动态调用动态语言之Java脚本API PDF转换可能丢失图片或格式，建议阅读原文
https://www.100test.com/kao_ti2020/293/2021_2022__E5_8A_A8_E6_80_81_E8_B0_83_E7_c67_293740.htm 我们不需要将动态语言编译为 Java字节码就可以在 Java 应用程序中使用它们。使用 Java Platform, Standard Edition 6 (Java SE)中添加的脚本包（并且向后兼容 Java SE 5），Java 代码可以在运行时以一种简单的、统一的方式调用多种动态语言。 本系列文章共分两个部分，第 1 部分将介绍 Java 脚本 API 的各种特性。文章将使用一个简单的 Hello World 应用程序展示 Java 代码如何执行脚本代码以及脚本如何反过来执行 Java 代码。第 2 部分将深入研究 Java 脚本 API 的强大功能。 Java 开发人员清楚 Java 并不是在任何情况下都是最佳的语言。今年，1.0 版本的 JRuby 和 Groovy 的发行引领了一场热潮，促使人们纷纷在自己的 Java 应用程序中添加动态语言。Groovy、JRuby、Rhino、Jython 和一些其他的开源项目使在所谓的脚本语言中编写代码并在 JVM 中运行成为了可能（请参阅 参考资料）。通常，在 Java 代码中集成这些语言需要对各种解释器所特有的 API 和特性有所了解。 Java SE 6 中添加的 javax.script 包使集成动态语言更加容易。通过使用一小组接口和具体类，这个包使我们能够简单地调用多种脚本语言。但是，Java 脚本 API 的功能不只是在应用程序中编写脚本；这个脚本包使我们能够在运行时读取和调用外部脚本，这意味着我们可以动态地修改这些脚本从而更改运行应用程序的行为。 Java 脚本 API 脚本与动态的对比 术语脚本 通常表示在解释器 shell 中运行的语言，它们往往没有单独的编译步骤。术语动态 通常表示等到运行时

判断变量类型或对象行为的语言，往往具有闭包和连续特性。一些通用的编程语言同时具有这两种特性。此处首选脚本语言 是因为本文的着重点是 Java 脚本 API，而不是因为提及的语言缺少动态特性。 2006 年 10 月，Java 语言添加了脚本包，从而提供了一种统一的方式将脚本语言集成到 Java 应用程序中去。对于语言开发人员，他们可以使用这个包编写粘连代码（glue code），从而使人们能够在 Java 应用程序中调用他们的语言。对于 Java 开发人员，脚本包提供了一组类和接口，允许使用一个公共 API 调用多种语言编写的脚本。因此，脚本包类似于不同语言（比如说不同的数据库）中的 Java Database Connectivity (JDBC) 包，可以使用一致的接口集成到 Java 平台中去。 以前，在 Java 代码中，动态调用脚本语言涉及到使用各种语言发行版所提供的独特类或使用 Apache 的 Jakarta Bean Scripting Framework (BSF)。BSF 在一个 API 内部统一了一组脚本语言（请参阅 参考资料）。使用 Java SE 6 脚本 API，二十余种脚本语言（AppleScript、Groovy、JavaScript、Jelly、PHP、Python、Ruby 和 Velocity）都可以集成到 Java 代码中，这在很大程序上依赖的是 BSF。 脚本 API 在 Java 应用程序和外部脚本之间提供了双向可见性。Java 代码不仅可以调用外部脚本，而且还允许那些脚本访问选定的 Java 对象。比如说，外部 Ruby 脚本可以对 Java 对象调用方法，并访问对象的属性，从而使脚本能够将行为添加到运行中的应用程序中（如果在开发时无法预计应用程序的行为）。 调用外部脚本可用于运行时应用程序增强、配置、监控或一些其他的运行时操作，比如说在不停止应用程序的情况下修改业务规则。脚本包可能的作用包括： 在比 Java 语言更简单的语言中

编写业务规则，而不用借助成熟的规则引擎。 创建插件架构，使用户能够动态地定制应用程序。 将已有脚本集成到 Java 应用程序中，比如说处理或转换文件文章的脚本。 使用成熟的编程语言（而不是属性文件）从外部配置应用程序的运行时行为。 在 Java 应用程序中添加一门特定于域的语言（domain-specific language）。 在开发 Java 应用程序原型的过程中使用脚本语言。 在脚本语言中编写应用程序测试代码。 你好，脚本世界 HelloScriptingWorld 类（本文中的相关代码均可从 下载部分 获得）演示了 Java 脚本包的一些关键特性。它使用硬编码的 JavaScript 作为示例脚本语言。此类的 main() 方法（如清单 1 所示）将创建一个 JavaScript 脚本引擎，然后分别调用五个方法（在下文的清单中有显示）用于突出显示脚本包的特性。 清单 1. HelloScriptingWorld main 方法public static void main(String[] args) throws ScriptException, NoSuchMethodException {ScriptEngineManager scriptEngineMgr = new ScriptEngineManager().ScriptEngine jsEngine = scriptEngineMgr.getEngineByName("JavaScript").if (jsEngine == null) {System.err.println("No script engine found for JavaScript").System.exit(1).}System.out.println("Calling invokeHelloScript...").invokeHelloScript(jsEngine).System.out.println("\nCalling defineScriptFunction...").defineScriptFunction(jsEngine).System.out.println("\nCalling invokeScriptFunctionFromEngine...").invokeScriptFunctionFromEngine(jsEngine).System.out.println("\nCalling invokeScriptFunctionFromJava...").invokeScriptFunctionFromJava(

jsEngine).System.out.println("\nCalling invokeJavaFromScriptFunction...").invokeJavaFromScriptFunction(jsEngine).} main() 方法的主要功能是获取一个 javax.script.ScriptEngine 实例（清单 1 中的前两行代码）。脚本引擎可以在特定的语言中加载并执行脚本。它是 Java 脚本包中使用最为频繁、作用最为重要的类。我们从 javax.script.ScriptEngineManager 获取一个脚本引擎（第一行代码）。通常，程序只需要获取一个脚本引擎实例，除非使用了很多种脚本语言。 ScriptEngineManager 类 ScriptEngineManager 可能是脚本包中惟一一个经常使用的具体类；其他大多数都是接口。它或许是脚本包中惟一的一个要直接或间接地（通过 Spring Framework 之类的依赖性注入机制）实例化的类。ScriptEngineManager 可以使用以下三种方式返回脚本引擎： 通过引擎或语言的名称，比如说 清单 1 请求 JavaScript 引擎。 通过该语言脚本共同使用的文件扩展名，比如说 Ruby 脚本的 .rb。 通过脚本引擎声明的、知道如何处理的 MIME 类型。 本文示例为什么要使用 JavaScript？ 本文中的 Hello World 示例使用了部分 JavaScript 脚本，这是因为 JavaScript 代码易于理解，不过主要还是因为 Sun Microsystems 和 BEA Systems 所提供的 Java 6 运行时环境附带有基于 Mozilla Rhino 开源 JavaScript 实现的 JavaScript 解释器。使用 JavaScript，我们无需在类路径中添加脚本语言 JAR 文件。 ScriptEngineManager 间接查找和创建脚本引擎。也就是说，当实例化脚本引擎管理程序时，ScriptEngineManager 会使用 Java 6 中新增的服务发现机制在类路径中查找所有注册的 javax.script.ScriptEngineFactory 实现。这些工厂类封装在 Java

脚本 API 实现中；也许您永远都不需要直接处理这些工厂类。 ScriptEngineManager 找到所有的脚本引擎工厂类之后，它会查询各个类并判断是否能够创建所请求类型的脚本引擎 清单 1 中为 JavaScript 引擎。如果工厂说可以创建所需语言的脚本引擎，那么管理程序将要求工厂创建一个引擎并将其返回给调用者。如果没有找到所请求语言的工厂，那么管理程序将返回 null，清单 1 中的代码将检查 null 返回值并做出预防。