

Jenkins2.x入门到精通实战(一) - pipeline介绍和基本groovy语法



ElliotG [关注](#) IP属地: 上海

0.102 2021.04.25 10:18:59 字数 919 阅读 2,153

0. 背景

什么是pipeline?

pipeline称为部署流水线。

```
1 | Jenkins 1.x只能通过界面来手动描述部署流水线。
2 | Jenkins 2.x开始支持pipeline as code, 可以通过代码来描述部署流水线。
```

pipeline as code的意义在于:

- 更好的版本化
将pipeline提交到版本库中进行版本控制。
- 更好的协作
pipeline的每个修改对团队成员可见, 另外还可以对pipeline进行代码审核。
- 更好的重用性
pipeline代码可以重用。

2. Jenkinsfile和pipeline语法

Jenkinsfile是什么?

Jenkinsfile是一个文本文件。

它是部署流水线在Jenkins中的表现形式, 所有部署流水线的逻辑都卸载Jenkinsfile中。

它存在的意义就像dockerfile对于Docker的意义。

Jenkins默认是不支持Jenkinsfile的, 我们需要安装pipeline插件。

pipeline语法的选择

Groovy脚本语言被作为pipeline的默认语言。

脚本中每个步骤我们称之为stage

一个基本的基于groovy语法的pipeline骨架如下:

```
pipeline {
    agent any

    stages {
        stage("Build") {
            // 构建
            steps {
                echo("Building Start...")
            }
        }
    }
}
```

```

12
13     stage("Test") {
14         // 测试
15         steps {
16             echo("Testing Start...")
17         }
18     }
19
20     stage("Deploy") {
21         // 部署
22         steps {
23             echo("Deploy Start...")
24         }
25     }
26 }

```

3. 创建第一个pipeline

```

1  1. 在首页创建项目时选择pipeline
2
3  2. 从版本控制库拉取pipeline
4      2-1) 安装git插件
5      2-2) 将git私钥放到Jenkins上
6              Jenkins->Creentials->System->Global credentials
7                  选择Kind为"SSH Username with private key"
8      2-3) 将Jenkinsfile推送到gitlab
9      2-4) Pipeline节点下的Destination选择"Pipeline script from SCM", SCM中选择Git,
10             Credentials中选择刚刚创建好的凭证
11
12  3. 用Maven来构建Java Application
13      3-1) 安装JDK和Maven
14              Manage Jenkins->Global Tool Configuration->JDK
15                  (可以选择外部的jdk目录, 如果是docker方式安装的jenkins一定要记得挂载外部jdk目录)
16              Manage Jenkins->Global Tool Configuration->Maven
17                  (可以选择外部的maven目录, 如果是docker方式安装的jenkins一定要记得挂载外部maven目录)

```

maven构建应用示例代码, 如下:

```

1  pipeline {
2      agent any
3
4      tools {
5          maven 'maven '3.6.3''
6      }
7
8      stages {
9          stage("Build") {
10             // Maven构建
11             steps {
12                 script {
13                     dir("${SOME_DIR}") {
14                         sh("mvn clean package -DskipTests=true")
15                         echo("Package xxx done")
16                     }
17                 }
18             }
19         }
20     }
21 }

```

4. 基本Groovy语法

- Groovy支持静态和动态类型

定义变量时, 我们还是习惯使用def

eg:

```
def a = 0
```

- Groovy语句不需要用分号结尾

- Groovy中方法调用可以省略括号

```
1 | print "Hello World"
```

- 支持命名参数

```
1 | def testMethod(String arg1, String arg2) {  
2 |     return arg1 + "_" + arg2  
3 | }  
4 |  
5 | // call the method  
6 | testMethod arg2 = "B", arg1 = "A"  
7 |  
8 | // return result  
9 | A_B
```

- 支持默认参数

```
1 | def printName(String name = "default name") {  
2 |     print "name is: ${name}"  
3 | }  
4 |  
5 | // call method  
6 | printName()  
7 |  
8 | // return result  
9 | name is: default name
```

- 同时支持双引号和单引号

要注意的是: 双引号支持占位符插值, 单引号不支持。

如下代码:

```
1 | def name = 'world'  
2 | print "hello ${name}"    // hello world  
3 | print 'hello ${name}'    // hello ${name}
```

- 支持三引号

三单引号和三双引号都支持换行
但是只有三双引号支持换行

代码如下:

```
1 def name = "Tom"
2
3 def tempStr = """
4 Today is a good day.
5 ${name} plans to go to the park.
6 He wants to have a good rest.
7 """
```

- 支持闭包

闭包可以作为参数传递给另一个方法

代码如下:

```
1 // 定义闭包
2 def codeBlock = { print "Hello Closure" }
3
4 def stage(String name, closure) {
5     println name
6     closure()
7 }
8
9 // 调用
10 stage("stage name", codeBlock)
11
12 // return result
13 stage name
14 Hello Closure
15
16 // Groovy的另一种简洁写法
17 stage("stage name") {
18     print "Hello Closure"
19 }
```

5. pipeline的组成和指令

pipeline的组成

关键字	含义
pipeline	整条流水线的逻辑
stage	流水线的阶段
steps	每个阶段的具体步骤
agent	指定流水线的执行位置(eg: 物理机/虚拟机/Docker容器)
post	pipeline执行失败后通过failure部分发送邮件到指定邮箱

6. 如何在pipeline中编写脚本

直接在steps块中编写Groovy代码是不可以的(eg: if-else)，会出错。

Jenkins pipeline专门提供了一个script步骤, 可以再script步骤中像写代码一样编写pipeline逻辑。

例子代码如下:

```
1 pipeline {
2   agent any
3   stages {
4     stage("Demo") {
5       steps {
6         script {
7           def browsers = ['chrome', 'firefox']
8           for (int i = 0; i < browsers.size(); i++) {
9             echo "Using the ${browsers[i]} browser"
10          }
11        }
12      }
13    }
14  }
15 }
```

pipeline内置基础步骤

- deleteDir

删除当前目录, 通常与dir一起使用。

- dir

切换到目录

(默认pipeline在工作空间目录下, dir可以切换到其它目录。)

```
1 dir("/myapp") {
2   ...
3 }
```

- fileExists

判断文件是否存在, 返回布尔类型。

- pwd

返回当前目录(同Linux的pwd命令)。

- writeFile

将内容写入指定文件。

```
1 参数列表:
2 file: 文件路径
3 text: 写入的内容
4 encoding(optional): 目标文件的编码(eg: Base64)
```

- readFile

读取文件的内容。

```
参数列表:
file: 文件路径
```

encoding(option): 读取文件的编码(eg: Base64)

- stash

将一些文件保存起来，以便同一次构建的其它步骤或阶段使用。

```
1 | stash(name: "test", includes: "test.txt")
```

- unstash

取出之前stash的文件。

```
1 | unstash("test")
```

命令相关步骤

- sh

执行shell命令。

参数名	含义
script	执行的shell脚本
encoding	输出日志的编码
returnStatus	脚本返回的状态码
returnStdout	true返回任务的标准输出

其它步骤

- error

主动报错，终止当前pipeline。

```
1 | error("error occurs")
```

- tool

使用预定义的工具。

参数名 | 含义

---- | ----

name | 工具名称

type(optional) | 工具安装类的全路径类名

```
1 | def t = tool name: 'docker', type: 'org.jenkinsci.plugins.docker.common.tools.DockerTool'
```

- timeout

代码块超时时间。

参数名 | 含义

---- | ----

time | 超时时间

unit(optional) | 时间单位 默认minutes 其它有milliseconds, seconds, hours, days

activity(optional) | true 当日志没有活动后，才算真正超时

```
1 | timeout(10)
```

- waitUntil

不断重复代码，直到条件满足。

- retry

重复执行块。

```
1 | // 重复10次
2 | retry(10) {
3 |   ...
4 | }
```

- sleep

让pipeline休眠一段时间。

默认是秒。

```
1 | // 休眠120秒
2 | sleep(120)
3 | // 休眠2分钟
4 | sleep(time: '2', unit: 'MINUTES')
```



2020-09-12 Xio

Groovy 和 Jenkins pipeline

公司项目一直使用的 Jenkins 作为 java/Android 的 CI/CD 工具, 之前需求比较简单, 直接使用 Jenkins 提供的管理后台 UI 操作就能完成。后来有一些新的构建需求, 我最初是使用 Python 处理的, 近期全部转到了 gradle 脚本 + Jenkins pipeline 的实现。这两者使用的都是 groovy, 使用过程中发现, 其实 groovy 很多操作比 Python 都要方便。

先介绍一些 groovy 操作便捷和需要注意的地方

File

join file path

```
1 def dataPath = Paths.get("/User/xxx","data/log")
2 def file = dataPath.toFile() // def file = new File(".")
3 file.eachFile { f ->
4
5 }
```

读写文件

```
1 new File(".").text
2 // even use `as String[]`
```

更多示例可以参考:[groovy-io](#)

正则

groovy 提供了很多字符串表示正则的写法:

- '\d+' 单引号
- "\d+" 双引号
- /\d+/ 斜杠
- \$\d+\$ 美元符号

*我使用斜杠较多, 可以应付很多需要转义的情况

groovy 还提供了很多特有的操作符:

- ~ 模式符, 后面紧跟正则表达式字符串即会转换为 `java.util.regex.Pattern` 对象, 如:

```
1  def p = ~/\d+/
2  // p 就是一个 Pattern 对象了
```

- =~ 正则搜寻符, 相当与执行正则的 `find` 操作, 返回 `Matcher` 对象, 在 groovy 里可以直接对其进行遍历:

```
1  def matcher = result =~ /versionCode='(\S+)'\s+versionName='(\S+)'/
2  // matcher[0] 就是搜索结果, 具体group结果在子数组里
3  def version_code = matcher[0][1]
4  def version_name = matcher[0][2]
```

- ==~ 正则匹配符, 相当与执行 `match` 操作, 返回值 `boolean` 类型

执行 shell 命令

直接 `'ls -al'.execute().text` 就可以执行并拿到结果, 非常方便, 更 Robust 的代码:

```
1  def result = new StringBuilder()
2  def error = new StringBuilder()
3  def cmd_newApkInfo = cmd.execute()
4  cmd_newApkInfo.consumeProcessOutput(result, error)
5  cmd_newApkInfo.waitFor()
```

List/Map

1. 创建

- list: `def a = []`
- map 就不太一样了: `def m = [:]`, `def m = [name: "John", age:123]`

2. find, findAll, collect find 和 findAll 起到过滤作用, 前者返回一个元素, 后者返回集合; 相当于其他语言里的 map 转换方法。

变量作用域的问题

```
1  // foo.groovy
2  x = 42
3  def y = 40
4  def f() {
5      println(x) // 42
6      println(y) // error
7  }
8
9  f()
```

这里top-level的俩变量是不一样的, x 没有类型声明, 将触发 [script binding](#), 变成“全局变量”(script-scope), 而 y 是本地变量, 除了在top-level直接使用外, 无法被其他方法调用。实际上看看 groovy 转换后的代码就一目了然了:

```
1  class foo{
2      // 成员变量, “全局可访问”
3      int x = 42;
4
5      void f() {
6          println(x) // 42
7          println(y) // error
8      }
9
10     // top-level 的代码都是放到 run 方法里的
11     void run(){
12         int y = 40;
13         f();
14     }
15
16     static main(args){
17         new foo().run();
18     }
19
20 }
```

但是, 不写任何声明的变量在 **gradle** 中是不能使用的, Android gradle 中如果实现 script-scope variable, 只能通过给 赋给 project/ext 的方式实现:

```
1  // build.gradle
2  ext.x = 42
3  def f(){
4      println x //42
5  }
```

其他

- 强制转换符 as , 这和 Kotlin 的还不一样:

```
1  Integer x = 123
2  // Java ,直接抛出 ClassCastException 异常
3  String s = (String) x
4  // Kotlin ,同 Java
5  String s = x as String
6  // Groovy ,正常转换到 String
7  String s = x as String
```

如果是不同类型, groovy 这实际上是创建了一个新的对象, 而这个转换方法(`asType()`)是需要实现的, 如果是自定义类型那需要我们自己实现。具体参考 [Coercion operator](#)

Jenkins Pipeline

Pipeline 是 Jenkins 提供的一种持续交付的模式, 我们可以通过编写 pipeline 脚本文件 (Jenkinsfile) 来高效地实现我们的 CD 流程。

Jenkinsfile 脚本文件分为两种:

Declarative Pipeline

集成了很多 Jenkins 新提供的很多 Api 功能, 可以声明式的便捷调用。

比如 jenkins 构建完成后你要发一条结果通知:

```
1  pipeline {
2      agent any
3      stages {
4          stage('No-op') {
5              steps {
6                  sh 'ls'
7              }
8          }
9      }
10     post {
11         always {
12             echo 'One way or another, I have finished'
13             deleteDir() /* clean up our workspace */
14         }
15         success {
16             echo 'I succeeded!'
17         }
18         unstable {
19             echo 'I am unstable :/'
20         }
21         failure {
22             echo 'I failed :('
23         }
24         changed {
25             echo 'Things were different before...'
26         }
27     }
28 }
```

post 就是完成后会执行的方法,

Scripted Pipeline

纯“脚本式”的风格, Jenkins 提供的 api 很少。

同样的上面的需求, scripted 实现是这样:

```
1  node {
2      try {
3          stage('No-op') {
4              sh 'ls'
```

```

5         }
6     }
7     catch (exc) {
8         echo 'I failed'
9     } finally {
10        if (currentBuild.result == 'UNSTABLE') {
11            echo 'I am unstable :/'
12        } else {
13            echo 'One way or another, I have finished'
14        }
15    }
16 }

```

注意, 只能使用 `try catch` 的方式实现(`currentBuild` 是由 `jenkins` 提供的, 可以直接使用)。

另外, 从上面俩例子也可以说明两者的格式也是不一样的, 前者整个构建过程需要这样表述:

```

1  pipeline {
2      agent any
3      stages {
4          stage("Sources"){
5              steps {
6
7              }
8          }
9          stage('Test') {
10             steps {
11                 //
12             }
13         }
14     }
15 }

```

pipeline 代码块是声明式的核心主体, 其包含多个 `stage`, 表示当前执行的阶段, 名称可以自定义, `stage` 里又由多个 `steps` 构成, 表示执行的具体步骤。

```

1  node {
2      stage('Build') {
3          //
4      }
5      stage('Test') {
6          //
7      }
8  }

```

node 代码块则是脚本式的核心主体, 其直接包含多个 `stage`(其实有没有 `stage` 不重要), 我们把具体的执行代码直接放到对应的 `stage` 下即可, 这里没有 `step` 等内容。写 `stage` 的主要目的是可以在 `jenkins` 有直观的 `ui` 展示, 比较方便。

pipeline 脚本主要基于 `groovy`, 声明式和脚本式的环境、插件都是共用的(调用方式上会有些不一样), 声明式出现的目的是 `Jenkins` 团队认为 `groovy` 的学习曲线还是比较陡峭, 所以整了个声明式的, 方便不了解 `groovy` 语法的开发者也可以使用 pipeline。[Syntax Comparison](#)

Jenkins 部分 Api 说明

1. dir

```
1  node {
2      // 工作目录 /workspace
3      stage("Build"){
4          dir('demo'){
5              // 创建并切换到 /workspace/demo 目录下
6              // git clone, credentialsId 需预先在 jenkins 中配置
7              git branch: 'develop',
8              url: '',
9              credentialsId: ''
10         }
11         // 自动回到工作目录
12         dir("demolib"){
13             // 创建并切换到 /workspace/demolib 目录下
14         }
15     }
16 }
```

2. Jenkins pipeline 里执行 shell 脚本注意事项:

- 插入 pipeline 中定义的变量, 需要用 `'''+variable+'''` 方式插入:

```
1  node {
2      def f = "rtm"
3      stage('Build'){
4          sh '''
5
6              ./gradlew '''+f+'''Release
7          '''
8      }
9
10 }
```

References

- [pipeline 简介](#)
- [pipeline examples](#)
- [Pipeline Syntax](#)
- [Trigger Jenkins pipeline for new tags in Github Repo](#)
- [Medium - Guide To Hacking Together Jenkins Scripted Pipeline](#)