器码，不是生成新的机器码（也有说法是逐条翻译成机器码？）。 - 由于逐条翻译，程序启动快，但是执行效率不高。

编译执行：运行期间，通过将字节码编译成对应的新的机器码（会将其缓存起来，通过参数-XX:ReservedCodeCacheSize），然后执行。 需要先编译出新的机器指令，所以程序启动较慢，但是执行效率高（因为执行的是机器指令）。

混合模式:现在主流的商用虚拟机（HotSpot（Oracle）、J9 VM（IBM））中几乎都同时包含解释器和编译器。二者在其中各有优势：当程序需要迅速启动和执行时，解释器可以首先发挥作用，省去编译的时间，立即执行；当程序运行后，随着时间的推移，编译器逐渐会返回作用，把越来越多的代码编译成本地代码后，可以获取更高的执行效率。解释执行可以节约内存，而编译执行可以提升效率。

在 Java7 之前，需要根据程序的特性来选择对应的 JIT，虚拟机默认采用解释器和其中一个编译器配合工作。HotSpot 虚拟机会根据自身版本与计算机的硬件性能自动选择运行模式，用户也可以使用 -client 和 -server 参数强制指定虚拟机运行在 Client 模式或者 Server 模式。这种配合使用的方式称为\*\*“混合模式”（Mixed Mode）\*\*，用户可以使用参数 -Xint 强制虚拟机运行于 “解释模式”（Interpreted Mode），这时候编译器完全不介入工作。另外，使用 -Xcomp 强制虚拟机运行于 “编译模式”（Compiled Mode），这时候将优先采用编译方式执行，但是解释器仍然要在编译无法进行的情况下接入执行过程。通过虚拟机 -version 命令可以查看当前默认的运行模式。

### Openjdk

#### 概述

OpenJDK 是 Java 平台标准版 (Java SE) 的免费开源实现。这是 Sun Microsystems (以下简称：Sun) 于2006年开始努力的结果。该实现已获得 GNU通用公共许可证（GNU GPL）版本2的许可。但有链接例外。除 GPL 特例链接之外，链接到 Java 类库的组件将受到 GPL 许可条款的约束。

OpenJDK Project 产生了许多组件：最重要的是虚拟机( HotSpot )，Java类库和Java编译器( javac )。

从 Java SE 7 开始，OpenJDK Project 成为了 Java SE 的官方参考实现。从 Java SE 10 开始，JDK Project（ OpenJDK Community 的下属项目） 成为了 Java SE 的官方参考实现。

OpenJDK 是由 OpenJDK Community 、Oracle、IBM 领导，连同 Alibaba，Amazon，Ampere，Azul，BellSoft，Canonical，Fujitsu，Google，Huawei，Intel，Java Community，JetBrains，London Java Community，Microsoft，Red Hat，SAP，SouJava，SUSE，Tencent，Twitter ，VMWare 等第三方共同开发、维护的 Java SE 开源参考实现。

OpenJDK 具体版本的开发标准是 Java Community Process（JCP，Java 社区进程） 发布的 Java Specification Requests（JSR，Java规范请求）。

OpenJDK Community 领导的 OpenJDK Project 是 Java SE 的官方参考实现，只产生 OpenJDK 源码，并不提供可以直接使用的二进制文件格式。现在能直接使用的二进制文件格式的 JDK 都是被编译之后的程序。 OpenJDK 官网指向的可下载二进制文件的地址，实际是 Oracle’s OpenJDK builds 下载的地址。

#### 使用原理

OpenJDK的工作原理和优点是什么？

Java诞生时形成了几个目标，包括：Java很简单，面向对象，大家都很熟悉;Java是健壮和安全的;Java语言与体系结构无关，JVM是可移植的;Java能够以高性能执行。

这些原则仍然是OpenJDK流行的原因，因此探索其中每一个原则并了解它们如何影响开发人员和最终用户的生活是很重要的。作为一个开源项目，OpenJDK的代码可供任何想要检查或增强其安全性的人使用。这意味着新发现的漏洞将不断通过每季度的CPU版本进行修补，对于Liberica JDK，我们的支持人员将修复这些漏洞。而且，有了广泛的安全工具选择，如漏洞扫描器，可以确保破坏应用程序和基础设施安全的尝试不会成功。

有很多方法可以提升OpenJDK应用程序。JVM中的编译器优化代码以获得最佳性能。如果启动时间更重要，则可以使用GraalVM本机映像技术，将应用程序与运行时打包到单个可执行二进制文件中。

**其中，OpenJDK的开发可以分为几个方向：**

* Bug修复；
* 新功能的实施和过时功能的弃用；
* 主要分支机构以外的项目。

对于错误修复来说，今天，当GitHub用于OpenJDK开发时，这个过程大部分是自动化的，比以前快了很多。为了报告新发现的错误，开发人员使用邮件列表来描述错误，然后给出一个数字。当开发人员希望修复bug时，他们会在存储库中创建一个fork，它基本上是整个项目的副本。然后在这个fork中，开发人员编辑代码的必要部分来修复bug。测试成功后，他们会发出将新代码拉回主分支的请求。然后对代码进行审查，如果符合要求，则将其实现到OpenJDK下一版本的主分支中。该系统允许任何开发人员尝试修复任何发现的bug。

对于新功能的实施和过时功能的弃用来说，新特性通常通过几个步骤来实现。首先，它是由开发人员作为概念（草案）提出的。名为OpenJDK lead的管理委员会成员决定该功能是否值得开发。批准后，草案更新了必要的说明和基本实施建议，成为JEP（JDK增强建议）。它在完成之前经历了许多阶段。实现过程本身类似于前面描述的bug修复，是通过GitHub上的fork分支完成的。由于新功能分阶段发布，此过程可能需要很长时间才能完成。预览涉及语义和语法，孵化器一 API更改。在该功能的最终实现发布之前，可能会有多达3个预览或孵化器。这样做是为了让开发人员能够测试新功能，并查看它与新代码和现有代码的配合情况，以免破坏已经在工作的应用程序。通常在两次预览或孵化器之后，新代码要么以最终形式添加到版本中，要么在被认为不安全或性能不佳时删除。

对于主要分支机构以外的项目来说，一些对Java功能影响很大的项目是在主分支之外开发的。如果成功的话，他们会彻底改变语言，但他们需要很多时间来完善。如果需要，它们仍然可以在当前状态下实现到OpenJDK中，一些开发人员通常会这样做。随着时间的推移，它们将成为JEP，并成为下一版本的一部分。