Java运行环境

更新时间：2020-08-28 19:57:30

本页目录

* [打印日志](https://help.aliyun.com/document_detail/113519.html?spm=a2c4g.11186623.6.585.7ba936afJQFwrG#title-zji-kc3-nl4)
* [错误处理](https://help.aliyun.com/document_detail/113519.html?spm=a2c4g.11186623.6.585.7ba936afJQFwrG#title-m68-80g-eao)
* [使用自定义模块](https://help.aliyun.com/document_detail/113519.html?spm=a2c4g.11186623.6.585.7ba936afJQFwrG#title-d1e-mv5-9t3)
* [使用IDEA安装依赖并打包上传](https://help.aliyun.com/document_detail/113519.html?spm=a2c4g.11186623.6.585.7ba936afJQFwrG#title-2p5-ayo-3tn)
* [使用Funcraft工具打包部署](https://help.aliyun.com/document_detail/113519.html?spm=a2c4g.11186623.6.585.7ba936afJQFwrG#title-dor-cq2-ule)
* [执行结果](https://help.aliyun.com/document_detail/113519.html?spm=a2c4g.11186623.6.585.7ba936afJQFwrG#h2-url-1)

函数计算目前支持Java OpenJDK 1.8.0（runtime=java8）运行环境。本文介绍了Java运行环境的打印日志、错误处理和自定义模块。

**打印日志**

函数通过context.getLogger()打印的内容会被收集到创建服务时指定的Logstore中。

package example;

**import** com.aliyun.fc.runtime.Context;

**import** com.aliyun.fc.runtime.StreamRequestHandler;

**import** java.io.IOException;

**import** java.io.InputStream;

**import** java.io.OutputStream;

public **class** **HelloFC** **implements** **StreamRequestHandler** {

@Override

public **void** handleRequest(

InputStream inputStream, OutputStream outputStream, Context context) throws IOException {

context.getLogger().info("hello world");

outputStream.write(**new** **String**("hello world").getBytes());

}

}

执行上面的代码输出的日志内容如下所示。

message:2017-07-05T05:13:35.920Z a72df088-f738-cee3-e0fe-323ad89118e5 [INFO] hello world

使用context.getLogger().warn和context.getLogger().error分别可以打印WARN、ERROR级别的日志。

**错误处理**

您的函数如果在执行过程中抛出异常，函数计算会把异常捕获并将异常信息返回。

**package** example;

**import** com.aliyun.fc.runtime.Context;

**import** com.aliyun.fc.runtime.StreamRequestHandler;

**import** java.io.IOException;

**import** java.io.InputStream;

**import** java.io.OutputStream;

**public** **class** **HelloFC** **implements** **StreamRequestHandler** {

@Override

**public** **void** **handleRequest**(InputStream inputStream, OutputStream outputStream, Context context) **throws** IOException {

**throw** **new** IOException("oops");

}

}

调用时收到的响应如下所示。

>>> invk hello-java -f /tmp/a.json

{

"errorMessage" : "oops",

"errorType" : "java.io.IOException",

"errorCause" : "oops",

"stackTrace" : [ "example.HelloFC.handleRequest(HelloFC.java:15)" ]

}

Error: Request id: 45dd8d90-6b78-cce3-087c-8bf4ebc6c9af. Error type: UnhandledInvocationError

发生异常时，函数调用的响应的HTTP header中会包含X-Fc-Error-Type: UnhandledInvocationError。

更多信息，请参见[错误类型](https://help.aliyun.com/document_detail/157704.html?spm=a2c4g.11186623.6.559.2e4d7a367X35GR#h2-u9519u8BEFu5904u74065)。

**使用自定义模块**

如果您需要使用自定义的模块，则需要在打JAR包时，将他们与代码一起打包。下文演示如何将OSS Java SDK打包到项目中。

1. 安装Java和Maven。

您可以通过 [官网](https://help.aliyun.com/document_detail/www.java.html)下载安装合适您系统的JDK。Maven安装请参见 [Installing Apache Maven](https://maven.apache.org/install.html)。

1. 创建一个Java项目，目录结构如下。

**test**/src/main/java/example/App.java

1. 在App.java文件内输入以下内容。
2. **package** example;
3. **import** java.io.IOException;
4. **import** java.io.InputStream;
5. **import** java.io.OutputStream;
6. **import** com.aliyun.fc.runtime.Context;
7. **import** com.aliyun.fc.runtime.StreamRequestHandler;
8. **import** com.aliyun.fc.runtime.FunctionInitializer;
9. /\*\*
10. \* Hello world!
11. \*
12. \*/
13. **public** **class** **App** **implements** **StreamRequestHandler**, **FunctionInitializer** {
14. **public** **void** **initialize**(Context context) **throws** IOException {
15. }
16. **public** **void** **handleRequest**(
17. InputStream inputStream, OutputStream outputStream, Context context) **throws** IOException {
18. outputStream.write(**new** String("hello world\n").getBytes());
19. }

}

1. 在项目文件夹根目录下创建pom.xml文件，在pom.xml中添加[maven-assembly-plugin](http://maven.apache.org/plugins/maven-assembly-plugin/)插件，内容如下。
2. <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3. xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4\_0\_0.xsd">
4. <modelVersion>4.0.0</modelVersion>
5. <groupId>example</groupId>
6. <artifactId>Java-example</artifactId>
7. <packaging>jar</packaging>
8. <version>1.0-SNAPSHOT</version>
9. <name>Java-example</name>
10. <properties>
11. <maven.compiler.target>1.8</maven.compiler.target>
12. <maven.compiler.source>1.8</maven.compiler.source>
13. <maven.test.skip>true</maven.test.skip>
14. </properties>
15. <build>
16. <plugins>
17. <plugin>
18. <artifactId>maven-assembly-plugin</artifactId>
19. <version>3.1.0</version>
20. <configuration>
21. <descriptorRefs>
22. <descriptorRef>jar-with-dependencies</descriptorRef>
23. </descriptorRefs>
24. <appendAssemblyId>false</appendAssemblyId> <!-- this is used for not append id to the jar name -->
25. </configuration>
26. <executions>
27. <execution>
28. <id>make-assembly</id> <!-- this is used for inheritance merges -->
29. <phase>package</phase> <!-- bind to the packaging phase -->
30. <goals>
31. <goal>single</goal>
32. </goals>
33. </execution>
34. </executions>
35. </plugin>
36. <plugin>
37. <groupId>org.apache.maven.plugins</groupId>
38. <artifactId>maven-compiler-plugin</artifactId>
39. <configuration>
40. <source>1.8</source>
41. <target>1.8</target>
42. </configuration>
43. </plugin>
44. </plugins>
45. </build>

</project>

1. 项目需要引用Maven Central的外部包，可以根据需要添加依赖，下文以OSS Java SDK为例，pom.xml文件示例内容如下。
2. <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3. xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4\_0\_0.xsd">
4. <modelVersion>4.0.0</modelVersion>
5. <groupId>example</groupId>
6. <artifactId>Java-example</artifactId>
7. <packaging>jar</packaging>
8. <version>1.0-SNAPSHOT</version>
9. <name>Java-example</name>
10. <dependencies>
11. <dependency>
12. <groupId>com.aliyun.fc.runtime</groupId>
13. <artifactId>fc-java-core</artifactId>
14. <version>1.3.0</version>
15. </dependency>
16. <dependency>
17. <groupId>com.aliyun.oss</groupId>
18. <artifactId>aliyun-sdk-oss</artifactId>
19. <version>2.6.1</version>
20. </dependency>
21. </dependencies>
22. <properties>
23. <maven.compiler.target>1.8</maven.compiler.target>
24. <maven.compiler.source>1.8</maven.compiler.source>
25. <maven.test.skip>true</maven.test.skip>
26. </properties>
27. <build>
28. <plugins>
29. <plugin>
30. <artifactId>maven-assembly-plugin</artifactId>
31. <version>3.1.0</version>
32. <configuration>
33. <descriptorRefs>
34. <descriptorRef>jar-with-dependencies</descriptorRef>
35. </descriptorRefs>
36. <appendAssemblyId>false</appendAssemblyId> <!-- this is used for not append id to the jar name -->
37. </configuration>
38. <executions>
39. <execution>
40. <id>make-assembly</id> <!-- this is used for inheritance merges -->
41. <phase>package</phase> <!-- bind to the packaging phase -->
42. <goals>
43. <goal>single</goal>
44. </goals>
45. </execution>
46. </executions>
47. </plugin>
48. <plugin>
49. <groupId>org.apache.maven.plugins</groupId>
50. <artifactId>maven-compiler-plugin</artifactId>
51. <configuration>
52. <source>1.8</source>
53. <target>1.8</target>
54. </configuration>
55. </plugin>
56. </plugins>
57. </build>

</project>

1. 在项目的根目录下执行mvn package命令打包，编译输出如下。
2. $mvn package
3. [INFO] Scanning **for** projects...
4. ... .... ....
5. [INFO] --------------------------< example:example >---------------------------
6. [INFO] Building Java-example 1.0-SNAPSHOT
7. [INFO] --------------------------------[ jar ]---------------------------------
8. ... .... ....
9. [INFO] ------------------------------------------------------------------------
10. [INFO] BUILD SUCCESS
11. [INFO] ------------------------------------------------------------------------
12. [INFO] Total time: 11.681 s
13. [INFO] Finished at: 2020-03-26T15:55:05+08:00

[INFO] ------------------------------------------------------------------------

如果显示编译失败，请根据输出的编译错误信息调整代码。

编译后的JAR包位于项目文件夹内的target目录内，并根据pom.xml内的**artifactId**、**version**字段命名为Java-example、1.0-SNAPSHOT.jar。

1. 在函数计算控制台上传JAR包。
   1. 登录[函数计算控制台](https://fc.console.aliyun.com)。
   2. 在顶部菜单栏，选择地域。
   3. 找到目标服务下的目标函数，单击函数名称。
   4. 在**代码执行**页面，选择**OSS上传**或者**代码包上传**的方式上传打包好的JAR包。

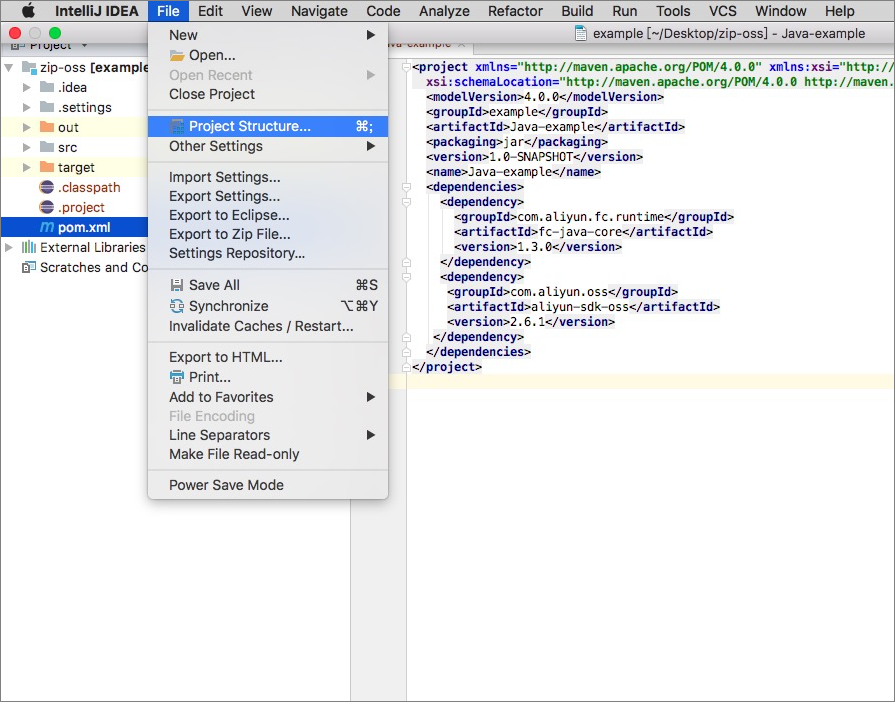
您也可以使用[maven-shade-plugin](http://maven.apache.org/plugins/maven-shade-plugin/)插件进行打包。 通常上述的打包能够满足大多数的使用场景。但是如果您想把某些没有被Maven管理的JAR包打入到最终的JAR包中，例如您在resources/lib下引入的其他非Maven仓库中的JAR包，此时可以使用[maven-jar-plugin](http://maven.apache.org/plugins/maven-jar-plugin/)和[maven-dependency-plugin](http://maven.apache.org/components/plugins/maven-dependency-plugin/)插件将其打入最终的JAR包中。

**使用IDEA安装依赖并打包上传**

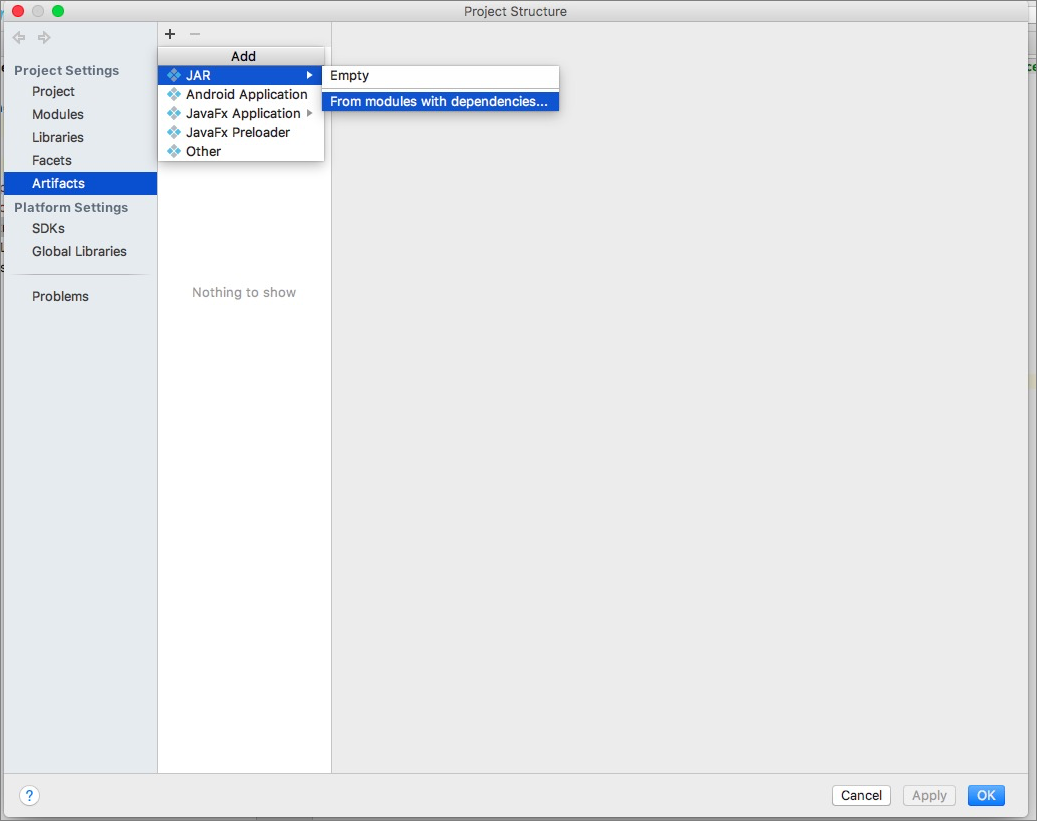
1. 本地创建Java项目，pom.xml文件内容如下。
2. <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3. xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4\_0\_0.xsd">
4. <modelVersion>4.0.0</modelVersion>
5. <groupId>example</groupId>
6. <artifactId>Java-example</artifactId>
7. <packaging>jar</packaging>
8. <version>1.0-SNAPSHOT</version>
9. <name>Java-example</name>
10. <dependencies>
11. <dependency>
12. <groupId>com.aliyun.fc.runtime</groupId>
13. <artifactId>fc-java-core</artifactId>
14. <version>1.3.0</version>
15. </dependency>
16. <dependency>
17. <groupId>com.aliyun.oss</groupId>
18. <artifactId>aliyun-sdk-oss</artifactId>
19. <version>2.6.1</version>
20. </dependency>
21. </dependencies>

</project>

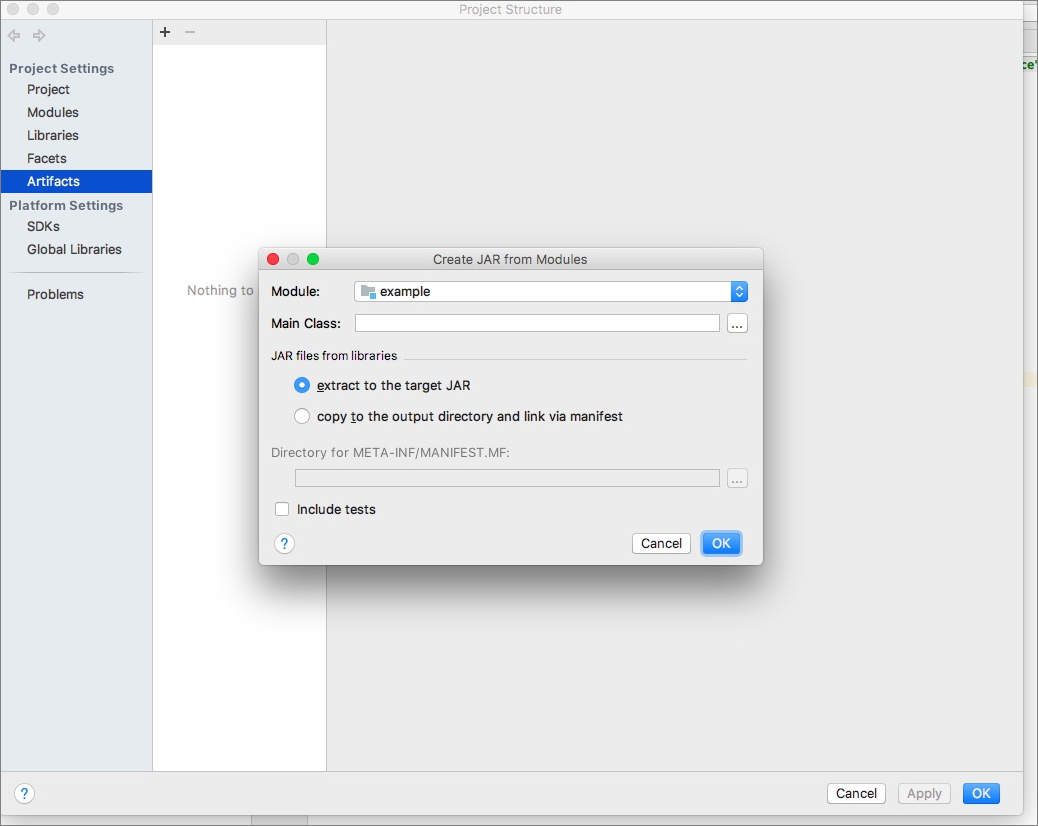
1. 使用IDEA导出JAR包相关配置。
   1. 选择**File > Project Structure**。

[](http://static-aliyun-doc.oss-cn-hangzhou.aliyuncs.com/assets/img/zh-CN/9485168951/p144050.png)

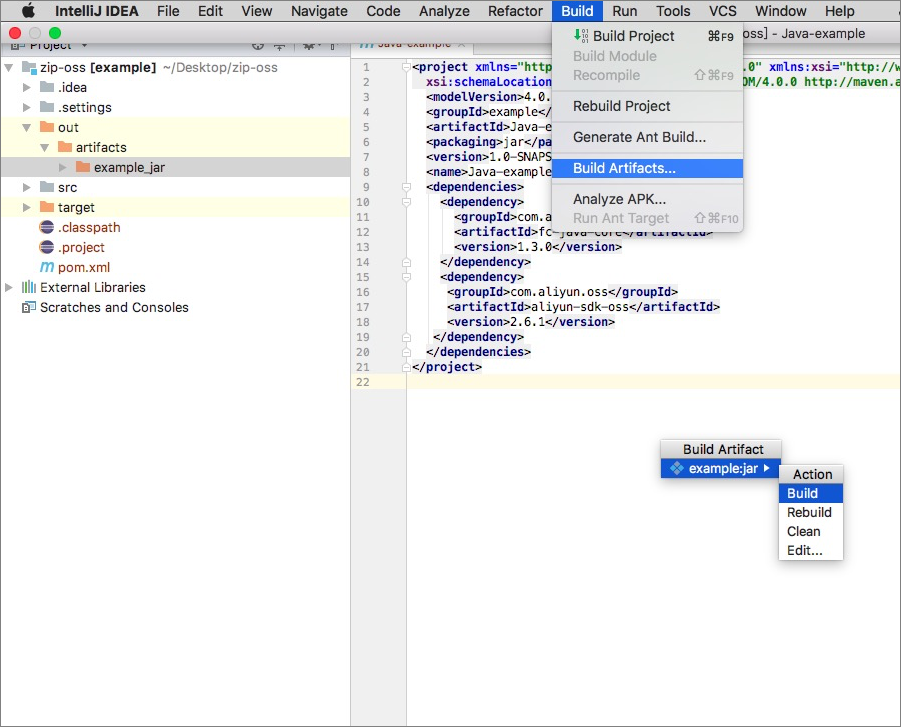
* 1. 在弹出的窗口中选择**Artifacts > + > JAR > from modules with dependencies**。

[](http://static-aliyun-doc.oss-cn-hangzhou.aliyuncs.com/assets/img/zh-CN/9485168951/p144070.png)

* 1. 在配置窗口中使用默认配置，单击**OK**。

[](http://static-aliyun-doc.oss-cn-hangzhou.aliyuncs.com/assets/img/zh-CN/9485168951/p146767.png)

* 1. 回到IDEA的主菜单，选择**Build > Build Artifacts**下的**Build**或者**Rebuild**即可生成JAR包。

[](http://static-aliyun-doc.oss-cn-hangzhou.aliyuncs.com/assets/img/zh-CN/9485168951/p146772.png)

1. 您可以在项目根目录下out/artifacts/example\_jar目录中将导出的JAR包上传函数计算。 您通过[函数计算控制台](https://fc.console.aliyun.com)的**代码执行**页面，选择**OSS上传**或者**代码包上传**方式上传导出的JAR包。

**使用Funcraft工具打包部署**

如果您使用Funcraft部署应用，可以使用[fun build](https://help.aliyun.com/document_detail/147039.html#multiTask2451)命令来构建。

1. 在工程的根目录下创建template.yml文件，内容示例如下。
2. ROSTemplateFormatVersion: '2015-09-01'
3. Transform: 'Aliyun::Serverless-2018-04-03'
4. Resources:
5. JavaDemo:
6. Type: 'Aliyun::Serverless::Service'
7. Properties:
8. Description: 'helloworld'
9. test:
10. Type: 'Aliyun::Serverless::Function'
11. Properties:
12. Handler: example.App::handleRequest
13. Initializer: example.App::initialize
14. Runtime: java8

CodeUri: './'

1. 执行以下命令将工程打包。

fun build

返回结果如下。

using template: template.yml

**start** building **function** dependencies **without** docker

building JavaDemo/**test**

running task flow MavenTaskFlow

running task: MavenCompileTask

running task: MavenCopyDependencies

running task: CopyMavenArtifacts

**Build** **Success**

Built artifacts: .fun/**build**/artifacts

Built **template**: .fun/**build**/artifacts/template.yml

Tips **for** **next** step

======================

\* Invoke **Event** **Function**: fun **local** invoke

\* Invoke **Http** **Function**: fun **local** **start**

\* Deploy Resources: fun deploy

1. 执行以下命令部署函数。

fun deploy -y

返回结果如下。

**using** template: .fun/build/artifacts/template.yml

**using** region: cn-qingdao

**using** accountId: \*\*\*\*\*\*\*\*\*\*\*3743

**using** accessKeyId: \*\*\*\*\*\*\*\*\*\*\*Ptgk

**using** timeout: 60

Collecting your services information, **in** order to caculate devlopment changes...

Resources Changes(Beta version! Only FC resources changes will be displayed):

\*\*\*\*\*\*\*\*\*\*\*

Waiting **for** service JavaDemo to be deployed...

Waiting **for** **function** **test** **to** **be** **deployed**...

Waiting **for** packaging **function** **test** **code**...

The **function** **test** **has** **been** **packaged**. **A** **total** **of** **3** **files** **were** **compressed** **and** **the** **final** **size** **was** **330**.**35** **KB**

**function** **test** **deploy** **success**

service JavaDemo deploy success