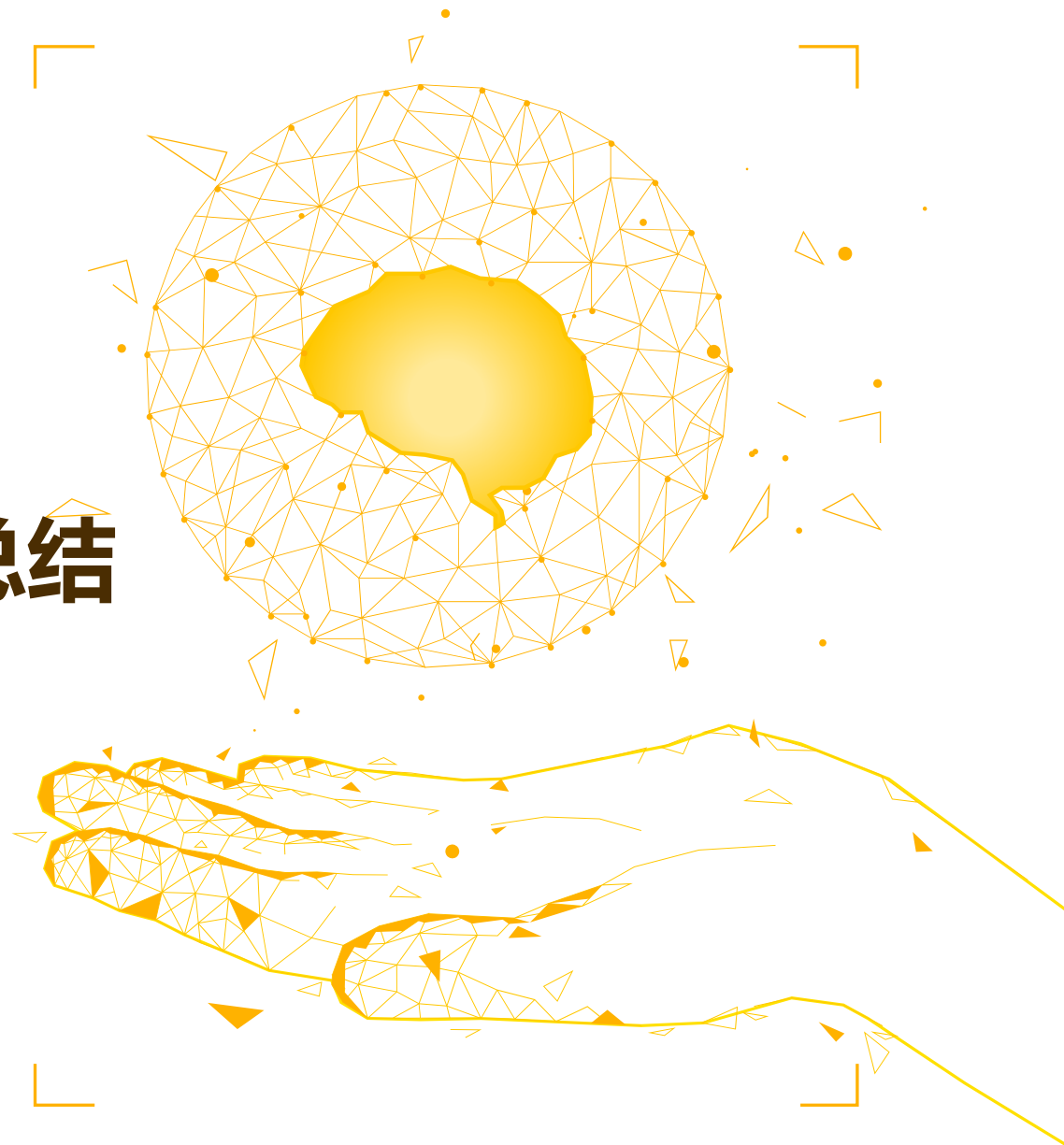


# RuyiSDK

## 上游镜像版本检测功能总结

第三测试小队 kina



# 报告内容

针对RuyiSDK上游镜像版本检测功能

我们所做的工作

以及过程中碰到的问题

# 目录

1

背景

2

所做的工作

3

存在的问题

4

未来期望

# 01



## 背景



# 背景

## RuyiSDK 介绍

### 介绍

RuyiSDK 是一个由 [PLCT Lab](#) 所启动的开源项目，该项目旨在为 RISC-V 开发者提供一个便捷、完善的开发环境。其提供了相关最新的硬件信息、软件支持，例如在支持的设备中有提供相关支持硬件情况；软件层面提供了镜像（如 [RevyOS](#)）、工具链、包管理等。

其最终目标是希望为 RISC-V 开发者提供一个完善、便捷的开发环境，使得 RISC-V 成为主流架构，以及建设并运营一个完善的社区以便开发者交流。最终希望 RuyiSDK 可以走向国际化，为全球的 RISC-V 开发者提供开发的便捷。

---

# 背景

## ruyi device provision 功能

<code>ruyi device provision</code>	下载所需系统镜像，并为设备安装系统。	按照引导进行系统安装。
------------------------------------	--------------------	-------------

# 背景

## ruyi device provision 演示

RuyiSDK Device Provisioning Wizard

This is a wizard intended to help you install a system on your device for your development pleasure, all with ease.

You will be asked some questions that help RuyiSDK understand your device and your intended configuration, then packages will be downloaded and flashed onto the device's storage, that you should somehow make available on this host system beforehand.

Note that, as Ruyi does not run as root, but raw disk access is most likely required to flash images, you should arrange to allow your user account sudo access to necessary commands such as dd. Flashing will fail if the sudo configuration does not allow so.

Continue? (y/N) y

The following devices are currently supported by the wizard. Please pick your device:

1. Allwinner Nezha D1
2. Canaan Kendryte K230
3. Milk-V Duo
4. Milk-V Pioneer Box
5. SiFive HiFive Unmatched
6. Sipeed Lichee RV
7. Sipeed LicheePi 4A
8. StarFive VisionFive
9. StarFive VisionFive2

Choice? (1-9)

Choice? (1-9) 7

The device has the following variants. Please choose the one corresponding to your hardware:

1. Sipeed LicheePi 4A (8G RAM)
2. Sipeed LicheePi 4A (16G RAM)

Choice? (1-2) 2

The following system configurations are supported by the device variant you have chosen to put on the device:

1. openEuler RISC-V (headless) for Sipeed LicheePi 4A (16G RAM)
2. openEuler RISC-V (XFCE) for Sipeed LicheePi 4A (16G RAM)
3. RevyOS for Sipeed LicheePi 4A (16G RAM)

Choice? (1-3) 3

We are about to download and install the following packages for your device:

- \* board-image/revyos-sipeed-lpi4a
- \* board-image/uboot-revyos-sipeed-lpi4a-16g

Proceed? (y/N)

# 背景

## 进行上游镜像版本检测功能工作的必要性

1. RuyiSDK ruyi device provision 功能上游系统镜像版本信息  
目前处于手动维护状态，镜像信息位于  
Github packages-index 仓库  
<https://github.com/ruyisdk/packages-index>
2. 手动维护很困难，同时存在诸如版本更新同步不及时，镜像下载  
链接失效等问题



# 02



## 所做的工作



# 所做的工作

## ruyi-reimu 介绍

ruyi-reimu 是第三测试小队在多测试工具、多测试环境下减轻人工运维和监控测试状态压力的辅助工具

当前尚未实现完全的自动化，但已经具有基本的功能，其中包括了 board image 上游版本的更新检测

<https://github.com/weilinfox/ruyi-reimu>

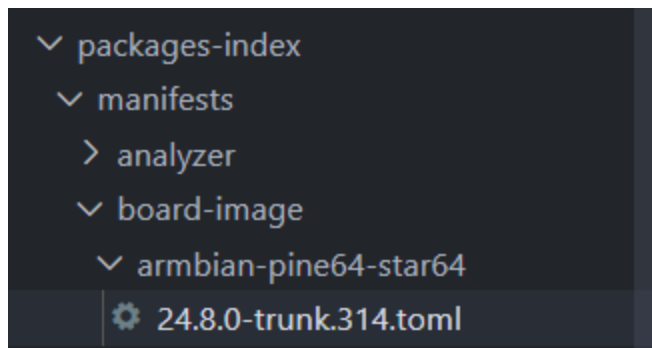
# 所做的工作

**board image 上游版本跟踪程序** 当前作为 **测试调度程序(ruyi-reimu)** 的一部分，大致流程如下：

1. 解析 packages-index 中的文件获取现版本信息
2. 根据 packages-index 中的 url 获取最新版本信息并进行对比
3. 判断现版本是否为最新版本，若否则自动发起issue

# 所做的工作

packages-index 信息 (以 armbian-pine64-star64 为例)



<https://github.com/ruyisdk/packages-index>

```
1  format = "v1"
2
3  [metadata]
4  desc = "Armbian 24.8.0-trunk.314 image for Pine64 Star64"
5  vendor = { name = "Armbian", eula = "" }
6
7  [[distfiles]]
8  name = "Armbian_community_24.8.0-trunk.314_Star64_noble_edge_5.15.0_minimal.img.xz"
9  size = 213362100
10 urls = [
11     "https://github.com/armbian/community/releases/download/24.8.0-trunk.314/Armbian_community_24.8.0-trunk.314_Star64_noble_edge_5.15.0_minimal.img.xz"
12 ]
13 restrict = ["mirror"]
14
15 [distfiles.checksums]
16 sha256 = "c5c7d091af27ede581113560c0e133da7b9c54a4a2529756ceb04e5407f87296"
17 sha512 = "0234becd82e36672afe48eab9714f8372733c940cbb622770193ff7e4d07936dd875d3d1adc2d0c956"
18
19 [blob]
20 distfiles = [
21     "Armbian_community_24.8.0-trunk.314_Star64_noble_edge_5.15.0_minimal.img.xz",
22 ]
23
24 [provisionable]
25 strategy = "dd-v1"
26
27 [provisionable.partition_map]
28 disk = "Armbian_community_24.8.0-trunk.314_Star64_noble_edge_5.15.0_minimal.img"
29
```

# 所做的工作

从 packages-index 中获取到的 RuyiSDK 上游镜像来源分为：

1. ISCAS 镜像站
2. 系统镜像发布方自行维护的上游源
3. Github Release

```
class RepoBoardImage:

    def __new__(cls, *args):
        image_type = ""
        board_image = args[1]

        if not isinstance(board_image, dict):
            return
        if len(board_image["files"]) == 1:
            image_type = "_SINGLE"
            if board_image["files"][0].host == "github.com":
                image_type = "GITHUB" + image_type
            else:
                image_type = "MIRROR" + image_type
```

# 所做的工作

针对镜像站或上游发布源识别使用的 Adapter (维护于 mirror.toml )

```
class MirrorAdapter:

    def __new__(cls, *args):

        host = args[1]
        if host == "mirror.iscas.ac.cn":
            cls = IscasMirrorAdapter
        elif host == "downloads.openwrt.org":
            cls = OpenWrtDownloadsMirrorAdapter
        elif host == "cdimage.ubuntu.com":
            cls = UbuntuCdimageMirrorAdapter
        elif host == "download.freebsd.org":
            cls = FreeBSDDownloadMirrorAdapter
        elif host == "releases.openkylin.top":
            cls = OpenkylinReleasesMirrorAdapter

        return object.__new__(cls)
```

ISCAS 镜像站

一些镜像的上游发布源

# 所做的工作

## 从 url 中获取版本信息的流程

### H3 执行逻辑

假设 packages-index 每个配置文件的 url 列表中，只有一个上游 url。维护一个上游 hostname 列表，从每个配置文件的 url 列表中找到这个上游链接，  
从该上游 url 中获取版本，再从对应页面获取版本列表。

以 RevyOS 为例，在这个 url

[https://mirror.iscas.ac.cn/revyos/extra/images/lpi4a/20240601/root-lpi4a-20240601\\_180941.ext4.zst](https://mirror.iscas.ac.cn/revyos/extra/images/lpi4a/20240601/root-lpi4a-20240601_180941.ext4.zst) 中，20240601 是版本，故可以从 <https://mirror.iscas.ac.cn/revyos/extra/images/lpi4a/> 页面获取整个版本列表，使用正则过滤非版本号路径。

上游 hostname 列表维护在 [mirrors.toml](https://mirrors.tpm.org.cn/mirrors.toml) 中，还是以 RevyOS 为例：

```
</>
1  ["mirror.iscas.ac.cn"]
2  repo="1"
3  repo_name="revyos"
4  version="-2"
5  version_match = "[0-9]+$"
```

从 url 得到一个 path 列表

`["mirror.iscas.ac.cn", "revyos", "extra", "images", "lpi4a", "20240601", "root-lpi4a-20240601_180941.ext4.zst"]`。

其中 `repo=1`，故显然 `repo_name` 为 `revyos`，这里由于同一 hostname 下还存在其他镜像（如 oErv），配置中指定了 `repo_name`；

其中 `version="-2"`，故 `version_name` 为 `20240601`，该 `version_name` 显然符合 `version_match` 的定义。

注意 `repo` 和 `version` 这类字段还可以定义为切片，使用 `..` 操作符，如 `repo="1..-2"` 得到 `repo_name` 为

`revyos/extra/images/lpi4a/20240601`。

# 所做的工作

生成判断信息，检测当前是否为最新版本

```
def check(self) -> dict:
    """

    :return: {
        "update": bool,
        "latest_version": str,
        "latest_url": str,
        "current_version": str,
        "upstream_repo": str
    } items are valid when "update" is False.
    """

    logger.warn("This is a father class, and IDK what to check")
    return {"update": True}
```



# 所做的工作

针对 Github Release 需要特殊处理 (按照 release tag 匹配检测版本)

```
def check(self) -> dict:
    logger.info("Check board image {} on github repo {}".format(self.title, self.upstream_repo))
    version_list = {}
    for f in self.files:
        if f.version_name in version_list.keys():
            version_list[f.version_name].append(f.filename)
        else:
            version_list[f.version_name] = [f.filename]

    upstream_releases = gh_op.get_repo_releases(self.upstream_repo)
    latest_version = ""
    ruyi_version = ""

    for u in upstream_releases:
        # get latest version
        # versions are sort in time,
        # but we cannot sort these version code
        # they could in invalid version format
        if latest_version == "":
            latest_version = u.tag_name # 使用 tag 进行版本比较

        # check assets
        for v in version_list.keys():
            if v == u.tag_name: # 直接使用字符串进行版本比较
                ruyi_version = v
                break

    if ruyi_version:
        break
```

# 所做的工作

版本检测后 判断结果 并利用 Github API 自动发送 issue

```
@staticmethod
def send_issue(board_image: RepoBoardImage, info: dict):
    """
    Send issue
    :param board_image:
    :param info:
    :return:
    """

    if info["update"]:
        logger.info("Board image {} already the latest".format(board_image.title))
        return

    title = "[ruyi-reimu] Board image {} need update".format(board_image.title)
    body = "## Description\n"

    # send issue
    logger.info("Info about the latest {}".format(info))
    body += ("\n+ In upstream repo <{}>, the latest version is [{}]({})"
            .format(info["upstream_repo"], info["latest_version"], info["latest_url"]))
    body += ("\n+ The current version in ruyi upstream is {}".format(info["current_version"]))

    packidx = ("{} /tree/{}/manifests/board-image/{}".format(
        reimu_config.ruyi_repo, reimu_config.ruyi_repo_branch, board_image.title))
    body += ("\n+ The packages-index info is [{}]({})".format(packidx, packidx))

    gh_op.create_issue(reimu_config.issue_to, title, body)
```

# 所做的工作

## 自动创建的 issue (示例)



delete-cloud commented last week

Owner



### Description

- In upstream repo <https://mirror.iscas.ac.cn/revyos/extra/images/lpi4a/>, the latest version is [20240601](#)
- The current version in ruyi upstream is 20231210
- The packages-index info is <https://github.com/ruyisdk/packages-index/tree/main/manifests/board-image/uboot-revyos-sipeed-lpi4a-8g>



# 03



## 存在的问题



# 存在的问题

## 1. 版本排序问题

在 GitHub Release 中，已经以发布时间排序，没有进行二次排序；在镜像源中，则简单以字符串排序的方式做了排序，当前可以适用，未来未知；而 packages-index 中的版本号符合 Semver 规范，故直接使用 semver 库进行比较排序。

```
=====
Check board image uboot-revyos-milkv-meles-4g on github repo milkv-meles/meles-images
Release: tag_name=v1.0.0, title=, created_at=2024-03-15 09:53:51+00:00
Release: tag_name=v2024-0417, title=v2024-0417, created_at=2024-03-15 09:53:51+00:00
Release: tag_name=v2024-0329, title=v2024-0329, created_at=2024-03-15 09:53:51+00:00
Board image uboot-revyos-milkv-meles-4g already the latest
```

默认按tag的话少了3款：  
uboot-revyos-milkv-meles-8g  
uboot-revyos-milkv-meles-4g  
revyos-milkv-meles  
图里是其中一款，这三个上游在同一个仓库，因为create\_at时间一样，似乎默认匹配的是第一个v1.0.0，就识别为不需要更新了

14:51 ✓✓

Github Release 部分镜像按 tag 匹配仍然存在问题

# 存在的问题

## 2. 从版本号推断镜像文件名

当前在跟踪到新版本后，还需要人工二次验证。

在一些镜像中，所有版本的文件名相同；

在一些镜像中，版本号被包含在文件名中，通过版本号可以推断文件名；

而在另一些镜像中，文件名变更随意，无法推断。这也导致自动的镜像更新实现有一定的困难。

# 存在的问题

## 3. 特定情况无法跟踪上游

### H3 执行逻辑

假设 packages-index 每个配置文件的 url 列表中，只有一个上游 url。维护一个上游 hostname 列表，从每个配置文件的 url 列表中找到这个上游链接，  
从该上游 url 中获取版本，再从对应页面获取版本列表。

能够跟踪检测的情况为假设 每个 board image 只有一个上游 url 存在

<https://github.com/ruyisdk/packages-index/blob/main/manifests/board-image/openbsd-riscv64-live/7.4.0.json>


中的情况，packages-index里没给出 OpenBSD 上游链接，只存在 tuna 镜像源的链接，因此无法跟踪

# 存在的问题

## 4. 上游的不可控性


例如: packages-index 中上游链接失效的情况

ruyi device provision 中 armbian-pine64-star64 镜像上游 url 失效

 Closed

 delete-cloud opened this issue last week · 1 comment

---

 delete-cloud commented last week

[package-index](#) 中 [armbian-pine64-star64](#) 的 镜像 url 已失效

**原因:**

armbian 仓库 [release](#) 更新, 删除了之前的 tag 导致链接失效, 目前有效 url 为 [24.8.0-trunk.314](#) 与 [24.8.0-trunk.205](#)

<https://github.com/ruyisdk/ruyi/issues/175>



# 04



## 未来期望



# 未来期望

统一的命名规范？

目前 RuyiSDK 处于高速发展期，各厂商 board image 变更情况也很频繁，  
对于版本检测因为上游情况变更频繁，导致目前只能跟随上游进行后续动作  
等待各方面成熟之后，完全实现自动化

# 参考资料

- |                       |  |
|-----------------------|--|
| [1] RuyiSDK 文档        | <a href="https://ruyisdk.github.io/docs/zh/"><u>https://ruyisdk.github.io/docs/zh/</u></a>               |
| [2] packages-index 仓库 | <a href="https://github.com/ruyisdk/packages-index"><u>https://github.com/ruyisdk/packages-index</u></a> |
| [3] ruyi-reimu 仓库     | <a href="https://github.com/weilinfox/ruyi-reimu"><u>https://github.com/weilinfox/ruyi-reimu</u></a>     |