Jenkins 持续集成与持续交付系列

Jenkins Pipeline全解析

主讲人：杨晓飞
yangxf@doudianyun.com

北京逗点云创科技有限公司

Learn Today
Lead Tomorrow

逗点云创

Jenkins Pipeline 全解析系列

第十一节： 声明式Pipeline语法之并行（一讲）：Sequential Stages（串行Stages）

Sequential Stages(串行Stages)

- 一个Stage有且仅有一个steps, parallel或stages
- Stages可以包含多个嵌套的Stage指令，且可以按顺序执行
- Parallel 块中不能再嵌套 Parallel 块
- 在并行(Parallel)块中的Stage区块可以使用该Stage的其他指令，如Agent，tool，when等

Sequential Stages示例

```
pipeline {
    agent none
    stages {
        stage('Non-Sequential Stage') {
            agent {
                label 'for-non-sequential'
            }
            steps {
                echo "On Non-Sequential Stage"
            }
        }
        stage('Sequential') {
            agent {
                label 'for-sequential'
            }
            environment {
                FOR_SEQUENTIAL = "some-value"
            }
```

```
stages {
    stage('In Sequential 1') {
        steps {
            echo "In Sequential 1"
        }
    }
    stage('In Sequential 2') {
        steps {
            echo "In Sequential 2"
        }
    }
    stage('Parallel In Sequential') {
        parallel {
            stage('In Parallel 1') {
                steps {
                    echo "In Parallel 1"
                }
            }
            stage('In Parallel 2') {
                steps {
                    echo "In Parallel 2"
                }
            }
        }
    }
```

Sequential Stages运行

Non-Sequential Stage	Sequential	In Sequential 1	In Sequential 2	Parallel In Sequential	In Parallel 1	In Parallel 2
51ms	48ms	47ms	39ms	69ms	80ms	75ms
43ms	46ms	33ms	31ms	44ms	62ms	59ms
59ms	51ms	61ms	48ms	95ms	99ms	92ms

```
                    In Sequential 2
                    [Pipeline] }
                    [Pipeline] // stage
                    [Pipeline] stage
                    [Pipeline] { (Parallel In Sequential)
                    [Pipeline] parallel
                    [Pipeline] { (Branch: In Parallel 1)
                    [Pipeline] { (Branch: In Parallel 2)
                    [Pipeline] stage
                    [Pipeline] { (In Parallel 1)
                    [Pipeline] stage
                    [Pipeline] { (In Parallel 2)
                    [Pipeline] echo
[In Parallel 1]     In Parallel 1
                    [Pipeline] }
                    [Pipeline] echo
[In Parallel 2]     In Parallel 2
                    [Pipeline] }
                    [Pipeline] // stage
                    [Pipeline] // stage
                    [Pipeline] }
                    [Pipeline] }
                    [Pipeline] // parallel
                    [Pipeline] }
                    [Pipeline] // stage
                    [Pipeline] }
                    [Pipeline] // withEnv
                    [Pipeline] }
                    [Pipeline] // stage
                    [Pipeline] End of Pipeline
                    Finished: SUCCESS
```

逗点云创

Jenkins Pipeline 全解析系列

第十一节： 声明式Pipeline语法之并行（二讲）：并行及示例

Parallel 并行块

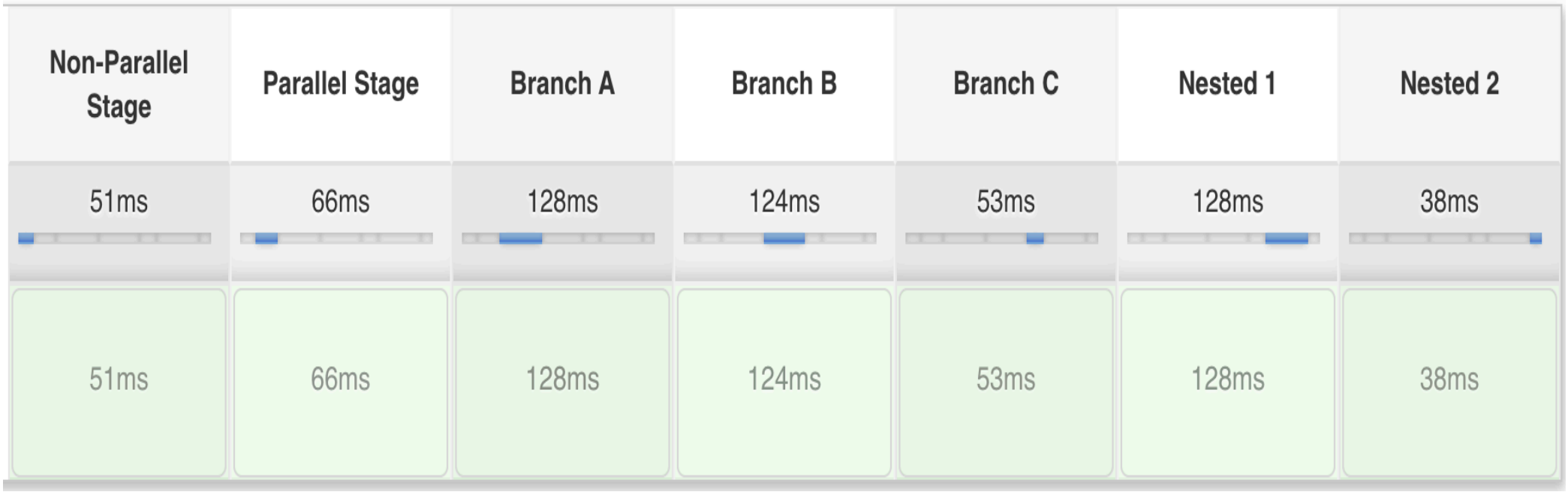
- 一个Stage有且仅有一个steps, parallel或stages
- Parallel 块中不能再嵌套 Parallel 块
- 任何包含 parallel 的Stage不能包含 Agent区块 或 tools指令
- 在Parallel 块中，通过设置 failFast 为 true ， 可以使当并行的其中一个Stage失败时，其他所有的 Parallel Stage都被终止
- 设置failfast的另一众方法是使用Pipeline 选项 parallelsAlwaysFailFast ( )

Parallel 示例

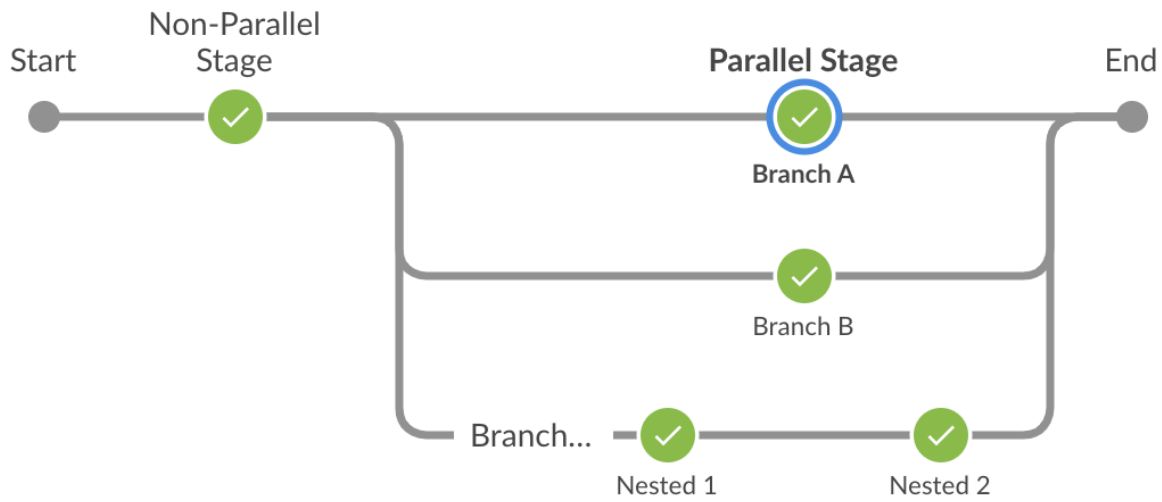
```
pipeline {
    agent any
    stages {
        stage('Non-Parallel Stage') {
            steps {
                echo 'This stage will be executed first.'
            }
        }
        stage('Parallel Stage') {
            when {
                branch 'master'
            }
            failFast true
            parallel {
                stage('Branch A') {
                    agent {
                        label "for-branch-a"
                    }
                    steps {
                        echo "On Branch A"
                    }
                }
```

```
                stage('Branch B') {
                    agent {
                        label "for-branch-b"
                    }
                    steps {
                        echo "On Branch B"
                    }
                }
                stage('Branch C') {
                    agent {
                        label "for-branch-c"
                    }
                    stages {
                        stage('Nested 1') {
                            steps {
                                echo "In stage Nested 1 within Branch C"
                            }
                        }
                        stage('Nested 2') {
                            steps {
                                echo "In stage Nested 2 within Branch C"
                            }
```

Parallel运行


Non-Parallel Stage
Parallel Stage
Branch A
Branch B
Branch C
Nested 1
Nested 2
51ms
66ms
128ms
124ms
53ms
128ms
38ms
51ms
66ms
128ms
124ms
53ms
128ms
38ms

Parallel运行




```
[Pipeline] stage
[Pipeline] { (Parallel Stage)
[Pipeline] parallel
[Pipeline] { (Branch: Branch A) (hide)
[Pipeline] { (Branch: Branch B)
[Pipeline] { (Branch: Branch C)
[Pipeline] stage
[Pipeline] { (Branch A)
[Pipeline] stage
[Pipeline] { (Branch B)
[Pipeline] stage
[Pipeline] { (Branch C)
[Pipeline] stage
[Pipeline] { (Nested 1)
[Pipeline] echo
```

Jenkins Pipeline 全解析系列

第十二节： Pipeline Steps 参考

Pipeline Step Reference

- Jenkins网站提供了大量的Pipeline Step Plugin 可以在Pipeline中根据需求来引用
- 参见 https://jenkins.io/doc/pipeline/steps/ 来查看完整的Pipeline Step Plugin 列表
- 也可访问 $Jenkins_host:8080/pipeline-syntax/html 查看Pipeline step 列表

Pipeline Step 参考页面

Pipeline Steps Refer

The following plugins offer Pipeline-compatible steps
for each step.

Read more about how to integrate steps into your P

- Acunetix 360 Scan Plugin
  - `NCScanBuilder`: Acunetix 360 Scan
- Agiletestware Pangolin Connector for TestRail
  - `pangolinTestRail`: Pangolin: Upload tes
- Alauda DevOps Pipeline Plugin
- GitLab Plugin
  - `acceptGitLabMR`: Accept GitLab Merge Request
  - `addGitLabMRComment`: Add comment on GitLab Merge Request
  - `gitlabBuilds`: Notify gitlab about pending builds
  - `gitlabCommitStatus`: Update the commit status in GitLab dependin
  - `updateGitlabCommitStatus`: Update the commit status in GitLab
- Google Chat Notification
  - `googlechatnotification`: Google Chat Notification
- Google Cloud Build Plugin
  - `googleCloudBuild`: Execute Google Cloud Build
- Google Cloud Storage plugin
  - `googleStorageUpload`: Google Storage Classic Upload
  - `googleStorageDownload`: Google Storage Download
  - `googleStorageBucketLifecycle`: Google Storage Bucket Lifecycle
  - `googleStorageBuildLogUpload`: Google Storage Build Log Upload

查看Pipeline Step 文档

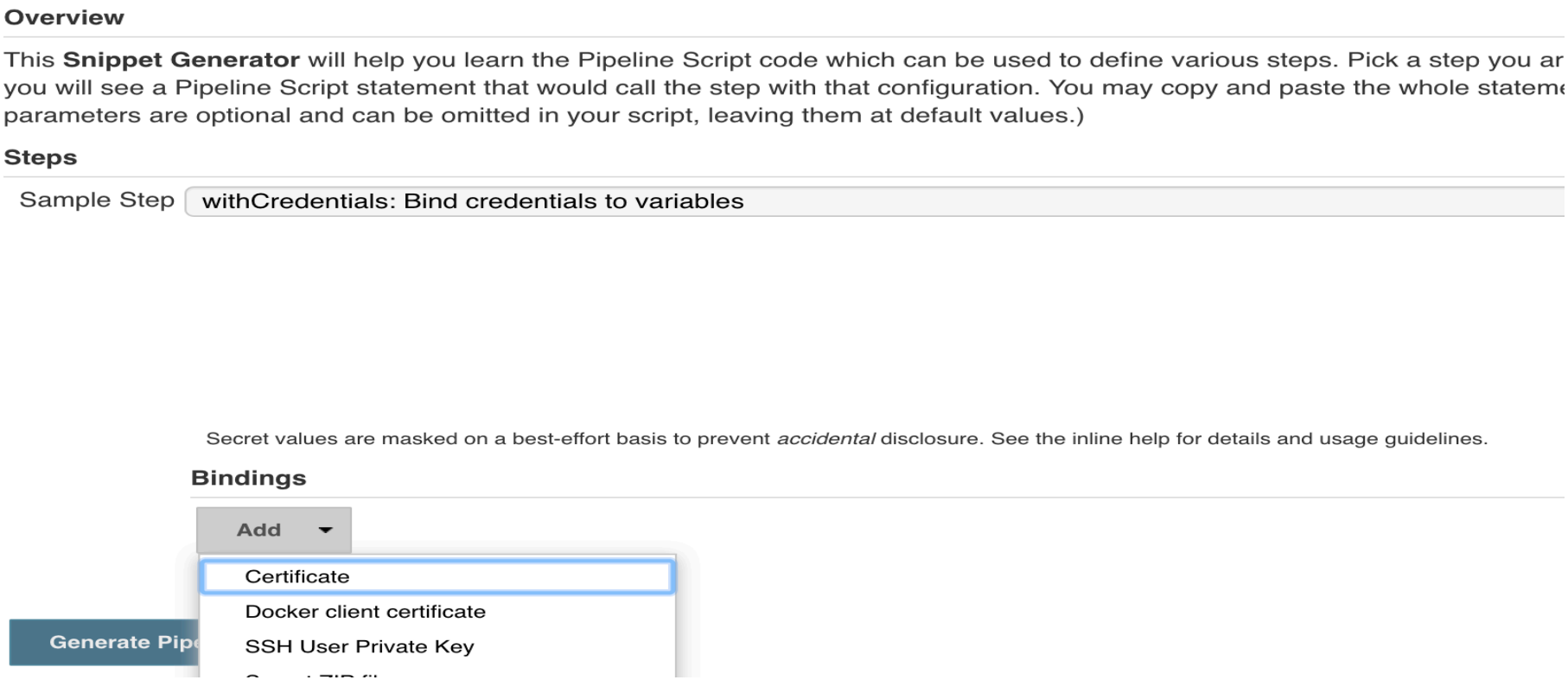
- withCredentials([certificate(aliasVariable: 'valias', credentialsId: 'cert_1', keystoreVariable: 'vkeystore', passwordVariable: 'vpwd')])

`certificate`
Sets one variable to the username and one variable to the password given in the credentials.
Warning: if the master or slave node has multiple executors, any other build running concurrently on the same node will be able to read the text of the secret, for example on Linux using `ps e`.

- `keystoreVariable`
  Name of an environment variable to be set to the temporary keystore location during the build.
  - **Type:** `String`
- `credentialsId`
  Credentials of an appropriate type to be set to the variable.
  - **Type:** `String`
- `aliasVariable` (optional)
  Name of an environment variable to be set to the keystore alias name of the certificate during the build.
  - **Type:** `String`
- `passwordVariable` (optional)
  Name of an environment variable to be set to the password during the build.
  - **Type:** `String`

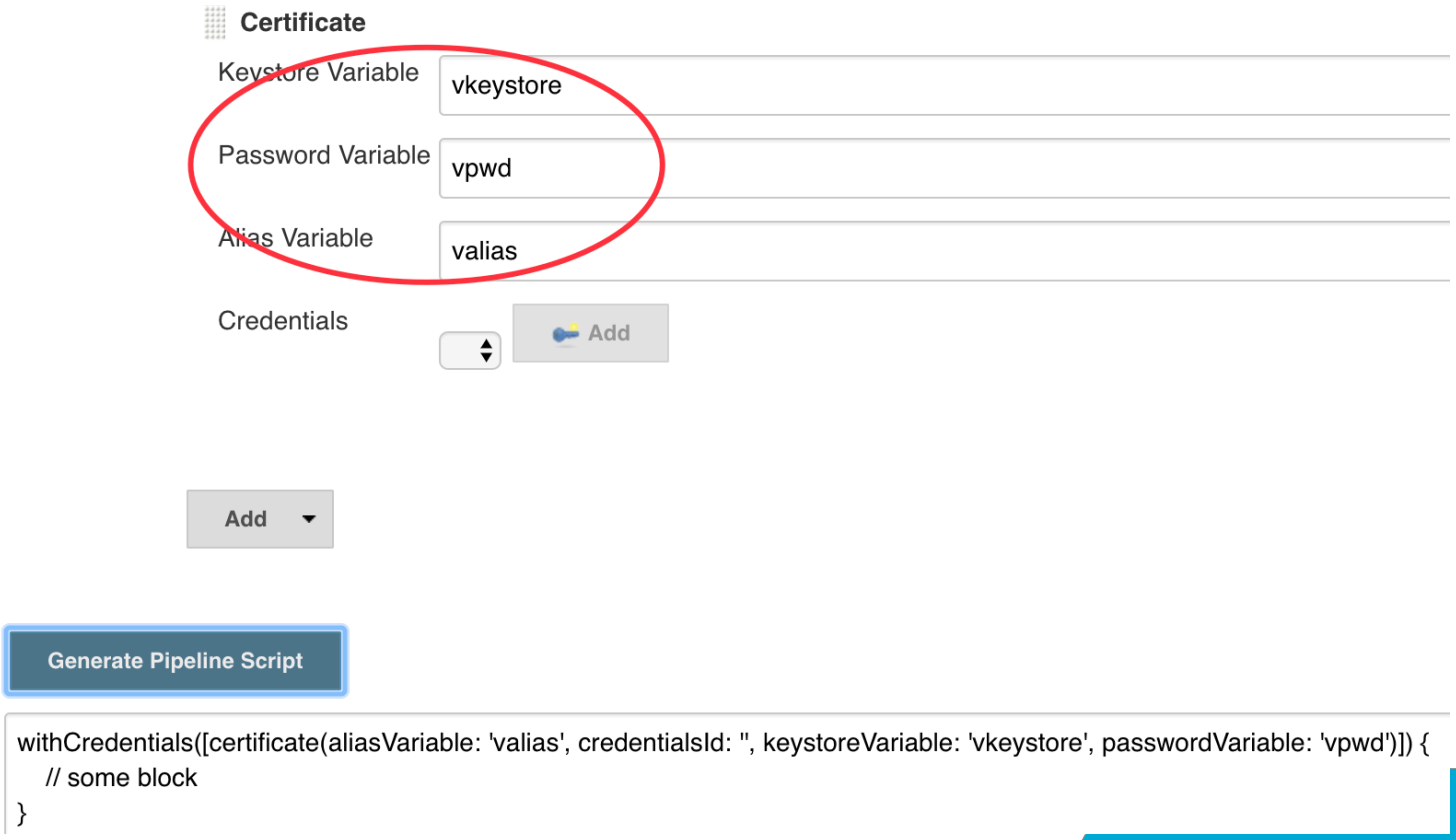
Pipeline 片段生成器示例

- Sample Step 选择 WithCredentials,选择“证书”安全凭据



Pipeline 片段生成器示例

- 输入必要的字段信息，点击生成Pipeline 脚本 按钮



复制并使用生成的Pipeline脚本

- 复制并使用生成的Pipeline脚本片段，注意替换相应的安全凭证ID，如下所示：

```
        }
    }

stage("cert") {
        steps {

withCredentials([certificate(aliasVariable: 'valias', credentialsId: 'cert_1', keys

    sh 'echo "keystore is $vkeystore"'

}

        echo "hello2"
    }
```

Use Groovy Sandbox

```
Started by user jenkin
Running in Durability level: MAX_SURVIVABILITY
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/jenkins_home/workspace/cert_example
[Pipeline] {
[Pipeline] stage
[Pipeline] { (foo)
[Pipeline] echo
hello
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { ("cert")
[Pipeline] withCredentials
Masking supported pattern matches of $valias or $vpwd or $vkeystore
[Pipeline] {
[Pipeline] sh
+ echo 'keystore is ****'
keystore is ****
[Pipeline] }
[Pipeline] // withCredentials
[Pipeline] echo
hello2
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

逗点云创

Jenkins Pipeline 全解析系列

第十三节： Groovy基础介绍

（一讲：Groovy基本语法和数据类型）

Groovy特点

- Groovy 使用方式基本与Java 代码的方式相同。在编写新应用程序时，Groovy 代码能够与 Java 代码很好地结合，也能用于扩展现有代码
- Groovy 的语法与 Java 语言的语法很相似，可将它想像成 Java 语言的一种更加简单、表达能力更强的变体
- 从学习的角度看，如果知道如何编写 Java 代码，那就已经了解 Groovy 了。Groovy 和 Java 语言的主要区别是：完成同样的任务所需的 Groovy 代码比 Java 代码更少
- 查看https://groovy-lang.org/ 了解更多内容

Groovy优点

- 简洁：完成相同的功能代码行数为java的1/3甚至更少，代码少，同时容易维护
- 兼容性：兼容java语法和包
- 便于测试和模拟，和JVM语言共存性较好
- 入门轻松 高效编程，注重实效思想
- Groovy背后有ThoughtWorks, SpringSource等公司的支持

Groovy “Hello World!”

```
package helloword

class HelloWorld {

    static main(args) {
        println"Hello World!!!"

    }

}
```

单独写println“Hello World!!!”也是能正确打印输出语句的

Groovy-Import语句

- import语句用于导入代码中使用的其他库包的功能，类似于Java语言
- 默认情况下，Groovy在代码中包含以下库包

```
import java.lang.*
import java.util.*
import java.io.*
import java.net.*

import groovy.lang.*
import groovy.util.*

import java.math.BigInteger
import java.math.BigDecimal
```

Groovy-注释

- 和Java语法类似
- 通过在行中的任何位置使用//标识来单行注释
- 多行注释以/ *开头，* /标识多行注释的结尾

```
class Example {
   static void main(String[] args) {
      // Using a simple println statement to print output to the console
      println('Hello World');
   }
}
```

```
class Example {
   static void main(String[] args) {
      /* This program is the first program
      This program shows how to display hello world */
      println('Hello World');
   }
}
```

Groovy-Identifiers（标识符）

- 标识符用于定义变量，函数或其他用户自定义变量
- 标识符以字母，美元符号或下划线开头，不能以数字开头
- def是Groovy中用于定义标识符的关键字

下面是一些合理的标识符例子：

```
def employeename
def student1
def student_name
```

```
class Example {
   static void main(String[] args) {
      // One can see the use of a semi-colon after each statement
      def x = 5;
      println('Hello World');
   }
}
```

Groovy-关键词

as	assert	break	case
catch	class	const	continue
def	default	do	else
enum	extends	false	Finally
for	goto	if	implements
import	in	instanceof	interface
new	pull	package	return
super	switch	this	throw
throws	trait	true	try
while			

Groovy-内置数据类型

- Byte:用于表示字节值
- Short:用于表示一个short数字, 一个例子是10
- Int:表示整数。 例如1234
- Long:用于表示长整数。 例如10000090
- Float:用于表示32位浮点数, 一个例子是12.34
- Double:用于表示64位浮点数，这些浮点数是有时可能需要的更长的十进制数表示形式。 一个示例是12.345656
- char:定义一个字符, 例如“ a”
- Boolean:表示布尔值，可以为true或false
- String:文本文字，例如“ Hello World”

逗点云创

Jenkins Pipeline 全解析系列

第十三节： Groovy基础介绍
（二讲：Groovy运算符）

数学运算符

Operator	Description	Example
+	Addition of two operands	1 + 2 will give 3
–	Subtracts second operand from the first	2 – 1 will give 1
*	Multiplication of both operands	2 * 2 will give 4
/	Division of numerator by denominator	3 / 2 will give 1.5
%	Modulus Operator and remainder of after an integer/float division	3 % 2 will give 1

++	Incremental operators used to increment the value of an operand by 1	int x = 5;
		x++;
		x will give 6

--	Incremental operators used to decrement the value of an operand by 1	int x = 5; x--; x will give 4

关系运算符

Operator	Description	Example
==	Tests the equality between two objects	2 == 2 will give true
!=	Tests the difference between two objects	3 != 2 will give true
<	Checks to see if the left objects is less than the right operand.	2 < 3 will give true
<=	Checks to see if the left objects is less than or equal to the right operand.	2 <= 3 will give true
>	Checks to see if the left objects is greater than the right operand.	3 > 2 will give true
>=	Checks to see if the left objects is greater than or equal to the right operand.	3 >= 2 will give true

逻辑运算符

Operator	Description	Example
&&	This is the logical “and” operator	true && true will give true
||	This is the logical “or” operator	true || true will give true
!	This is the logical “not” operator	!false will give true

```
class Example {
   static void main(String[] args) {
      boolean x = true;
      boolean y = false;
      boolean z = true;

      println(x&&y);
      println(x&&z);

      println(x||z);
      println(x||y);
      println(!x);
   }
}
```

```
false
true
true
true
false
```

位与运算符

Sr.No	Operator & Description
1	**&** This is the bitwise “and” operator
2	**|** This is the bitwise “or” operator
3	**^** This is the bitwise “xor” or Exclusive or operator
4	**~** This is the bitwise negation operator

p	q	p & q	p | q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

赋值运算符

Operator	Description	Example
+=	This adds right operand to the left operand and assigns the result to left operand.	def A = 5 A+=3 Output will be 8
-=	This subtracts right operand from the left operand and assigns the result to left operand	def A = 5 A-=3 Output will be 2
*=	This multiplies right operand with the left operand and assigns the result to left operand	def A = 5 A*=3 Output will be 15
/=	This divides left operand with the right operand and assigns the result to left operand	def A = 6 A/=3 Output will be 2
%=	This takes modulus using two operands and assigns the result to left operand	def A = 5 A%=3 Output will be 2

范围运算符

- Groovy支持范围运算符（..），下面给出了范围运算符的一个简单示例

```
class Example {
   static void main(String[] args) {
      def range = 5..10;
      println(range);
      println(range.get(2));
   }
}
```

```
[5, 6, 7, 8, 9, 10]
7
```

逗点云创

Jenkins Pipeline 全解析系列

第十三节： Groovy基础介绍
（三讲：Groovy循环和条件判断）

Groovy- While 循环

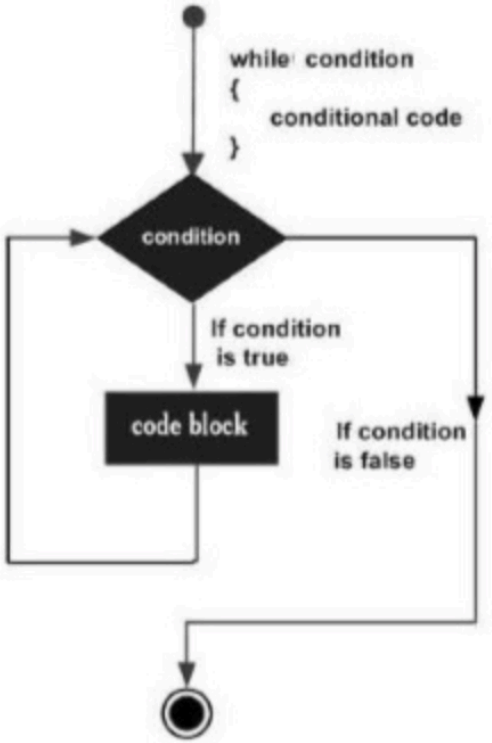
和Java 类似，语法如下：

```
while(condition) {
   statement #1
   statement #2
   ...
}
```

```
class Example {
   static void main(String[] args) {
      int count = 0;

      while(count<5) {
         println(count);
         count++;
      }
   }
}
```

```
0
1
2
3
4
```



Groovy- For 循环

和Java 类似，语法如下：

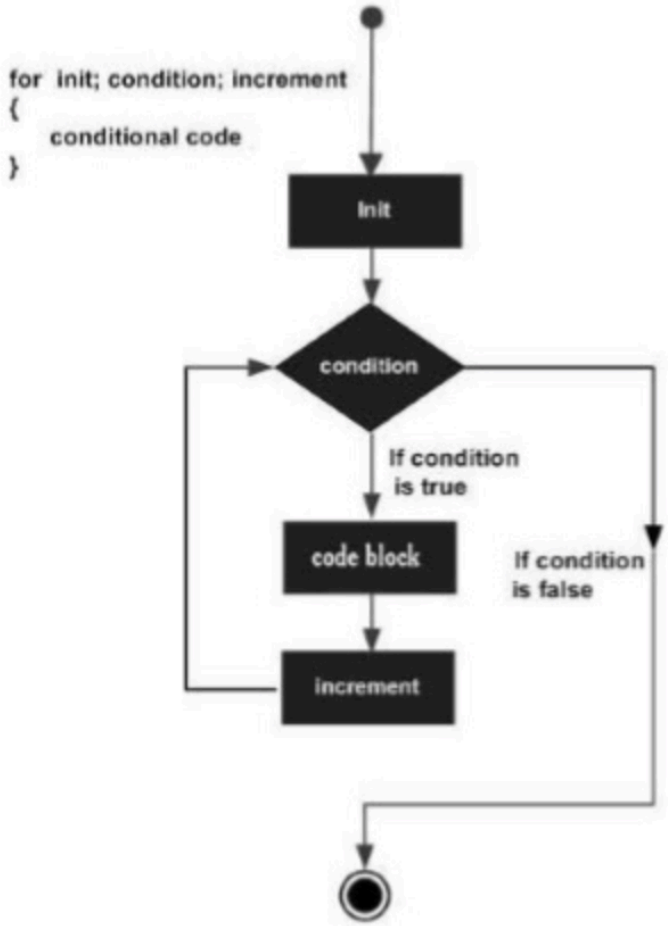
```
for(variable declaration;expression;Increment) {
   statement #1
   statement #2
   …
}
```

```
class Example {
   static void main(String[] args) {

      for(int i = 0;i<5;i++) {
         println(i);
      }

   }
}
```

```
0
1
2
3
4
```



Groovy- For In 循环

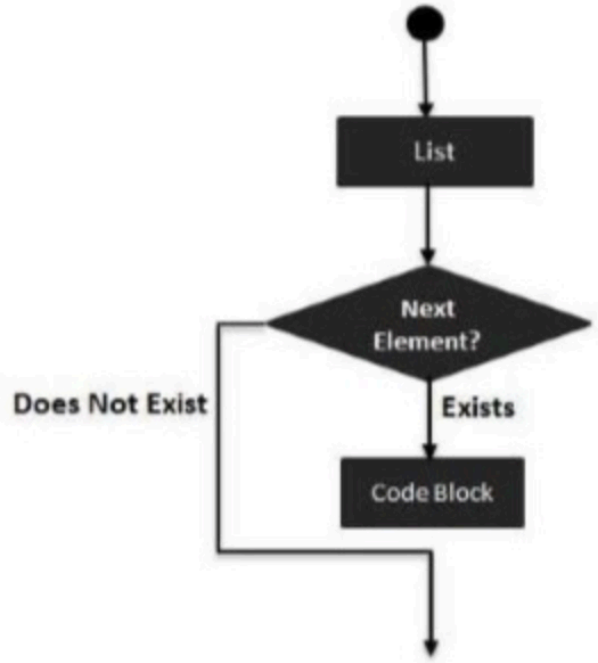
- for-in语句用于迭代一组数值，如下：

```
for(variable in range) {
   statement #1
   statement #2
   …
}
```

```
class Example {
   static void main(String[] args) {
      int[] array = [0,1,2,3];

      for(int i in array) {
         println(i);
      }
   }
}
```

```
0
1
2
3
```



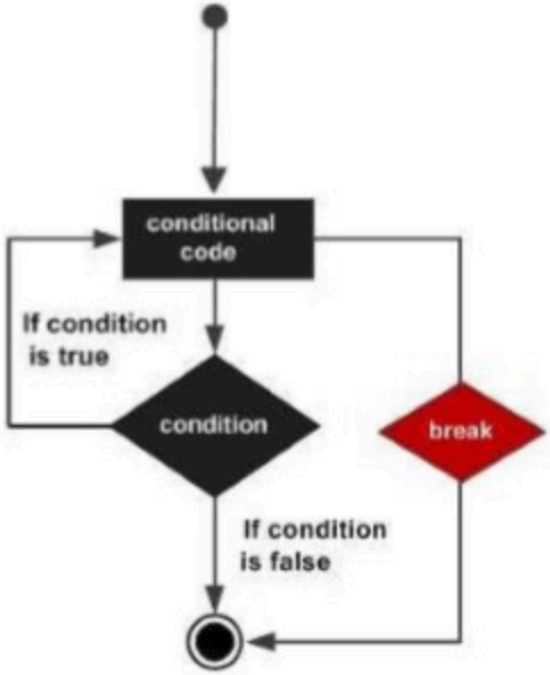
Groovy- Break 语句

- Break语句用于更改循环内的控制流，与while和for语句一起使用
- 循环中的任何一个执行break语句都会导致该循环立即终止

```
class Example {
   static void main(String[] args) {
      int[] array = [0,1,2,3];

      for(int i in array) {
         println(i);
         if(i == 2)
         break;
      }
   }
}
```

```
0
1
2
```



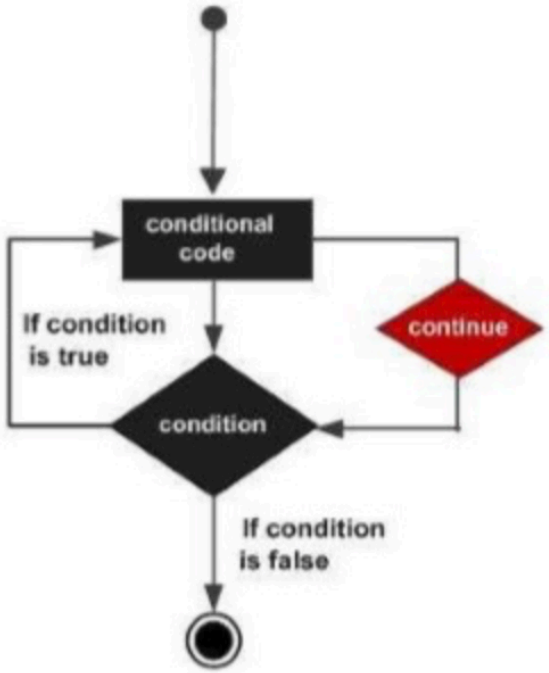
Groovy- Continue语句

- Continue语句是对break语句的补充，与while和for循环一起使用
- 当执行continue语句时，控制立即传递到最近的封闭循环的是否循环判断条件，以确定循环是否应继续

```
class Example {
   static void main(String[] args) {
      int[] array = [0,1,2,3];

      for(int i in array) {
         println(i);
         if(i == 2)
         continue;
      }
   }
}
```

```
0
1
2
3
```



逗点云创

Jenkins Pipeline 全解析系列

第十三节： Groovy基础介绍
（四讲：Groovy方法和异常处理）

Groovy- Method方法

- Groovy中的方法使用返回值类型和def关键字定义，方法可接收任意数量的参数，定义参数时不必明确定义类型
- 可加诸如public，private和protected的修饰符。 默认情况下，如未提供方法修饰符，则该方法为public

  如下所示最简单的方法定义：

```
def methodName() {
   //Method code
}
```

```
class Example {
   static def DisplayName() {
      println("This is how methods work in groovy");
      println("This is an example of a simple method");
   }

   static void main(String[] args) {
      DisplayName();
   }
}
```

Groovy- Method 参数

方法定义：

```
def methodName(parameter1, parameter2, parameter3) {
   // Method code goes here
}
```

```
def someMethod(parameter1, parameter2 = 0, parameter3 = 0) {
   // Method code goes here
}
```

```
class Example {
   static void sum(int a,int b) {
      int c = a+b;
      println(c);
   }

   static void main(String[] args) {
      sum(10,5);
   }
}
```

```
class Example {
   static void sum(int a,int b = 5) {
      int c = a+b;
      println(c);
   }

   static void main(String[] args) {
      sum(6);
   }
}
```

Groovy- Method 返回值

和Java语法一样

```
class Example {
   static int sum(int a,int b = 5) {
      int c = a+b;
      return c;
   }

   static void main(String[] args) {
      println(sum(6));
   }
}
```

Groovy- Try/Catch 块

- 使用try和catch关键字的组合来捕获异常，在可能产生异常的代码前后放置了try / catch块，如下：

```
try {
   //Protected code
} catch(ExceptionName e1) {
   //Catch block
}
```

```
class Example {
   static void main(String[] args) {
      try {
         def arr = new int[3];
         arr[5] = 5;
      } catch(Exception ex) {
         println("Catching the exception");
      }

      println("Let's move on after the exception");
   }
}
```

Groovy- Finally 块

- finally块位于try/catch块之后，无论是否发生异常，始终都会执行finally代码块

```
try {
   //Protected code
} catch(ExceptionType1 e1) {
   //Catch block
} catch(ExceptionType2 e2) {
   //Catch block
} catch(ExceptionType3 e3) {
   //Catch block
} finally {
   //The finally block always executes.
}
```

```
class Example {
   static void main(String[] args) {
      try {
         def arr = new int[3];
         arr[5] = 5;
      } catch(ArrayIndexOutOfBoundsException ex) {
         println("Catching the Array out of Bounds exception");
      }catch(Exception ex) {
         println("Catching the exception");
      } finally {
         println("The final block");
      }

      println("Let's move on after the exception");
   }
}
```

Jenkins Pipeline 全解析系列

第十四节： Scripted Pipeline基本语法简介

Scripted Pipeline语法基本概念

Pipeline通过Domain Specific Language（DSL）syntax定义Pipeline as Code并且实现持续交付的目的。Pipeline的代码定义了整个构建过程，通常包括构建应用程序，测试然后交付应用程序的阶段。下面是Scripted Pipeline语法中基本概念：

- Stage：一个Pipeline可以划分成若干个Stage，每个Stage代表一组操作，例如：“Build”，“Test”，“Deploy”。注意，Stage是一个逻辑分组的概念，可以跨多个Node（Agent）
- Node：一个Node就是一个Jenkins节点，或者是Master，或者是Agent，是执行Step的具体运行环境
- Step：Step是最基本的操作单元，小到创建一个目录，大到构建一个Docker镜像，由各类Jenkins Plugin提供，例如：sh ‘make’

Scripted Pipeline注意事项

- Scripted Pipeline 是基于Groovy语言实现
- 在一个节点上执行Pipeline 定义的任务有如下好处：
  - 将要执行的Step被添加到Jenkins队列中，如Node 上Executor被释放，Step就立即运行
  - 创建工作区，在该工作区中可以处理从源代码管理中检出的文件

Pipeline基础语法示例

Jenkinsfile (Declarative Pipeline)

```
pipeline {
    agent any (1)
    stages {
        stage('Stage 1') {
            steps {
                echo 'Hello world!' (2)
            }
        }
    }
}
```

Jenkinsfile (Scripted Pipeline)

```
node { (3)
    stage('Stage 1') {
        echo 'Hello World' (2)
    }
}
```

- Agent:Jenkins为Pipeline分配的执行空间和工作空间
- Echo在控制台输出字符串
- Node作用等同于Agent

Scripted Pipeline基本结构

Jenkinsfile (Scripted Pipeline)

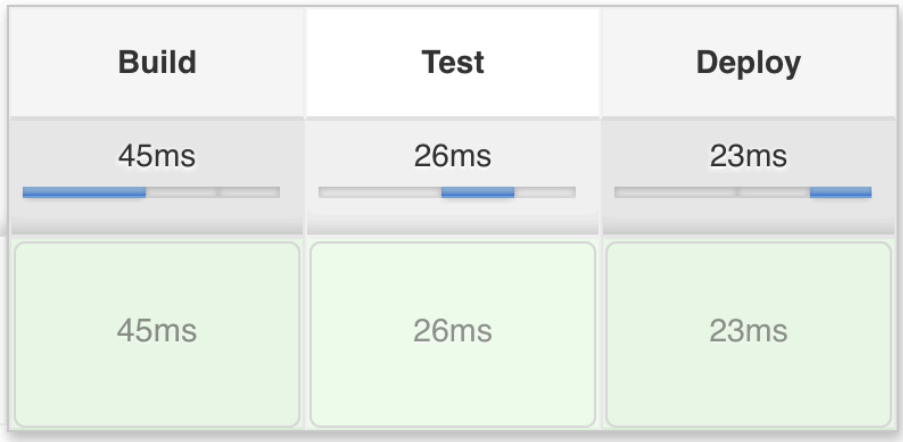
```
node {  ❶
    stage('Build') { ❷
        // ❸
    }
    stage('Test') { ❹
        // ❺
    }
    stage('Deploy') { ❻
        // ❼
    }
}
```

1. 在任何可用的Node上执行此Pipeline或其任何阶段
2. 定义“构建”阶段
3. 执行与“构建”阶段相关的步骤
4. 定义“测试”阶段
5. 执行与“测试”阶段相关的步骤
6. 定义“部署”阶段
7. 执行与“部署”阶段相关的步骤

Scripted Pipeline示例

Jenkinsfile (Scripted Pipeline)

```
node {
    stage('Build') {
        echo 'Building....'
    }
    stage('Test') {
        echo 'Testing....'
    }
    stage('Deploy') {
        echo 'Deploying....'
    }
}
```




```
Building....
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Test)
[Pipeline] echo
Testing....
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Deploy)
[Pipeline] echo
Deploying....
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

Jenkins Pipeline 全解析系列

第十五节： Pipeline 脚本示例及文档

Pipeline 文档

- Pipeline_Basic
- Pipeline_Script 部分
  包括Groovy 基本语法部分

Pipeline 示例


cert.groovy


credential.groovy


cron.groovy


env.groovy


input.groovy


upstream.groovy


when.groovy


pkcs12.cmd


post.groovy


script.groovy


sequential.groovy


stages.groovy

Pipeline 示例


```
pipeline {
  agent {
    dockerfile {
      /*
        * The Default is "Dockerfile" but this can be changed.
        * This will build a new container based on the contents o
        * and run the pipline inside this container
        */
      filename "Dockerfile.alternate"
      args "-v /tmp:/tmp -p 8000:8000"
    }
  }
  stages {
    stage("foo") {
      steps {
        sh 'cat /hi-there'
        sh 'echo "The answer is 42"'
      }
    }
```

感谢大家

逗点云创专注于计算机领域和相关技术领域（大数据集成，云计算和人工智能等）的知识分享和线上线下的知识推广

联系我们：www.doudianyun.com

contact@doudianyun.com