Jlink и Custom Runtime Image - мастерская Франкенштейна

Юрий Артамонов
CUBA Platform

Толстота

Пользователи скачивают:

JDK 8 - 202 MB

JDK 11 - 179 MB

Как доставить Java?

Как сделать меньше?

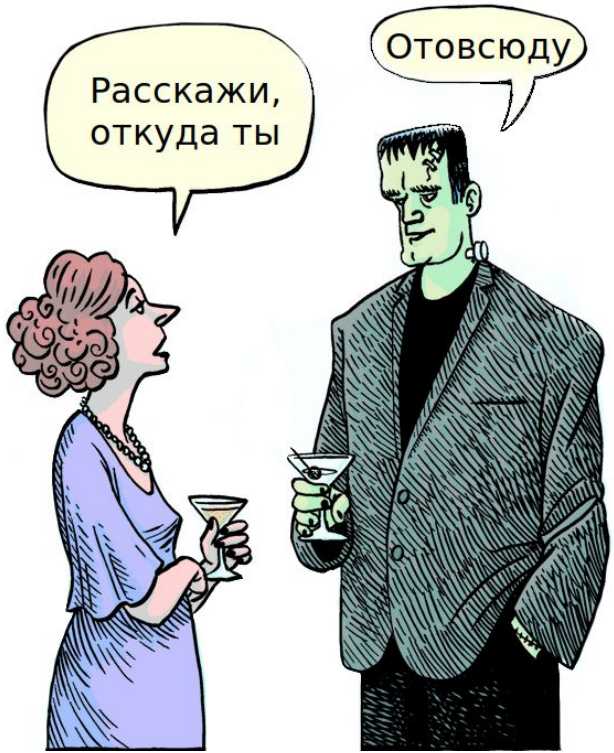
История вопроса

Собираем всё:

- Launch4j
- IzPack / Install4j
- Java Packager
- Custom JRE Bundle

Минимизируем:

JEP 161: Compact Profiles



javac -profile

Server JRE 8	all	53 MB
compact 3	jmx, crypto, naming, …	21 MB
compact 2	rmi, sql, xml, …	18 MB
compact 1	lang, util, io, …	14 MB

Проблемы

- Монструозность
- Хрупкость

> jdeps -P HelloWorld.class

```
HelloWorld.class -> /binaries/linux-i586/jre/lib/rt.jar
   <unnamed> (HelloWorld.class)
        -> java.io                                    compact1
        -> java.lang                                  compact1
```

Java Packager

https://docs.oracle.com/javase/8/docs/technotes/tools/unix/javapackager.html

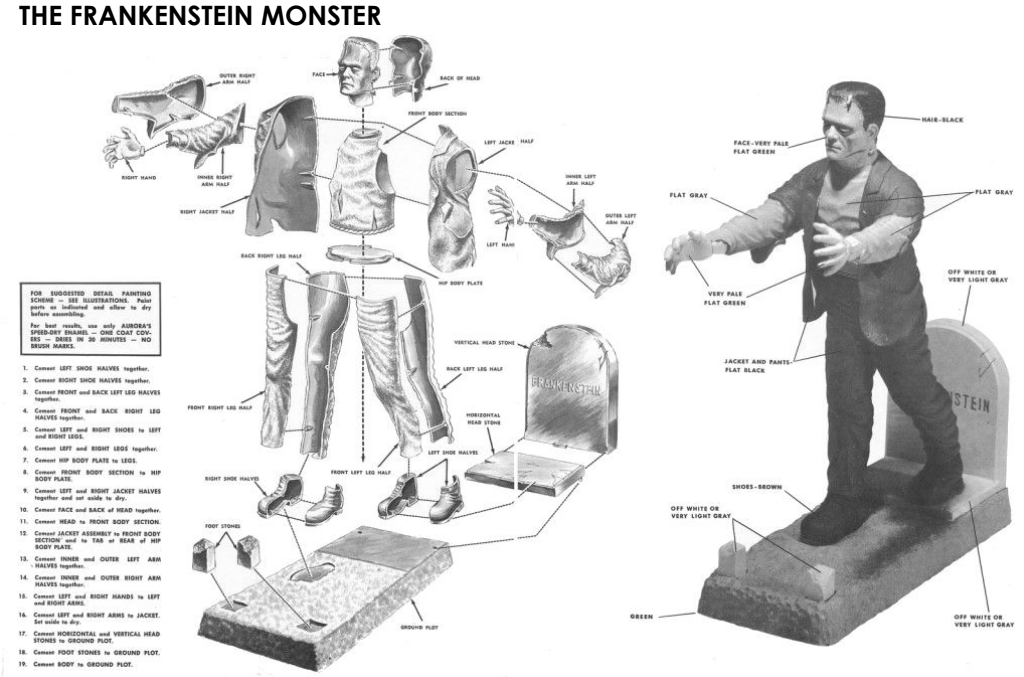
```
> javapackager -deploy <…>
> ls -lah out/deploy/bundles
54M pack-me-1.0.deb -
```

А внутри 200 MB JRE!

- Существует давно
- Пакует приложение с JRE
- Работает с JAVA и JAVA FX
- Выпиливается из Java 11 вместе с Java FX

Чего не хватает?

- JRE 8 - довольно большая штука (rt.jar)
- В Compact Profile нельзя менять состав пакета
- Проблемы Class Path:
  Jar Hell,
  NoClassDefFoundError



А если мы хотим собрать монстра сами?

Зачем это всё?

JPMS - модульная система Java:

- Надёжная конфигурация
- Сильная инкапсуляция
- Модуляризация JDK

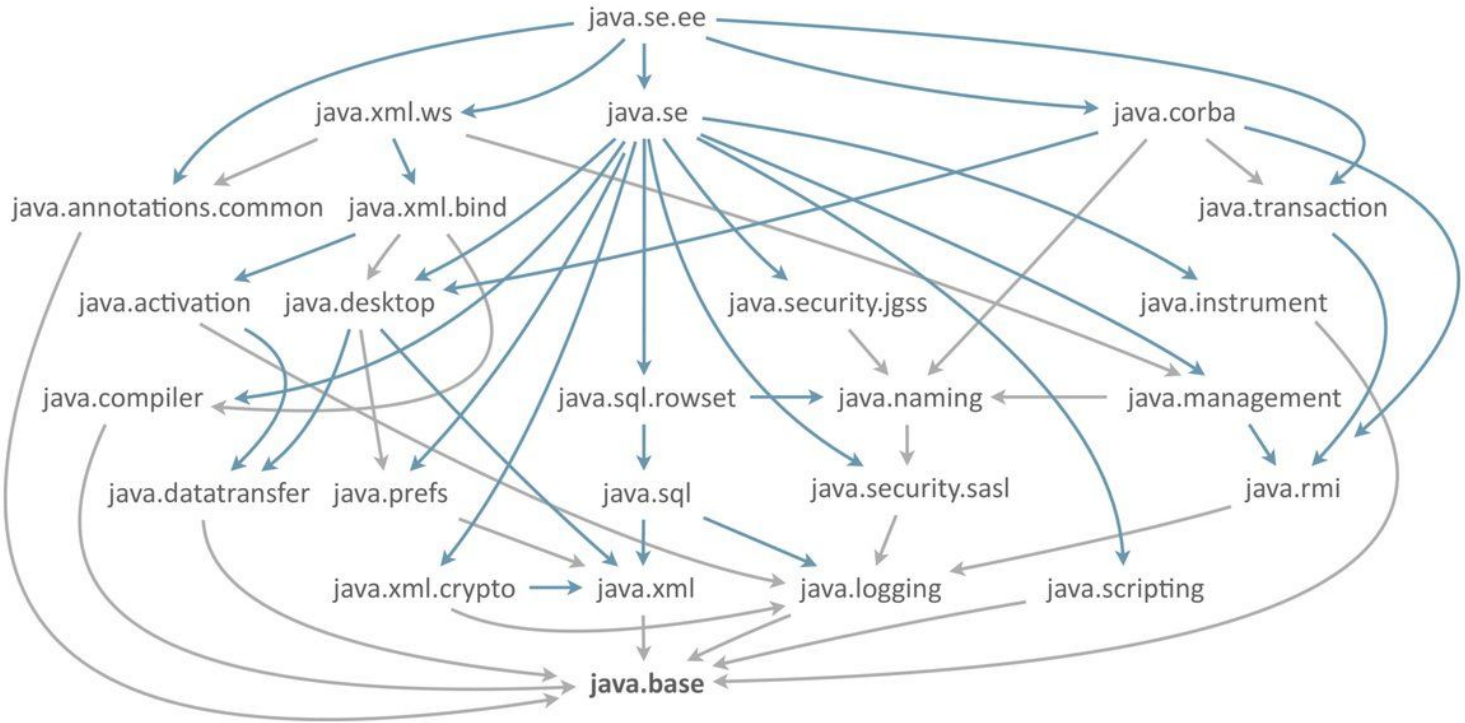
- Модульные приложения надёжнее и их проще развивать и поддерживать
- Можно использовать Хитрые трюки: AOT, Compile Time IoC, Bundling

Новый мир Java 9+

```
> jlink --module-path <…>
        --add-modules <…>
        --launcher <…>
```

- утилита для линковки ваших приложений с минимальной JRE
- Java 9+
- Требуется модульное окружение !
- На выходе - нативное приложение без зависимостей

Java Platform Modules



Custom Runtime Images

Новый формат распространения JRE с вашими приложениями

```
> ./bin/java -version

openjdk version "10.0.1" 2018-04-17


> ./bin/java --list-modules

java.base@10.0.1
com.haulmont.powercli
```

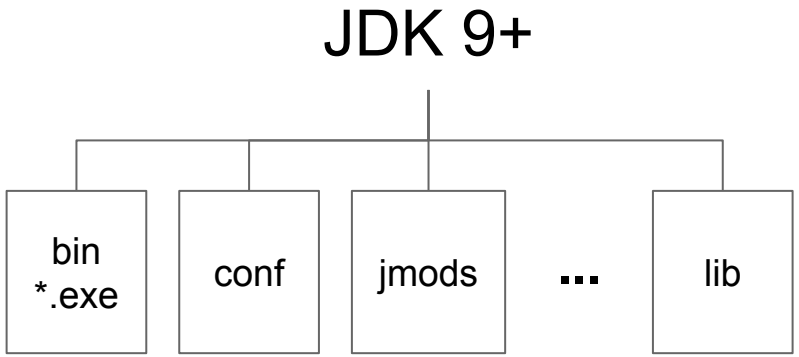
```
dist/
  bin/
    launch ← shell script
    java
    keytool
  conf/
    logging.properties
    net.properties
  lib/
    modules ← Modules BLOB
    libjava.so
    libnet.so
    …
```

Файлы JMOD

- Неисполняемые файлы
- Замена JAR для самой JDK
- Используются в рантайме или для компиляции / линковки

JMOD включает в себя:

- Java классы
- Ресурсы и Конфигурационные файлы
- Нативный код




ЗАВИСИМОСТИ !

Jdeps - узнаём правду о зависимостях

```
> $JAVA_HOME/bin/jdeps --module-path . --generate-module-info .
guava-26.0-jre.jar

Error: Module jsr305 contains package javax.annotation, module
java.xml.ws.annotation exports package javax.annotation to jsr305

Error: missing dependencies
   com.google.common.base.package-info                  ->
javax.annotation.ParametersAreNonnullByDefault     not found
   com.google.common.cache.package-info                 ->
javax.annotation.ParametersAreNonnullByDefault     not found
```


FAIL !

JPMS Patch Modules

Инструмент устранения Split Packages (временное решение):

```
> java
    --add-modules java.xml.ws.annotation
    --patch-module java.xml.ws.annotation=jsr305-3.0.2.jar
    --class-path $dependencies
    -jar $appjar
```

Полезный хак:

- Позволяет пересобрать существующие JAR файлы с modules-info.java

Сборка legacy проектов с jlink

Придётся всё модулиризовать самим !

1. Компилируем

```
> $JAVA_HOME/bin/javac -p build/lib \
                       -d build/modules \
                       --patch-module <moduleName>=<jarPath> \
                       module-info.java
```

2. Патчим JAR:

```
> $JAVA_HOME/bin/jar uf ./build/lib/<jarName> \
                      -C ./build/modules module-info.class
```

Патчим библиотеки на лету

1. Храним соответствие модулей и JAR файлов
2. Патчим всё перед линковкой
3. Делаем вид, что так и было

Build.gradle:

github.com/jreznot/power-cli/

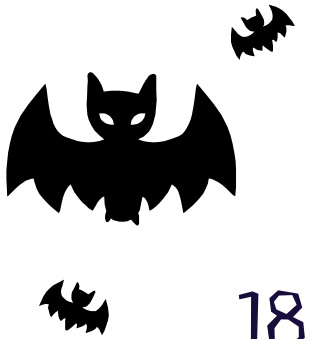
Кто готов к модульной Java ?

Готовность популярных фреймворков и библиотек к работе в модульном окружении

- Spring 5 - не модуляризован
- Hibernate - Не модуляризован
- Guava - Не модуляризована

... и т.д.

Имена Модулей и Automatic-module-names:
https://github.com/jodastephen/jpms-module-names



Отладка приложений, собранных с jlink

Отладчик - это тоже модуль!

```
module com.haulmont.powercli {
    requires jdk.jdwp.agent;
```

Правим скрипт запуска:

```
JLINK_VM_OPTIONS="-verbose -Xdebug -Xnoagent
-Xrunjdwp:server=y,transport=dt_socket,address=4000,suspend=y"
```

Поддержка jlink в инструментах сборки

Слабая поддержка в инструментах сборки:

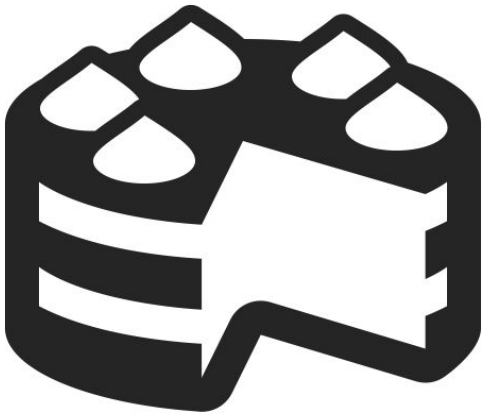
- Maven Compiler Plugin 3.7 +
- Gradle 4+:
  Java Library Plugin
  https://guides.gradle.org/building-java-9-modules/
- Для линковки - Сторонние плагины

Совместимость с Groovy / Kotlin

- Kotlin можно модуляризовать самим
- Groovy - Пока отстаёт
  JIRA/GROOVY-8339

```
> groovy -v
WARNING: An illegal reflective access
operation has occurred
...
```

А что ещё ?



Кросс-компиляция для Windows / Mac OS

```
> $JAVA_HOME/bin/jlink \
    --module-path libs:$TARGET_JAVA_HOME/jmods \
    --add-modules <moduleName>
    --launcher launch=<moduleName>/<mainClassName>
```

- Jlink поддерживает линковку с JRE для Windows / Linux / Mac OS.
- Собирайте приложение для всех ОС на Linux (особенности файловых систем *NIX)
- Добавьте CI по вкусу

Файловая система JRT

- Имеет префикс jrt://
- Используется для получения Ресурсов в Custom Runtime Image
- Фреймворки, которые сканируют Class Path, должны явно поддержать Jrt (Spring component scan сломается)
- Можно выполнять сканирование ресурсов при помощи

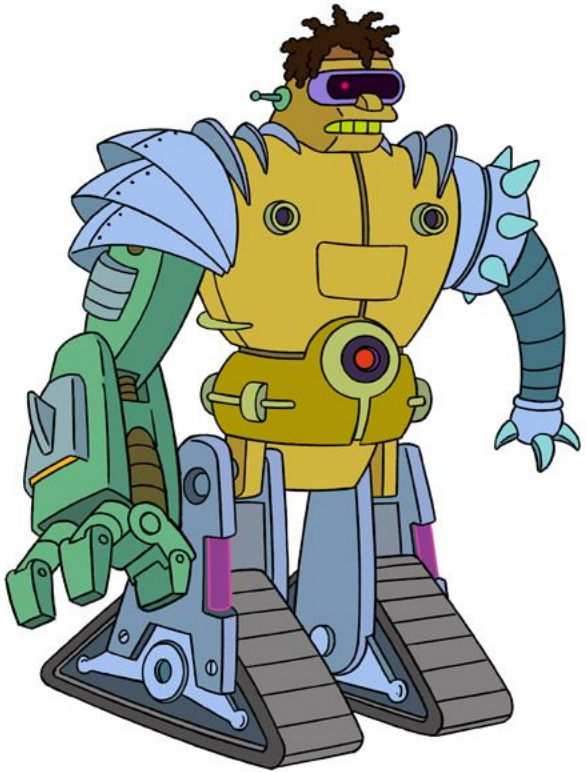
```
Files.walk(Path, depth)
```

Jlink Plugin API

См. `jdk.tools.jlink.plugin.Plugin`

Механизм плагинов позволяет реализовать:

- Удаление неиспользуемого кода
- Создание индексов аннотаций и сервисов
- IoC/DI во время линковки



Credits: Gunnar Morling
http://in.relation.to/2017/12/12/exploring-jlink-plugin-api-in-java-9/

А что с производительностью?

- Время Старта - Консольное приложение
  (250msec - 200 msec = 50 msec)
- Размер бандла - MIN 30 MB (11 MB - ZIP)
- Потребление памяти -
  Зависит от приложения !
  MIN - 2MB Heap
  (Разницы не замечено)

См. `ManagementFactory.getRuntimeMXBean().startTime`

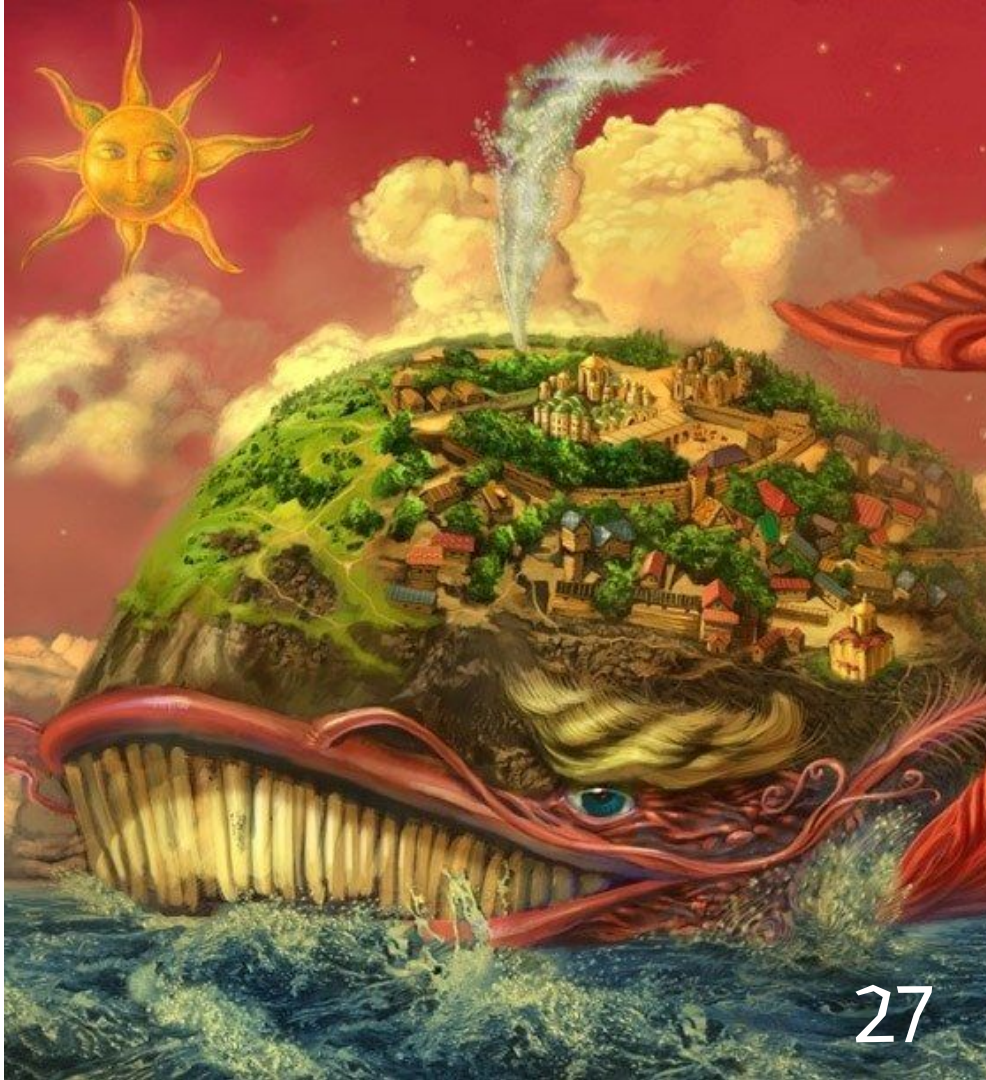
Кто сказал Docker?

Java 10 наконец-то нормально исполняется в Docker.

- Минимальный Alpine 3.8 +
- Custom Runtime Image

= 35 MB!

Сравним с Go:
<1 MB бинарник + Linux = 10 MB !



Область применимости

- Настольные приложения
- Консольные утилиты
- Серверные решения
- Развёртывание Контейнерами

Нужно всем!

Используем jlink для CLI

CUBA CLI

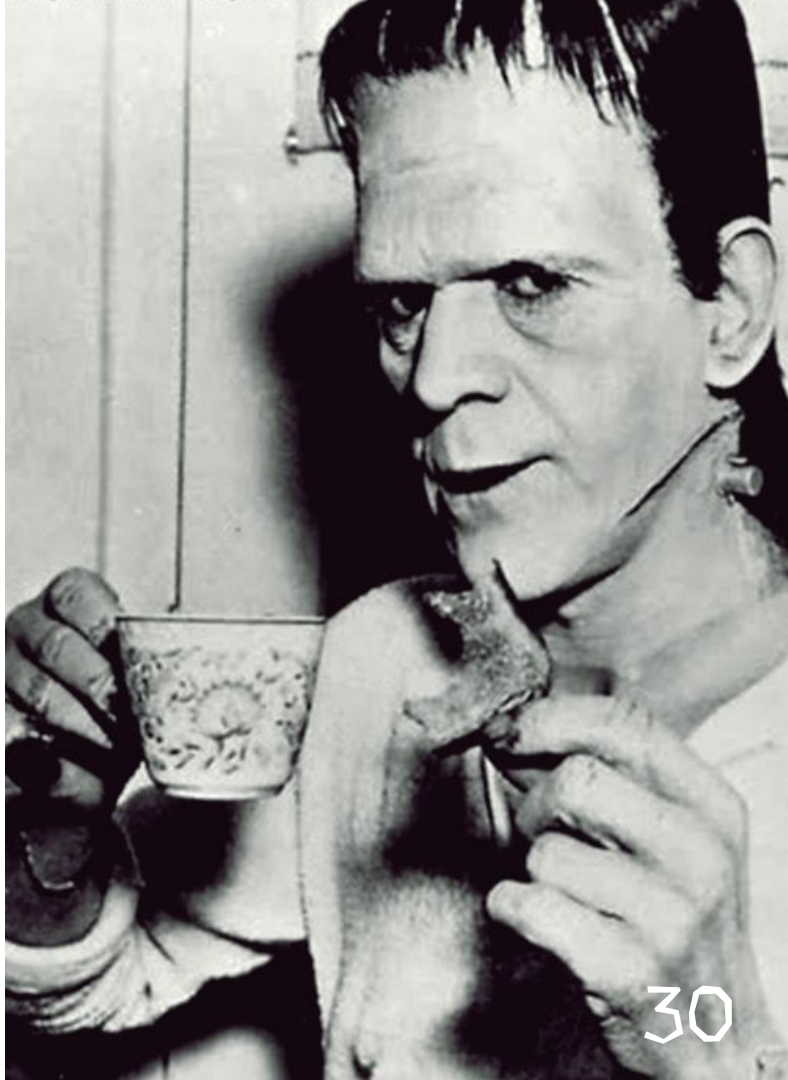
github.com/cuba-platform/cuba-cli

```
> sdk install cuba
```

- Написан на Kotlin
- Размер архива ~ 30 MB
- Поддерживает динамическую загрузку плагинов:
  ModuleFinder, ModuleLayer, ServiceLoader

Выводы

1. Формат доставки - это важно
2. Полагаться на установленную версию Java - больше не безопасно
3. Модуляризируйте свои приложения
4. JLink - полезный инструмент для всех




Вопросы ?
Слайды: <Speaker Deck Link>
Twitter: @Yuriy_Artamonov