10分钟搞定让你困惑的 Jenkins 环境变量

日拱一兵 发布于 2021-01-08

前言

Jenkins， DevOps 技术栈的核心之一，CI/CD 离不开编写 Pipeline 脚本，上手 Jenkins ，简单查一下文档，你就应该不会被 agent，stages，step 这类关键词弄懵，也能很快构建出 pipeline 的骨架

但是当向骨架中填充内容的时候，尤其如何利用**环境变量**（系统内置 | 自定义），多数人都会变得比较混乱，浪费很多时间，本文就帮助大家快速通关环境变量

认识 Jenkins 环境变量

> Jenkins 环境变量就是通过 `env` 关键字暴露出来的**全局变量**，可以在 Jenkins 文件的**任何位置使用**

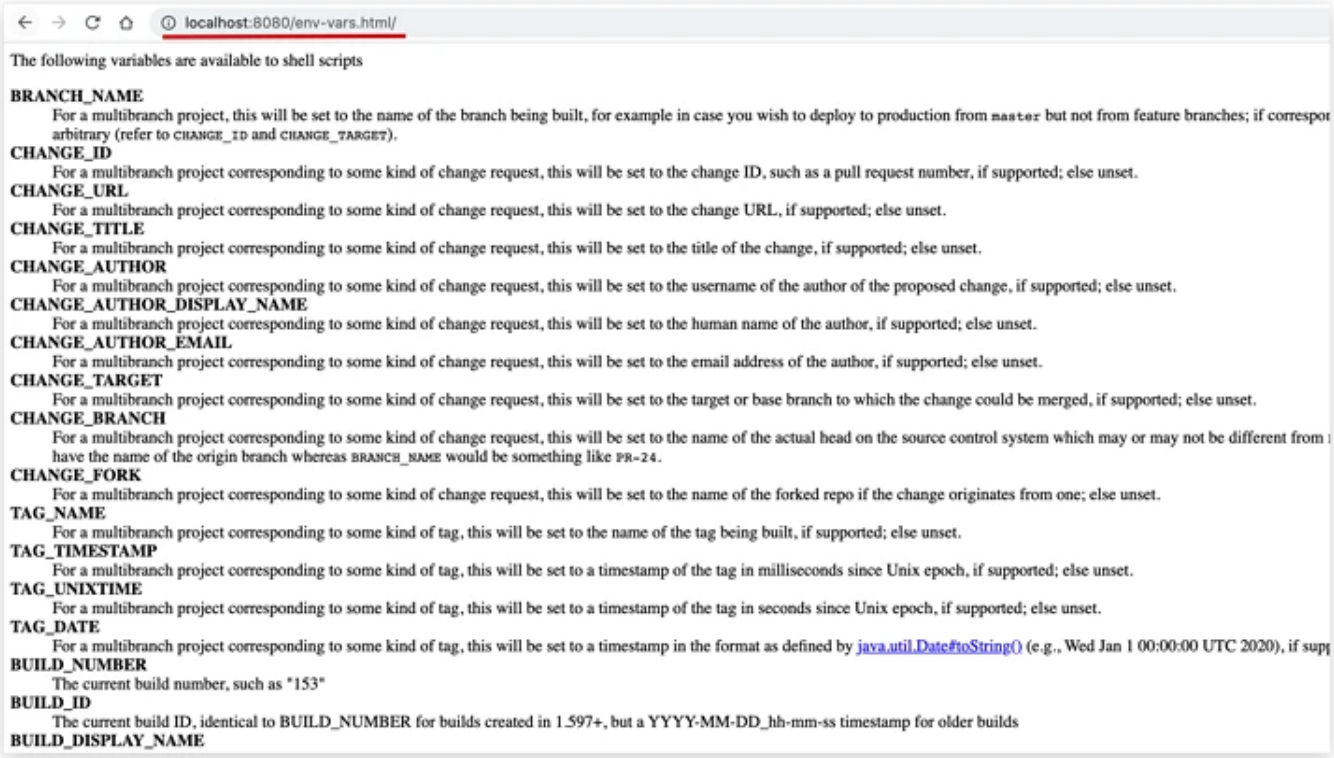
其实和你使用的编程语言中的全局变量没有实质差别

查看 Jenkins 系统内置环境变量

Jenkins 在系统内置了很多环境变量方便我们快速使用，查看起来有两种方式：

方式一：

直接在浏览器中访问 `${YOUR_JENKINS_HOST}/env-vars.html` 页面就可以，比如 `http://localhost:8080/env-vars.html` ，每个变量的用途写的都很清楚



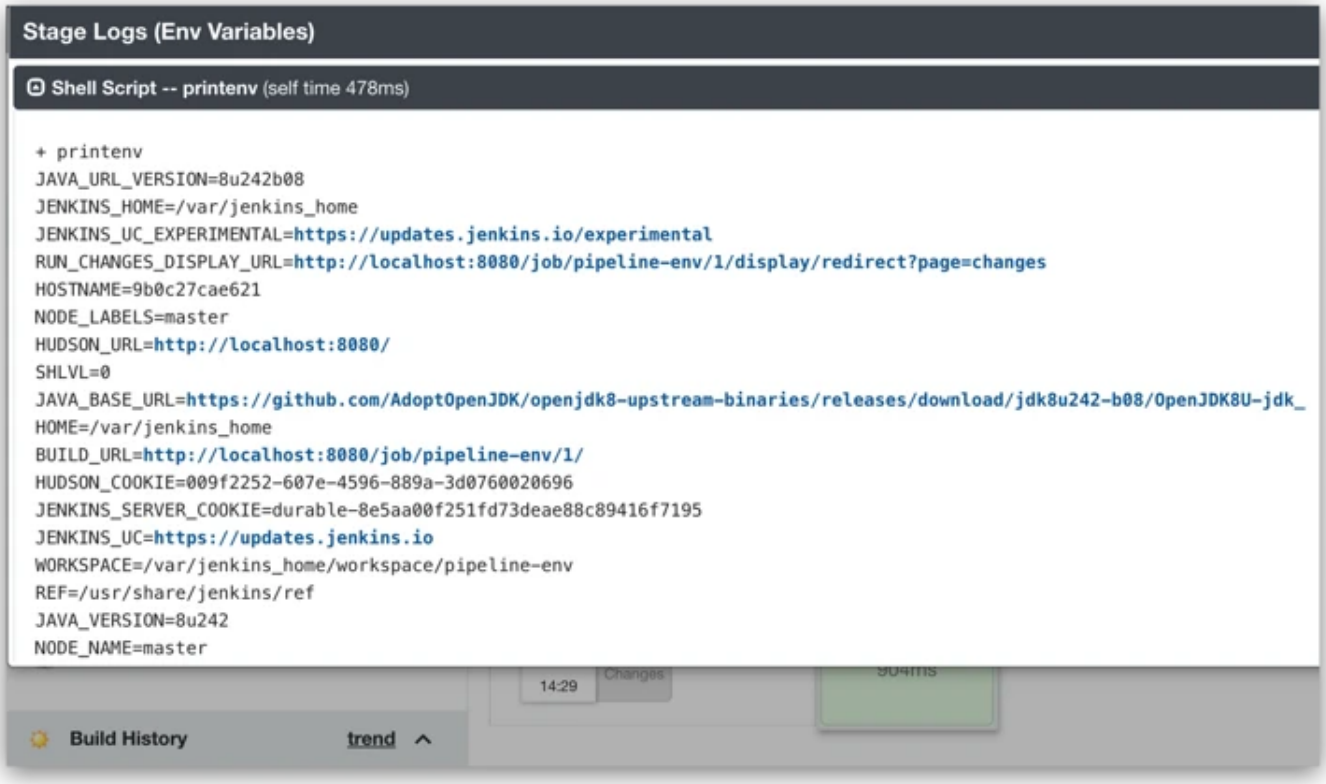
方式二

通过执行 `printenv` shell 命令来获取：

```
pipeline {
    agent any

    stages {
        stage("Env Variables") {
            steps {
                sh "printenv"
            }
        }
    }
}
```

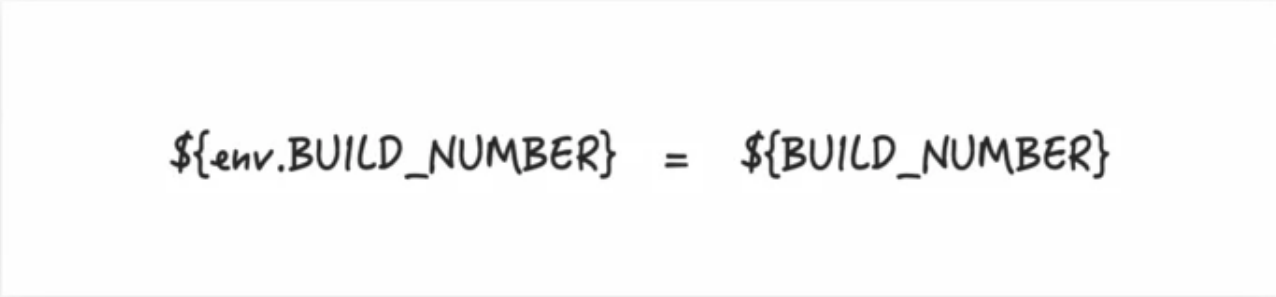
直接 Save - Build, 在终端 log 中你会看到相应的环境变量，并且可以快速看到他们当前的值



通常这两种方式可以结合使用

读取环境变量

上面我们说了 `env` 是环境变量的关键字，但是读取 Jenkins 内置的这些环境变量，`env` 关键字是可有可无, 但不能没了底裤，都要使用 `${xxx}` 包围起来。以 `BUILD_NUMBER` 这个内置环境变量举例来说明就是这样滴：



如果你在 Jenkins 文件中使用 shell 命令，使用这些内置环境变量甚至可以不用 `{}`， 来看一下：

```
pipeline {
    agent any

    stages {
        stage("Read Env Variables") {
            steps {
```

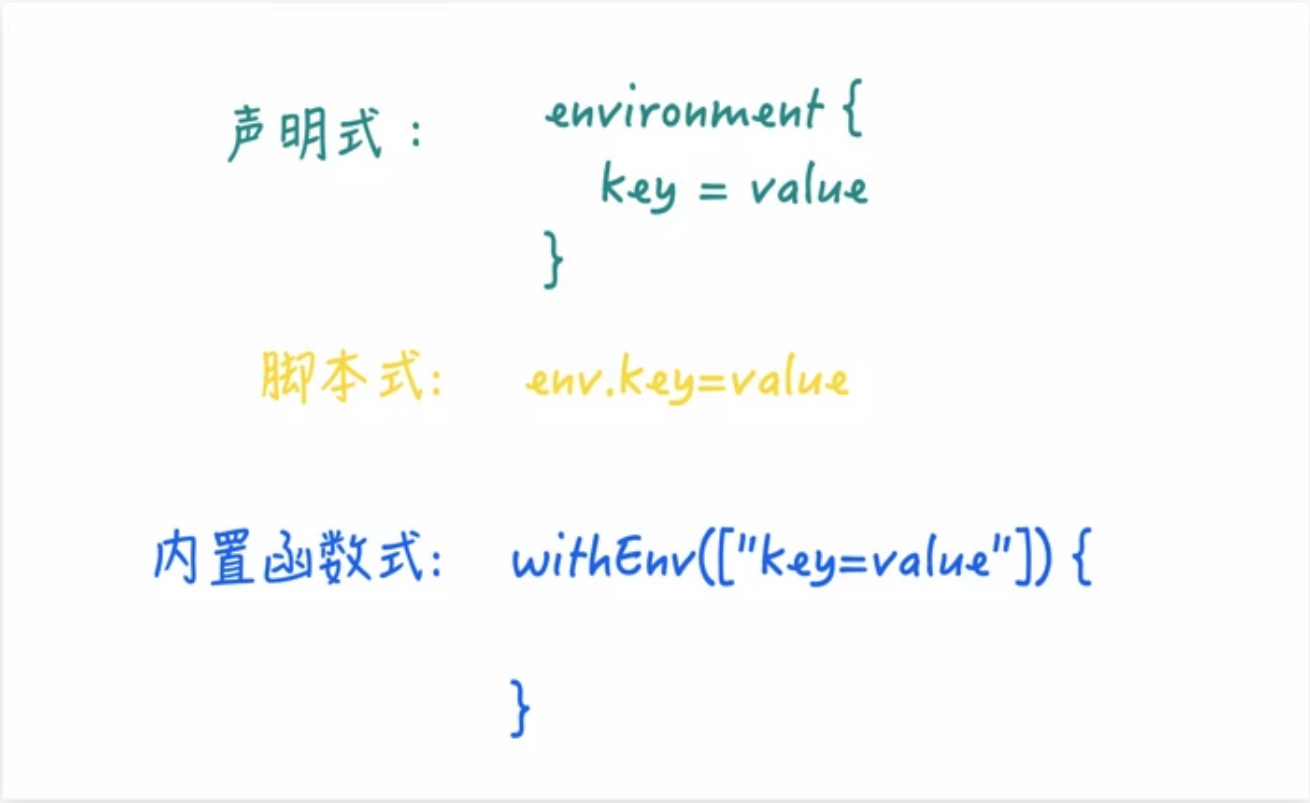
```
                echo "带 env 的读取方式：${env.BUILD_NUMBER}"
                echo "不带 env 的读取方式：${BUILD_NUMBER}"
                sh 'echo "shell 中读取方式 $BUILD_NUMBER"'
            }
        }
    }
}
```

可以看到结果是一样一样滴，**不管有几种，记住第一种最稳妥**

内置的环境变量虽好，但也不能完全满足我们自定义的 pipeline 的执行逻辑，所以我们也得知道如何定义以及使用自定义环境变量

自定义 Jenkins 环境变量

Jenkins pipeline 分声明式（Declarative）和 脚本式（imperative）写法，相应的环境变量定义方式也略有不同，归纳起来有三种方式：



还是看个实际例子吧：

```
pipeline {
    agent any

    environment {
        FOO = "bar"
    }

    stages {
        stage("Custom Env Variables") {
            environment {
                NAME = "RGYB"
            }

            steps {
                echo "FOO = ${env.FOO}"
                echo "NAME = ${env.NAME}"

                script {
                    env.SCRIPT_VARIABLE = "Thumb Up"
                }

                echo "SCRIPT_VARIABLE = ${env.SCRIPT_VARIABLE}"

                withEnv(["WITH_ENV_VAR=Come On"]) {
                    echo "WITH_ENV_VAR = ${env.WITH_ENV_VAR}"
```

```
                }
            }
        }
    }
}
```

来看运行结果：

> 注意：`withEnv(["WITH_ENV_VAR=Come On"]) {}` **这里的 = 号两侧不能有空格**，必须是 `key=value` 的形式

一个完整的 pipeline 通常会有很多个 stage，环境变量在不同的 stage 有不同的值是很常见的，知道如何设置以及读取环境变量后，我们还得知道如何重写环境变量

重写 Jenkins 环境变量

Jenkins 让人相对困惑最多的地方就是重写环境变量，但是只要记住下面这三条规则，就可以搞定一切了

1. `withEnv(["WITH_ENV_VAR=Come On"]) {}` 内置函数的这种写法，可以重写任意环境变量
2. 定义在 `environment {}` 的环境变量不能被脚本式定义的环境变量（`env.key="value"`）重写
3. 脚本式环境变量只能重写脚本式环境变量

这三点是硬规则，没涵盖在这 3 点规则之内的也就是被允许的了

三条规则就有点让人头大了，农夫选豆种，举例为证吧

```
pipeline {
    agent any

    environment {
        FOO = "你当像鸟飞往你的山"
        NAME = "Tan"
    }

    stages {
        stage("Env Variables") {
            environment {
                  // 会重写第 6 行 变量
                NAME = "RGYB"
                  // 会重写系统内置的环境变量 BUILD_NUMBER
                BUILD_NUMBER = "10"
            }

            steps {
                  // 应该打印出 "FOO = 你当像鸟飞往你的山"
                echo "FOO = ${env.FOO}"
                  // 应该打印出 "NAME = RGYB"
                echo "NAME = ${env.NAME}"
                  // 应该打印出 "BUILD_NUMBER = 10"
                echo "BUILD_NUMBER =  ${env.BUILD_NUMBER}"

                script {
                      // 脚本式创建一个环境变量
                    env.SCRIPT_VARIABLE = "1"
                }
            }
        }

        stage("Override Variables") {
            steps {
                script {
```

```
                    // 这里的 FOO 不会被重写，违背 Rule No.2
                    env.FOO = "Tara"
                    // SCRIPT_VARIABLE 变量会被重写，符合 Rule No.3
                    env.SCRIPT_VARIABLE = "2"
                }

                // FOO 在第 37 行重写失败，还会打印出 "FOO = 你当像鸟飞往你的山"
                echo "FOO = ${env.FOO}"
                // 会打印出 "SCRIPT_VARIABLE = 2"
                echo "SCRIPT_VARIABLE = ${env.SCRIPT_VARIABLE}"

                // FOO 会被重写，符合 Rule No.1
                withEnv(["FOO=Educated"]) {
                    // 应该打印 "FOO = Educated"
                    echo "FOO = ${env.FOO}"
                }

                // 道理同上
                withEnv(["BUILD_NUMBER=15"]) {
                    // 应该打印出 "BUILD_NUMBER = 15"
                    echo "BUILD_NUMBER = ${env.BUILD_NUMBER}"
                }
            }
        }
    }
}
```

来验证一下结果吧

看到这，基本的设置应该就没有什么问题了，相信你也发现了，Jenkins 设置环境变量和编程语言的那种设置环境变量还是略有不同的，后者可以将变量赋值为对象，但 Jenkins 就不行，因为**在 Jenkins 文件中，所有设置的值都会被当成 String**， 难道没办法应用 Boolean 值吗?

Jenkins 中使用 Boolean 值

如果设置一个变量为 `false` ， Jenkins 就会将其转换为 `"false"`, 如果想使用 Boolean 来做条件判断，必须要调用 `toBoolean()` 方法做转换

```
pipeline {
    agent any

    environment {
        IS_BOOLEAN = false
    }

    stages {
        stage("Env Variables") {
            steps {
                script {
                      // Hello 会被打印出来，因为非空字符串都会被认为是 Boolean.True
                    if (env.IS_BOOLEAN) {
                        echo "Hello"
                    }

                      // 真正的 Boolean 比较
                    if (env.IS_BOOLEAN.toBoolean() == false) {
                        echo "日拱一兵"
                    }

                      // 真正的 Boolean
                    if (!env.IS_BOOLEAN.toBoolean()) {
```

```
                    echo "RGYB"
                }
            }
        }
    }
}
```

来看运行结果：

如果你写过 Pipeline，你一定会知道，写 Pipeline 是离不开写 shell 的，有些时候，需要将 shell 的执行结果赋值给环境变量，Jenkins 也有方法支持

Shell 结果赋值给环境变量

实现这种方式很简单，只需要记住一个格式：`sh(script: 'cmd', returnStdout:true)`

```
pipeline {
    agent any

    environment {
          // 使用 trim() 去掉结果中的空格
        LS_RESULT = "${sh(script:'ls -lah', returnStdout: true).trim()}"
    }

    stages {
        stage("Env Variables") {
            steps {
                echo "LS_RESULT = ${env.LS_RESULT}"
            }
        }
    }
}
```