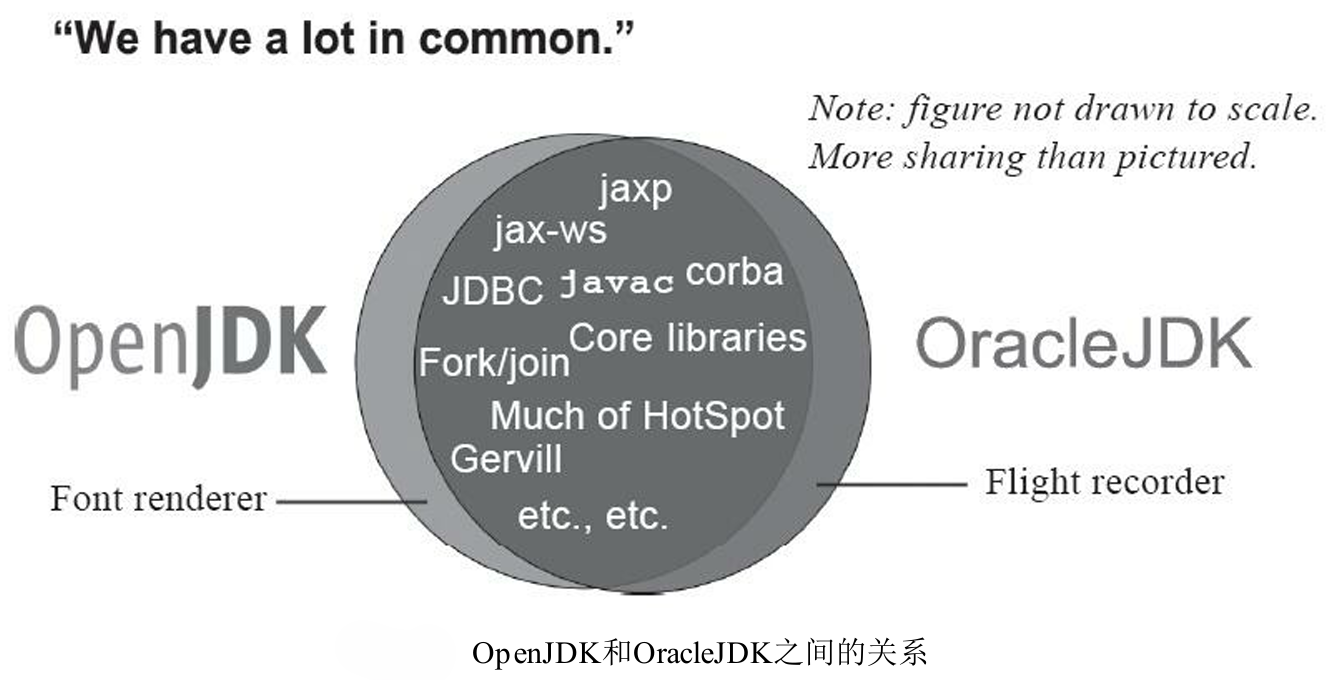
# 获取源码

编译源码之前，我们要先明确OpenJDK和OracleJDK之间，以及OpenJDK的各个不同版本之间存在什么联系。

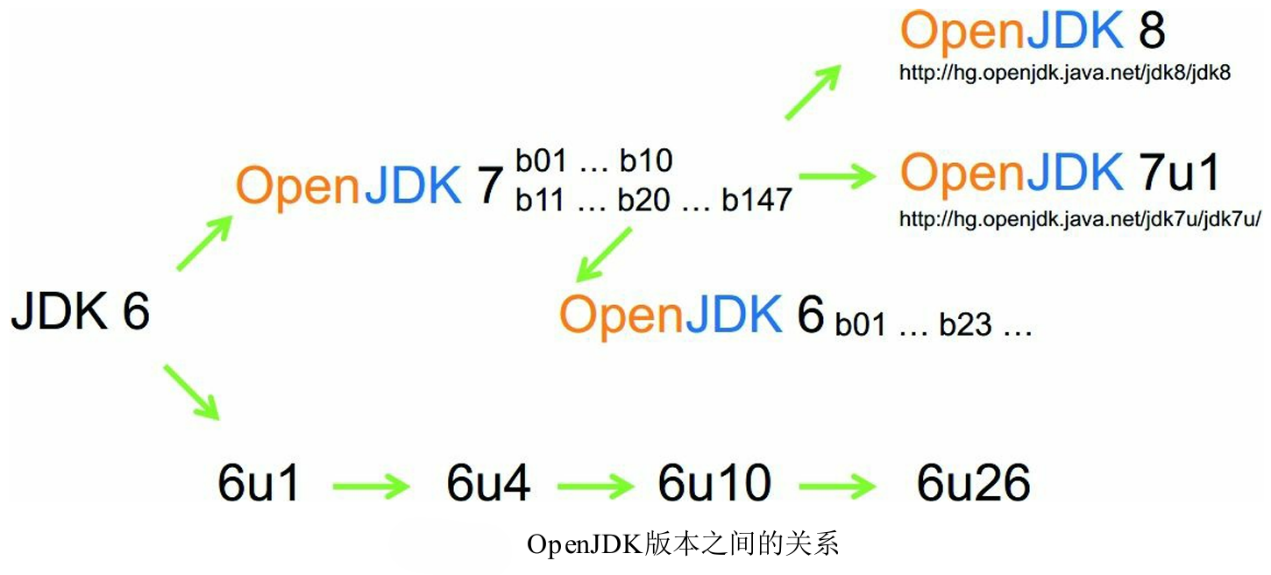
OpenJDK是Sun公司在2006年年末把Java开源而形成的项目，这里的“开源”是通常意义上的源码开放形式，即源码是可被复用的，例如OracleJDK、Oracle OpenJDK、AdoptOpenJDK、Azul Zulu、SAP SapMachine、Amazon Corretto、IcedTea、UltraViolet等都是从OpenJDK源码衍生出的发行版。但如果仅从“开源”字面意义（开放可阅读的源码）上讲的话，其实Sun公司自JDK 5时代起就曾经以JRL（Java Research License）的形式公开过Java的源码，主要是开放给研究人员阅读使用，这种JRL许可证的开放源码一直持续到JDK 6 Update 23才因OpenJDK项目日渐成熟而终止。如果拿OpenJDK中的源码跟对应版本的JRL许可证形式开放的Sun/OracleJDK源码互相比较的话，会发现除了文件头的版权注释之外，其余代码几乎都是相同的，只有少量涉及引用第三方的代码存在差异，如字体栅格化渲染，这部分内容OracleJDK采用了商业实现，源码版权不属于Oracle自己，所以也无权开源，而OpenJDK中使用的是同样开源的FreeType代替。

OpenJDK中的源码仓库只包含了标准Java SE的源代码，而一些额外的模块，典型的如JavaFX，虽然后来也是被Oracle开源并放到OpenJDK组织进行管理（OpenJFX项目），但是它是存放在独立的源码仓库中，因此OracleJDK的安装包中会包含JavaFX这种独立的模块，而用OpenJDK的话则需要单独下载安装。

此外，在JDK 11以前，OracleJDK中还会存在一些OpenJDK没有的、闭源的功能，即OracleJDK的“商业特性”。例如JDK 8起从JRockit移植改造而来的Java Flight Recorder和Java Mission Control组件、JDK 10中的应用类型共享功能（AppCDS）和JDK 11中的ZGC收集器，这些功能在JDK 11时才全部开源到了OpenJDK中。到了这个阶段，我们已经可以认为OpenJDK与OracleJDK代码实质上已达到完全一致的程度。



下OpenJDK内部不同版本之间的关系，在OpenJDK接收Sun公司移交的JDK源码时，Java正处于JDK 6时代的初期，JDK 6 Update 1才刚刚发布不久，JDK 7则还完全处于研发状态的半成品。OpenJDK的第一个版本就是来自于当时Sun公司正在开发的JDK 7，考虑到OpenJDK 7的状况在当时完全不足以支持实际的生产部署，因此又在OpenJDK 7 Build 22的基础上建立了一条新的OpenJDK 6分支，剥离掉所有JDK 7新功能的代码，形成一个可以通过TCK 6测试的独立分支，先把OpenJDK 6发布出去给公众使用。等到OpenJDK 7达到了可正式对外发布的状态之后，就从OpenJDK 7的主分支延伸出用于研发下一代Java版本的OpenJDK 8以及用于发布更新补丁的OpenJDK 7 Update两条子分支，按照开发习惯，新的功能或Bug修复通常是在最新分支上进行的，当功能或修复在最新分支上稳定之后会同步到其他老版本的维护分支上。后续的JDK 8和JDK 9都重复延续着类似的研发流程。



获取OpenJDK源码：[jdk/jdk12: 06222165c35f / (openjdk.org)](https://hg.openjdk.org/jdk/jdk12/file/06222165c35f)

# 系统需求

尽量在Linux或者MacOS上构建OpenJDK，这两个系统在准备构建工具链和依赖项上要比在Windows或Solaris平台上要容易许多。

在本次编译中采用的是64位操作系统，默认参数下编译出来的也是64位的OpenJDK，如果需要编译32位版本（推荐在64位的操作系统上进行，理由是编译过程可以使用更大内存，受4G内存限制），通过编译参数（--with-target-bits=32）来指定需要生成32位编译结果即可。

在官方文档上要求编译OpenJDK至少需要2～4GB的内存空间（CPU核心数越多，需要的内存越大），而且至少要6～8GB的空闲磁盘空间，不要看OpenJDK源码的大小只有不到600MB，要完成编译，过程中会产生大量的中间文件，并且编译出不同优化级别（Product、FastDebug、SlowDebug）的HotSpot虚拟机可能要重复生成这些中间文件，这都会占用大量磁盘空间。

# 构建编译环境

在MacOS和Linux上构建OpenJDK编译环境相对简单，对于MacOS，需要MacOS X 10.13版本以上，并安装好最新版本的XCode和Command Line Tools for XCode（在Apple Developer网站上可以免费下载），这两个SDK提供了OpenJDK所需的CLang编译器以及Makefile中用到的其他外部命令。

对于Linux系统，要准备的依赖与MacOS类似，在MacOS中CLang编译器来源于XCode SDK，而Ubuntu里用户可以自行选择安装GCC或CLang来进行编译，但必须确保最低的版本为GCC 4.8或者CLang 3.2以上，官方推荐使用GCC 7.8或者CLang 9.1来完成编译。在Ubuntu系统上安装GCC的命令为：

|  |
| --- |
| sudo apt-get install build-essential |

在编译过程中需要依赖FreeType、CUPS等若干第三方库：

|  |
| --- |
| sudo apt-get install libfreetype6-dev  sudo apt-get install libcups2-dev  sudo apt-get install libx11-dev libxext-dev libxrender-dev libxrandr-dev libxtst-dev libxt-dev  sudo apt-get install libasound2-dev  sudo apt-get install libffi-dev  sudo apt-get install autoconf  sudo apt-get install libfontconfig1-dev |

最后，假设要编译大版本号为N的JDK，我们还要另外准备一个大版本号至少为N-1的、已经编译好的JDK，这是因为OpenJDK由多个部分（HotSpot、JDK类库、JAXWS、JAXP……）构成，其中一部分（HotSpot）代码使用C、C++编写，而更多的代码则是使用Java语言来实现，因此编译这些Java代码就需要用到另一个编译期可用的JDK，官方称这个JDK为“Bootstrap JDK”。编译OpenJDK 12时，Bootstrap JDK必须使用JDK 11及之后的版本。在Ubuntu中使用以下命令安装OpenJDK 11：

|  |
| --- |
| sudo apt-get install openjdk-11-jdk |

# 进行编译

按照默认配置来开始编译了，但通常编译OpenJDK的目的都不仅仅是为了得到在自己机器中诞生的编译成品，而是带着调试、定制化等需求，这样就必须了解OpenJDK提供的编译参数才行，这些参数可以使用“bash configure--help”命令查询：

|  |
| --- |
| --enable-debug：等效于--with-debug-level=fastdebug  --with-native-debug-symbols=：确定调试符号信息的编译方式，可选值为none、internal、external、zipped  --with-version-string=：设置编译JDK的版本号，譬如java-version的输出就会显示该信息。这个参数还有--with-version-=的形式，其中part可以是pre、opt、build、major、minor、security、patch之一，用于设置版本号的某一个部分。  --with-jvm-variants=[，...]：编译特定模式（Variants）的HotSpot虚拟机，可以多个模式并存，可选值为server、client、minimal、core、zero、custom。  --with-jvm-features=[，...]：针对--with-jvm-variants=custom时的自定义虚拟机特性列表（Features），可以多个特性并存，由于可选值较多，请参见help命令输出。  --with-target-bits=：指明要编译32位还是64位的Java虚拟机，在64位机器上也可以通过交叉编译生成32位的虚拟机。  --with-=：用于指明依赖包的具体路径，通常使用在安装了多个不同版本的Bootstrap JDK和依赖包的情况。其中lib的可选值包括boot-jd、freetype、cups、x、alsa、libffi、jtreg、libjpeg、giflib、libpng、lcms、zlib。  --with-extra-=：用于设定C、C++和Java代码编译时的额外编译器参数，其中flagtype可选值为cflags、cxxflags、ldflags，分别代表C、C++和Java代码的参数。  --with-conf-name=：指定编译配置名称，OpenJDK支持使用不同的配置进行编译，默认会根据编译的操作系统、指令集架构、调试级别自动生成一个配置名称，譬如“linux-x86\_64-server-release”，如果在这些信息都相同的情况下保存不同的编译参数配置，就需要使用这个参数来自定义配置名称。 |

参数使用方法：

|  |
| --- |
| bash configure [options] |

configure命令承担了依赖项检查、参数配置和构建输出目录结构等多项职责，如果编译过程中需 要的工具链或者依赖项有缺失，命令执行后将会得到明确的提示，并且给出该依赖的安装命令。

解压下载的openjdk12并进入：

|  |
| --- |
| tar -zxvf /development/jdk12-06222165c35f.tar.gz -C /development |

编译FastDebug版、仅含Server模式的HotSpot虚拟机，命令应为：

|  |
| --- |
| bash configure --enable-debug --with-jvm-variants=server |

执行过程中会提示有哪些错误，比如哪些依赖没安装

如果配置成功，提示：

|  |
| --- |
| A new configuration has been successfully created in  /development/jdk12-06222165c35f/build/linux-x86\_64-server-fastdebug  using configure arguments '--with-debug-level=fastdebug --with-jvm-variants=server'.  Configuration summary:  \* Debug level: fastdebug  \* HS debug level: fastdebug  \* JVM variants: server  \* JVM features: server: 'aot cds cmsgc compiler1 compiler2 dtrace epsilongc g1gc graal jfr jni-check jvmci jvmti management nmt parallelgc serialgc services shenandoahgc vm-structs zgc'  \* OpenJDK target: OS: linux, CPU architecture: x86, address length: 64  \* Version string: 12-internal+0-adhoc.root.jdk12-06222165c35f (12-internal)  Tools summary:  \* Boot JDK: openjdk version "11.0.24" 2024-07-16 OpenJDK Runtime Environment (build 11.0.24+8-post-Ubuntu-1ubuntu322.04) OpenJDK 64-Bit Server VM (build 11.0.24+8-post-Ubuntu-1ubuntu322.04, mixed mode, sharing) (at /usr/lib/jvm/java-11-openjdk-amd64)  \* Toolchain: gcc (GNU Compiler Collection)  \* C Compiler: Version 7.5.0 (at /usr/bin/gcc)  \* C++ Compiler: Version 7.5.0 (at /usr/bin/g++)  Build performance summary:  \* Cores to use: 2  \* Memory limit: 5878 MB |

在configure命令以及后面的make命令的执行过程中，会在“build/配置名称”目录下产生如下目录结构。不常使用C/C++的读者要特别注意，如果多次编译，或者目录结构成功产生后又再次修改了配置，必须先使用“make clean”和“make dist-clean”命令清理目录，才能确保新的配置生效。编译产生的目录结构以及用途如下所示：

|  |
| --- |
| buildtools/：用于生成、存放编译过程中用到的工具  hotspot/：HotSpot虚拟机编译的中间文件  images/：使用make \*-image产生的镜像存放在这里  jdk/：编译后产生的JDK就放在这里  support/：存放编译时产生的中间文件  test-results/：存放编译后的自动化测试结果  configure-support/：这三个目录是存放执行configure、make和test的临时文件 |

执行整个OpenJDK编译：

|  |
| --- |
| make images |

报错：

|  |
| --- |
| No rule to make target 'linux-x86\_64-server-fastdebug/make-support/vardeps/make/ModuleWrapper.gmk/java.base/ORDERED\_CFG\_VARIANTS.vardeps' |

解决：

|  |
| --- |
| 根据<https://bugs.gentoo.org/706012>的提示，  找到bug修复方式<https://hg.openjdk.org/jdk/jdk/rev/0743e1d49930>  修改make/common/MakeBase.gmk的代码即可 |

执行整个OpenJDK编译：

|  |
| --- |
| make images |

这里“images”是“product- images”编译目标（Target）的简写别名，这个目标的作用是编译出整个JDK镜像，也可以编译其他目标：

|  |
| --- |
| hotspot：只编译HotSpot虚拟机  hotspot-<variant>：只编译特定模式的HotSpot虚拟机  docs-image：产生JDK的文档镜像  test-image：产生JDK的测试镜像  all-images：相当于连续调用product、docs、test三个编译目标  bootcycle-images：编译两次JDK，其中第二次使用第一次的编译结果作为Bootstrap JDK  clean：清理make命令产生的临时文件  dist-clean：清理make和configure命令产生的临时文件 |

编译过程有点久，编译完成之后，进入OpenJDK源码的“build/配置名称/jdk”目录下就可以看到OpenJDK的完整编译结果了，把它复制到JAVA\_HOME目录，就可以作为一个完整的JDK来使用，如果没有人为设置过JDK开发版本的话，这个JDK的开发版本号里默认会带上编译的机器名。

