**OpenBmc配置文档**

修订记录

|  |  |  |  |
| --- | --- | --- | --- |
| **日期** | **版本** | **描述** | **作者** |
| 2022/5/5 | 0.1 | 初稿 | 郑东雄 |

目 录

[修订记录 I](#_Toc102828115)

[目 录 II](#_Toc102828116)

[1. 简介 1](#_Toc102828117)

[2. Yocto 1](#_Toc102828118)

[2.1 官方参考文档 1](#_Toc102828119)

[2.2 介绍 1](#_Toc102828120)

[2.3 工作流程 1](#_Toc102828121)

[3. BitBake介绍 2](#_Toc102828122)

[3.1 官方参考文档 2](#_Toc102828123)

[3.2 BitBake配置文件语法 2](#_Toc102828124)

[3.2.1 运算符 2](#_Toc102828125)

[3.2.2 通用功能 3](#_Toc102828126)

[3.2.3 函数 4](#_Toc102828127)

[3.2.3.1 Shell函数 4](#_Toc102828128)

[3.2.3.2 BitBake样式的python函数 5](#_Toc102828129)

[3.2.3.3 python函数 6](#_Toc102828130)

[3.2.3.4 匿名python函数 6](#_Toc102828131)

[3.3 任务 7](#_Toc102828132)

[3.3.1 查看Recipes任务 7](#_Toc102828133)

[3.3.2 增加任务 8](#_Toc102828134)

[3.3.3 删除任务 8](#_Toc102828135)

[3.4 配置文件 8](#_Toc102828136)

[3.4.1 Configuration 8](#_Toc102828137)

[3.4.2 Recipes 10](#_Toc102828138)

[3.4.3 Class 11](#_Toc102828139)

[4. Openbmc 11](#_Toc102828140)

[4.1 目录结构 11](#_Toc102828141)

[4.2 下载与编译 13](#_Toc102828142)

[4.2.1 源码 13](#_Toc102828143)

[4.2.2 编译 13](#_Toc102828144)

[4.2.3 启动 14](#_Toc102828145)

[4.3 自定义镜像 14](#_Toc102828146)

[4.3.1 传感器 14](#_Toc102828147)

[4.3.1.1 温度传感器 15](#_Toc102828148)

[4.3.1.2 电压传感器 16](#_Toc102828149)

[4.3.1.3 风扇 16](#_Toc102828150)

[4.3.2 ipmi 16](#_Toc102828151)

[4.3.2.1 yaml文件 18](#_Toc102828152)

1. 简介

BMC硬件中的硬件资源非常有限，因此，OpenBMC被设计为完整的Linux发行版，可以灵活地定制以支持不同的BMC SoC或板卡

OpenBmc使用Yocto Project作为底层构建和发行的框架，并结合OpenEmbedded，systemd以及D-BusOpen等技术来轻松定制管理平台。同时，OpenBMC包含一个用于与固件堆栈进行交互的Web应用程序，并添加了Redfish对硬件管理的支持。它支持常见的主机状态查看和控制、主机固件更新等功能

1. Yocto

## 官方参考文档

[Welcome to the Yocto Project Documentation — The Yocto Project ® 3.2 documentation](https://docs.yoctoproject.org/3.2/index.html)

## 介绍

Yocto 项目是一个开源协作项目，能帮助开发者为不同硬件架构的嵌入式产品，制作基于Linux的定制系统。Yocto Project 提供了灵活的工具集和开发环境，使世界各地的嵌入式设备开发人员通过共享技术、软件堆栈、配置和用于创建这些定制 Linux 映像的最佳实践进行协作

## 工作流程

Yocto 利用bitbake解析一系列配置文件（主要为recipes，即bb/bbappend文件）来创建任务列表，并根据依赖关系依次执行，执行流程大致如下:

Fetch get the source code

Extract unpack the sources

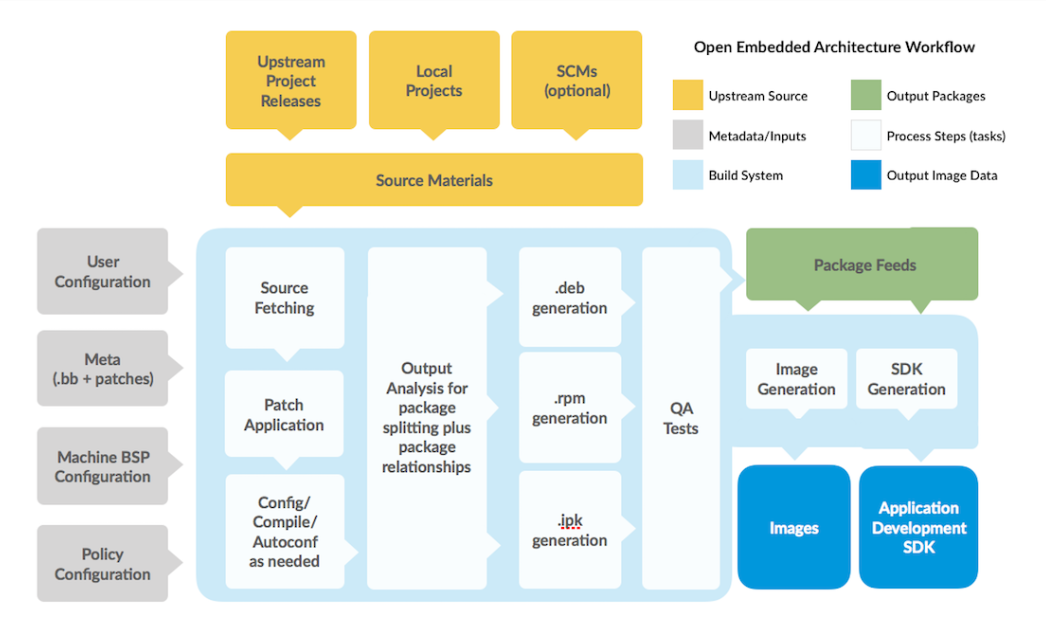
Patch apply patches for bug fixes and new capability

Configure set up your environment specifications

Build compile and link

Install copy files to target directories

Package bundle files for installation



1. BitBake介绍

## 官方参考文档

[BitBake User Manual — Bitbake dev documentation (yoctoproject.org)](https://docs.yoctoproject.org/bitbake/index.html)

## BitBake配置文件语法

### 运算符

|  |  |  |
| --- | --- | --- |
| **运算符** | **例子** | **说明** |
| ??= | VARIABLE ??= “value” | 弱默认赋值(如果变量没有赋值操作, 则赋值为最后一个“??=”赋值) |
| ?= | VARIABLE ?= “value” | 默认赋值(如果没有赋值将使用默认值) |
| = | VARIABLE = “value” | 直接赋值(最后的赋值将覆盖前面的赋值) |
| += | VARIABLE += “test” | 带空格的追加，VARIABLE最终结果为”value test” |
| =+ | VARIABLE =+ “value” | 带空格前新增，VARIABLE最终结果为”test value |
| .= | VARIABLE .= “value” | 不带空格的追加 |
| =. | VARIABLE =. “test” | 不带空格前新增 |
| :append | VARIABLE:append = “append” | 不带空格追加 |
| :prepend | VARIABLE:prepend = “value” | 不带空格前新增 |
| :remove | VARIABLE:remove = “value” | 移除(变量中以空格分列匹配) |
| @ | DATE = "${@time.strftime('%Y%m%d',time.gmtime())}" | 将后面的表达式视作python代码 |

### 通用功能

BitBake允许通过包含文件（.inc）和类文件（.bbclass）共享元数据。 例如，假设您具有一项通用功能，例如要在多个配方之间共享的任务定义。 在这种情况下，创建一个包含通用功能的.bbclass文件，然后在配方中使用inherit指令来继承该类将是共享任务的一种常用方法

BitBake使用BBPATH变量来查找所需的include文件和class文件。 此外，对于include和require指令，BitBake还会在当前目录中搜索

|  |  |  |
| --- | --- | --- |
| **指令** | **例子** | **说明** |
| include | include test.inc | 解析您指定的任何文件，并将该文件插入该位置，找不到文件不会报错 |
| inherit | inherit autotools | 仅支持在recipe和class文件中使用, 例子中继承autotools.bbclass文件，可以使用Autoconf和Automake工具构建软件包的任务 |
| INHERIT | INHERIT += “test” | 创建配置文件（.conf）时，可以使用INHERIT配置指令来继承类。 在配置文件中，BitBake仅支持此指令 |
| require | require foo.inc | 该指令的行为与include指令相同，不同之处在于，如果找不到要包含的文件，则BitBake会引发解析错误 |

### 函数

BitBake支持以下类型的函数

Shell函数：用Shell脚本编写并直接作为函数/任务执行的函数。也可以由其他Shell函数调用它们。

BitBake风格的Python函数：用Python编写并由BitBake执行的函数或其他使用bb.build.exec\_func() 调用执行的Python函数。

Python函数：用Python编写并由Python执行的函数。

匿名Python函数：在解析过程中自动执行的Python函数

#### Shell函数

用Shell脚本编写并直接作为函数、任务执行的函数。 也可以由其他Shell函数调用它们，同样支持追加语法

do\_foo() {

bbplain first

fn

}

fn:prepend() {

bbplain second

}

fn() {

bbplain third

}

do\_foo:append() {

bbplain fourth

}

执行结果

recipename do\_foo: first

recipename do\_foo: second

recipename do\_foo: third

recipename do\_foo: fourth

#### BitBake样式的python函数

这一类的函数是用Python编写的，由BitBake或其他Python函数使用bb.build.exec\_func()执行，同样支持追加语法

python do\_foo:prepend() {

bb.plain("first")

}

python do\_foo() {

bb.plain("second")

}

python do\_foo:append() {

bb.plain("third")

}

执行结果

recipename do\_foo: first

recipename do\_foo: second

recipename do\_foo: third

#### python函数

这些函数是用Python编写的，由其他Python代码执行。Python函数的一个例子是您打算从内联Python或其他Python函数中调用的实用函数

def get\_depends(d):

if d.getVar('SOMECONDITION'):

return "dependencywithcond"

else:

return "dependency"

SOMECONDITION = "1"

DEPENDS = "${@get\_depends(d)}"

执行结果

DEPENDS = “dependencywithcond”

注意:

Python函数可以带参数。

BitBake数据存储不是自动可用的。因此，必须将其作为参数传递给函数。

“bb”和“os”Python模块是自动可用的。您不需要导入它们

#### 匿名python函数

有时在解析过程中设置变量或可编程的执行其他操作很有用。 为此，您可以定义特殊的Python函数（称为匿名Python函数），该函数在解析结束时运行

python () {

d.setVar('FOO', 'foo 2')

}

FOO = "foo 1"

python () {

d.appendVar('BAR',' bar 2')

}

BAR = "bar 1"

执行后等价于:

FOO = "foo 1"

BAR = "bar 1"

FOO = "foo 2"

BAR += "bar 2"

## 任务

任务是BitBake执行单元，构成了BitBake执行给定配方的步骤。 仅配方和类（即.bb文件以及从.bb文件包含或继承的文件）支持任务。 按照约定，任务的名称以“do\_”开头

### 查看Recipes任务

使用说明:

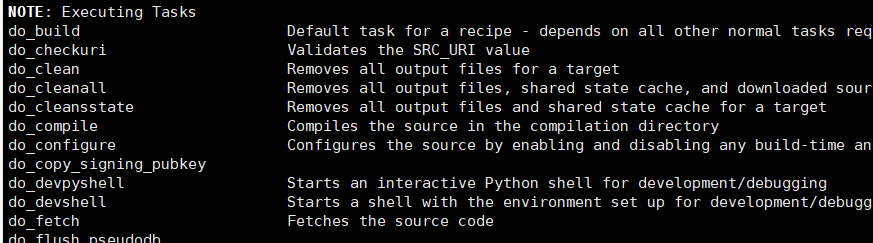
bitbake -c listtasks <recipes-name>

使用例子:

查看构建openbmc任务

bitbake -c listtasks obmc-phosphor-image

部分任务



### 增加任务

任何函数都可以使用addtask命令提升为任务。addtask命令还描述了任务间的依赖关系。下面的例子使用了上一节的函数，但是使用了addtask将它提升为一个任务，并定义了一些依赖关系

python do\_printdate () {

import time

print time.strftime('%Y%m%d', time.gmtime())

}

addtask printdate after do\_fetch before do\_build

说明:

在do\_build之后在do\_fetch之后执行addtask打印日期

### 删除任务

除了可以添加任务之外，您还可以删除它们。 只需使用deltask命令删除任务。 例如，要删除前面的示例任务

deltask printdate

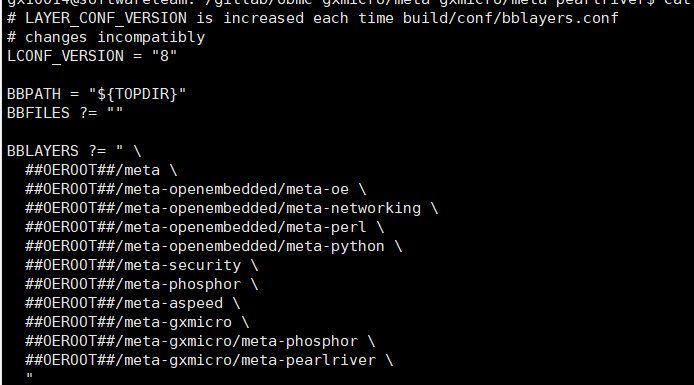
## 配置文件

### Configuration

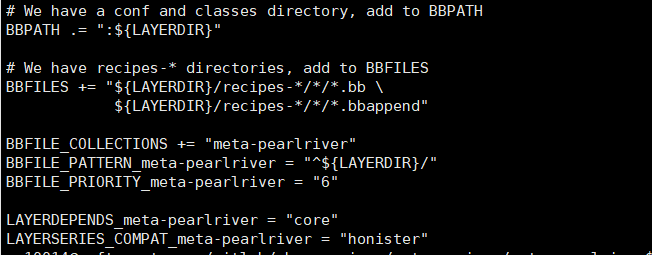
.conf文件，即配置文件，我们可以用它来改变构建方式，比如conf/local.conf可以配置机器配置选项，分发配置选项，编译器调整选项，常规通用配置选项和用户配置选项

|  |  |
| --- | --- |
| **变量** | **说明** |
| [BBPATH](https://docs.yoctoproject.org/3.2/ref-manual/ref-variables.html#term-BBPATH) | 定义此变量，指定bitbake的搜索路径 |
| [BBFILES](https://docs.yoctoproject.org/3.2/ref-manual/ref-variables.html#term-BBFILES) | 定义.bb/.bbappend位置 |
| [BBFILE\_COLLECTIONS](https://docs.yoctoproject.org/3.2/ref-manual/ref-variables.html#term-BBFILE_COLLECTIONS) | 通过在整个 OpenEmbedded 构建系统中使用的唯一标识符建立当前层，以引用该层 |
| [BBFILE\_PATTERN](https://docs.yoctoproject.org/3.2/ref-manual/ref-variables.html#term-BBFILE_PATTERN) | 解析时提供Layer目录 |
| [BBFILE\_PRIORITY](https://link.zhihu.com/?target=http%3A//www.yoctoproject.org/docs/2.7/ref-manual/ref-manual.html%23var-BBFILE_PRIORITY) | OE构建系统在不同Layer找到相同名字recipe时所参考的使用优先级 |
| [LAYERVERSION](https://link.zhihu.com/?target=http%3A//www.yoctoproject.org/docs/2.7/ref-manual/ref-manual.html%23var-LAYERVERSION) | Layer的版本号。可以通过LAYERDEPENDS变量指定使用特定版本号的Layer |
| [LAYERSERIES\_COMPAT](https://link.zhihu.com/?target=http%3A//www.yoctoproject.org/docs/2.7/ref-manual/ref-manual.html%23var-LAYERSERIES_COMPAT) | 列出当前版本兼容的Yocto Project释放版本。它可以表示Layer是否有效 |
| DL\_DIR | 指定下载文件存放路径，一般默认取值${TOPDIR}/downloads，如果已下载子在本地中,则在local.conf.sample指明本地路径，同时设置BB\_NO\_NETWORK = "1" |
| SOURCE\_MIRROR\_URL | 尝试从 [SRC\_URI](https://docs.yoctoproject.org/3.2/ref-manual/ref-variables.html#term-SRC_URI) 中指定的路径获取源之前，首先从中获取源，在local.conf.sample中同时添加INHERIT += "own-mirrors" |

bblayers.conf.sample文件



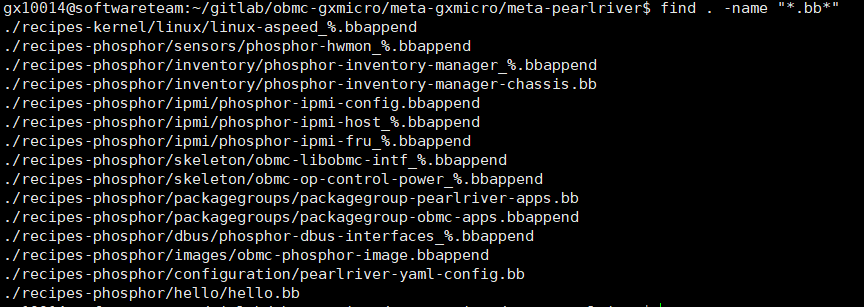
layer,conf文件:



### Recipes

.bb/.bbappend文件，即配方文件，描述了从哪获取软件源码，如何配置，如何编译。

.bbappend是对.bb文件的补充和覆盖，例如存在foo-1.bb和 foo-2.bb文件，可以用foo-%.bbappend来替代foo-1.bb和foo-2.bb，其中%为通配符



常用变量:

|  |  |
| --- | --- |
| **变量** | **说明** |
| DESCRIPTION | 软件包描述 |
| SRC\_URI | 文件下载(获取)路径 |
| BP | 基本软件名称和版本${BPN}-${PV}, 没有任何特殊的recipes名称后缀 |
| [PN](https://docs.yoctoproject.org/3.2/ref-manual/ref-variables.html#term-BBFILE_PATTERN) | package name 用于构建包的配方名称或包的名称 |
| BPN | PN变量移除常见的前缀或者后缀 |
| LICENSE | 指定recipes的一些列的license |
| DEPENDS | 指定构建依赖 |
| S | 假设bb文件中指定源码来源于git，则在 do\_fetch 期间，源码将被被克隆到 ${WORKDIR}/git目录中  S路径与默认值不同, 故此需要设置S = ${WORKDIR}/git |

### Class

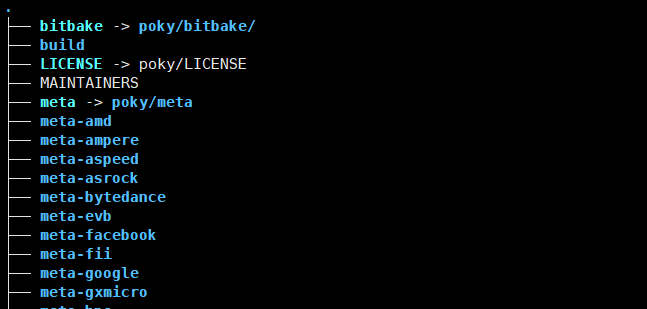
.bbclass文件，即类文件，包含在配方文件之间共享的有用信息。比如autotools该类，其中包含Autotools使用的任何应用程序的常用设置

1. Openbmc

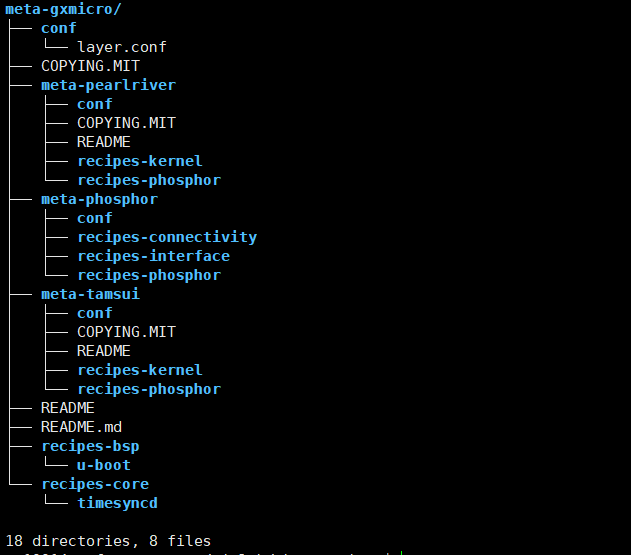
.conf文件，即配置文件，我们可以用它来改变构建方式，比如conf/local.conf可以配置机器配置选项，分发配置选项，编译器调整选项，常规通用配置选项和用户配置选项

## 目录结构

|  |  |
| --- | --- |
| **目录** | **说明** |
| meta-xxx | 厂商文件夹。有些厂商有多个机型或者使用多种BMC的soc，会在该文件夹下建立meta-xxx的文件夹 |
| meta | 核心部分 |
| meta-openembedded | 对meta的补充 |
| meta-phosphor | openbmc整个应用框架和应用软件的配置，我们编译镜像时，一定会编译该文件夹下的一些配置 |
| meta-aspeed | bmc的soc芯片相关的layer，主要包括soc编译优化、u-boot、kernel等相关的配置(不同芯片不同的) |
| meta-openpower | 电源管理相关的layer，依赖于meta-phosphor |



不同厂商下目录结构



## 下载与编译

### 源码

源码地址

git@code.gxmicro.com:bmc/obmc-gxmicro.git

下载源码

git clone git@code.gxmicro.com:bmc/obmc-gxmicro.git

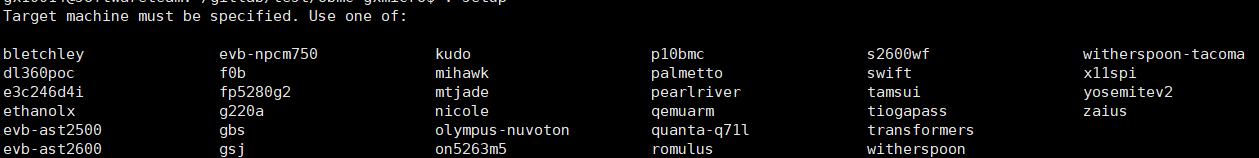
切换分支

cd obmc-gxmicro/ &&

### 编译

进入源码目录下，查看可编译的机型

. setup



选择机型，设置环境变量，

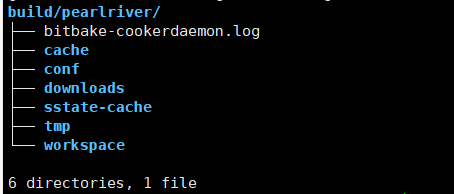
. setup pearlriver

使用bitbake命令编译

bitbake obmc-phosphor-image

编译输出目录

build/<machine>/



镜像文件位置

build/<machine>/tmp/deploy/images/<machine>

### 启动

编译后镜像文件位置

build/<machine>/tmp/deploy/images/<machine>

本地端口映射启动

qemu-system-arm -m 256 -M ast2500-evb -nographic -drive file=./flash-pearlriver,if=mtd -net nic -net user,hostfwd=:127.0.0.1:2222-:22,hostfwd=:127.0.0.1:2443-:443,hostname=qemu

安装docker后创建镜像后下载qemu-system-arm, docker创建容器, 将容器2443端口映射到本机16716端口

docker run --name="openbmc" -it -p 16716:2443 -v E:\docker\openbmc:/root/win ubuntu:22.04 env LANG=C.UTF-8 /bin/bash

容器中运行qemu, 将镜像中443端口映射到容器中2443端口

qemu-system-arm -m 256 -M ast2500-evb -nographic -drive file=./flash,if=mtd -net nic -net user,hostfwd=:0.0.0.0:2443-:443,hostname=qemu

参数:

drive 添加映像文件，一般用于块设备

hostfwd=[tcp|udp]:[hostaddr]:hostport-[guestaddr]:guestport

不指定hostaddr或guestaddr，则主机或虚拟机会默认绑定在0.0.0.0，即对所有IP地址都实现转发

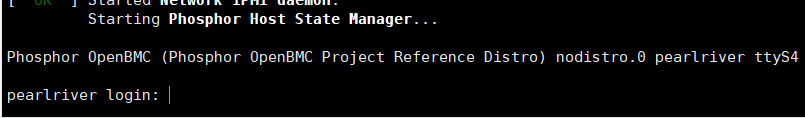
net 指定网络模式通常为user 和 tap

user模式下，客户机和宿主机之间可以建立tcp/udp连接，且可以配置端口转发将客户机中的端口转发到宿主机上

tap即在宿主机上创建一个虚拟网卡设备tap0，客户机使用这个网卡设备进行网络通信

退出qemu

ctrl + a, x



用户名与密码

用户名: root

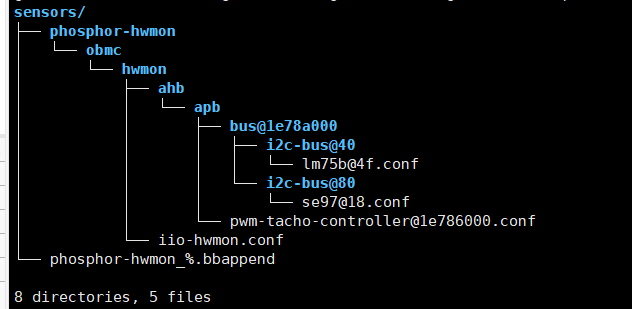
密码: 0penBmc

## 自定义镜像

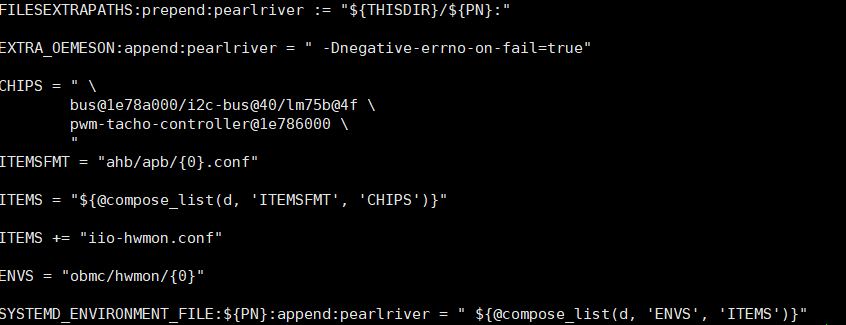
### 传感器

传感器配置文件存放目录，hwmon/子目录定义与设备树定义的相关

<machine>/<machine>/recipes-phosphor/sensors/phosphor-hwmon/obmc/hwmon/\*



同时自定义phosphor-hwmon\_%.bbappend对源码.bb文件扩展，定义如下



.conf配置文件内容定义

xxx\_yyyn = “value”

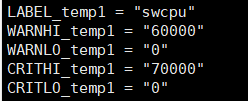
xxx : 属性

属性包括: LABEL, WARNHI, WARNLO CRITHI, CRITLO

n : 序号

yyy : 类型

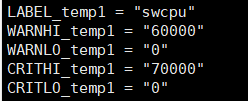
类型包括: temp, in(电压), \*(所有其它名称) 等



#### 温度传感器

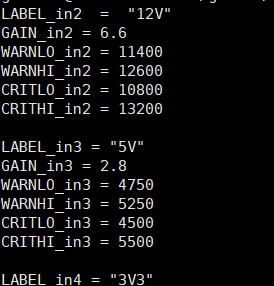
Lm75为例, 定义配置文件需要存放在sensors/phosphor-hwmon/obmc/hwmon下当前设备树描述Lm75挂载在ahb/apb/bus@1e78a000/i2c-bus@40/, 因此需要在sensors/phosphor-hwmon/obmc/hwmon下创建同样的目录结构





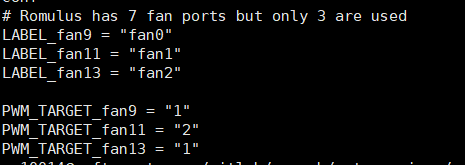
#### 电压传感器

iio-hwmon.conf部分描述如下



#### 风扇

[pwm-tacho-controller@1e786000.conf](mailto:pwm-tacho-controller@1e786000.conf)配置文件如下



### ipmi

phosphor-ipmi-hostd实现了ipmi大部分功能，比如常用的传感器信息获取、fru信息等等，是ipmi的[守护进程](https://so.csdn.net/so/search?q=%E5%AE%88%E6%8A%A4%E8%BF%9B%E7%A8%8B&spm=1001.2101.3001.7020" \t "_blank)，并提供dbus接口来响应和处理ipmi命令

源码

[https://github.com/openbmc/phosphor-host-ipmid](https://github.com/openbmc/phosphor-host-ipmid" \o "https://github.com/openbmc/phosphor-host-ipmid)

配置文件

meta-phosphor/recipes-phosphor/ipmi/phosphor-ipmi-host\_git.bb文件，该文件是描述软件包源码配置文件

meta-phosphor/recipes-phosphor/ipmi/phosphor-ipmi-config.bb文件，该文件中指定的是软件运行需要的配置文件（json）

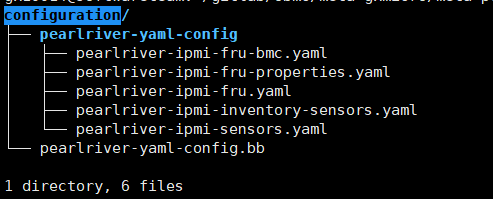
meta-phosphor/recipes-phosphor/ipmi/phosphor-ipmi-sensor-xxx.bb的几个文件，该文件中会指定yaml文件，yaml文件在编译阶段会通过scripts文件夹下的脚本生成源码并编译

软件包下有一个scripts文件夹，这里边是python脚本、模板、示例yaml文件以及使用说明，本内容介绍sensor的yaml文件。在configure阶段，通过autoconf工具，解析configure.ac以及Makefile.am生成Makefile文件，在编译阶段执行sensor\_gen.py的python脚本，利用writesensor.mako.cpp模板，并根据sensor.yaml文件生成sensor-gen.cpp文件

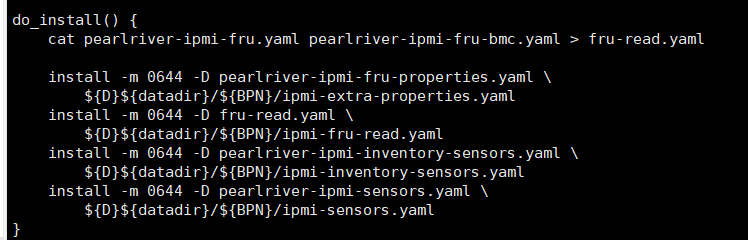
指定自定义的yaml文件

通过安装覆盖的方式，将yaml文件添加到以下路径

<machine>/<machine>/recipes-phosphor/configuration/<machine>-yaml-config



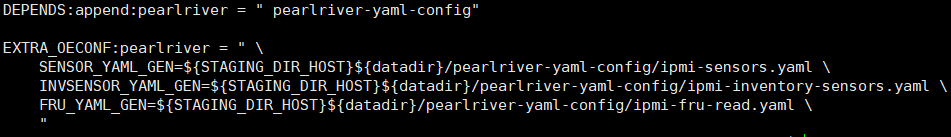
.bb文件部分内容如下



在以下路径添加.bbappend文件，可参考其它机型的.bbappend文件

<machine>/<machine>/recipes-phosphor/ipmi/phosphor-ipmi-host\_%.bbappend

内容如下



#### yaml文件

参考源码scripts目录下example

