

pipeline进阶

Groovy语法

概述

Groovy 是一种基于Java平台的面向对象语言。Groovy 的语法和 Java 非常的相似，可以使用现有的 Java 库来进行 Groovy 开发。可以将它想像成 Java 语言的一种更加简单、表达能力更强的变体。用 Groovy 编写的任何内容都可以编译成标准的 Java 类文件并在 Java 代码中重用。类似地，用标准 Java 代码编写的内容也可以在 Groovy 中重用。所以，可以轻易地使用 Groovy 为 Java 代码编写单元测试。而且，如果用 Groovy 编写一个方便的小工具，那么也可以在 Java 程序中使用这个小工具。

我们为什么要学习 Groovy 语言呢？

Groovy 是一种更有生产力的语言。它具有松散的语法和一些特殊功能，能够加快编码速度。

Jenkins的pipeline脚本都是由groovy开发的，想要更完善的掌握pipeline就必须了解groovy语法，好在我们都有编程语言的经验，因此学习Groovy语言的难度并不高

Groovy 语言的一些特点：

- Groovy 的松散的 Java 语法允许省略分号和 return 关键字。
- 变量的类型和方法的返回值也是可以省略的。
- 方法调用时，括号也是可以省略的。
- 除非另行指定，Groovy 的所有内容都为 public。
- Groovy 允许定义简单脚本，同时无需定义正规的 class 对象。
- Groovy 在普通的常用 Java 对象上增加了一些独特的方法和快捷方式，使得它们更容易使用。
- Groovy 语法还允许省略变量类型。

[官方网站](#)

[Groovy API 文档](#)：遇到不懂的类或者方法，这个是好帮手。

字符串

在Groovy中有两种风格的字符串：String (java.lang.String) 和GString (groovy.lang.GString) 。GString允许有占位符而且允许在运行时对占位符进行解析和计算。

这里我们只介绍对占位符的一种使用：嵌入表达式

嵌入表达式

```
def worldStr = "world"
def helloStr = "Hello ${worldStr}";
println helloStr
println "value: ${3+3}"
```

除了\${}占位符的{}其实是可以省去的。

占位符里面可以包含一个闭包表达式。

变量类型定义和方法声明

在 Java 中，变量是必须指定类型的，但是在 Groovy 中，所有的变量类型都可以用 `def` 去指定，Groovy 会根据对象的值来判断它的类型。

```
def helloStr = "Hello world";
def a = 1, b = 2
println helloStr
println a + b
```

函数的返回值的类型当然也可以用 `def` 来声明：

```
def getStr() {
    return "Hello world"
}
```

在声明函数时，参数变量的类型是可以省略的：

```
def add(arg1, arg2) { //在声明函数时，参数变量的类型是可以省略的：
    return arg1+arg2
}
```

前面我们说过，方法返回值的关键字 `return` 是可以省略的：

```
def add(arg1, arg2) {
    arg1+arg2 //方法返回值的关键字 `return` 是可以省略的：
}
```

方法调用时括号是可以省略的，见 `println a + b` 的调用。

在 Groovy 中，类型是弱化的，所有的类型都可以动态推断，但是 Groovy 仍然是强类型的语言，类型不匹配仍然会报错；

上述两个类完全一致，只要有属性就有Getter/Setter；同理，只要有Getter/Setter，那么它就有隐含属性。

groovy内置数据类型

Groovy提供多种内置数据类型。以下是在Groovy中定义的数据类型的列表 -

- ***byte*** -这是用来表示字节值。例如2。
- ***short*** -这是用来表示一个短整型。例如10。
- **int** -这是用来表示整数。例如1234。
- ***long*** -这是用来表示一个长整型。例如10000090。
- ***float*** -这是用来表示32位浮点数。例如12.34。
- ***double*** -这是用来表示64位浮点数，这些数字是有时可能需要的更长的十进制数表示。例如12.3456565。
- ***char*** -这定义了单个字符文字。例如“A”。
- ***Boolean*** -这表示一个布尔值，可以是true或false。
- ***String*** -这些是以字符串的形式表示的文本。例如，“Hello World”的。

Groovy条件判断与循环

条件判断if(condition)...else...

```
if(worldStr == "中文"){
    println "正确"
}else{
    println "错误"
}
```

普通for循环 和java其他语言类似

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

for in循环

```
def list1=[1,2,3,4,6] //定义一个列表
for(i in list1){      //遍历列表
    println(i);
}
```

while循环

```
int count = 0;

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

Groovy文件操作

读文件

```
def file = new File('Jenkinsfile') //获取文件对象
println file.text                    //打印文件对象文本
```

写文件

```
File file2 = new File('testfile')
def writer = file2.newPrintWriter('utf-8') //定义一个文件书写器
writer.write('测试开发运维课程') //写入内容
writer.flush() // 内容输出到文件
writer.close() // 关闭文件流
```

简洁操作

```
def file = new File("testfile")

file.withWriter('utf-8') { writer ->
    writer.write('写入中文') //不带回车
    writer.writeLine('写入中文') //带回车
}
```

Groovy处理yaml文件

读yaml: 将yaml文件转化成groovy对象格式, 如: 字典, 列表, 字符串, 数字等

test.yml

```
build: build1
test: test1
push: push1
deploy: deploy1
```

groovy

```
// 导包
import groovy.yaml.YamlSlurper

def file = new File('test.yml')
def ys = new YamlSlurper() //yaml文件转化器
def data = ys.parseText(file.text) // 解析yaml文件内容
println data.keySet() //获取字典所有的 key
println data.values() // 获取所有的value
println data['build'] // 取key对应的value
println data.build // 取key对应的value
```

Groovy异常处理

try catch语句块

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

```
try {
    def file = new File("abcfile")
    println file.text
} catch(Exception e1) {
    println('没找到文件')
}
```

Groovy 面向对象

在Groovy中，如在任何其他面向对象语言中一样，存在类和对象的概念以表示编程语言的对象定向性质。Groovy类是数据的集合和对该数据进行操作的方法。在一起，类的数据和方法用于表示问题域中的一些现实世界对象。

Groovy中的类声明了该类定义的对象的状态（数据）和行为。因此，Groovy类描述了该类的实例字段和方法。

以下是Groovy中的一个类的示例。类的名称是Student，它有两个字段 - **StudentID**和**StudentName**。在main函数中，我们创建一个这个类的对象，并将值分配给对象的**StudentID**和**StudentName**。

```
class Student {
    int StudentID;
    String StudentName;

    static void main(String[] args) { //入口方法，想测试该类或者其他类的方法可以声明一个main函数
        Student st = new Student();
        st.StudentID = 1;
        st.StudentName = "Joe"
        println st.StudentID //也可以加上括号
        println st.StudentName
    }
}
```

外部调用类属性和方法

```
class Student {
    int StudentID;
    String StudentName;

    def study(course){
        println "学习 ${course}"
    }
}

Student st = new Student();
st.StudentID = 1;
st.StudentName = "Joe"
println st.StudentID //也可以加上括号
println st.StudentName

st.study('测试开发')
```

使用自定义库

写脚本就会遇到重复的情况，就会封装，如果我们写的函数还OK，那么可以封装到一个库文件中，然后其他库引用即可。

库文件 utils.groovy

```
import groovy.yaml.YamlSlurper

def readYml(path){
    def file = new File(path)
    def ys = new YamlSlurper()
    def data = ys.parseText(file.text)
    return data
}
```

主函数 main.groovy

```
evaluate(new File("utils.groovy")) // 导入库文件路径

def tools = new utils()           // 对象形式调用库
println tools.readYml('test.yml')
```

Jenkins共享库

Jenkinsfile

```
node('haiwen_linux'){ //node表示任务执行所在的机器--通过节点标签指定
    checkout scm
    def tools = load ('utils.groovy')
    tools.run_pipe()
}
```

utils.groovy

```
def run_pipe(){
    stage('单元测试'){
        def str = '执行单元测试'
        println str
    }
    stage('构建镜像'){
        println '执行构建镜像'
        //writefile('testfile','执行构建镜像')
        //println readfile('testfile')
    }
    stage('推送镜像'){
        println '执行推送镜像'
    }
    stage('更新服务'){
        sh 'touch testfile'
        sh "echo '执行更新服务' > testfile"
    }
}

return this //加上这句话
```

附录

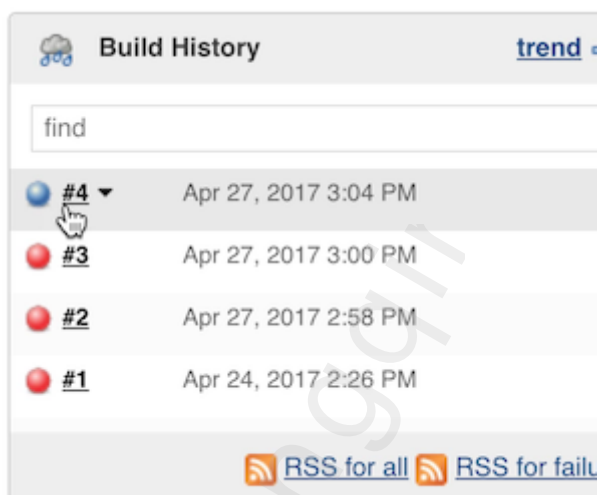
使用回放功能调试流水线

典型的流水线将在经典的 Jenkins web UI 中被定义, 或者提交一个 `Jenkinsfile` 到源代码控制。不幸的是, 这两种方法都不适合于流水线的快速迭代或原型开发。"Replay" 特性允许快速修改和执行现有流水线, 而不需要修改流水线配置或创建新的提交。

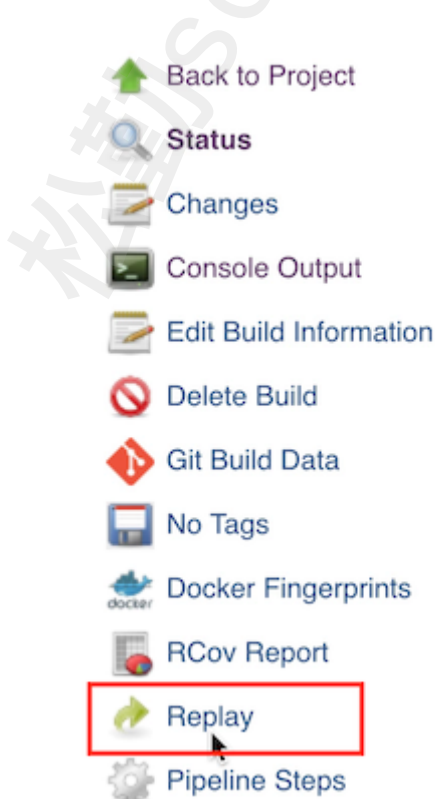
用法

使用"Replay" 特性:

1. 在构建历史中选择一个之前已完成的运行。



2. 点击右侧菜单的 "Replay" 。



3. 修改并点击 "Run"。在该示例中, 将 "ruby-2.3" 改为 "ruby-2.4"。

Replay #4

Allows you to replay a Pipeline build with a modified script. If any load steps were run, you can also modify the scripts they loaded.

```

Main Script
1  #!groovy
2
3  pipeline {
4    agent {
5      // Use docker container
6      docker {
7        image 'ruby:2.4'
8      }
9    }
10   options {
11     // Only keep the 10 most recent builds
12     buildDiscarder(logRotator(numToKeepStr: '10'))
13   }
14   stages {
15     stage ('Install') {

```

[Pipeline Syntax](#)

Run

4. 检查更改的结果

一旦你满意本次更改, 你可以使用回放来再次查看他们, 复制他们到你的流水线作业或 `Jenkinsfile` 中, 然后使用平常的工程流程提交它们。

特性

- **Can be called multiple times on the same run** - 允许对不同的更改进行简单的并行测试。
- **Can also be called on Pipeline runs that are still in-progress** - 只要流水线包含语法正确的 Groovy 并且能够启动, 就可以回放。
- **Referenced Shared Library code is also modifiable** - 如果一个流水线运行引用了一个 [Shared Library](#), 那么共享库中的代码也将作为回放页面的一部分显示和修改。

限制

- **Pipeline runs with syntax errors cannot be replayed** - 意思是他们的代码不能被查看, 任何在他们里面的改变都不能被检索。 当使用回放进行更重要的修改时, 在运行之前保存你的变更到一个 Jenkins 之外的文件或编辑器中。 参考 [JENKINS-37589](#)
- **Replayed Pipeline behavior may differ from runs started by other methods** - 对于不属于多分支流水线的流水线, 原始运行和回放运行的 提交信息可能会不同。 参考 [JENKINS-36453](#)

vscode 编写 groovy

- Groovy 需要 Java 的支持, 所以需要安装 JDK (<https://www.oracle.com/technetwork/java/javase/downloads/index.html>) 下载一个 jdk 版本, 比如 jdk1.8, 安装完成后, 配置一个环境变量: JAVA_HOME=C:\java\jdk1.8\bin (假设 jdk 安装在 C:\java\jdk1.8)

去 Groovy 官网 <https://bintray.com/artifact/download/groovy/maven/apache-groovy-binary-3.0.7.zip> 下载 Groovy SDK, 目前最新的稳定版 3.0.8, 网速慢的可以到课程云盘下载

下载后解压到目录中, 在环境变量中配置一个变量: GROOVY=C:\Tools\groovy-3.0.8\bin (假设 groovy 解压在 C:\Tools\groovy-3.0.8) 或者配置到 path 中

打开 CMD 敲 groovy 检测


```

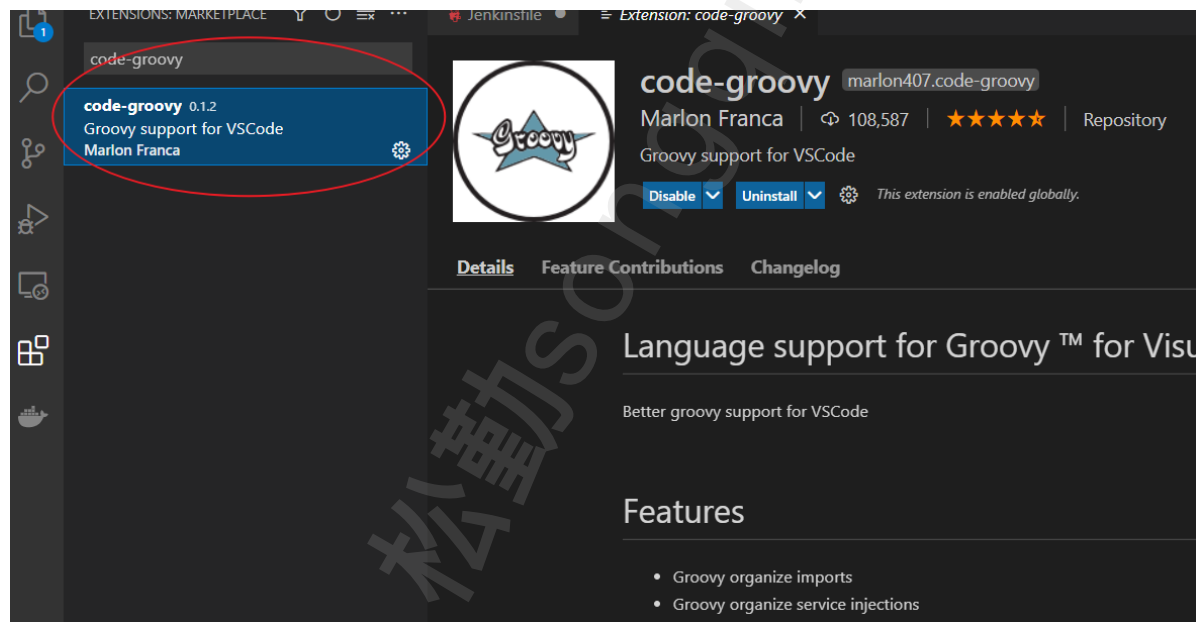
Microsoft Windows [版本 10.0.19042.928]
(c) Microsoft Corporation. 保留所有权利。

C:\Users\Shone>groovy
error: neither -e or filename provided
Usage: groovy [options] [filename] [args]
The Groovy command line processor.
    -cp, -classpath, --classpath=<path>
        Specify where to find the class files - must be
        first argument
    -D, --define=<property=value>
        Define a system property
    --disableopt=optlist[,optlist...]
        Disables one or all optimization elements; optlist
        can be a comma separated list with the elements:
        all (disables all optimizations),
        int (disable any int based optimizations)
    -d, --debug
        Debug mode will print out full stack traces
    -c, --encoding=<charset>
        Specify the encoding of the files
    -e=<script>
        Specify a command line script
    -i=<extension>
        Modify files in place; create backup if extension
        is given (e.g. '.bak')
    -n
        Process files line by line using implicit 'line'
        variable
    -p
        Process files line by line and print result (see
        also -n)
    -pa, --parameters
        Generate metadata for reflection on method
        parameter names (jdk8+ only)
    -pr, --enable-preview
        Enable preview Java features (JEP 12) (jdk12+ only)
    -l=<port>
        Listen on a port and process inbound lines

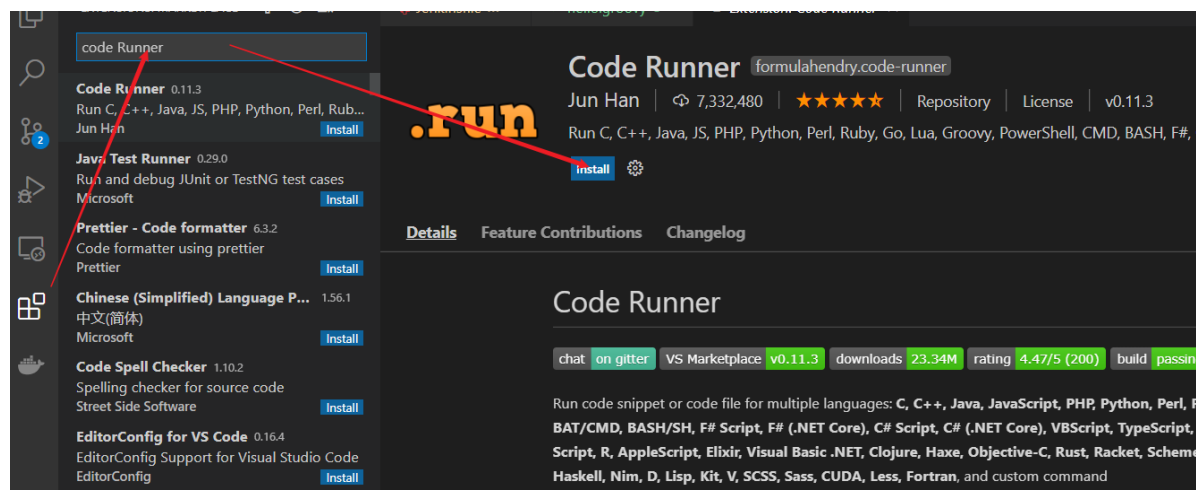
```

可以看到 Groovy，说明配置就OK了。接下来，在VS Code中开始一个经典的hello world试试看。

开始之前，在VS Code 中搜索安装groovy插件

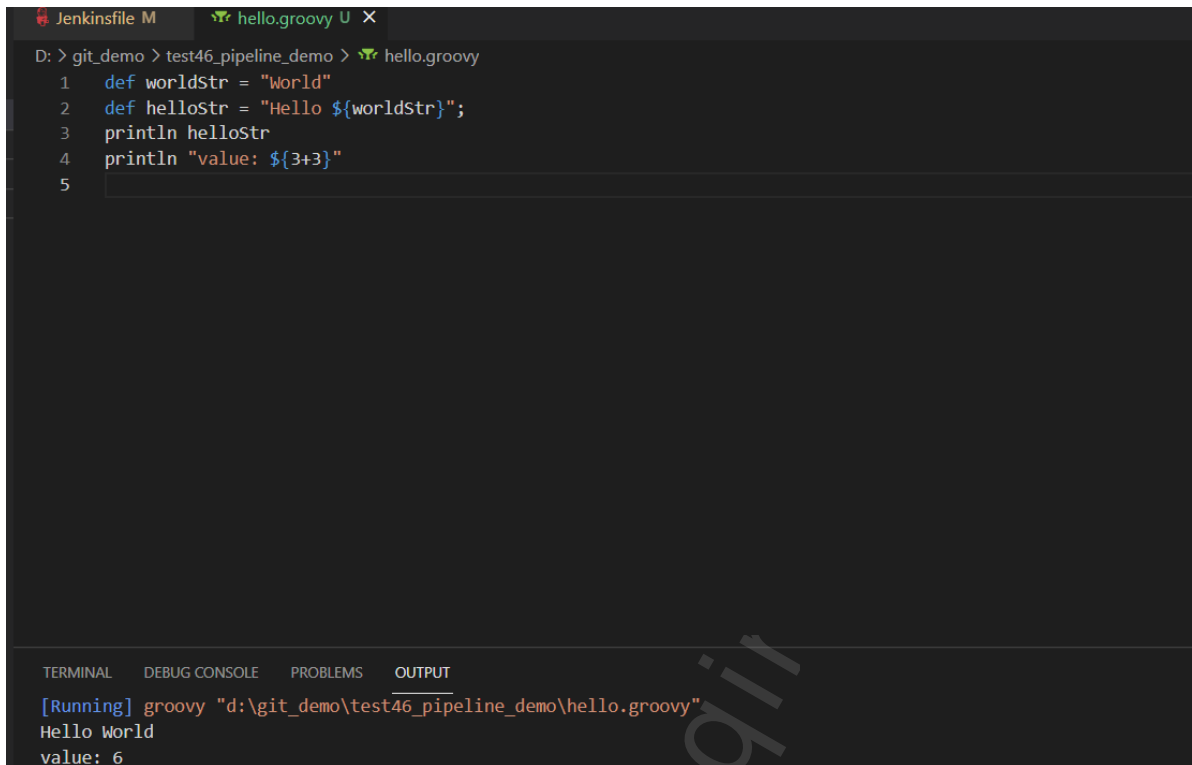


再安装一个CodeRunner插件，方便右键直接运行



这时候新建一个文件，后缀名保存为groovy,然后用vscode打开

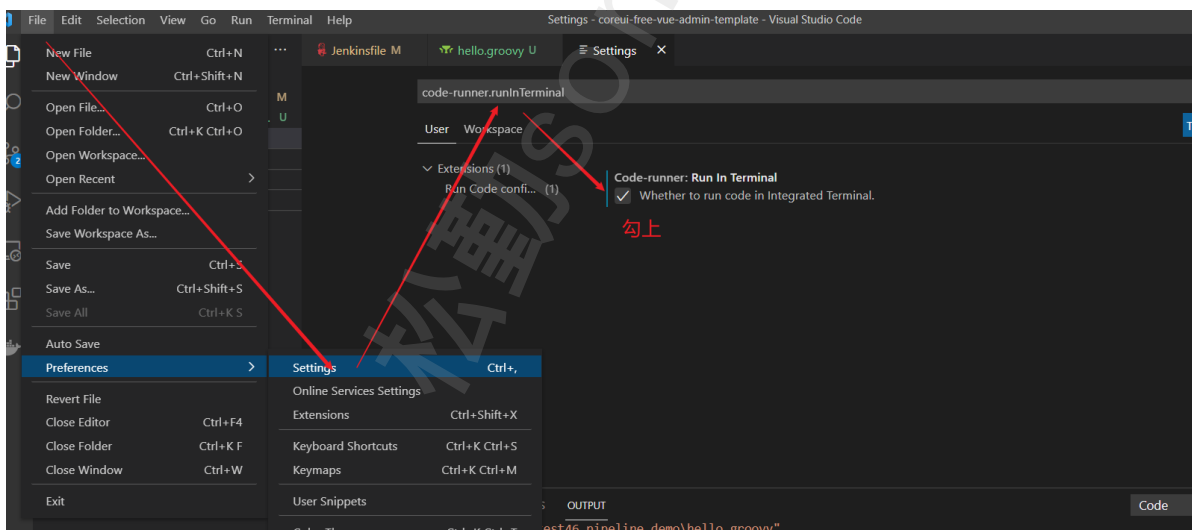
敲第行代码然后右键执行



```
D: > git_demo > test46_pipeline_demo > hello.groovy
1  def worldStr = "World"
2  def helloStr = "Hello ${worldStr}";
3  println helloStr
4  println "value: ${3+3}"
5

[Running] groovy "d:\git_demo\test46_pipeline_demo\hello.groovy"
Hello World
value: 6
```

Note: 如果Code Runner执行中文有乱码，可以更改一下设置
进入 File -> Preference -> setting, 然后在输入框搜索 code-runner.runInTerminal 勾选复选框即可



再次右键运行，输出会切到终端这里

The image shows a Visual Studio Code editor window with a file named `hello.groovy` open. The script contains the following code:

```
1 def worldStr = "中文"
2 def helloStr = "Hello ${worldStr}";
3 println helloStr
4 println "value: ${3+3}"
5
```

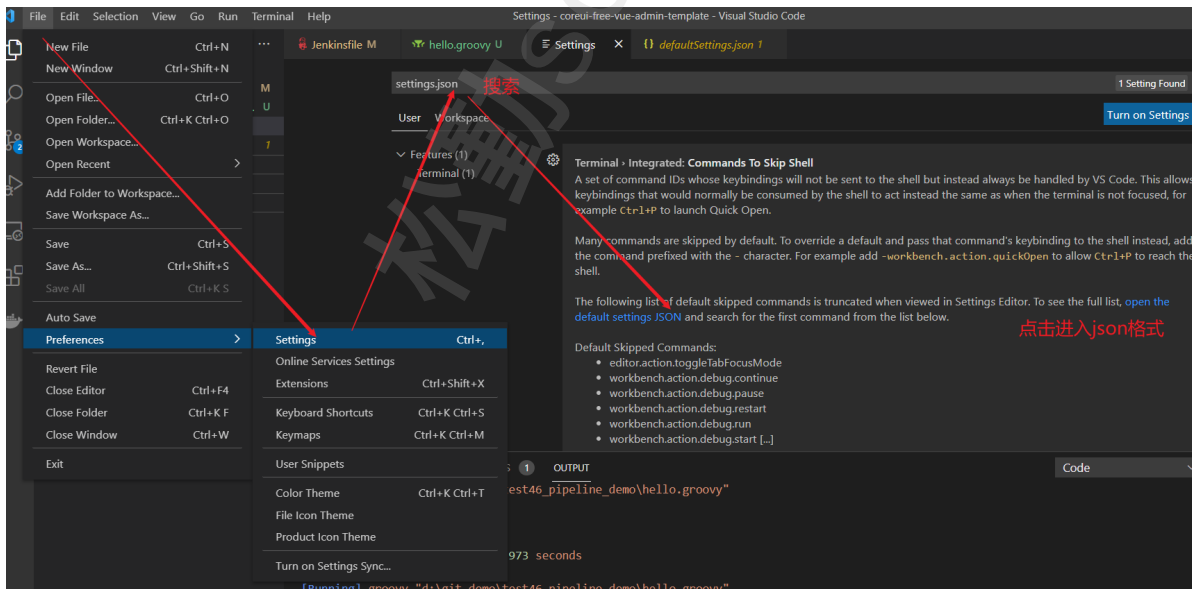
Below the editor, the **TERMINAL** panel is active, showing the output of running the script:

```
Windows PowerShell
版权所有 (C) Microsoft Corporation。保留所有权利。

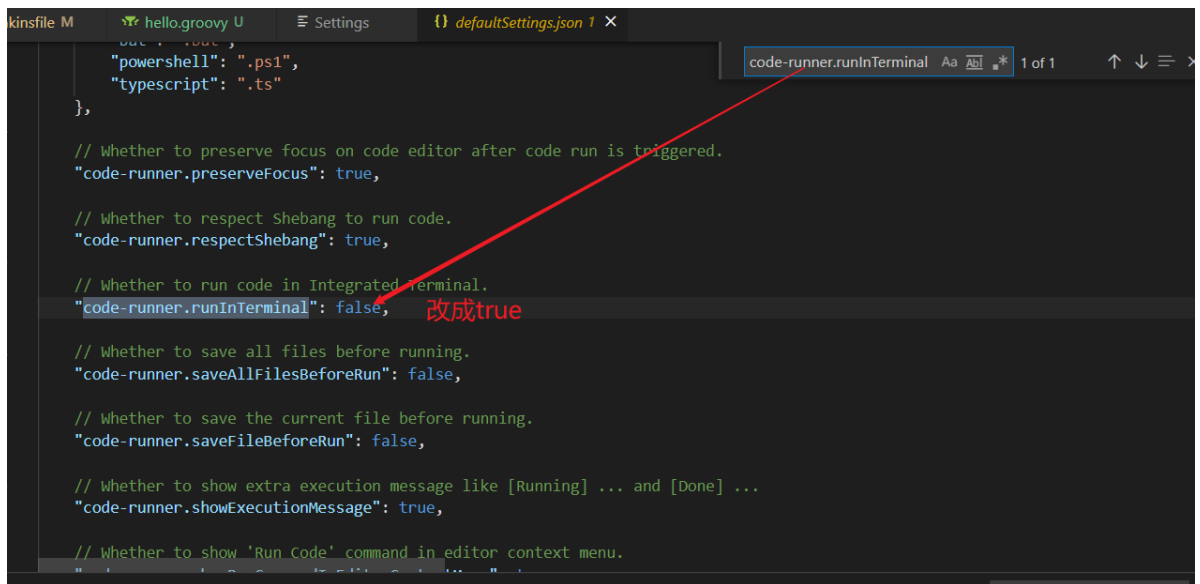
尝试新的跨平台 PowerShell https://aka.ms/pscore6

PS D:\Course\course_django\coreui-free-vue-admin-template> groovy "d:\git_demo\test46_pipeline_demo\hello.groovy"
Hello 中文
value: 6
PS D:\Course\course_django\coreui-free-vue-admin-template>
```

网上其他的参考方案



在文件中搜索code-runner.runInTerminal



```

{
  "code-runner.runInTerminal": true,
  "code-runner.preserveFocus": true,
  "code-runner.respectShebang": true,
  "code-runner.runInTerminal": false,
  "code-runner.saveAllFilesBeforeRun": false,
  "code-runner.saveFileBeforeRun": false,
  "code-runner.showExecutionMessage": true,
  "code-runner.showRunCodeCommand": true
}
```

这里修改发现文件只读修改不了。

松勤songqin