

JVM 动态执行 Groovy 脚本片段的方法

目录

一、	背景	2
二、	方法	2
(一)	利用 JShell 执行代码	2
(二)	调试模式下动态执行代码	3
(三)	利用接口执行 Groovy 脚本	4
1、	javax.script 包使用	4
2、	javax.script 包结构	5
3、	接口实现实例	8
4、	打包发布	10
三、	总结	13
四、	注意点	13

一、 背景

如果我们想要调试正在运行的程序的某个类的具体某个方法，而该方法又没有发布接口通过网络调用，或者发布的网络接口调用触发的该方法的参数不满足我们想要进行测试的参数，基于该想法，并参考 hybris 提供的执行 groovy 脚本的接口，一下将实现一个自定义接口用在程序运行时执行传入的脚本片段。

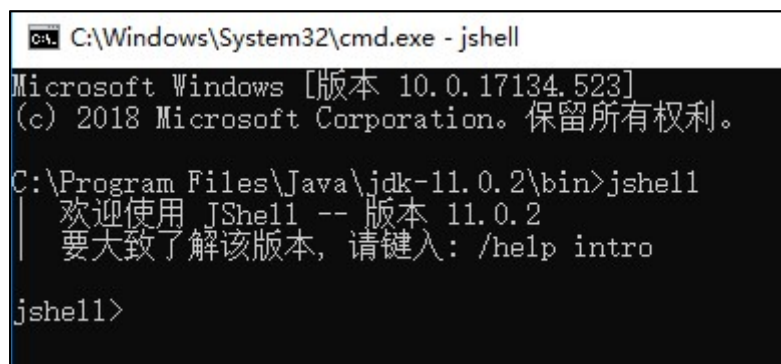
二、 方法

(一) 利用 JShell 执行代码

Java9 提供了 JShell 功能，JShell 可以输入 Java 代码片段，然后立即看到结果，并可以根据需要做出调整。某些小的测试即可不用在 ide 中编写测试类再运行了，可以直接在 JShell 中运行。

JShell 对我们的代码片段测试很方便，但是不能连接我们正在运行的 JVM 动态与更改内存内容。即没有上下文环境，只能执行无状态的代码。

1. 安装 java9 并配置环境变量后，在 cmd 中输入 jshell 即可打开。

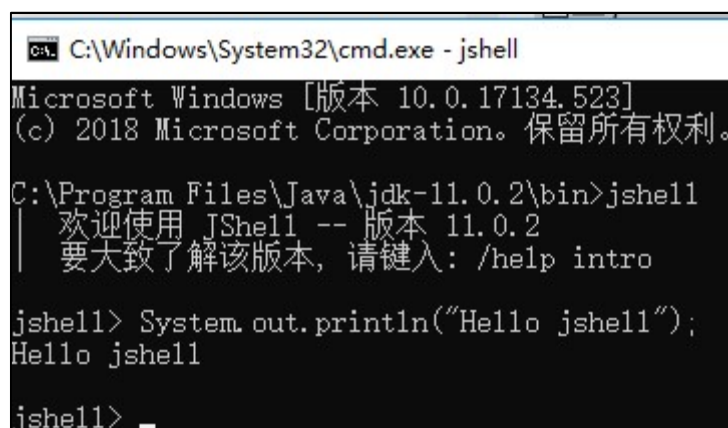


```
C:\Windows\System32\cmd.exe - jshell
Microsoft Windows [版本 10.0.17134.523]
(c) 2018 Microsoft Corporation。保留所有权利。

C:\Program Files\Java\jdk-11.0.2\bin>jshell
| 欢迎使用 JShell -- 版本 11.0.2
| 要大致了解该版本，请键入: /help intro

jshell>
```

2. 执行示例片段



```
C:\Windows\System32\cmd.exe - jshell
Microsoft Windows [版本 10.0.17134.523]
(c) 2018 Microsoft Corporation。保留所有权利。

C:\Program Files\Java\jdk-11.0.2\bin>jshell
| 欢迎使用 JShell -- 版本 11.0.2
| 要大致了解该版本，请键入: /help intro

jshell> System.out.println("Hello jshell");
Hello jshell

jshell> _
```

3. 语法地址:

<https://docs.oracle.com/javase/9/jshell/toc.htm>

(二) 调试模式下动态执行代码

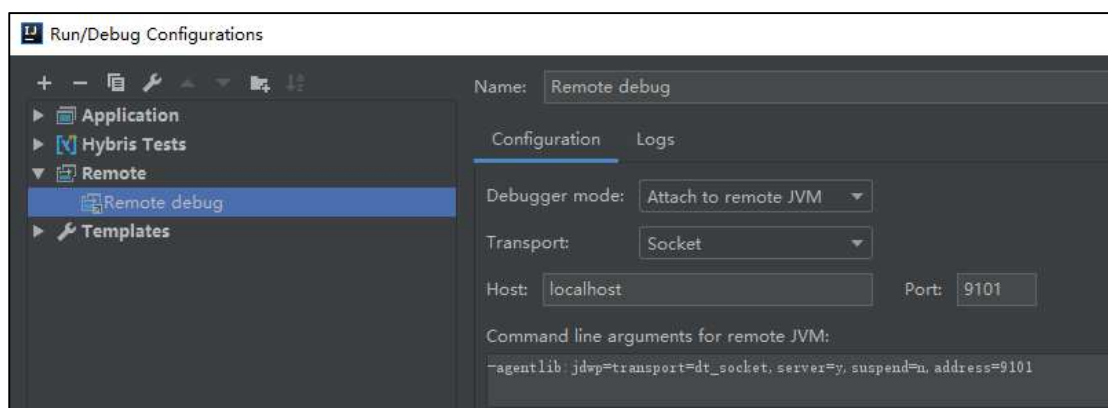
还有一种方法可以进行代码片段动态执行。即利用断点暂停线程，利用 IntelliJ Idea 工具在该线程中执行 java 代码片段。

1、在 tomcat 配置文件中打开调试模式。

配置 tomcat.debugjavaoptions 项，配置 jrebel 通知可在程序更改编译后自动动态更改断点行。

```
#hybris JREBEL调试
tomcat.debugjavaoptions=-noverify -agentpath:"${REBEL_HOME}" -Xdebug -Xnoagent -Xrunjdpw:transport=dt_socket,
server=y,address=9101,suspend=n
```

2、在 idea 中添加程序 IP 和监听端口的调试配置:

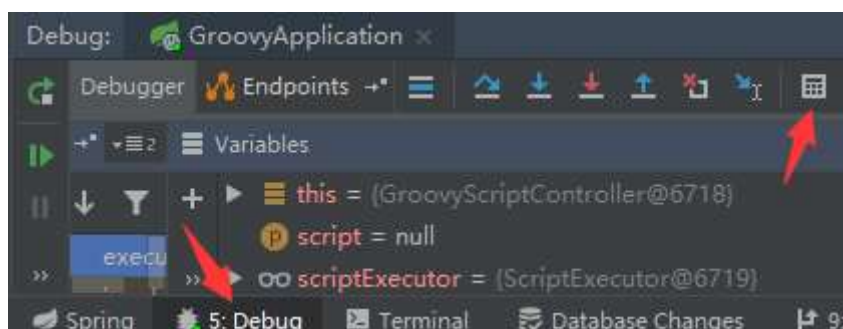


3、启动程序并使用调试模式进入任意断点。

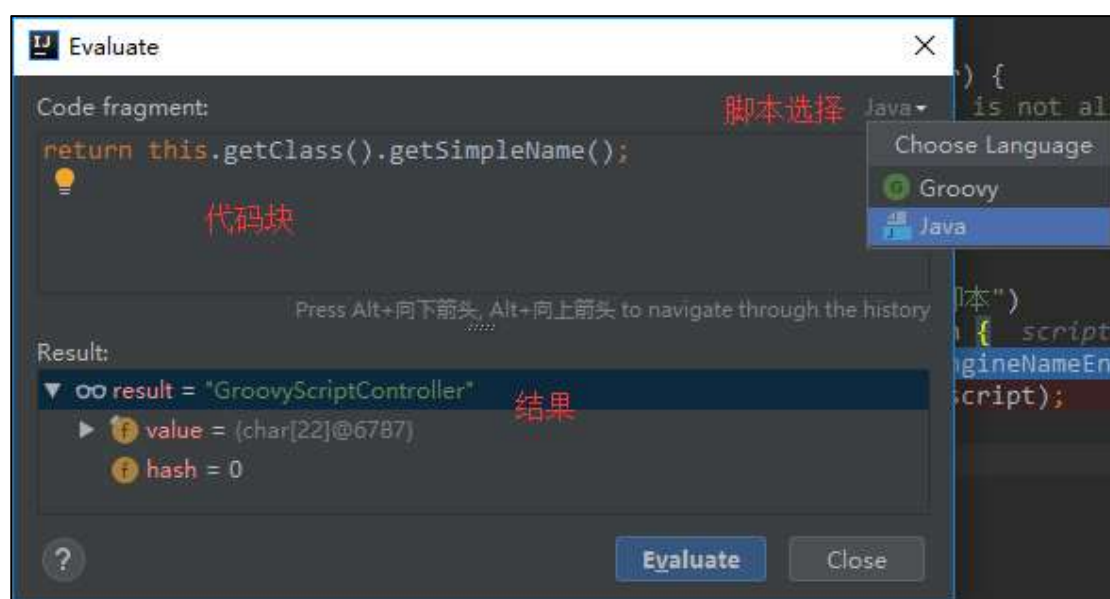
```
@GetMapping(path = "/execute")
@ApiOperation(notes = "执行Groovy脚本", value = "执行groovy脚本")
public String execute(String script) throws ScriptException {
    ScriptEngine engine = scriptExecutor.getEngineByName(EngineNameEnum.GROOVY);
    CompiledScript script1 = ((Compilable)engine).compile(script);
    return null;
}
```

4、点击 debug 小窗口，右上角即有表达式执行框，可执行动态的执行 java

或者 groovy 代码片段。



5、执行示例，可以不写 return 字段，会默认返回最后一行的执行结果：



(三) 利用接口执行 Groovy 脚本

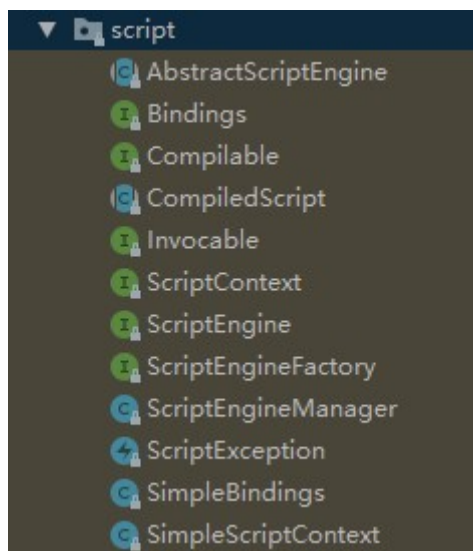
1、javax.script 包使用

Java 提供了 javax.script 包以支持对脚本的支持。该包下定义了脚本支持相关的接口、类、异常。Javax.script 的使用方法为以下几个基本的步骤：

- 创建 ScriptEngineManager 对象；
- 从创建的 ScriptEngineManager 获取一个脚本引擎 ScriptEngine；
- 调用脚本引擎 ScriptEngine 的 eval 方法执行脚本。

2、Javax.script 包结构

a) 包结构图



b) ScriptEngineManager

ScriptEngineManager 是脚本引擎管理类, 该类在所有的构造函数中都调用了 `private void init(final ClassLoader loader)` 方法, 该方法实例化了几个引擎工程的映射 Map 和调用了 `private void initEngines(final ClassLoader loader)` 以加载 classpath 中所有实现的脚本引擎工厂类 ScriptEngineFactory 该方法在初始化的时候会通过 `java.util.ServiceLoader` 类自动加载 ScriptEngineFactory 的实现类, 从而获取需要的脚本引擎, 放入引擎脚本集 `engineSpis` 中。

```
private void init(final ClassLoader loader) {  
    globalScope = new SimpleBindings();  
    engineSpis = new HashSet<ScriptEngineFactory>();  
    nameAssociations = new HashMap<String, ScriptEngineFactory>();  
    extensionAssociations = new HashMap<String, ScriptEngineFactory>();  
    mimeTypeAssociations = new HashMap<String, ScriptEngineFactory>();  
    initEngines(loader);  
}
```

```
private void initEngines(final ClassLoader loader) {
    Iterator<ScriptEngineFactory> itr = null;
    try {
        ServiceLoader<ScriptEngineFactory> sl = AccessController.doPrivileged(
            new PrivilegedAction<ServiceLoader<ScriptEngineFactory>>() {
                @Override
                public ServiceLoader<ScriptEngineFactory> run() { return getServiceLoader(loader); }
            });
        itr = sl.iterator();
    } catch (ServiceConfigurationError err) {
        System.err.println("Can't find ScriptEngineFactory providers: " +
            err.getMessage());
        if (DEBUG) {
            err.printStackTrace();
        }
        // do not throw any exception here. user may want to
        // manage his/her own factories using this manager
        // by explicit registration (by registerXXX) methods.
        return;
    }

    try {
        while (itr.hasNext()) {
            try {
                ScriptEngineFactory fact = itr.next();
                engineSpis.add(fact);
            } catch (ServiceConfigurationError err) {
                System.err.println("ScriptEngineManager providers.next(): " +
                    err.getMessage());
                if (DEBUG) {
                    err.printStackTrace();
                }
                // one factory failed, but check other factories...
                continue;
            }
        }
    } catch (ServiceConfigurationError err) {
```

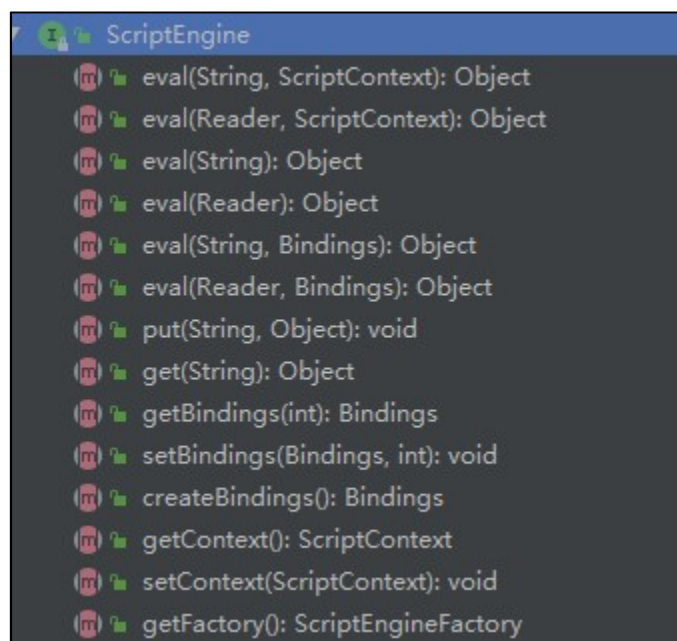
```
private ServiceLoader<ScriptEngineFactory> getServiceLoader(final ClassLoader loader) {
    if (loader != null) {
        return ServiceLoader.load(ScriptEngineFactory.class, loader);
    } else {
        return ServiceLoader.loadInstalled(ScriptEngineFactory.class);
    }
}
```

该类提供了以下三个方法以获取脚本引擎实例。获取原理是先从对应方法名映射的 Map 中查找，如果没有找到再从所有的脚本引擎集和中查找。

```
m getEngineByName(String): ScriptEngine
m getEngineByExtension(String): ScriptEngine
m getEngineByMimeType(String): ScriptEngine
```

脚本执行从该类开始获取具体脚本的引擎，即 ScriptEngine 的实现类。

c) ScriptEngine



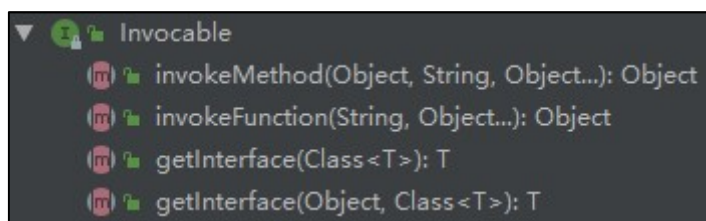
ScriptEngine 提供了多个重载的 eval 方法以执行脚本。

d) ScriptContent

脚本内容封装类，能为脚本设置参数与变量，并能提供流的输入与输出。

e) Invocable

该接口提供了两类方法，获取接口与调用方法。脚本引擎实现类需要实现该接口。



f) Compilable

ScriptEngine 实现的可选接口，作用是将脚本方法编译成可重复执行的代码。

g) ScriptEngineFactory

脚本引擎工厂类, 获取脚本引擎的相关信息, 包括版本, 媒体类型等。

h) Bindings

Bindings 接口继承了 Map 接口, 用于键值对映射, 所有的 key 类型都为 String

3、接口实现实例

a) 加 maven 相关依赖

```
<dependencies>
  <dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger2</artifactId>
    <version>2.9.2</version>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <scope>runtime</scope>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.springframework.restdocs</groupId>
    <artifactId>spring-restdocs-mockmvc</artifactId>
    <scope>test</scope>
  </dependency>
  <!-- https://mvnrepository.com/artifact/org.codehaus.groovy/groovy-all -->
  <dependency>
    <groupId>org.codehaus.groovy</groupId>
    <artifactId>groovy-all</artifactId>
    <version>3.0.0-alpha-4</version>
    <type>pom</type>
  </dependency>
</dependencies>
```


b) 定义脚本引擎管理类:

```
@Configuration
@ComponentScan(basePackages = "com.wauil.groovy")
public class JavaConfig {
    @Bean
    public ScriptEngineManager scriptEngineManager() {
        return new ScriptEngineManager();
    }
}
```

c) 定义接口:

```
@RestController
@Api(tags = "Groovy脚本执行接口")
@RequestMapping(path = "/groovy")
public class GroovyScriptController {
    private ScriptEngineManager scriptEngineManager;
    private ApplicationContext applicationContext;

    @Autowired
    public GroovyScriptController(ScriptEngineManager scriptEngineManager, ApplicationContext applicationContext) {
        Assert.notNull(scriptEngineManager, "scriptEngineManager is not allowed null.");
        Assert.notNull(applicationContext, "applicationContext is not allowed null.");
        this.scriptEngineManager = scriptEngineManager;
        this.applicationContext = applicationContext;
    }

    @PostMapping
    @ApiOperation(notes = "执行Groovy脚本", value = "执行groovy脚本")
    public Object execute(String script) throws ScriptException {
        ScriptEngine engine = scriptEngineManager.getEngineByName("groovy");
        ScriptContext context = new SimpleScriptContext();
        context.setBindings(new SimpleBindings(new HashMap<String, Object>(InitialCapacity: 1) {{
            put("spring", applicationContext);
        }}), ScriptContext.ENGINE_SCOPE);
        return engine.eval(script, context).toString();
    }
}
```

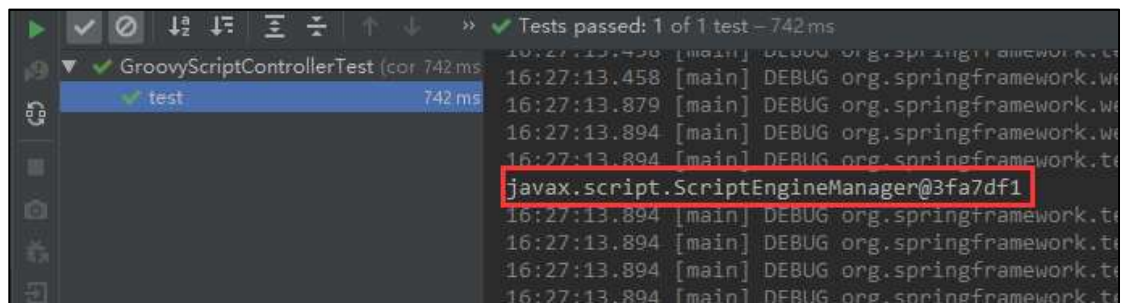
d) 编写测试类:

```
/**
 * @Description: groovy 脚本引擎测试
 * @Author: Bin.Zhou
 * @Email: dxld@qq.com
 * @Date: 2019/02/14 16:16
 * @Version: 1.0
 */
@RunWith(SpringJUnit4ClassRunner.class)
@WebAppConfiguration
@ContextConfiguration(classes = JavaConfig.class)
public class GroovyScriptControllerTest {
    private MockMvc mockMvc;
    @Autowired
    private WebApplicationContext context;
    @Autowired

    @Before
    public void setup() { this.mockMvc = MockMvcBuilders.webAppContextSetup(context).build(); }

    @Test
    public void test() throws Exception {
        String groovyScript = "spring.getBean(\"scriptEngineManager\").toString()";
        MvcResult mvcResult = mockMvc.perform(post( urlTemplate: "/groovy").param( name: "script",
groovyScript))
            .andExpect(status().isOk())
            .andReturn();
        System.out.println(mvcResult.getResponse().getContentAsString());
    }
}
```

e) 测试结果:



Tests passed: 1 of 1 test - 742ms

✓ GroovyScriptControllerTest (cor 742 ms)

✓ test 742 ms

16:27:13.458 [main] DEBUG org.springframework.w...
16:27:13.458 [main] DEBUG org.springframework.w...
16:27:13.879 [main] DEBUG org.springframework.w...
16:27:13.894 [main] DEBUG org.springframework.w...
16:27:13.894 [main] DEBUG org.springframework.w...
javax.script.ScriptEngineManager@3fa7df1
16:27:13.894 [main] DEBUG org.springframework.w...
16:27:13.894 [main] DEBUG org.springframework.w...
16:27:13.894 [main] DEBUG org.springframework.w...
16:27:13.894 [main] DEBUG org.springframework.w...

f) 测试成功，并得到预期结果

4、打包发布

给 groovy 配置文件类的 spring bean 重命名，以免别的项目引入后冲突

```
package com.wauil.groovy;

import ...

/**
 * @Description:
 * @Author: Bin.Zhou
 * @Email: dxld@qq.com
 * @Date: 2019/01/30 15:48
 * @Version: 1.0
 */
@Configuration(value = "groovyJavaConfig")
@ComponentScan(basePackages = "com.wauil.groovy")
public class JavaConfig {

    @Bean
    public ScriptEngineManager scriptEngineManager() { return new ScriptEngineManager(); }
}
```

Maven 打包插件中增加跳过配置的选项，解决打包后的 jar 产生 BOOT-INF 文件夹而导致别的项目引用包后找不到类的问题。

```
<plugin>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-maven-plugin</artifactId>
  <configuration>
    <skip>true</skip>
  </configuration>
</plugin>
```

新建测试项目，并加入发布的 groovy 依赖：

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>com.wauil</groupId>
    <artifactId>groovy</artifactId>
    <version>0.0.1-SNAPSHOT</version>
  </dependency>
  <dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger-ui</artifactId>
    <version>2.9.2</version>
  </dependency>
</dependencies>

<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>
```

在配置文件中加入 groovy 的包扫描项以引入 groovy 项目的 spring 配置。

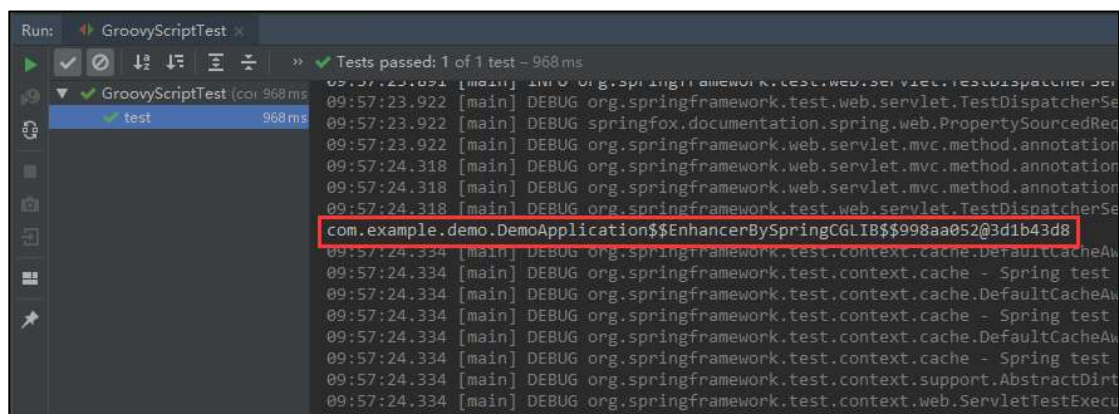
```
@Configuration
@EnableSwagger2
@ComponentScan(basePackages = {"com.wauil", "com.example"})
public class JavaConfig {
    @Bean
```

编写测试类，测试能否获取当前项目的 spring bean：

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(classes = JavaConfig.class)
@WebAppConfiguration
public class GroovyScriptTest {
    private MockMvc mockMvc;
    @Autowired
    private WebApplicationContext context;
    @Before
    public void setup() { this.mockMvc = MockMvcBuilders.webAppContextSetup(context).build(); }
    @Test
    public void test() throws Exception {
        String groovyScript = "spring.getBean(\"demoApplication\").toString()";
        MvcResult mvcResult = mockMvc.perform(post( uriTemplate: "/groovy").param( name: "script", groovyScript))
            .andExpect(status().isOk())
            .andReturn();
        System.out.println(mvcResult.getResponse().getContentAsString());
    }
}
```

启动测试项目进行测试：

测试成功，并得到预期结果。



```
Run: GroovyScriptTest x
Tests passed: 1 of 1 test - 968 ms
GroovyScriptTest (col 968 ms)
test 968 ms
09:57:23.922 [main] DEBUG org.springframework.test.web.servlet.TestDispatcherSe
09:57:23.922 [main] DEBUG springfox.documentation.spring.web.PropertySourcedReq
09:57:23.922 [main] DEBUG org.springframework.web.servlet.mvc.method.annotation
09:57:24.318 [main] DEBUG org.springframework.web.servlet.mvc.method.annotation
09:57:24.318 [main] DEBUG org.springframework.web.servlet.mvc.method.annotation
09:57:24.318 [main] DEBUG org.springframework.test.web.servlet.TestDispatcherSe
com.example.demo.DemoApplication$$EnhancerBySpringCGLIB$$998aa052@3d1b43d8
09:57:24.334 [main] DEBUG org.springframework.test.context.cache.DefaultCacheAw
09:57:24.334 [main] DEBUG org.springframework.test.context.cache - Spring test
09:57:24.334 [main] DEBUG org.springframework.test.context.cache.DefaultCacheAw
09:57:24.334 [main] DEBUG org.springframework.test.context.cache - Spring test
09:57:24.334 [main] DEBUG org.springframework.test.context.cache.DefaultCacheAw
09:57:24.334 [main] DEBUG org.springframework.test.context.cache - Spring test
09:57:24.334 [main] DEBUG org.springframework.test.context.support.AbstractDir
09:57:24.334 [main] DEBUG org.springframework.test.context.web.ServletTestExecu
```

三、 总结

Javax.script 包提供了 java 对脚本语言的支持，该包提供了脚本语言支持的一些规范，具体的实现还需要依赖第三方包或者自定义实现。本文档只介绍了对 groovy 脚本支持的具体实现，其他脚本语言如 javascript 等，可以自己探索实现。

脚本语言的嵌入对我们编码调试或者生产环境下对内存的动态调整都有着非常大的帮助。基于脚本语言的嵌入可以实现非常强大的功能，本文档只是做了一个简单的实现，更多更强大的功能可以自己探索，如对接口代码片段事务控制等。

项目地址：<https://gitee.com/Wauil/groovyscript>

四、 注意点

Maven 项目打包后需要添加包扫描才能引入 jar 中的 spring 环境。