华为IoT平台NB编解码插件开发教程

本文通过一个比较简单又不失全面的例子，说明一下华为IoT平台编解码插件线下开发的整个开发流程。对于环境的搭建尽量一笔带过，对核心编码部分做比较详细的讲解。第二章Profile部分会先出给一个小例子作为demo，编解码的编写按照该pfofile中定义的字段来解析，以方便大家理解编解码插件中的代码。本文编写过程中主要参考了“华为IoT平台NB-IoT设备集成开发指南.pdf”。

# 环境搭建

开发编解码插件使用的IDE是eclipse，语言是Java。本文尽量以简单的方式告诉你如何将华为提供的编解码样例修改为适合自己的编解码插件。即使你没有什么Java基础，只要懂得编程逻辑就好。跟着一起来吧！

## 1、安装JDK

JDK版本1.8以上。参考网络教程：

<https://blog.csdn.net/u012934325/article/details/73441617/>

## 2、安装eclipse

下载并安装eclipse。eclipse下载后解压缩到本地即可运行。

## 3、安装Maven

下载地址是： <http://maven.apache.org/download.cgi>

下载后的文件解压缩即可，然后添加环境变量。请参考：

<https://www.cnblogs.com/liuhongfeng/p/5057827.html>

在eclipse中配置maven插件。打开eclipse，点击window->Preferences，弹出如下窗口：

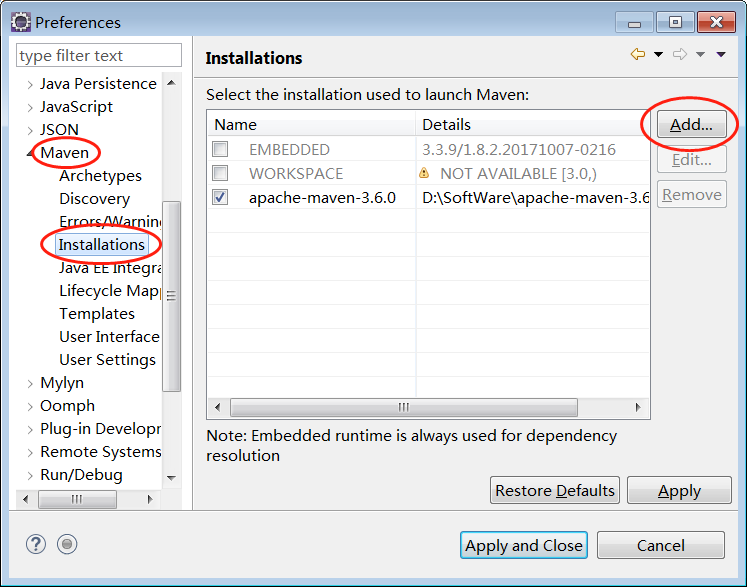


图1 Maven配置1

在弹出的窗口左侧栏中，依次找到并点击Maven->Installations，在右侧点击Add按钮，弹出如下图所示窗口：

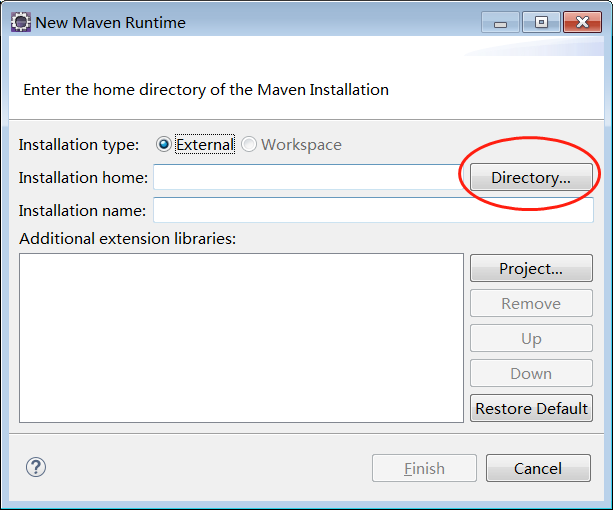


图2 Maven配置2

点击Directory，选择Maven的路径，然后Finish。到此，Maven配置完成。

# Profile说明

Profile实际上是一系列关于设备模型的描述文件，每个文件都使用JSON格式（键值对）。

Profile中首先需要说明设备的基本信息，包括厂商ID，厂商名称，设备类型，接入协议，以及设备可以提供的哪些服务等；其次，profile中要针对每一项服务，用一个独立的文件进行详细描述。服务，可以理解为是对设备消息（上下行）功能的一个分类，一个服务就代表一类功能；每个服务下包含若干属性和命令，每个属性对应上报消息中的某一个数据，每个命令字段则对应下行消息中的某些字段。比如，一个电表设备，会上报电池电量、功率、电能、电压等，可以将电池电量放在Battery服务中，属性值为batteryLevel，将功率、电能、电压都放在Transmission服务中，分别对应该服务下的Power、Energy、Voltage属性。可以在一条消息中上报所有服务的所有属性，也可以分开上报。

本例中提供的profile信息，基本信息如下：

设备类型： MyType

设备型号： MyModel

厂商ID ： ThirdParty

厂商名称： ThirdParty

协议： Coap

数据服务有两项：Battery，包含一个属性BatteryLevel，两个字节；Transmission，包含一个属性upData，不定长数组（profile中表示不定长数组，需将属性类型定义为string，长度设为一个比较大的数）；包含一条下行命令CLOUDREQ，有两个命令字段；cmdType，一个字节；downData，不定长数组。

profile采用在线开发的方式，如下图所示：

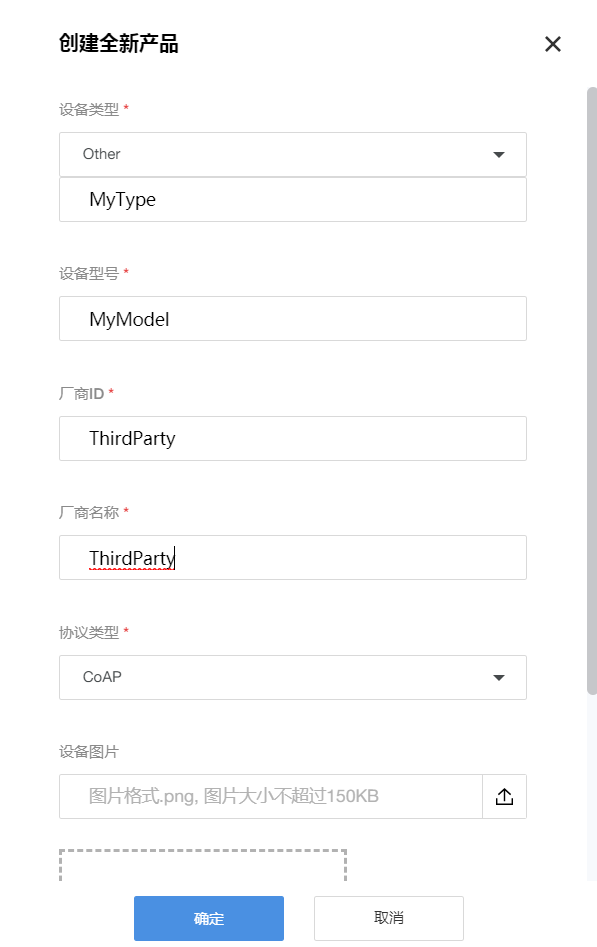


图3 设备信息

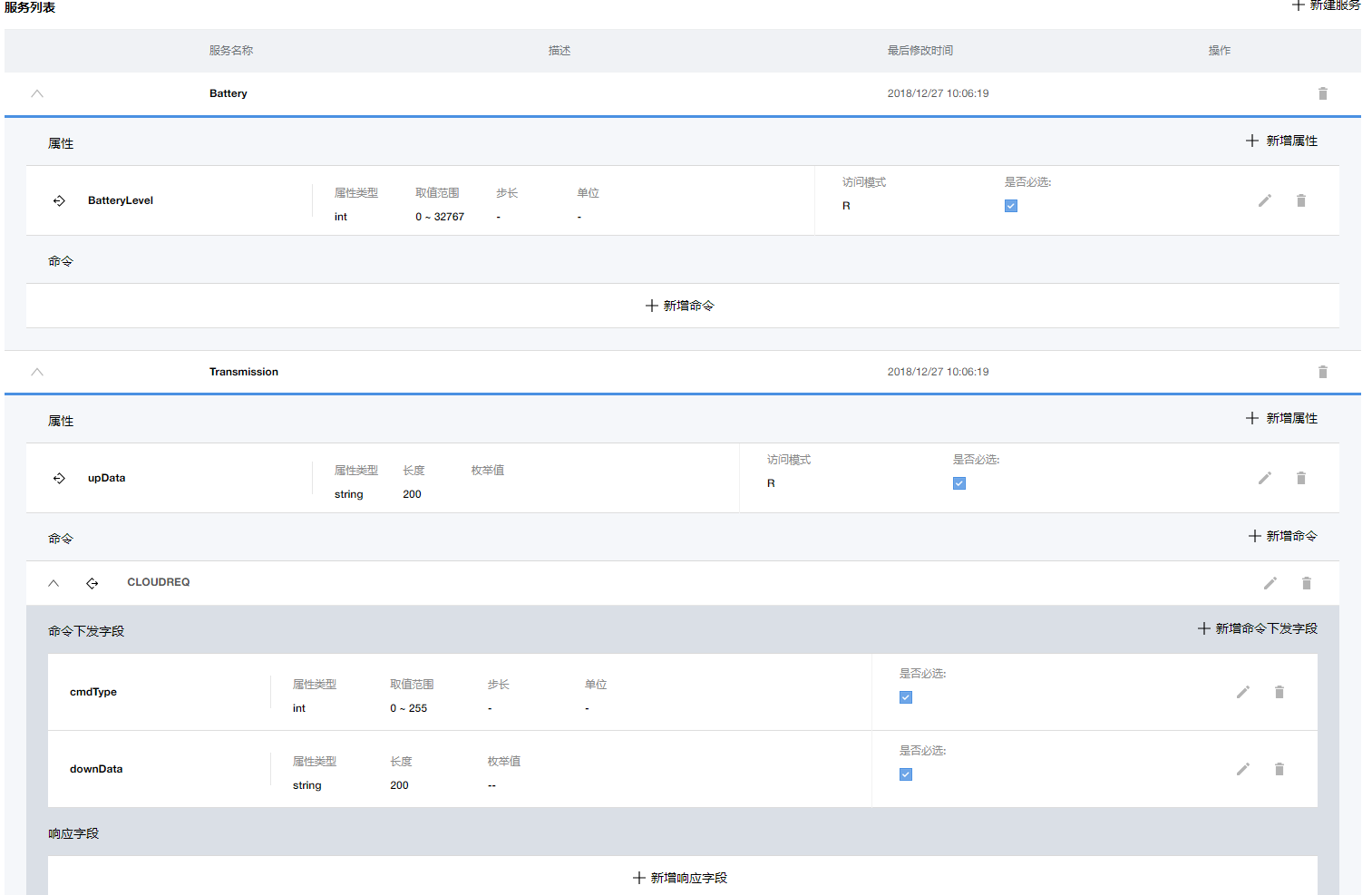


图4 服务（service）

本例中，采用一条消息上报所有服务属性的方式。上报数据格式为：前两个字节表示batteryLevel属性，大端方式；第三个字节表示后续数据长度；第四个字节至最后，表示upData属性。下行命令数据格式为：第一个字节表示cmdType；第二个字节至最后，表示downData。（请仔细理解该数据格式，插件的编写就是按照数据格式解析出对应的属性值）。

# 插件编写

## 1、源文件说明

从华为资源中心下载编解码插件Demo，并解压到本地。文件结构如下图所示：

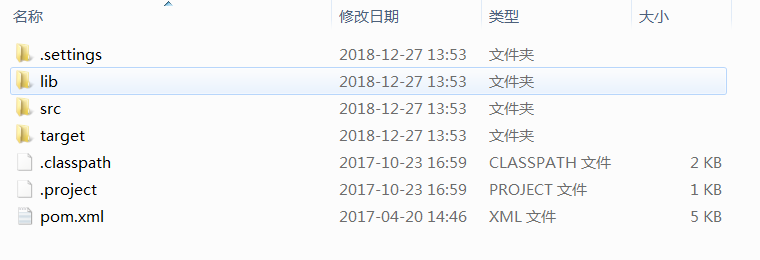


图5 编解码插件工程文件结构

源代码在src文件夹下；编译生成的插件包在target文件夹下。src 文件夹包含 main 、test 两个子文件夹，main下存放源码，test下是单元测试代码。官网下载的Demo中，源码的路径是：src\main\java\com\Huawei\NBIoTDevice\WaterMeter，单元测试代码的路径是：src\test\java\com\Huawei\NBIoTDevice\WaterMeter。

插件源码文件有5个：

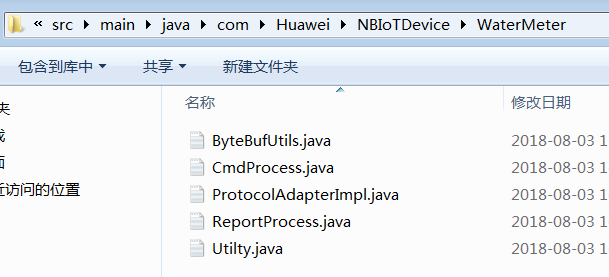


图6 源文件

1. ProtocolAdapterImpl.java 可以理解为是插件的入口文件，对外提供调用接口。该文件只需要修改两个字符串的定义即可：

// 厂商名称

**private** **static** **final** String ***MANU\_FACTURERID*** = "Huawei";

// 设备型号

**private** **static** **final** String ***MODEL*** = "NBIoTDevice";

修改为profile当中定义的厂商ID和设备型号。

1. CmdProcess.java 实现下行命令的编码工作，将从收到的服务器报文中提取出命令字段对应的内容，并将其转换成字节流。需要实现的函数是： **public byte[]** toByte()。
2. ReportProcess.java 实现将收到的二进制码流按照格式解码出对应profile中的属性值，并生成JSON格式。需要实现的函数是：

**public** ReportProcess(**byte**[] binaryData)，根据二进制码流的格式，从中取出对应字节，转换成profile中对应属性的值。

**public** ObjectNode toJsonNode()，将解码出来的属性值封装成JSON格式。

1. ByteBufUtils.java 和 Utilty.java文件封装了一些公共方法，不用做修改。也不会使用到。

## 2、修改文件路径（包名）

插件包名的要求是：com.厂商名称.设备型号.设备类型。因此下载下来的代码，要根据自己的设备修改下文件路径。即将Huawei文件夹重命名为profile中定义的厂商名称，NBIoTDevice文件夹重命名为profile中定义的设备型号，WaterMeter文件夹重命名为profile中定义的设备类型。注意：src\main 和src\test 下都要修改。在本例中，需要修改为：

src\main\java\com\ThirdParty\MyModel\MyTyp，

src\test\java\com\ThirdParty\MyModel\MyType

## 3、修改pom.xml

打开pom.xml文件，修改第7行“artifactId”和第88行“Bundle-SymbolicName”的值为：设备类型-厂商ID-设备型号。在本例中，需要修改为：MyType-ThirdParty-MyModel。

## 4、导入工程

打开eclipse，点击file->import，在弹出窗口中选择maven工程，如下图所示：

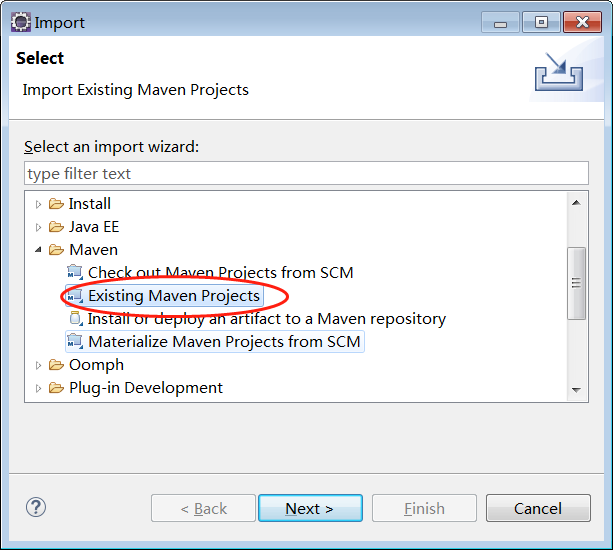


图7 工程导入

之后在弹出的窗口中，点击Browse，选择工程路径（pom.xml文件所在路径）。工程导入后如下图8所示：

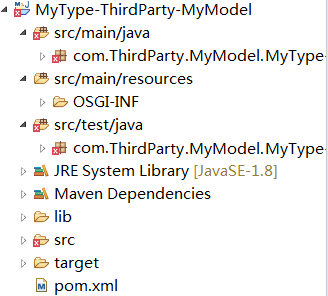


图8 工程导入

从图6可以看到首次导入工程后是有错误的。这是因为我们在第2节中将文件路径修改了，与代码里面的包路径不一致引起的。解决方法为：依次打开源文件，将第一行的

**package** com.Huawei.NBIoTDevice.WaterMeter;

修改为

**package** com.ThirdParty.MyModel.MyType;

打开OSGI\_INF目录下的CodeProvideHandler.xml 文件：

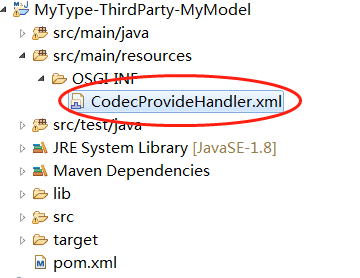


图9 CodeProvideHandler.xml路径

打开后，文件内容如下图所示：

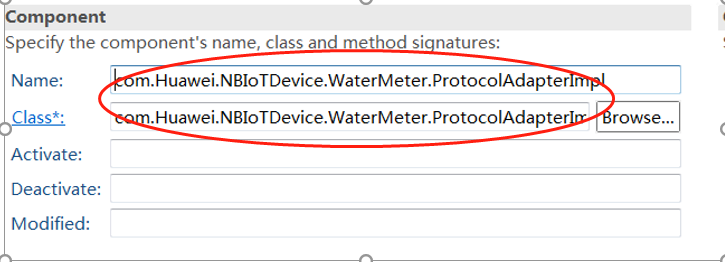


图10 CodeProvideHandler.xml内容

将Name 、 Class\* 内的路径也修改为对应的包路径：

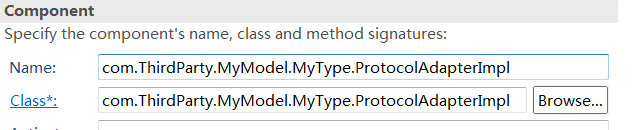


图11 CodeProvideHandler.xml修改

## 5、代码实现

第1节中说明了各个源文件要修改的地方，本节中具体讲解实现的方法。

1. 修改ProtocolAdatpterImpl.java文件

在文件中找到如下两行：

// 厂商名称

**private** **static** **final** String ***MANU\_FACTURERID*** = "Huawei";

// 设备型号

**private** **static** **final** String ***MODEL*** = "NBIoTDevice";

将***MANU\_FACTURERID*** 和 ***MODEL***定义修改为profile中定义的厂商ID和设备型号，本例中需要修改为：

// 厂商名称

**private** **static** **final** String ***MANU\_FACTURERID*** = "ThirdParty";

// 设备型号

**private** **static** **final** String ***MODEL*** = "MyModel";

1. 解码实现

解码，是将NB模组上报的二进制码流按格式解析出对应字段的过程。解码的代码在ReportProcess.java 文件中。

第一个函数：**public** ReportProcess(**byte**[] binaryData) 入参 byte[] binaryData就是NB模组上报的二进制码流。解码得到数据存储在成员变量当中。本例中的代码实现如下：



图12 解码函数实现

NB上报二进制数据的格式为：前两个字节表示batteryLevel，大端，整型；第三个字节表示后边还有多少字节；第四个字节往后表示不定长字段upData。因此，解码的思路便是：

1. 首先判断数据长度是否合法，至少应为3个字节，对应第37行代码；数据长度应不小于第3个字节的值加上3，对应第42行代码。该部分代码属于保护性代码。
2. 将前两个字节拼成一个16位的整型数据，表示batteryLevel，对应第47行代码。
3. 根据第三个字节的值，创建一个Byte数组，将第四个字节往后的内容拷贝至该数组内，得到upData。对应第50~53行代码。

System.*arraycopy* 是JDK提供的数组拷贝函数:第一个参数是源数组，第二个参数是偏移，表示从源数组的第几个字节开始拷贝，第三个参数是目的数组，第四个参数是目的数组的偏移，第5个参数表示拷贝的长度。

第二个函数：**public** ObjectNode toJsonNode() 返回一个ObjectNode对象（JSON）。该函数的功能，是将解码后得到的数据，按照规定格式填入一个JSON对象中。本例中，生成的JSON对象的内容格式如下图所示：

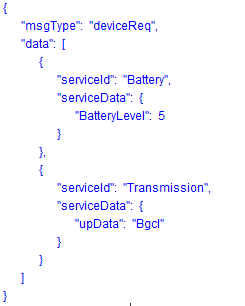


图13 上报数据格式

JSON对象的内容格式要求是：

"msgType": "deviceReq", 表示设备上报数据，固定不动；“data”：数组对象，数组中的每个元素分别对应profile中的一个服务；“serviceID”的值是profile中定义的服务名称；“serviceData”的值是该服务下所有的属性值。（本例中，profile定义了两个服务，Battery服务中有一个BatteryLevel属性;Transmission服务中有一个upData属性）。由图13的“upData”的值可以看出，数组类型的值，需要将二进制流转成base64编码的格式。

该函数的代码实现如下图所示：



图14 toJsonNode函数实现

该函数代码比较简单，主要是用到了 ObjectMapper 这个类，该类提供了JAVA中操作JSON数据的方法，可对照图13上报数据格式，仔细理解该部分代码。

1. 编码实现

编码，是将IoT平台收到的服务器下行数据（服务器下行数据是http或者https协议），从中提取出下行字段，并将其拼成二进制码流。编码部分的代码在 CmdProcess.java 文件中。需要实现的函数是：

**public** **byte**[] toByte()

本例中，该函数的实现代码如下图所示：



图15 toByte函数实现

服务器下行命令的JSON数据格式是：

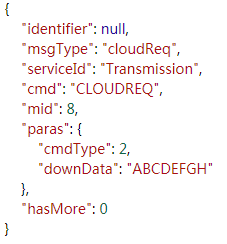


图16 服务器下行命令JSON数据格式

"msgType": "cloudReq", 固定值，表示服务器下行命令；

"serviceId"，profile中对应的服务，本例中是"Transmission",

"cmd"，profile中定义的下行命令，本例中是"CLOUDREQ",

"paras"，profile中定义的下行命令的各个字段，本例中是cmdType和downData两个字段；图16中，cmdType的值是2，downData是一个不定长数组，base64编码格式。

因此，编码的思路是：

1. 判断下行命令是否是profile中定义的。对应第77行代码。
2. 获取cmdType字段的值，该值占一个字节。对应第78行代码。
3. 获取downData的值，该值是base64编码形式，需转成二进制码流，对应第79行代码。
4. 将两个字段的值拼接成一个二进制数组，并返回。对应第81~84行代码。

## 6、生成Jar包

经过前面的工作后，代码就已经准备好了，接下来是生成JAR包。在DOS窗口中进入pom.xml文件所在路径，执行 mvn package 命令，最后弹出如下图所示的结果，则表明生成Jar包成功。如果有错误，则根据提示再去修改代码，然后重新执行 mvn package。

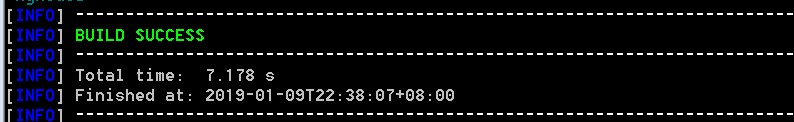


图17 生成JAR包

在工程目录的target文件夹下，存放生成的JAR包“MyType-ThirdParty-MyModel-1.0.0.jar”。JAR包的命名规则是：

设备类型-厂商ID-设备型号-版本号.jar

# 插件打包

1. 新建package文件，包含一个“preload”子文件夹，将上一章中生成的JAR包拷贝至preload文件夹下。
2. 在package文件夹中新建“package-info.json”文件（文本格式）。打开该文件，以UTF-8无BOM格式编辑，将以下大括号内容拷入该文件中并保存。

{

"specVersion": "1.0",

"fileName": "package.zip",

"version": "1.0.0",

"deviceType": "MyType",

"manufacturerName": "ThirdParty",

"model": "MyModel",

"description": "CIG codec plugin auto-generated by sps.",

"platform": "linux",

"packageType": "CIGPlugin",

"date": "Tue Nov 27 07:55:49 GMT 2018",

"ignoreList": [],

"bundles": [{

"bundleName": "MyType-ThirdParty-MyModel",

"bundleVersion": "1.0.0",

"priority": 5,

"fileName": "MyType-ThirdParty-MyModel-1.0.0.jar",

"bundleDesc": "",

"versionDesc": ""

}]

}

注： 在移植到别的项目中的时候，该文件需要修改的地方有：

"deviceType"，需根据实际的profile填写设备类型

"manufacturerName"，需根据实际的profile填写厂商名称

"model"，需根据实际的profile填写设备型号

"bundleName"，根据实际的profile填写，设备型号-厂商ID-设备类型

"fileName"，jar包的名称

1. 选中"package"文件夹中的全部文件，打包成zip格式。（“package.zip”，该压缩包内不能包含“package”目录）

说明：本章内容可参考 “华为IoT平台NB-IoT设备集成开发指南.pdf” 6.5.4.2.3章节的“制作插件包”部分的内容。

**package.zip 文件即为制作好的编解码插件包。**

# 插件质检

插件质检，是指插件开发完后之后，对插件的编解码功能的一个验证的过程，用于检测编解码插件中存在的问题。（当然，如果你对自己开发的插件足够自信的话，也可以不用进行质检咯，直接上传平台使用。本文强烈建议上传插件前进行质检工作。）

插件质检的过程比较简单，本文不再赘述，可通过下面的文章学习。



# 插件签名

插件签名，是指对生成的插件包进行数字签名，防止别人恶意篡改你的插件包，保证文件的完整性。

插件签名不属于开发过程，仅在需要把插件发送给别人的时候（比如IoT平台维护人员），一般要求要对插件进行签名。签名过程如下：

1. 解压“signtool.zip”，运行“signtool.exe”，运行界面如下图18所示。工具有三块功能，最上边用于生成公私密钥对，中间用于对文件进行签名，最下边用于校验。



图18 签名工具

1. 生成公私密钥对。点击“签名算法”选择区域，选择一个算法，在“私钥加密口令”输入框中输入私密加密口令（由自己设定的一个密码），然后点击 “生成公私密钥”按钮，在弹出的窗口中选择密钥保存的位置。在该位置会生成两个文件：

公钥： public.pem

私钥： private.pem

1. 签名。点击 “导入私钥文件”按钮，在弹出窗口中选择上一步中生成的私钥文件（private.pem）；点击“需要数字签名的软件”右侧的文件夹按钮，在弹出窗口中选择要签名的插件包（插件包路径不能有中文）；点击 “进行数字签名”按钮，会在插件包所在的路径生成签名后的文件，名称为 “xxx-signed.xxx”。
2. 验签。发送插件包给别人时，要将公钥和签名文件一并发送过去。

点击 “导入公钥文件”，在弹出窗口中选择公钥文件；点击 “需要验签的软件” 右侧的文件夹按钮，在弹出窗口中选择收到的已签名的插件包（插件包路径不能有中文），点击 “进行软件包验签”按钮。

验证成功则弹出 “验证签名成功”。

验证失败则弹出 “验签异常”，说明收到的插件包可能被异常修改过。

# 附件

附件中给出本例中的profile和插件工程、插件包，以及质检工具、签名工具等。

## 1、profile



## 2、插件工程源码



## 3、插件包



## 4、插件质检工具



## 5、签名工具



## 6、华为IoT平台NB-IoT设备集成开发指南

