Atitit 提升扩展性 通用架构 一个关键特性是可编程性

目录

[业务脚本 工作流 规则引擎 v2 u00.docx 1](#_Toc21578)

[嵌入式dsl 1](#_Toc20873)

[Go中调用Lua业务脚本，实现终端应用的热更新机制\_独行猫A的沉淀 ... 1](#_Toc8213)

[第一章 动态编译 2](#_Toc18500)

通用计算机 特性

通用计算机的一个关键特性是可编程性。必须存在某个将指令集引入计算机的方法，使计算机能响应这个指令集。现代计算机将这样的指令集作为字节存储在内存里，称为****机器码****。在楚泽的机器里，指令集的编码用35mm的电影胶带上的打孔来表示。巴贝奇的机器则使用打孔的卡片，与控制纺织机的卡片有点像。

一些早期的计算机只能用很不灵活的指令序列进行编程。一个通用计算机必须能够根据上几步的计算结果跳过一些指令序列。这个特性我们现在称为****条件分支****，这对于实现条件循环是必要的。

## 业务脚本 工作流 规则引擎 v2 u00.docx

## 嵌入式dsl

### [Go中调用Lua业务脚本，实现终端应用的热更新机制\_独行猫A的沉淀 ...](https://blog.csdn.net/qq8864/article/details/108502558)

[blog.csdn.net › article › details](https://blog.csdn.net/qq8864/article/details/108502558)

[Translate this page](https://translate.google.com/translate?hl=en&sl=zh-CN&u=https://blog.csdn.net/qq8864/article/details/108502558&prev=search&pto=aue)

二是脱机类，空闲联机的业务，终端可实现轻量化，热更新应用。把业务模块化，组件化，脚本化。每次升级不用整个都升级，可能仅需要写好业务脚本把轻量的 .

修改Java代码，JVM没有重启，输入参数也没有任何改变，仅仅改变脚本函数即可产生不同的结果。这就是脚本语言对系统设计最有利的地方：可以随时发布而不用重新部署；这也是我们Javaer最喜爱它的地方—即使进行变更，也能提供不间断的业务服务。

Java 6不仅仅提供了代码级的脚本内置，还提供了一个jrunscript命令工具，它可以在批处理中发挥最大效能，而且不需要通过JVM解释脚本语言，可以直接通过该工具运行脚本。想想看，这是多么大的诱惑力呀！而且这个工具是可以跨操作系统的，脚本移植就更容易了

# 动态编译

动态编译一直是Java的梦想，从Java 6版本它开始支持动态编译了，可以在运行期直接编译.java文件，执行.class，并且能够获得相关的输入输出，甚至还能监听相关的事件。不过，我们最期望的还是给定一段代码，直接编译，然后运行，也就是空中编译执行（on-the-fly），来看如下代码：

public class Client{

public static void main（String[]args）throws Exception{

//Java源代码

String sourceStr="public class Hello{public String sayHello（String name）

{return\"Hello，\"+name+\"！\"；}}"；

//类名及文件名

String clsName="Hello"；

//方法名

String methodName="sayHello"；

//当前编译器

JavaCompiler cmp=ToolProvider.getSystemJavaCompiler（）；

//Java标准文件管理器

StandardJavaFileManager fm=cmp.getStandardFileManager（null, null, null）；

//Java文件对象

JavaFileObject jfo=new StringJavaObject（clsName, sourceStr）；

//编译参数，类似于javac＜options＞中的options

List＜String＞optionsList=new ArrayList＜String＞（）；

//编译文件的存放地方，注意：此处是为Eclipse工具特设的

optionsList.addAll（Arrays.asList（"-d"，"./bin"））；

//要编译的单元

List＜JavaFileObject＞jfos=Arrays.asList（jfo）；

//设置编译环境

JavaCompiler.CompilationTask task=cmp.getTask（null, fm, null，

optionsList, null, jfos）；

//编译成功

if（task.call（））{

//生成对象

Object obj=Class.forName（clsName）.newInstance（）；

Class＜?extends Object＞cls=obj.getClass（）；

//调用sayHello方法

Method m=cls.getMethod（methodName, String.class）；

String str=（String）m.invoke（obj，"Dynamic Compilation"）；

System.out.println（str）；

}

}

}

//文本中的Java对象

class StringJavaObject extends SimpleJavaFileObject{

//源代码

private String content=""；

//遵循Java规范的类名及文件

public StringJavaObject（String\_javaFileName, String\_content）{

super（\_createStringJavaObjectUri（\_javaFileName），Kind.SOURCE）；

content=\_content；

}

//产生一个URL资源路径

private static URI\_createStringJavaObjectUri（String name）{

//注意此处没有设置包名

return URI.create（"String：///"+name+Kind.SOURCE.extension）；

}

//文本文件代码

@Override

public CharSequence getCharContent（boolean ignoreEncodingErrors）

throws IOException{

return content；

}

}

上面的代码较多，这是一个动态编译的模板程序，读者可以拷贝到项目中使用，代码中的中文注释也较多，相信读者看得懂，不多解释，读者只要明白一件事：只要是在本地静态编译能够实现的任务，比如编译参数、输入输出、错误监控等，动态编译就都能实现。

Java的动态编译对源提供了多个渠道。比如，可以是字符串（例子中就是字符串），可以是文本文件，也可以是编译过的字节码文件（.class文件），甚至可以是存放在数据库中的明文代码或是字节码。汇总成一句话，只要是符合Java规范的就都可以在运行期动态加载，其实现方式就是实现JavaFileObject接口，重写getCharContent、openInputStream、openOutputStream，或者实现JDK已经提供的两个SimpleJavaFileObject、ForwardingJavaFileObject，具体代码可以参考上个例子。

动态编译虽然是很好的工具，让我们可以更加自如地控制编译过程，但是在我目前所接触的项目中还是使用得较少。原因很简单，静态编译已经能够帮我们处理大部分的工作，甚至是全部的工作，即使真的需要动态编译，也有很好的替代方案，比如JRuby、Groovy等无缝的脚本语言。

Atitit 如何在java中其他语言提升扩展性的必要性

建议16：易变业务使用脚本语言编写