Ruyi v0.2

为了开箱即用的开发环境，我们都做了什么

xen0n@PLCT

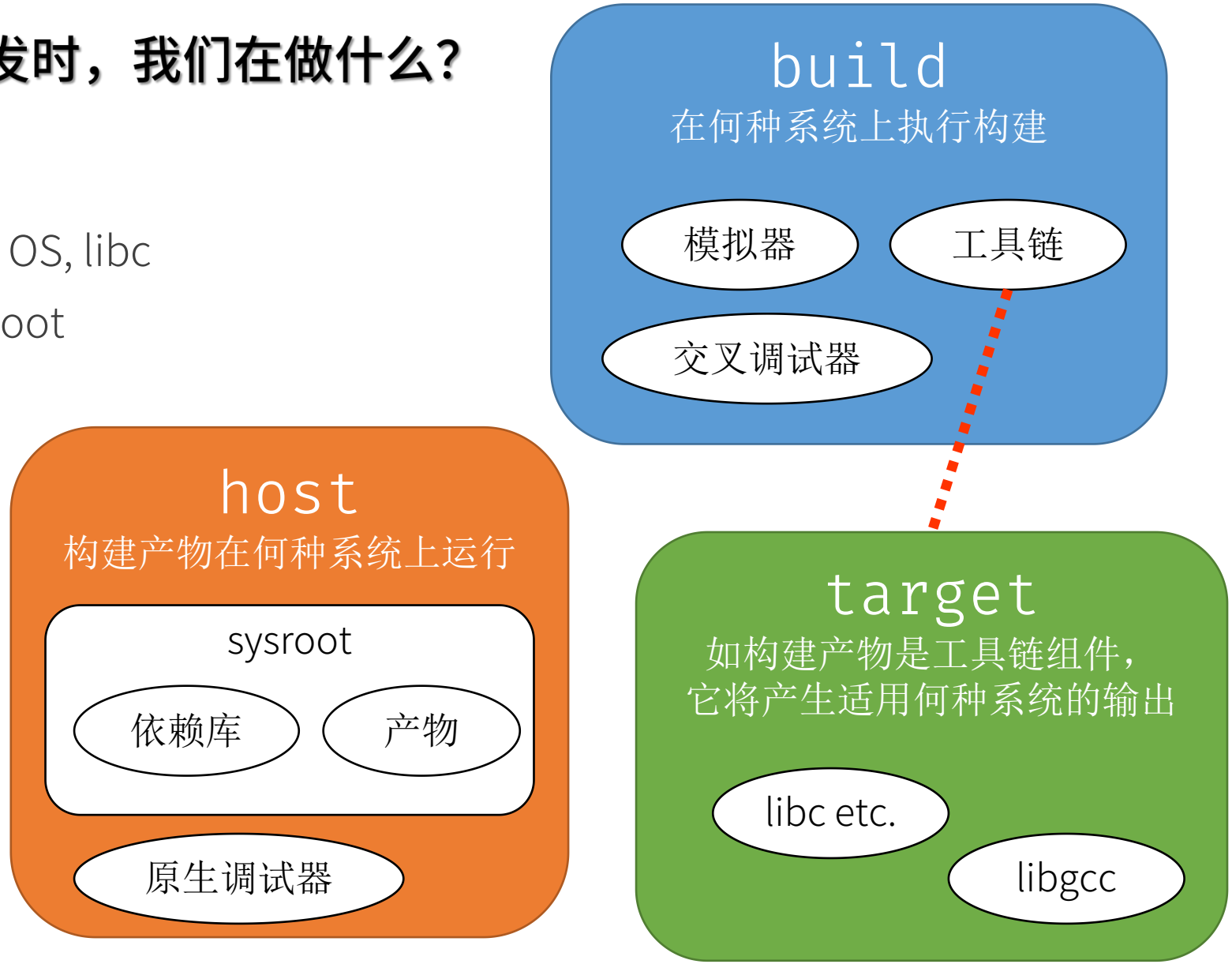
2023.12.15

我是谁?

- https://github.com/xen0n
- 以前是全栈研发，CRUD boy ~~包管理器的很多业务也会是 CRUD~~
- 是 Gentoo dev
  - https://wiki.gentoo.org/wiki/User:Xen0n
  - 从 Portage 借鉴了不少设计思想

当我们为一个环境做应用开发时，我们在做什么？

- 确定目标环境的 host tuple: ISA, OS, libc
- 寻找、部署一个目标环境的 sysroot
- 设定正确的交叉编译参数
  - host tuple
  - prefix = sysroot
  - CFLAGS/CXXFLAGS etc.
- 如需模拟：目标 CPU 型号的相应配置
  - 如 qemu linux-user 的 QEMU_CPU 配置



这里涉及了哪些问题？

- 如何定义**目标环境**？
- 如何**构建**适用的工具链、模拟器等组件？
  - 如果以**二进制**形式分发，如何实现跨发行版、某一发行版不同版本的**兼容性**？
- 如何**定义、分发**这些组件？
  - 对一个包管理器而言，它的“**软件源**”形态为何？
- Putting it together -- 如何为具体项目**适用**相应的编译参数，但**减少用户感知**？
- 如何分发、管理 Ruyi **包管理器本身**？

设计灵感的来源（他山之石）

- 软件源的形态上
  - Gentoo Portage: `category/package-name`、atom
  - crates.io: 软件源是 **Git** 储存库
  - crates.io, Go Modules: 采用 **SemVer** 规范
- 使用方式上
  - Python virtualenv
    - **分立的**虚拟环境，互不相关，每个环境提供 `bin/` 供放入 PATH，内含该环境可用的**命令 symlinks**
    - 需要激活或总之感知到 **venv root** 才能正常工作
  - rustup
    - 包管理器的分发方式：**单可执行文件**
    - 工具链命令的代理方式：链接到**同一可执行文件**；根据 **`argv[0]`** 进入不同入口点，调用业务逻辑
    - rustup 无 virtualenv 概念，先前设想完全以 rustup 方式提供工具链，但在广泛的场景不可行
      - 工具链的选择：每次构建时根据当前工作目录与全局配置匹配，或由命令行参数 `+foo` 直接指定
      - Rust 工具名称统一，不随 target tuple 而变化，项目目录结构统一，故可行
      - C/C++ 工具名称多随 target tuple 变化；项目结构、构建系统无统一标准，工作目录不定

RuyiSDK 意图提供什么？

- **Ruyi 包管理器** —— 本机架构的可执行文件；不需区分 targets
- IDE —— 同上
- 工具链 —— 本机架构的可执行文件；需要区分 targets
- 调试工具 —— 同上
- 模拟器 —— 同上
- 运行环境（sysroot / rootfs） —— 本地数据；需要区分 targets
- 文档 —— URL / 本地数据
- 代码（库、示例 etc.） —— 本地数据

目标环境定义：profiles

- 在 Ruyi 软件源中定义
- 针对 RISC-V 的架构相关适配
  - CFLAGS: `-march`, `-mabi`, `-mcpu`
  - 根据工具链 flavor，可选映射 `-mcpu` 取值

```
{
  "arch": "riscv64",
  "generic_opts": {
      "march": "rv64gc",
      "mabi": "lp64d",
      "mcpu": ""
  },
  "profiles": [
    {
      "name": "sipeed-lpi4a",
      "need_flavor": ["xthead"],
      "mcpu": "thead-c910"
    },
    {
      "name": "milkv-duo",
      "mcpu": "thead-c906"
    }
  ],
  "flavor_specific_mcpus": {
    "xthead": {
      "thead-c906": "c906",
      "thead-c910": "c910"
    }
  },
> "emulator_presets": {...
  }
}
```

目标环境定义：profiles

- 在 Ruyi 软件源中定义
- 针对 RISC-V 的架构相关适配
  - CFLAGS: `-march`, `-mabi`, `-mcpu`
  - 根据工具链 flavor，可选映射 `-mcpu` 取值
  - 同理，支持为模拟器定制配置

```
25    "emulator_presets": {
26      "generic": {
27        "qemu-linux-user": {
28          "env": {
29            "QEMU_CPU": "rv64"
30          }
31        }
32      },
33      "thead-c906": {
34        "qemu-linux-user": {
35          "env": {
36            "QEMU_CPU": "thead-c906"
37          }
38        }
39      },
40      "thead-c910": {
41        "qemu-linux-user": {
42          "need_flavor": ["xthead"],
43          "env": {
44            "QEMU_CPU": "c910v"
45          }
46        }
47      }
48    }
49  }
```

如何构建广泛兼容的工具链、模拟器等组件？

- **可复现**
  - 容器化构建环境
- **兼容性**
  - 需求：Ubuntu 22.04 LTS & openEuler 23.09
  - 选择略微放宽的兼容基线：Ubuntu 20.04
  - 对容易漂移或不一定广泛存在的可选依赖：
    - 很有必要带上的，静态链接 e.g. LLVM: libz.so.1, libtinfo.so.5
    - 相对较不重要的，暂时关掉 e.g. GDB: libpython3.x; LLVM: libedit, libxml
- **可维护性**
  - GNU 工具链
    - 涉及 binutils、gcc、glibc、linux 四种项目，需精确到特定来源（上游、PLCT、T-Head）的特定版本
    - 使用 crosstool-ng 管理配置、自动化。期间发现并修复了 crosstool-ng 上游的 multilib bug 一枚
  - LLVM、QEMU etc.
    - crosstool-ng LLVM support TODO, 后期考虑为其实现
    - 目前是手搓的构建脚本，参考了发行版打包的通常做派
    - 后续可能参考 Arch PKGBUILD、Gentoo ebuild 等现有 recipe 格式，方便、统一地描述源码构建过程

Ruyi 软件源

- 规范文档：https://github.com/ruyisdk/ruyi/blob/0.2.0/docs/repo-structure.md
- 官方软件源：https://mirror.iscas.ac.cn/git/ruyisdk/packages-index
- 包的种类
  - binary
    - 可匹配本机架构的包
  - source
    - 可原地解压的包
  - toolchain
    - 为某个 target tuple 生成代码的工具链包，带有内含组件的种类、版本信息
    - 支持指定 flavors: xthead
  - emulator
    - 支持模拟某个/某些架构的包，目前支持 QEMU linux-user 模拟器的用法
    - 提供 Linux `binfmt_misc` 集成
    - 支持指定 flavors: xthead again

当你创建虚拟环境时，`ruyi` 都在干什么？

```
$ ruyi venv -t llvm-upstream --sysroot-from gnu-plct -e qemu-user-riscv-upstream generic /tmp/foo
```

- 检索 **profile**，记录其配置、**flavor**
- 匹配工具链包
  - 检查 **flavor** 是否满足
  - 检查组件内容：是 LLVM 工具链
  - 目前 LLVM 工具链不含 sysroot 与 libgcc，必须搭配 GCC 工具链包使用
    - 匹配 sysroot 包
    - 检查：是 GCC 工具链，内含 sysroot
    - 后续可能也要检查 **flavor** 是否满足
- 匹配模拟器包
  - 检查 **flavor** 是否满足
- 复制 sysroot 入 venv root
- 创建命令的符号链接
  - 遍历工具链包 `bin/` 下所有可执行文件
  - 对 LLVM：创建兼容 binutils 的命令别名
  - 对 Clang：创建兼容 gcc 的命令别名
  - 如指定了模拟器包：`ruyi-qemu` wrapper
- 将预先解析的 **profile** 配置写入 venv
- 渲染其余的支持文件
  - `ruyi-activate` 脚本
  - `binfmt_misc` 配置样例
  - CMake toolchain file & Meson cross file
  - README

当你使用虚拟环境时，ruyi 都在干什么？

- 找到自己可执行文件的路径 `self_exe`
- 检查 `argv[0]`
  - `ruyi`：Ruyi 包管理器本体模式
  - `ruyi-qemu`：QEMU 转发模式
  - 像 C/C++ 编译器名字：编译器转发模式
  - 其他：单纯转发模式
- 转发模式
  - 显式指定：环境中存在 RUYI_VENV 变量
  - 隐式指定：如 `self_exe` 所在目录的上级目录含 `ruyi-venv.toml` 文件
  - 必须要感知到虚拟环境，否则报错退出
- QEMU 转发模式
  - 按照 venv 配置，灌入环境变量
  - 如本 venv 内含 sysroot：灌入 QEMU_LD_PREFIX 变量
- 编译器转发模式
  - 按照 venv 配置，向 `argv` 前部灌入 CFLAGS
  - 如编译器是 Clang：灌入 `--target`、`--gcc-install-dir` 选项
  - 如本 venv 内含 sysroot：灌入 `--sysroot` 选项
  - 确保本 venv 的 `bin/` 在 PATH 中
    - 如不在，灌入 PATH 前部

`ruyi` 本体的构建与分发

- 由于需要**降低贡献门槛**，要求使用 Python/Shell 等开发，**无法使用常见编译型语言**
  - 目前是基于 Python 3.11 的常规 Python 项目，以 Poetry 管理依赖
- 如何做出单可执行文件？
  - https://github.com/Nuitka/Nuitka comes to rescue!
  - 简单说来：在容器里 `poetry install && python -m nuitka ...`
  - 二进制依赖：已由 Nuitka 帮忙处理；兼容性：取决于构建容器的发行版版本；启动性能：很差
- 如何分发？
  - 目前由用户手工从分发渠道下载，确保文件名为 `ruyi` 即可
    - GitHub Releases
    - ISCAS 镜像源：https://mirror.iscas.ac.cn/ruyisdk/ruyi/releases
  - 后续计划提供类似 oh-my-zsh、Homebrew、rustup 等明星项目的安装脚本，支持 `curl | sh` 形式的“一键安装”

欢迎来上游 & Question time!

https://github.com/ruyisdk/ruyi

Thanks for listening!