Linux组件化初步实验的结果和想法

(所属题目：扩展arceos等，形成unikernel形态，支持Linux系统调用和应用程序)

石磊 . 乾云启创

2023.2.12

提纲

- 目标与思路

  Linux kernel分解建立组件库，按照方案能够组合成各种特性OS。
- 实验内容和工作项

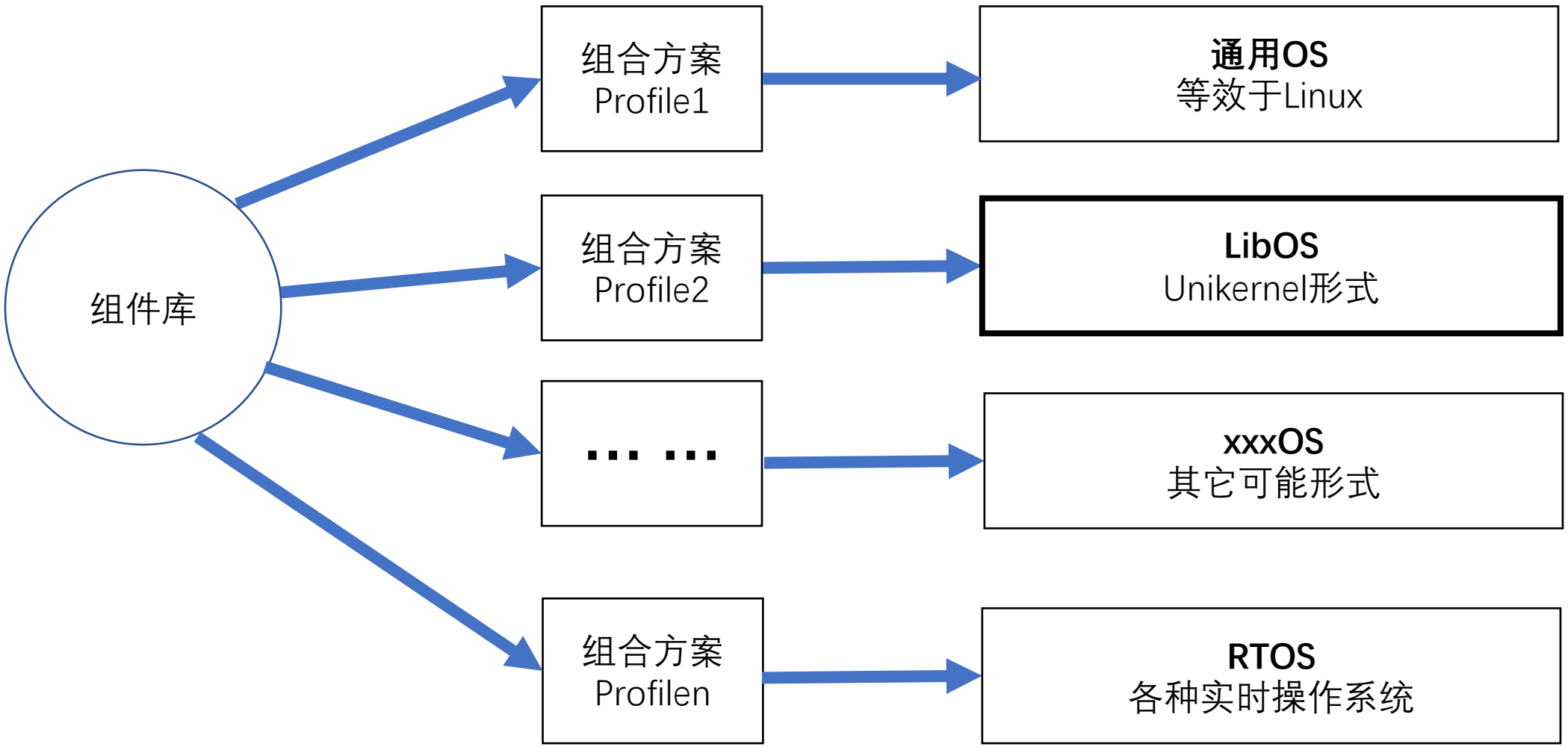
  具体实验项与完成进度。
- 总结和想法

  阶段总结、里程碑划分和下步工作的设想。
- 环境

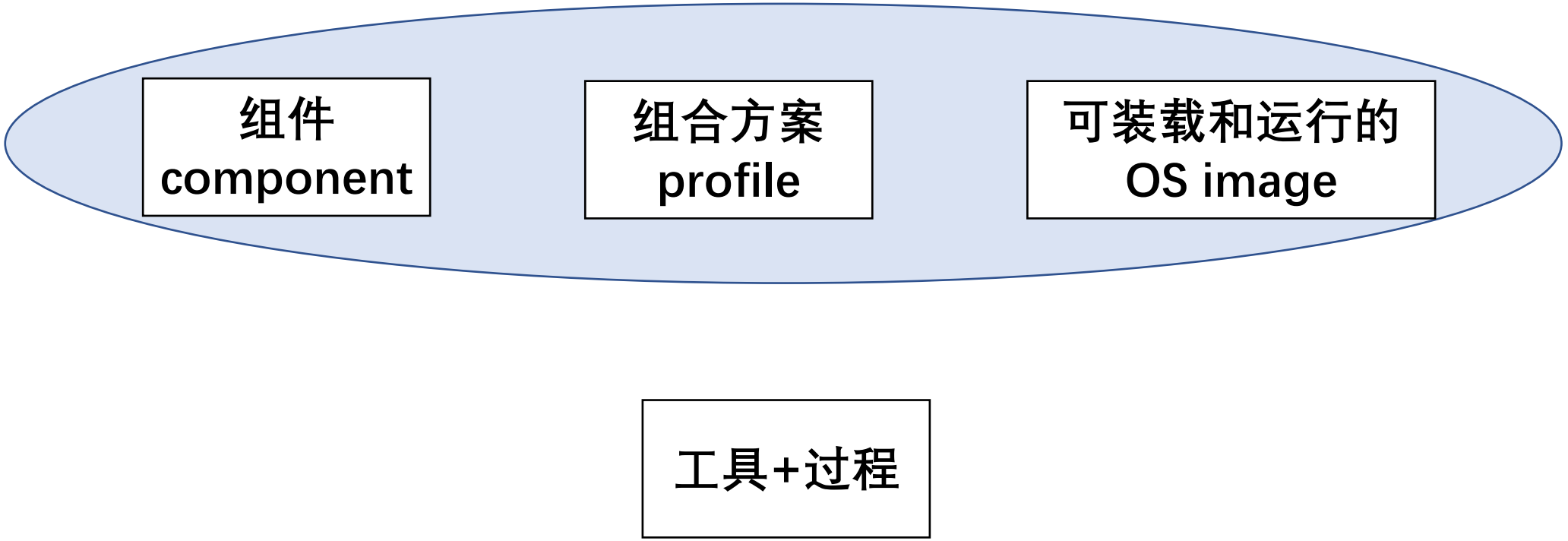
  实验环境描述，代码目录设置

限定：目标平台riscv64，最终的实现以rust语言为主

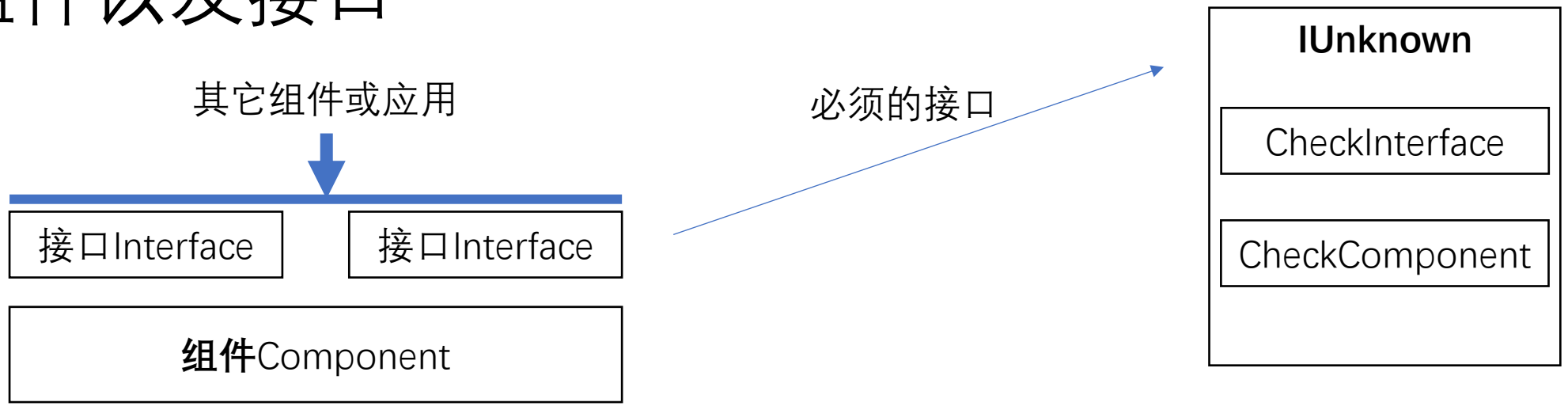
课题目标：通过组装组件来构造各种特性OS



四个中心要素



组件以及接口



参考：MS COM
格式：Linux kernel module (*.ko) ELF格式

主要Sections:
Symtab (undef) 引用的外部符号
Export symbols 对外公开的符号
Meta sections for interfaces(**provide/require**)
Meta section for component itself组件自身信息

Provide标记组件对外提供的接口ID
Require标记组件依赖的接口ID

接口：基于OMG IDL定义

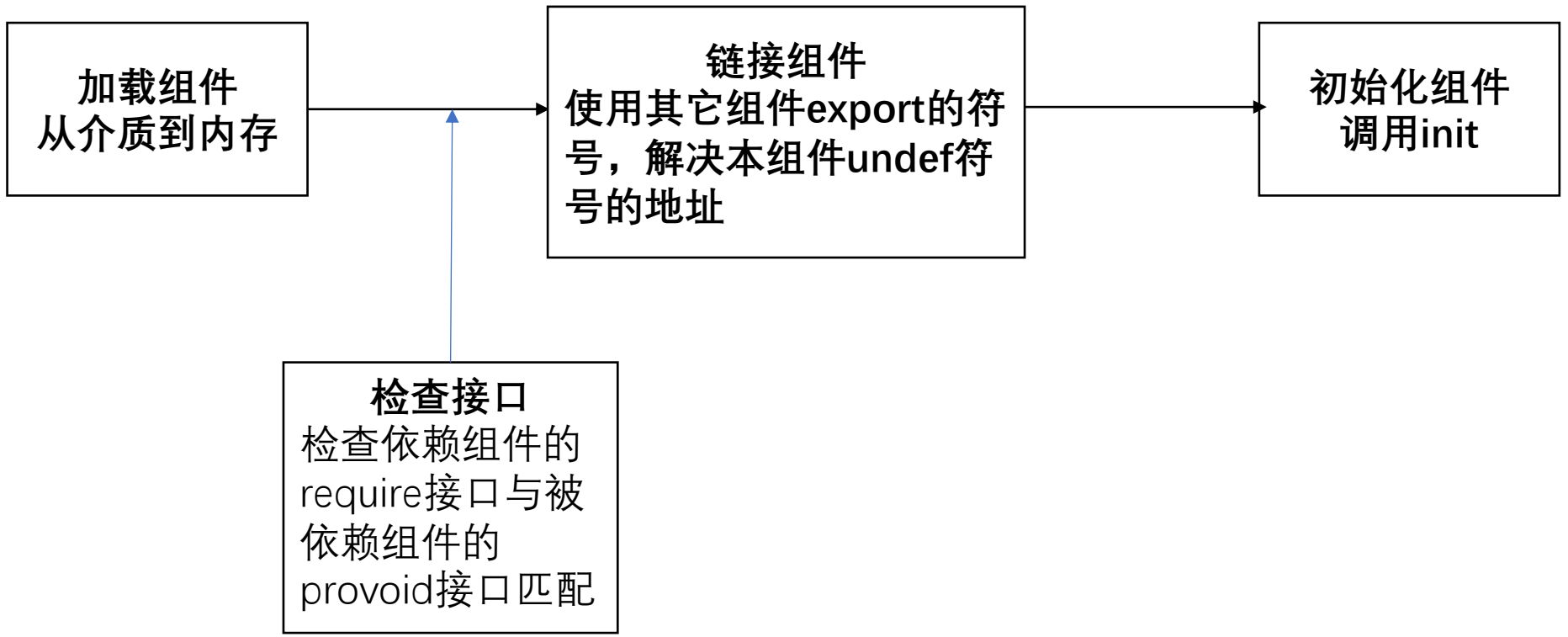
```
interface IUnknown
{
  boolean CheckInterface(in string itf_id);
  boolean CheckComponent(in string com_id);
}
```

组件和接口在编译时和链接时的关系

接口体现在编译时刻和组件动态加载后的链接时刻的检查。

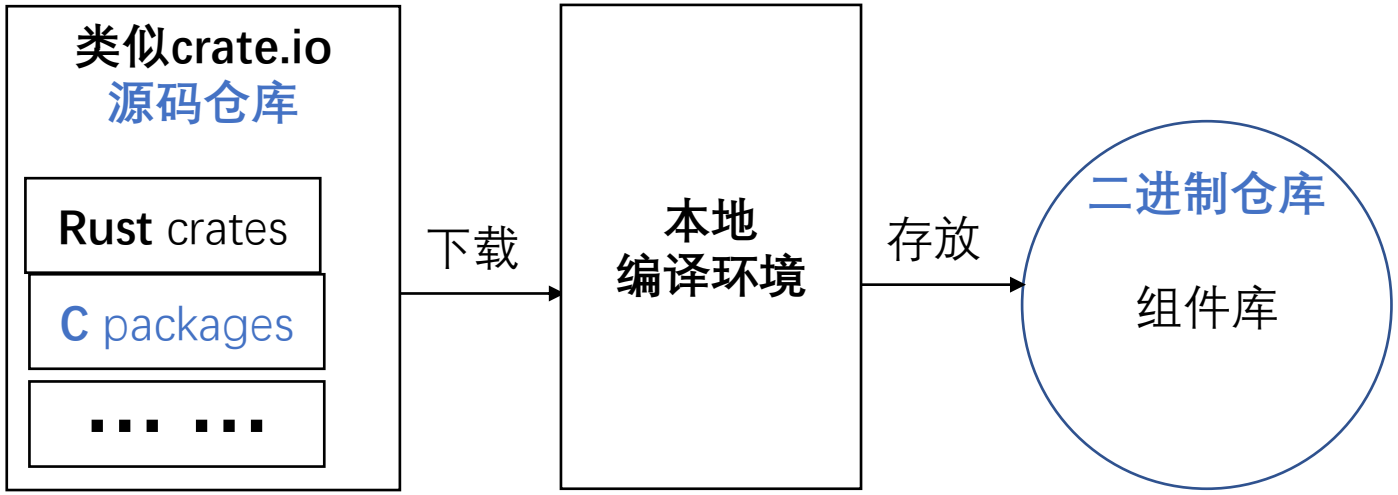
1. 组件编译时，就像rust中实现实现Trait那样，由编译器检查组件是否完全实现了接口。

2.组件链接时

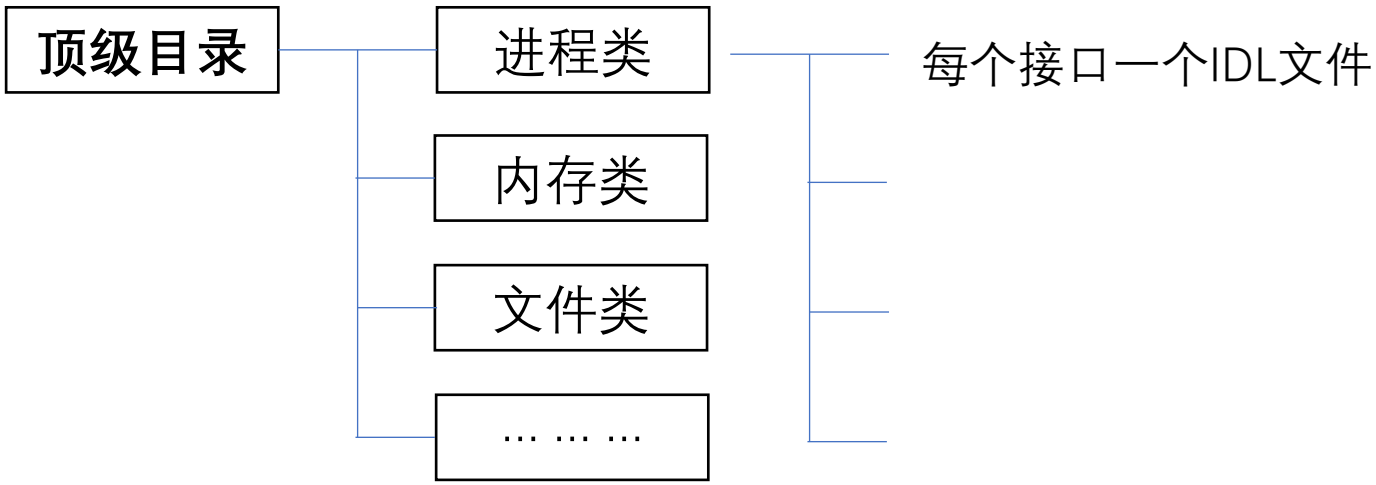


组件库以及接口库

组件库：
类似Cargo管理的源码库，或直接用cargo，但是不限于管理rust crate；
编译后形成二进制的组件库。
最终用于构建OS的组件库是二进制。



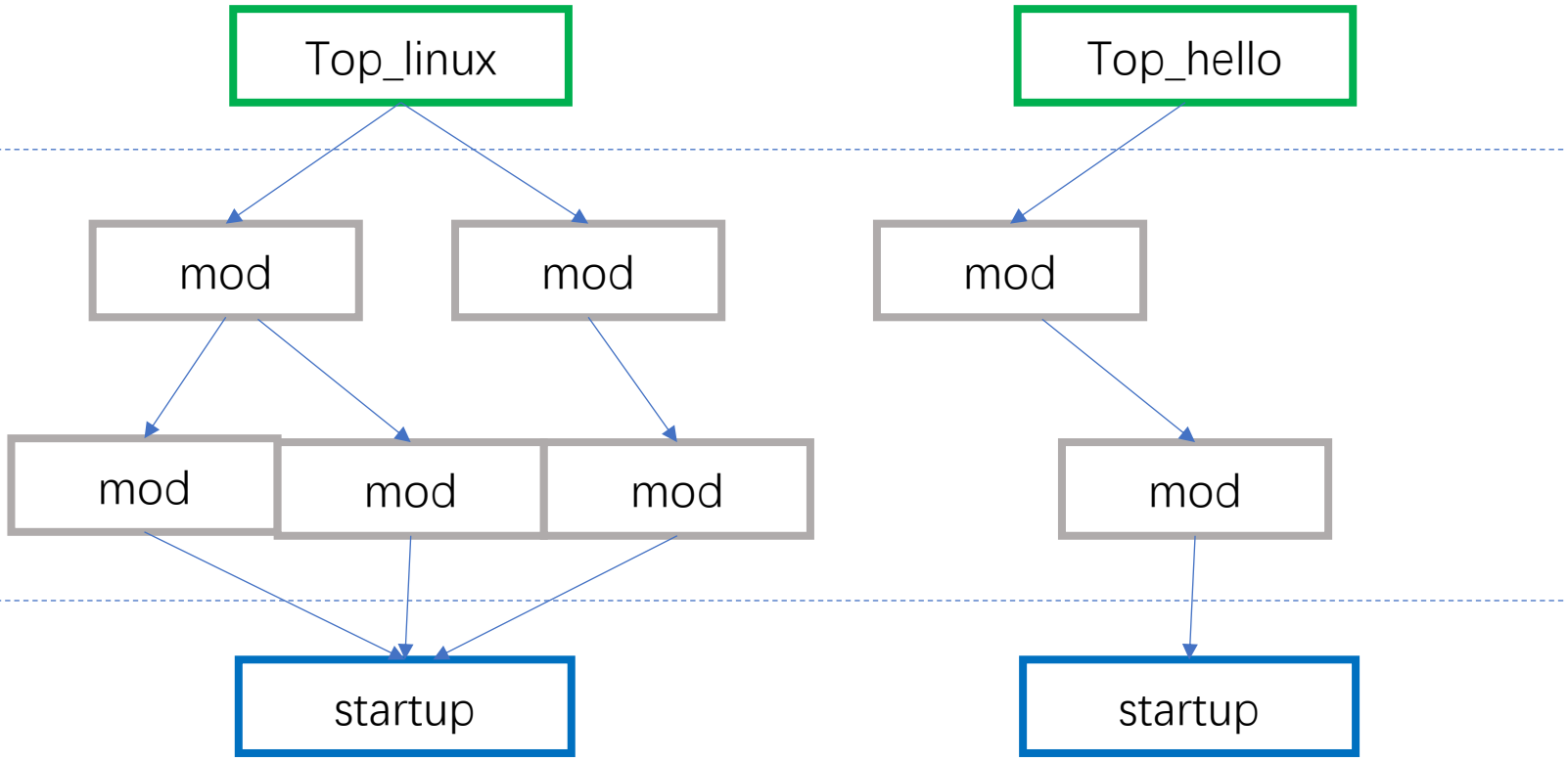
接口库：分级目录+IDL文件形式



组合方案Profile

站在组合方案的角度：三层

描述：(1)**包含那些组件** (2)**组件间的依赖关系**
展示形式：Json及可视化(面向人） 特定格式二进制(面向设备)



Profile_linux

Profile_hello

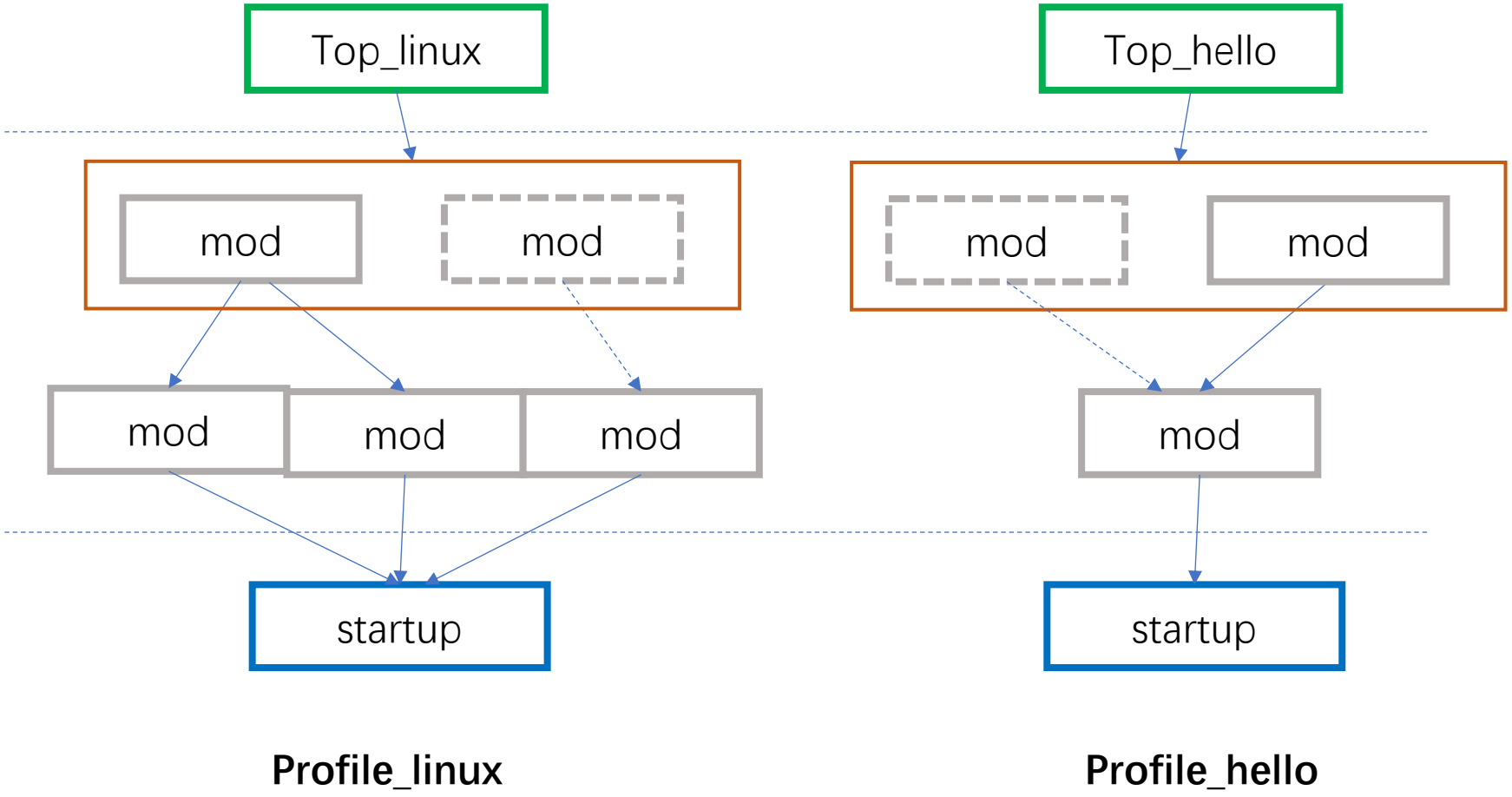
第三层 Top_XXX
代表一种特性的OS
对应一种方案

第二层 各种功能组件
组件之间通过接口交互

第一层 startup
各种方案的共有组件
第一个启动的组件
负责加载其它组件(根据profile)

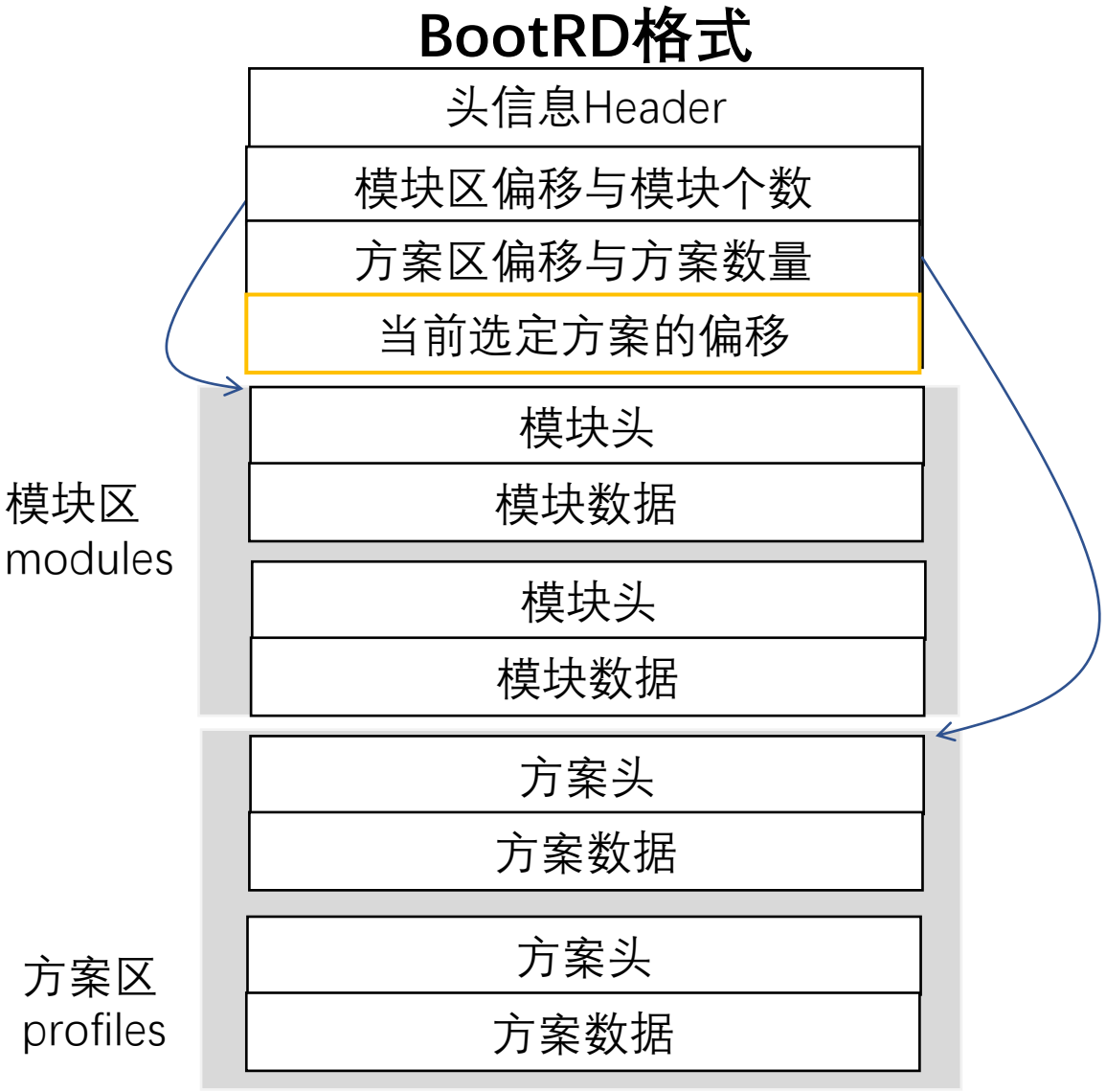
组合方案Profile – 互备组件

互备组件：组件库中，实现同一接口的多个并存组件
(1) 每个profile中，必须明确指定选择哪一个
(2) 不同profile中，可以有不同的选择



互备组件差异
(1) 不同的版本：生产/试运行
(2) 不同的语言：Rust/C
(3) 不同的算法：来源/策略

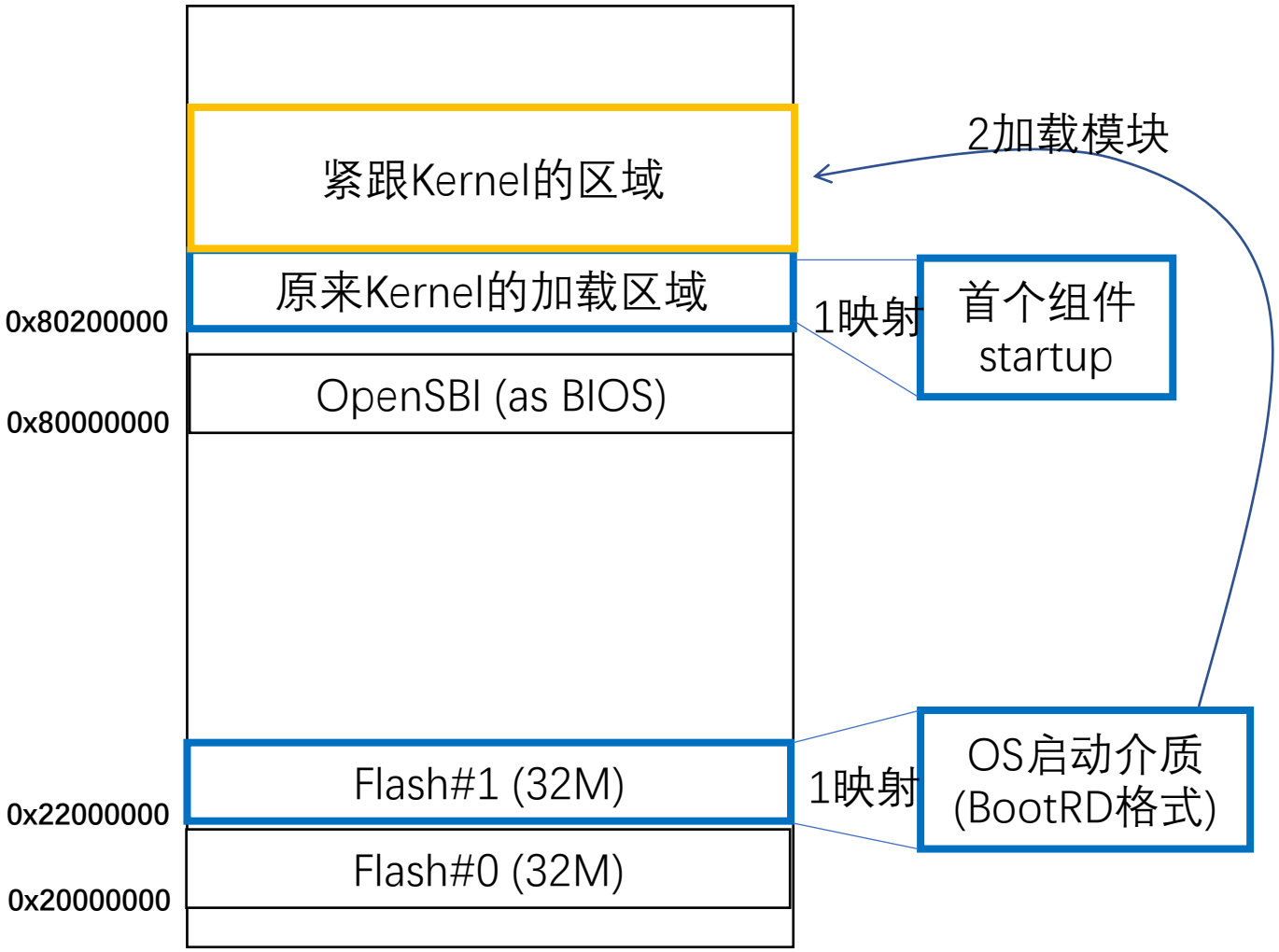
OS启动介质BootRD – 构成



模块section
存放组件库中所有的模块，头描述模块的长度等信息，数据即模块的二进制内容。

方案section
头描述方案profile的长度等信息，数据是一个数组，按照依赖顺序存放涉及模块的偏移位置。

OS启动介质BootRD – 启动流程中的位置

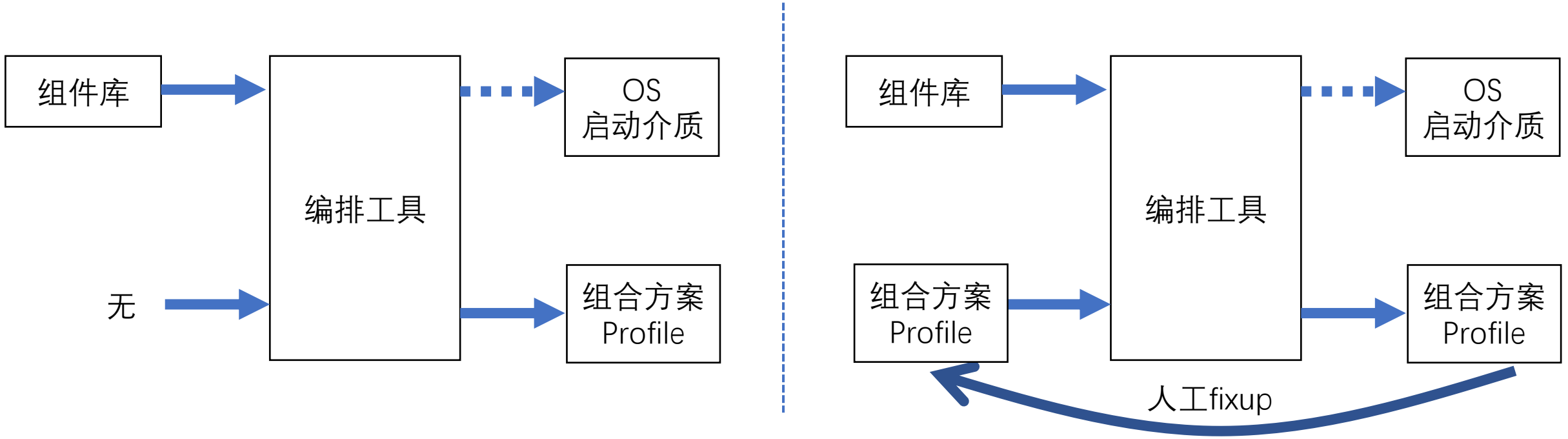


物理地址空间(qemu-版本小于7.2)

步骤

1. 通过qemu参数-pflash/-kernel/-bios加载各个部分到物理地址空间(左图)
2. 首组件startup启动后，读启动介质的头信息，按照**当前生效**的profile，顺序加载模块到随后的区域。然后按照顺序执行它们的init初始化。

工具-编排工具



Profile不存在

编排工具搜索组件库中所有top_xxx.ko，
自动对应生成profiles。

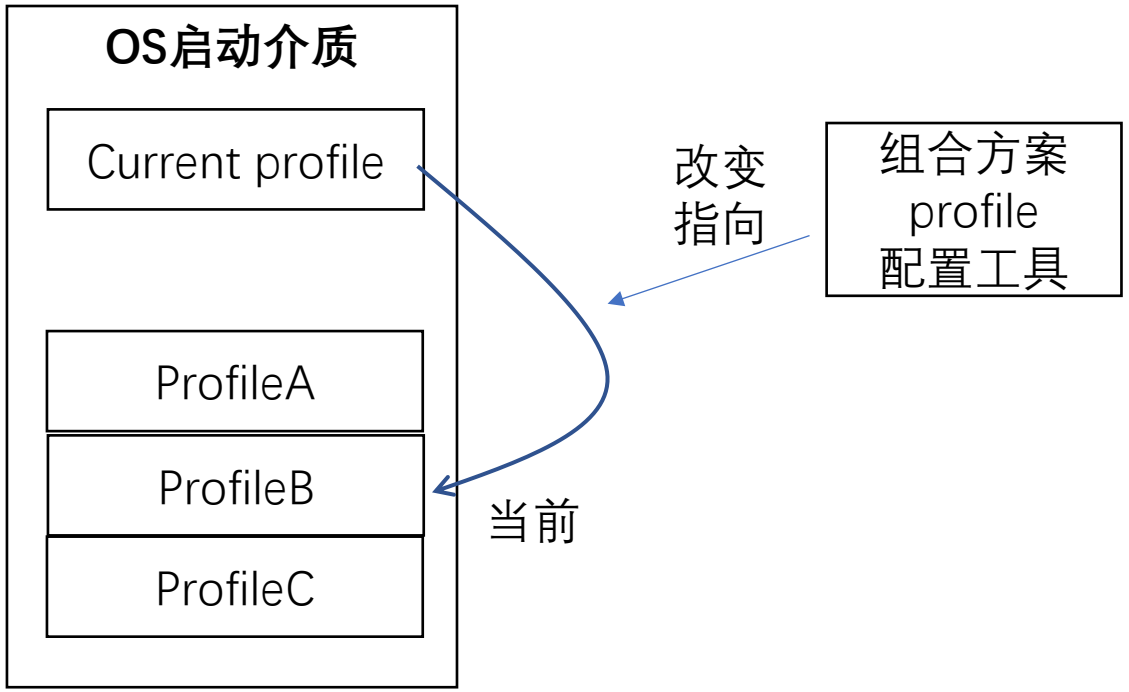
Profile已存在

编排工具检查校验profiles，
确认依赖的有效性。

需要人工干预的情况

发现有实现同样接口的互备组件存在，
人工在Profile中作标记后，再次运行编排工具验证。

工具-方案Profile配置



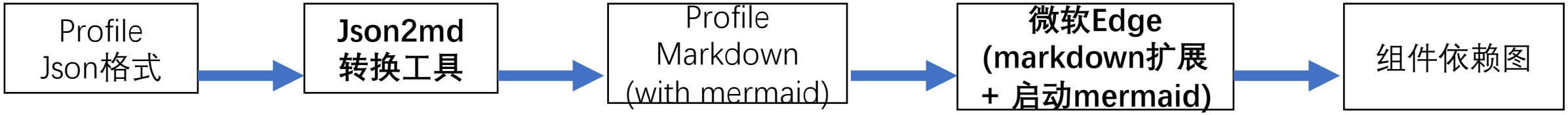
配置工具

目前功能：
在启动介质中打包的多个Profile中，选择当前的，重启qemu生效。

将来功能：

1. 在profile中增加/修改参数
2. 修改组件的依赖链

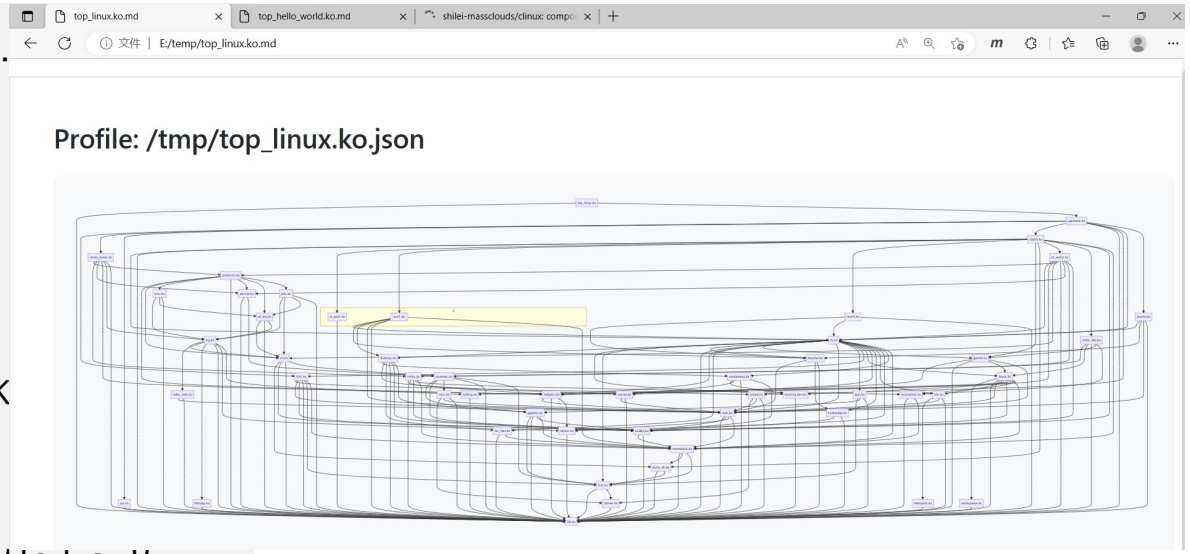
工具-Json2md转换实现Profile可视化




```
{
  "dependencies":   {
    "lib.ko": [],
    "rbtree.ko": [
      "lib.ko"
    ],
    … … …
    "top_linux.ko": [
      "lib.ko",
      "userboot.ko"
    ]
  },
}
```

```
### Profile: /tmp/top_linux.ko.
```mermaid
rbtree.ko --> lib.ko
mm.ko --> lib.ko & rbtree.ko

… … …
procfs.ko --> fs.ko & dcache.k
slab.ko & lib.ko
userboot.ko --> fork.ko &
virtio_mmio.ko & sys.ko & rootfs.ko &
lib.ko & sched.ko & fs.ko & procfs.ko
top_linux.ko --> lib.ko & userboot.ko
```
```



实验项即本阶段的工作内容与进度

1. 组件
   1.1 基于Linux5.9源码，分解为一系列模块，作为组件化实验的基础。(已完成)
   1.2 实现C组件与Rust组件之间的互操作性，作为互备组件进行对比。(进行中)
   1.3 引入接口的概念，体现为组件的meta数据；OMG IDL 定义接口，支持转Rust Trait和C的函数集合(未开始)。
   1.4 组件依赖从符号依赖改为对接口的依赖；组件编译时检查接口实现的完备性，加载链接时基于接口确定组件之间的依赖。(未开始)
2. 组合方案
   2.1 互备组件的支持。(已完成)
   2.2 类Linux profile，从OS启动到首个用户应用init完成。(已完成)
   2.3 HelloWorld profile，最简内核态应用能够启动。(已完成)
   2.4 组件测试profile：组件UnitTests的框架。(进行中)

实验项

3. OS启动介质
    3.1 qemu使用OS启动介质，正常识别和启动。(已完成)
4. 工具与过程
    4.1 编排工具：创建和管理profile与OS介质的核心工具。(已完成)
    4.2 配置工具：在OS介质备选profile中选择当前profile。(已完成)
    4.3 可视化工具：图形化显示profile。(已完成)
    4.4 rootfs生成工具：生成rootfs，包含首个应用进程init。(已完成)

总结和想法 – 简单总结

第1阶段尚未完成，就目前的情况做一下简单总结。说一下个人的看法：

1. 组件的模型应该是课题的核心点，包括组件自身构成，组件之间依赖关系的处理等，其它工作都要基于这个模型展开。但是目前认识较浅，仍需要一个在实验中加深认识的过程。希望第2阶段结束时能够把这个明确并固定下来。
2. 应该是需要单独维护一个接口的规格库，无论是像前面所说的目录+IDL的形式，还是采用其它方式。这个也是下一步让更多人参与进来，并分工协作的一个重要基础。
3. 需要深入RustForLinux，在理解的基础上，把它的实现引入进来并加以改造。

总结和想法 – 里程碑

通过近两周的工作， 已经可以大致的划分出里程碑。

第一阶段(当前阶段)

目标
基于对Linux的组件化分解实验，构造一个初步的框架，准备基本的工具。经过该阶段，对课题的总体目标以及可能面临的困难有一个初步的认识，制定大致的时间表。

核心任务
(1)确定组件的基本形式
(2)Rust和C组件的互操作性

预计完成时间：2月20日

第二阶段

目标
实现UniKernel方案，基于arceos等现有组件，逐步完成Rust组件对C组件的替换。支持该形式OS原型，对Linux syscall的支持。

可能的任务
可能需要2到3轮针对框架的大的调整，涉及组件与接口，组件与组件之间的关系、协调等。过程中，需要照应Rust的特点，还要参考RustForLinux的实现方法。
预计周期：两个月

第三阶段

目标
实现类Linux的OS方案，能够在用户态执行一些常见的应用。有相对完善的文档和配套工具。

可能的任务
待梳理

预计周期
待第二阶段总结时，应该能够估计出来。

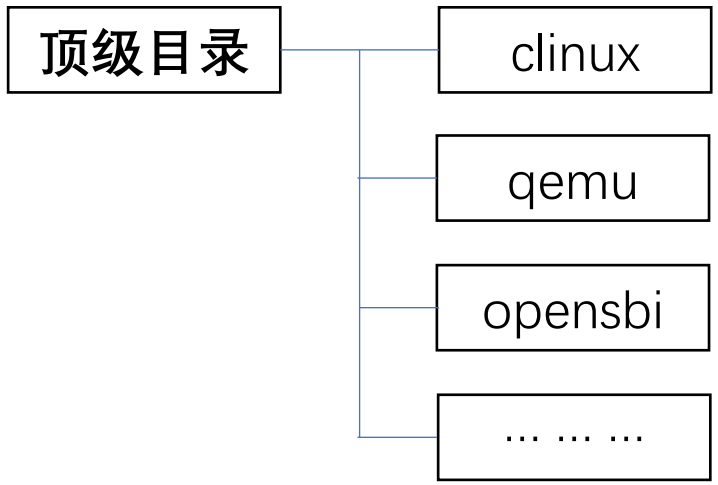
总结和想法 – 对下阶段粗略的想法

1. 组件是二进制还是源码的形态，目前倾向于采用前者，相关工作也是基于这个设定展开的。下步工作中继续验证可行性和识别其中的问题。
2. 目前采用的是动态加载组件的策略。将来可以实现静态的策略，即基于组件库可以直接编译生成OS image。这两种策略通过配置进行切换。
3. 打破循环依赖的方案。要求组件库中的各种组件是不存在循环依赖的。目前采取的方案是，如果出现循环依赖，就把部分弱引用作为回调Hook前置到startup中。这个可能导致startup不断膨胀，下步可能要考虑分散一下；或者研究是否存在更好的打破循环依赖的方法。

实验环境

WSL2
Unbuntu 18.04.01
riscv64-linux-gnu 8.4.0 (Ubuntu 8.4.0-1ubuntu1~18.04)
Qemu (版本小于7.2) + OpenSBI

实验代码的位置：
https://github.com/shilei-massclouds/clinux



谢谢！