source:

https://blog.csdn.net/JavaMonsterr/article/details/125411423

当我们执行 java -version 命令时，通常会看到如下信息。

```
java version "1.8.0_201"
Java(TM) SE Runtime Environment (build 1.8.0_201-b09)
Java HotSpot(TM) 64-Bit Server VM (build 25.201-b09, mixed mode
```

当然，这是 oracle jdk 8u201 版本的输出结果。

如果我们是用的是 openJDK 构建出的 jdk 来看，它会是这样的。

```
openjdk version "1.8.0-internal"
OpenJDK Runtime Environment (build 1.8.0-internal-root_2022_06_08_12_25-b00)
OpenJDK 64-Bit Server VM (build 25.71-b00, mixed mode
```

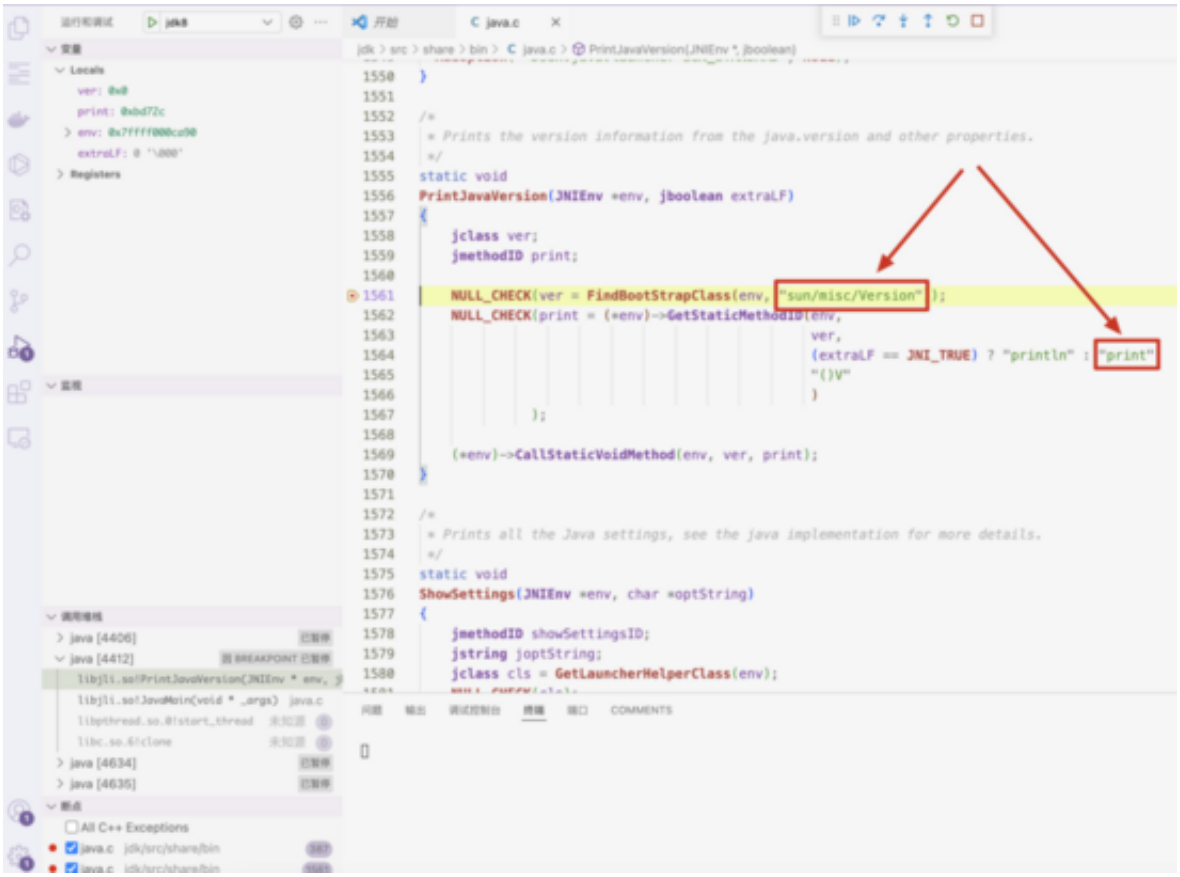
当然，这是在没有修改源码且没有增加多余 configure 参数构建出的结果。

今天我想改一改这个输出信息，让它成为我自己拥有的 jdk，应该怎么办呢?

首先，我们下载 openjdk 8u312 版本的源码，可以从 openjdk 官网下载，也可以从一个更开发者友好的 adoptopenjdk 网站下载。

https://github.com/AdoptOpenJDK/openjdk8-upstream-binaries

下载好源码后，我们一顿操作猛如虎，配置、编译、打断点、运行调试，进入了 java -version 命令所执行到的关键源码位置。



可以看出，它最终执行了 **sun/misc/Version** 类的 **print** 方法，我们打开这个类。

```
public class Version {
    private static final String launcher_name =
        "openjdk";
 private static final String java_version =
        "1.8.0-internal-debug";
    private static final String java_runtime_name =
        "OpenJDK Runtime Environment";
    private static final String java_profile_name =
        "";
    private static final String java_runtime_version =
        "1.8.0-internal-debug-root_2022_06_09_16_41-b00";

    public static void print() {
        print(System.err);
    }

    public static void print(PrintStream ps) {
      ...
        /* First line: platform version. */
        ps.println(launcher_name + " version \"" + java_version + "\"");
  ...
```

```
        /* Second line: runtime version (ie, libraries). */
        ps.print(java_runtime_name + " (build " + java_runtime_version);
  ...
        /* Third line: JVM information. */
        String java_vm_name    = System.getProperty("java.vm.name");
        String java_vm_version = System.getProperty("java.vm.version");
        String java_vm_info    = System.getProperty("java.vm.info");
        ps.println(java_vm_name + " (build " + java_vm_version + ", " +
java_vm_info + ")");
    }

```

这是段 Java 代码，所以非常好理解，print 函数里面就是简单粗暴地输出了三行文字。

而这些输出信息，是通过一个个变量拼接的，变量的定义就在上方，比如 **launcher_name** ， **java_version** 等等。

那看来我们只需要修改这个文件的 print 函数，或者修改上面定义的变量的值，就可以做到修改输出信息了。

但是呢，这个 Version 文件并不在 openJDK 源码里，而在 openJDK 源码构建出的产物 build 文件夹里。

也就是说，这个 Version 文件是构建过程中生成的，想要修改这个 Version 文件，就得找到这个 Version 文件是怎么构造出来的。

实际上，Version 文件是由 **Version.java.template** 文件生成的，这个文件是在源码里。

```
// jdk/src/share/classes/sun/misc/Version.java.template
public class Version {
    private static final String launcher_name =
        "@@launcher_name@@";
    private static final String java_version =
        "@@java_version@@";
    private static final String java_runtime_name =
        "@@java_runtime_name@@";
    private static final String java_profile_name =
        "@@java_profile_name@@";
    private static final String java_runtime_version =
        "@@java_runtime_version@@";

    public static void print(PrintStream ps) {
        ...
```

```
        /* First line: platform version. */
        ps.println(launcher_name + " version \"" + java_version + "\"");
        ...
        /* Second line: runtime version (ie, libraries). */
        ps.print(java_runtime_name + " (build " + java_runtime_version);
        ...
        /* Third line: JVM information. */
        String java_vm_name    = System.getProperty("java.vm.name");
        String java_vm_version = System.getProperty("java.vm.version");
        String java_vm_info    = System.getProperty("java.vm.info");
        ps.println(java_vm_name + " (build " + java_vm_version + ", " +
                   java_vm_info + ")");
    }


```

可以看出，这个文件是个模板文件，与刚刚生成的 Version 文件只差了上面那些常量的值，是通过 **@@XXX@@** 这种占位符替换做到的。

那我们接下来就应该寻找，这些占位符变量的值，分别是什么，就知道应该如何修改它了。

探索后发现，Version.java.template 这个文件里的 @@XXX@@ 占位符，会由 GensrcMisc.gmk 文件进行替换。

```
// jdk/make/gensrc/GensrcMisc.gmk
################################################################################
#
# Install the launcher name, release version string, full version
# string and the runtime name into the Version.java file.
# To be printed by java -version

$(JDK_OUTPUTDIR)/gensrc/sun/misc/Version.java \
$(PROFILE_VERSION_JAVA_TARGETS): \
    $(JDK_TOPDIR)/src/share/classes/sun/misc/Version.java.template
 $(MKDIR) -p $(@D)
 $(RM) $@ $@.tmp
 $(ECHO) Generating sun/misc/Version.java $(call profile_version_name, $@)
 $(SED) -e 's/@@launcher_name@@/$(LAUNCHER_NAME)/g' \
     -e 's/@@java_version@@/$(RELEASE)/g' \
     -e 's/@@java_runtime_version@@/$(FULL_VERSION)/g' \
     -e 's/@@java_runtime_name@@/$(RUNTIME_NAME)/g' \
     -e 's/@@java_profile_name@@/$(call profile_version_name, $@)/g' \
     $< > $@.tmp
 $(MV) $@.tmp $@
```

可以看出，这里用了 shell 脚本中的 **sed** 命令做占位符的替换，比如，将 @@java_version@@ 的值替换为 **$(RELEASE)** 。

很可惜，$(RELEASE) 也是个 shell 脚本的变量，我们还得寻找这部分变量的来源。

再经过一番探索，我们发现，这些变量会在 **spec.gmk.in** 以及 **version-numbers** 中定义，比如

```
// common/autoconf/spec.gmk.in
JDK_VERSION:=@JDK_VERSION@
BUILD_VARIANT_RELEASE:=@BUILD_VARIANT_RELEASE@
MILESTONE:=@MILESTONE@

ifeq ($(MILESTONE), fcs)
  RELEASE=$(JDK_VERSION)$(BUILD_VARIANT_RELEASE)
else
  RELEASE=$(JDK_VERSION)-$(MILESTONE)$(BUILD_VARIANT_RELEASE)
endif

ifneq ($(USER_RELEASE_SUFFIX), )
  FULL_VERSION=$(RELEASE)-$(USER_RELEASE_SUFFIX)-$(JDK_BUILD_NUMBER)
else
  FULL_VERSION=$(RELEASE)-$(JDK_BUILD_NUMBER)
endif
JRE_RELEASE_VERSION:=$(FULL_VERSION

```

很是可惜，这里面仍然是变量替换，我们继续往下寻找。

拿 **JDK_VERSION** 这个变量来说，找到它在 **generated-configure.sh** 中有这样的赋值方式。

```
// common/autoconf/generated-configure.sh
...
# Check whether --with-update-version was given.
JDK_UPDATE_VERSION="$with_update_version"
...
if test "x$JDK_UPDATE_VERSION" != x; then

JDK_VERSION="${JDK_MAJOR_VERSION}.${JDK_MINOR_VERSION}.${JDK_MICRO_VERSION}_
${JDK_UPDATE_VERSION}"
else

JDK_VERSION="${JDK_MAJOR_VERSION}.${JDK_MINOR_VERSION}.${JDK_MICRO_VERSION}"
```

```
    f
```

这里的 **--with-update-version** 就是我们在编译 openJDK 时，./configure 传入的参数。

好了，我们现在终于找到根了！

也就是说，假如我们在 configure 的时候传入了

--with-update-version=XXX

这样的参数：

那么 JDK_UPDATE_VERSION 将会等于 XXX

进而导致 JDK_VERSION = 1.8.0_XXX

进而导致 RELEASE = 1.8.0_XXX-internal

进而导致 @@java_version@@ = 1.8.0_XXX-internal

进而导致 java -version 打印出的第一行字符串为

openjdk version "1.8.0_XXX-internal"

而不再是

openjdk version "1.8.0-internal"

其他内容同理可知，我们直接说结论。

我们重新编译 openJDK，加入编译参数。

```
./configure \
--with-milestone=fcs \
--with-update-version=312 \
--with-build-number=b00

make al

```

在得出的产物中，执行 java -version，将会得到如下信息。

```
openjdk version "1.8.0_312"
```

```
OpenJDK Runtime Environment (build 1.8.0_312-b07)
OpenJDK 64-Bit Server VM (build 25.71-b00, mixed mode

```

这和一开始的默认版本，就不一样了！

```
openjdk version "1.8.0-internal"
OpenJDK Runtime Environment (build 1.8.0-internal-root_2022_06_08_12_25-b00)
OpenJDK 64-Bit Server VM (build 25.71-b00, mixed mode

```

OK！我们这就成功得到了一个我们自己定制的 openJDK 发行版！

不过先别高兴得太早，我们来看几个业界知名的企业 JDK 发行版的 java -verison 信息。

腾讯 Kona 的是这样的。

```
openjdk version "1.8.0_322"
OpenJDK Runtime Environment (Tencent Kona 8.0.9) (build 1.8.0_322-b1)
OpenJDK 64-Bit Server VM (Tencent Kona 8.0.9) (build 25.322-b1, mixed mode,
sharing
```

阿里 Dragonwell 的是这样的。

```
openjdk version "1.8.0_322"
OpenJDK Runtime Environment (Alibaba Dragonwell 8.10.11) (build 1.8.0_322-
b01)
OpenJDK 64-Bit Server VM (Alibaba Dragonwell 8.10.11) (build 25.322-b01,
mixed mode
```

华为 bisheng 的是这样的。

```
openjdk version "1.8.0_322"
OpenJDK Runtime Environment BiSheng (build 1.8.0_322-b06)
OpenJDK 64-Bit Server VM BiSheng (build 25.322-b06, mixed mode

```

我们可以看出，它们不但修改了我们刚刚所说的那几个配置参数，还在第二行的中间，加入了自己独特的 JDK 名称，很是酷炫。

那他们是怎么做的呢?

别急，你都知道了 Version.java.template 文件可以生成最终的 Version 文件里的 print 方法，那你直接在方法里，打印第二行的中间写死一个字符串，不就行了么?

当然可以，你的 JDK 你做主，但是，我们还是来看看他们是怎么优雅地修改源码的。

以腾讯 Kona 为例，它分别修改了如下地方的源码。

一、改造了 **Version.java.template** 里的 print 代码，但是没有写死字符串，而是像其他常量一样，留了个占位符。

```
// https://github.com/Tencent/TencentKona-8/blob/master/jdk/src/share/classes/sun/misc/Version.java.template
public class Version {
    ...
    private static final String java_distro_name =
         "@@java_distro_name@@";

    private static final String java_distro_version =
         "@@java_distro_version@@";

    public static void print(PrintStream ps) {
        ...
        /* First line: platform version. */
        ...
        /* Second line: runtime version (ie, libraries). */
        ps.print(java_runtime_name +
            " (" + java_distro_name + " " + java_distro_version + ")" +
            " (build " + java_runtime_version);...
        /* Third line: JVM information. */
        ...
    }
}
```

二、改造了 **GensrcMisc.gmk** 文件，将这些占位符同样用 sed 命令替换。

```
// https://github.com/Tencent/TencentKona-8/blob/master/jdk/make/gensrc/GensrcMisc.gmk
$(JDK_OUTPUTDIR)/gensrc/sun/misc/Version.java \
 $(PROFILE_VERSION_JAVA_TARGETS): \
     $(JDK_TOPDIR)/src/share/classes/sun/misc/Version.java.template
  $(MKDIR) -p $(@D)
```

```
  $(RM) $@ $@.tmp
  $(ECHO) Generating sun/misc/Version.java $(call profile_version_name, $@)
  $(SED) -e 's/@@launcher_name@@/$(LAUNCHER_NAME)/g' \
      -e 's/@@java_version@@/$(RELEASE)/g' \
      -e 's/@@java_runtime_version@@/$(FULL_VERSION)/g' \
      -e 's/@@java_runtime_name@@/$(RUNTIME_NAME)/g' \
      -e 's/@@java_profile_name@@/$(call profile_version_name, $@)/g' \
      -e 's/@@java_distro_name@@/$(DISTRO_NAME)/g' \
      -e 's/@@java_distro_version@@/$(DISTRO_VERSION)/g' \
      $< > $@.tmp
  $(MV) $@.tmp $@
```

三、改造了 **spec.gmk.in** 文件，在里面做了变量的定义。

```
// https://github.com/Tencent/TencentKona-8/blob/master/common/autoconf/spec.gmk.in
...
# Include TencentJDK version information
DISTRO_NAME=Tencent Kona
...
DISTRO_VERSION=8.0.9
```

所以最终，这里的 **Tencent Kona 8.0.9** 就显示在了 java -version 信息里了。

其实本质上，就是改变最终 print 方法的代码而已。