链滴

Kotlin 使用 GraalVM+Picocli 开发原生命令行应用

作者：crick77

原文链接：https://ld246.com/article/1598800089928

来源网站：链滴

代码已开源： https://github.com/wangyuheng/ddl2plantuml

背景

之前用 kotlin开发过一款根据建表DDL语句生成plantuml ER图的应用。被问如何使用，答曰"给你一jar包，然后执行 java -jar ddl2plantuml.jar ./ddl.sql ./er.puml 就可以了。是不是so easy?"

结果被吐槽了一番，

1. 为什么不能像命令行应用一样提供相关帮助信息?
2. 为什么是Java， 而不是一个原生命令行应用?

这个吐槽带来了一个思考： 为什么Java很少用于开发原生命令行(**CLI**)应用呢？我认为主要问题有2个

1. Java通过JVM实现跨平台。也就是说，如果要使用Java应用需要先安装JRE。
2. Java的优势在于JVM热点代码检测和运行时编译及优化，所以这是一门程序运行时间越长，速度越的神奇语言。而付出的代价则是应用启动速度较慢。这与一次性启动运行的命令行应用的场景需求正反。

方案

为了解决上述问题，引入2个名词

1. Picocli
2. GraalVM

Picocli

Picocli 致力于提供“最简便的方式来创建富命令行应用，这种应用可以在 JVM 上和 JVM 之外运行”

使用起来非常简单

```
fun main(args: Array<String>) {

    val cmd = CommandLine(Convert())
    when {
        args.isEmpty() -> {
            cmd.usage(System.out)
        }
        else -> {
            val exitCode = cmd.execute(*args)
            exitProcess(exitCode)
        }
    }
}

@CommandLine.Command(name = "ddl2plantuml",
        version = ["软件名称：Ddl2plantuml\n版本：V1.1.0"],
        description = ["convert sql ddl to plantuml er"],
        mixinStandardHelpOptions = true
```

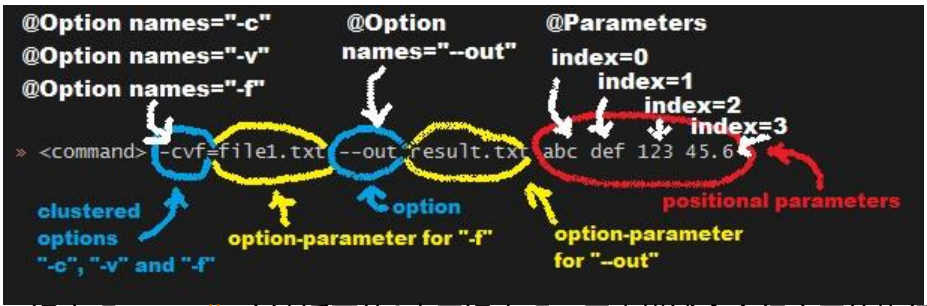
```
)
class Convert : Callable<Int> {

    @CommandLine.Parameters(index = "0", description = ["The sql ddl file that should be con
ert to plantuml er."])
    lateinit var src: Path

    @CommandLine.Option(names = ["-o", "--output"], description = ["The file where the plan
uml file to be saved. default is console "])
    private var target: Path? = null

    override fun call(): Int {
        require(src.toFile().exists()) { "ddl file must be existed!" }
        when (target) {
            null -> {
                FileReader(src).read()
                        .apply { ConsoleWriter(this).write() }
            }
            else -> {
                FileReader(src).read()
                        .apply { FileWriter(target!!, this).write() }
            }
        }
        return 0
    }

}
```

效果

```
# wangyuheng @ wangyuhengdeMacBook-Pro in ~/universe/code/wangyuheng_github/ddl2plantuml on git:shell x
$ java -jar target/ddl2plantuml-1.1.0.jar -h
Usage: ddl2plantuml [-hV] [-o=<target>] <src>
convert sql ddl to plantuml er
      <src>              The sql ddl file that should be convert to plantuml
                           er.
  -h, --help             Show this help message and exit.
  -o, --output=<target>  The file where the plantuml file to be saved. default
                           is console
  -V, --version          Print version information and exit.

# wangyuheng @ wangyuhengdeMacBook-Pro in ~/universe/code/wangyuheng_github/ddl2plantuml on git:shell x
$ java -jar target/ddl2plantuml-1.1.0.jar -V
软件名称: Ddl2plantuml
版本: V1.1.0

# wangyuheng @ wangyuhengdeMacBook-Pro in ~/universe/code/wangyuheng_github/ddl2plantuml on git:shell x
$ java -jar target/ddl2plantuml-1.1.0.jar ddl.sql
@startuml
!define Table(name,desc) class name as "desc" << (T,#FFAAAA) >>
!define primary_key(x) <color:red><b>x</b></color>
!define unique(x) <color:green>x</color>
!define not_null(x) <u>x</u>
hide methods
hide stereotypes

Table(table_1, "table_1\n(This is table 1)"){
 not_null(`id`) bigint  'column_1'
not_null(`prod_name`) varchar  'column_2'
not_null(`prod_type`) tinyint '0' 'column_3 0:活期 1:定期'
not_null(`start_time`) time  '停止交易开始时间'
not_null(`end_time`) time  '停止交易结束时间'
not_null(`online_type`) tinyint '0' '0:上线 1:未上线'
not_null(`prod_info`) varchar '' '产品介绍'
not_null(`over_limit`) tinyint '0' '超额限制 0:限制 1:不限制'
not_null(`created_time`) datetime CURRENT_TIMESTAMP
not_null(`updated_time`) datetime CURRENT_TIMESTAMP
 }
Table(table_2, "table_2\n(This is table 2)"){
 not_null(`id`) bigint
not_null(`user_id`) bigint  '用户id'
not_null(`user_name`) varchar  '用户名称'
not_null(`prod_id`) bigint  '产品id'
`interest_date` date NULL '计息日期'
not_null(`created_time`) datetime CURRENT_TIMESTAMP '创建时间'
not_null(`updated_time`) datetime CURRENT_TIMESTAMP '更新时间'
 }


@enduml
```

这里介绍用到的几个注解及概念

- @Parameters 和@Options 都是用来定义参数，区别在于@Parameters根据位置区分，而@Options可以指定名称



- 退出码。 call()方法返回的0表示退出码，用来描述命令行应用的执行结果。通常用0表示成功，其数字为自定义异常。退出码不会影响程序的执行，但是有一个很实用的功能是当你通过连接的方式同执行多个应用时，一个非零的退出码会中断这个组合。如：./ddl2plantuml_mac ddl.sql |grep "table"

- 版本及帮助信息。可以自定义并指定样式，version可以通过 versionProvider自定义生成。

GraalVM

Go的一个宣传点是可以将程序编译为一个静态可执行文件，而Java也可以通过 GraalVM做到这一点

GraalVM: Run Programs Faster Anywhere

这个slogan和Java的"Write Once, Run Anywhere"遥相呼应，同时又展示了极大的野心，准备带来一个20年的辉煌。

GraalVM 是一个高性能的通用虚拟机，可以运行使用 JavaScript，Python 3，Ruby，R，基于 JVM 的语言以及基于 LLVM 的语言开发的应用。 GraalVM 消除了编程语言之间的隔离性，并且通过共享行时增强了他们的互操作性。它可以独立运行，也可以运行在 OpenJDK，Node.js，Oracle，MySQL 等环境中。

可以看到 GraalVM提供了非常强大的功能，这里我们不做展开介绍，只看如何解决我们遇到的问题主要用到了2个功能特性

1. 即时编译，提升程序启动速度
2. Native Image，将应用编译为单个静态可执行文件

使用方式

1. 安装GraalVM
2. 安装 native-image 工具 gu install native-image
3. 编译应用 native-image -jar target/ddl2plantuml-1.1.0.jar ddl2plantuml

编译后的native image不运行在Java VM上，但是包含了必要的组件，如内存管理和线程调度，这些件来自另一个 Substrate VM。这个过程称为提前编译

此时我们已经得到了一个可以直接执行的原生命令行应用

./ddl2plantuml_mac ddl.sql

注意:

native image不支持Java的所有特性，尤其是对 reflection的限制。在这次改造过程中，原来通过阿的 druid进行sql解析，但是 druid使用了大量的 reflection导致native image编译失败，所以改用 jsq parser。

其他

1. Picocli提供了maven插件 native-image-maven-plugin，用于编译阶段进行native image构建但是建议分离开发和构建，在CICD中执行构建过程，可以节省开发时间，并构建不同平台的应用，决开发环境局限
2. 除了构建命令行应用，GraalVM也带来了更多的可能性，比如Java在 FAAS中的应用。