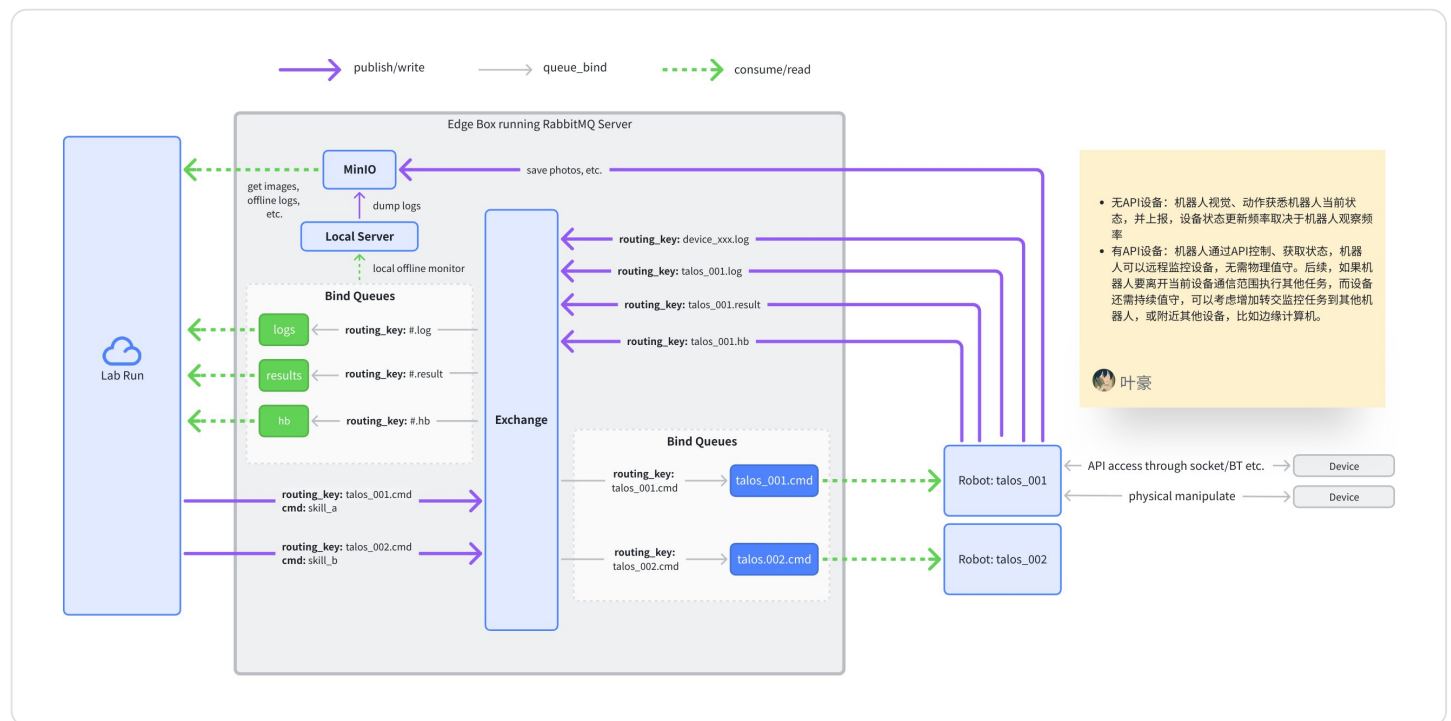


v0.3 Skill-level Runtime Service

一、通讯方案

1.1 定义



如上图所示，所有消息通过一个总的Exchange进行交换。采用Exchange模式的方便之处在于消费端比较灵活，可以有任意多个消费端，本地可以有一个Local Server做一些事情，后续LabRun到云端了也可以同步消费所有消息。

其中：

LabRun

- publish：下发机器人Skill命令
- consume/read：
 - 绑定所有#.log，#.result，#.hb，分别用于接受机器人/设备的日志，机器人技能执行结果，机器人心跳。
 - 从minio根据uri获取机器人保存的照片或其他对象。

Robot

- publish/write：
 - 设备状态更新日志、自身运行日志、技能执行结果、自身心跳。

- 拍摄的照片或其他对象存储到minio。
- consume：绑定针对自身个体的<robot_id>.cmd，接受命令下发。

此外，还与技能执行过程中所需要交互的设备通讯，目前分为两种形式：

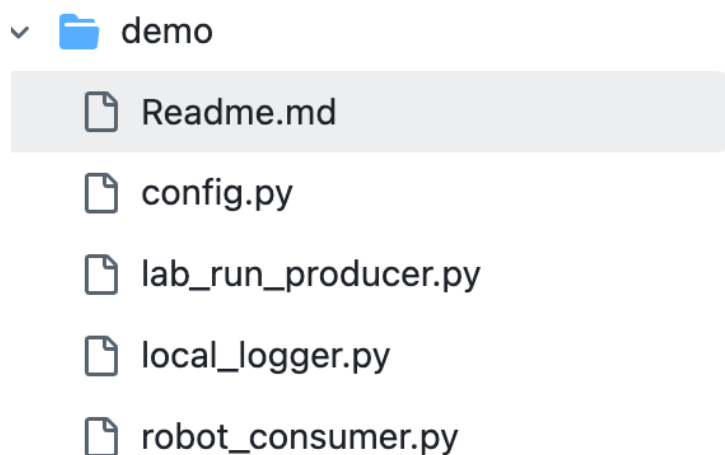
- physically：物理接触，比如按钮操作、触屏控制。设备状态的获取，通过触觉、视觉。其中：
 - 一些复杂的界面，机器人本体无法进行解析或设计上不放在机器人本体解析的，比如过柱完成的界面内容解析，需要在技能层面在这里进行断开，机器人只负责拍照，上传，然后接收上层的下发的下一步指令行动。
 - 一些简单的界面，比如旋蒸机UI上的转速、温度等的读取，虽然目前还未实现，但后续可以考虑实现后，把模型直接部署到机器人里，属于机器人对这个设备完整操作技能的一部分。因此，这些设备的当前状态，由机器人观察解析后，进行上报，告知LabRun。

Local Server

- 目前labrun如果部署在同一个局域网下，其实就不需要这个。
- 后面如果labrun部署在云端，那么考虑到云端与lab的通讯可靠性和频率，可以在实验室边缘计算机上部署一个local的offline的service，根据需要可以做一些本地的简单的事件记录、管理等。

1.2 代码示例

https://github.com/c12-ai/mars_service/blob/main/demo/Readme.md



192.168.12.212这台机器是机器人研发这边用的一台用于仿真的PC，起了一个rabbitmq的docker实例，这段时间每天上班时间都会开机的，可以暂时用来做测试研发。

1.3 演示示例

如下视频所示，从上到下5个窗口分别是：

- local_logger.py：运行在EdgePC上，LabRun侧也可以有这样一个，用于监听日志，向上给用户提

- lab_run_producer.py service: 启动一个mock的labRun的service，里面主要是附带了一个简单的对所有robot的心跳监听，目前是robot会每2秒发送一个心跳，而这里的监听如果发现超过5秒没有心跳，认为robot下线。上线/下线都会发送一条日志。
- robot_consumer.py talos.001: 启动mock的001号机器人worker。
- robot_consumer.py talos.002: 启动mock的002号机器人worker。
- lab_run_producer.py send_cmd <robot_id> <cmd_str>: mock的命令下发。其中，视频里演示时：
 - 命令只是字符串，下面我们会具体定义目前实际支持的命令内容格式。
 - robot接收到命令后假装执行，sleep3秒后完成，这期间都会发送日志到<robot_id>.log，完成后发送结果到<robot_id>.result
 - local_logger.py会监听#.log和#.result，以及来自lab_run_producer.py的日志，并打印出来。

```

lumi@LUMR-MBP-7:~/workspace/robot/mars_service/demo
..service/demo (-zsh)
[LOG] [2026-01-12 15:58:50.257] [system.log] talos.001 online
(robot) + demo git:(main) #
(robot) + demo git:(main) python local_logger.py
[*] Logger waiting for logs/results. CTRL+C to exit
[LOG] [2026-01-12 15:58:53.103] [system.log] talos.002 online
[LOG] [2026-01-12 15:58:54.104] [system.log] talos.002 offline
(robot) + demo git:(main) python local_logger.py

..service/demo (-zsh)
(robot) + demo git:(main) # python lab_run_producer.py service
[*] Lab service running. CTRL+C to exit
(robot) + demo git:(main)
(robot) + demo git:(main)
(robot) + demo git:(main) python lab_run_producer.py service
[*] Lab service running. CTRL+C to exit
(robot) + demo git:(main)

..service/demo
t.join(timeout=2)
File "/Users/lumi/miniforge3/envs/robot/lib/python3.10/threading.py", line 1100, in join
  self._wait_for_tstate_lock(timeout=max(timeout, 0))
File "/Users/lumi/miniforge3/envs/robot/lib/python3.10/threading.py", line 1116, in _wait_for_tstate_lock
  if lock.acquire(block, timeout):
KeyboardInterrupt
(robot) + demo git:(main)
(robot) + demo git:(main)

..service/demo (-zsh)
File "/Users/lumi/workspace/robot/mars_service/demo/robot_consumer.py", line 121, in start_robot
  t.join(timeout=2)
File "/Users/lumi/miniforge3/envs/robot/lib/python3.10/threading.py", line 1100, in join
  self._wait_for_tstate_lock(timeout=max(timeout, 0))
File "/Users/lumi/miniforge3/envs/robot/lib/python3.10/threading.py", line 1116, in _wait_for_tstate_lock
  if lock.acquire(block, timeout):
KeyboardInterrupt
(robot) + demo git:(main) #
(robot) + demo git:(main)

..service/demo (-zsh)
(robot) + demo git:(main) # python lab_run_producer.py send_cmd talos.002 "laugh"
[SEND CMD] [talos.002.cmd] laugh
(robot) + demo git:(main) # python lab_run_producer.py send_cmd talos.001 "laugh"
[SEND CMD] [talos.001.cmd] laugh
(robot) + demo git:(main) # python lab_run_producer.py send_cmd talos.002 "laugh"
[SEND CMD] [talos.002.cmd] laugh
(robot) + demo git:(main) #
(robot) + demo git:(main)
(robot) + demo git:(main)
(robot) + demo git:(main)

```

二、Skill API定义

这里的技能清单我们唯物主义，根据目前机器人已经扎实实现了的动作流程，先做拆解，后续一些未实现的，后面再做加法。以及，需要对这版定义的一些流程做切分或其他修改的，到时候再修改，成本不会太高的。

2.1 数据定义

这里我先草拟一个初步的草稿。过程中需要拍照的时机如何指定，暂时没有在接口里设计，等后续再补。

备料架

每个样品柱支架、硅胶柱支架、试管架、桶的物理位置，在哪个备料架、通风橱的第几层、第几个位置，这些信息，都是物理层面上的信息，labrun可以存储这些信息，但在进行指令下发的时候，不需要进行这样物理意义上的描述。就好像地铁的每个门，机场厕所每个包间都有编号一样，LabRun对机器人下发技能指令时，涉及到这些位置的，直接给编号就行了。机器人自己需要知道这些编号对应的在这个实验室里的哪个位置，自己去完成这些物品的pick。

2.2 技能清单

- 以下各个技能的request里不包含robot_id，因为robot_id已经在publish到具体的routing_key时就被指定了。
- 关于设备状态以及技能指令如果存在冲突的：
 - 通常一些不严重的冲突，由于机器人是不对设备状态做全程严格负责的，有些状态并不感知，只管执行，比如：
 - 要求机器人把过柱机通风橱右边的桶拿到回收处，但其实那个桶还没被使用，但机器人不管，你让我拿就拿。上层调度为此负责。
 - 可能导致安全性的严重冲突的，机器人会在做一些可能有风险的动作前尽量做安全检查（现在很多能力都还不具备，需要后面慢慢加上），如果发现情况不对，会终止执行并上报异常，比如：
 - 要求机器人去取两个柱子并架设到过柱机外置机构，但柱子不在指定的位置，或者规格错误。或者，到了外置机构前，发现上面已经有柱子了，或者盖子不在最上面，无法完成架设的动作。
- 以下主要都先围绕happy flow来定义response，各种异常的response，后续再补上。
- 所有设备的state的流转，感觉需要专门进行完整的设计规划，以下技能清单的response里示例的关于一些设备state的update回传，可以视作一个暂时的比较草率的版本，是缺乏系统性设计的。
- 写着写着感觉request和response都不太简短，还有点小重的，不知道是不是

setup_cartridges

代码块

```
1  {
2      "task_id": "xxx", // 任务ID, labrun分配提供, 用于robot后续反馈log/result时回传,
    标记消息绑定在哪个任务上
3      "task_name": "setup_tubes_to_column_machine", // 任务名, enum, 由robot整理提供
    给labrun来选择调用。这是架设柱子到过柱机外置机构上, 任务包含取一个硅胶柱, 一个拌样柱, 以及
    去哪个目标通风橱去架设。硅胶柱没有特殊性, 只需要指定型号规格即可, 机器人自己看着拿一个 (目
    前只支持一个, 以后会多的), 拌样柱里面有实验样品, 所以是有特殊性的, 需要明确指定取的
    location_id
4      "params": {
5          "silica_cartridge_location_id": "bic_09B_l3_001", // 硅胶住的位置, enum。
    由于硅胶柱没有特殊性, 这个参数可以不提供, 就交给机器人自己看着随便拿一根同规格的就好了。虽
    然目前机器人还没这个能力, 目前只能从一个固定的位置拿, 但这由机器人来消化, labrun可以不管
6          "silica_cartridge_type": "sepaflash_40g", // 硅胶柱类型, 这个字段现在、以后
    应该都是需要的, 是必要的
7          "silica_cartridge_id": "sepaflash_40g_001", // 硅胶柱没有特殊性, id字段我
    们可以先省略
8          "sample_cartridge_location_id": "bic_09B_l3_001", // 表示bic这栋楼的9楼,
    B表示L型备料区右边那个架子, l3表示第三层, 001表示这一层上某一个特定位置的编号
9          "sample_cartridge_type": "ilok_40g", // 样品柱类型
10         "sample_cartridge_id": "ilok_40g_001", // 样品柱有特殊性, 需要唯一ID, 由
    labrun分配并提供
11         "work_station_id": "fh_ccs_001" // 要架设柱子的目标工作站点, enum, 目前就这
    一个值, 表示BIC这边的过主机通风橱
12     }
13 }
```

其中:

- **location_id**: 可能可以展开为 [customer_id]_[building_id]_[floor_id]_[rack_id]_[position_id]。每个location_id, 物理意义上可能是我们的L型货架, 可能是通风橱, 可能是普通桌面, 以后为了通用性、泛化性之类的考虑, 可能还会做一些更复杂的设计, 但现在其实想太多我觉得到时候未必是那样的, 结合时间关系, 暂时我们简单点, 现阶段这里location_id对应机器人接下来去哪里、怎么取, 由机器人内部维护一个很小的映射关系来进行转义。
- **sample_cartridge_type**: 目前我们用的丐版的柱子就这一种型号, 回头BIC那边豪华版柱子可以另外起一个type名。不过对机器人动作而言大概率是一样的。
- **sample_cartridge_id**: 目前我们用的丐版的柱子就这一种型号, 回头BIC那边豪华版柱子可以另外起一个type名。不过对机器人动作而言大概率是一样的。

- **silica_cartridge_id, sample_cartridge_id**: 可以是持久的，也可以是临时的一个id，由labrun产生，通过下发任务时告知robot，这样后续robot回传这些柱子的状态更新时，能有一个唯一的id。
- **work_station_id**: fh_ccs_001表示fume_hood_column_chromatography_system_001，目前我们就一个用于过柱的通风橱。
 - 目前，我们研发的用于过柱的通风橱，内部布局是有我们的固定模式的，因此只需要指定是哪个站点id就行了，物理上这个站点必须是符合我们要求布局的一个通风橱。
 - 后续，如果是其他型号的通风橱，也许体积、布局不同，可能会对通风橱布局产生一定影响，届时也许会有多套略有差异的动作流程，那么这里技能接口可能会需要多一个参数，但也可能让机器人自己消化，到时候再说。



Robot Motion

- 机器人会根据silica_cartridge_location_id和sample_cartridge_location_id所在位置，前往对应站点，完成这两个柱子的获取，然后根据work_station_id的位置，前往对应站点，完成两个柱子的架设，然后在原地回到idle姿态待命。



Response

代码块

```
1  {
2      "code": 200,
3      "msg": "success",
4      "task_id": "xxx",
5      "updates": [
6          {
7              "type": "robot",
8              "id": "talos_001",
9              "properties": {
10                 "location": "work_station_id",
11                 "state": "idle"
12             }
13         },
14         {
15             "type": "silica_cartridge", // 硅胶柱
16             "id": "sepaflash_40g_001",
17             "properties": {
18                 "location": "work_station_id",
19                 "state": "mounted"
20             }
21         }
22     ]
23 }
```

```

21     },
22     {
23         "type": "sample_cartridge", // 拌样柱
24         "id": "ilok_40g_001",
25         "properties": {
26             "location": "work_station_id",
27             "state": "mounted"
28         }
29     },
30     {
31         "type": "ccs_ext_module", // 过柱机外置机构
32         "id": "ccs_ext_module_001",
33         "properties": {
34             "state": "using" // 表示这个外置机构正在被使用
35         }
36     }
37 ]
38 }

```

setup_tube_rack

Request

代码块

```

1  {
2      "task_id": "xxx", // 同上
3      "task_name": "setup_tube_rack", // 架设试管架task，可用试管架在哪机器人时知道的，且试管架没有特殊性，因此机器人会自己去对应备料区拿一个干净的试管架的（目前只支持一个具体的位置，后面会支持全部，并且机器人自己观察哪个位置上有空的试管架，自己去拿）。但是，需要指定机器人拿了试管架后，去放到哪个通风橱的过柱机上。尽管目前我们也只有唯一一个，但以后很可能会扩展的，而让机器人去哪个通风橱过柱，这应当是一个labrun进行调度的事情，所以由labrun来指定
4      "params": {
5          "tube_rack_location_id": "bic_09C_l3_002", // 试管架无特殊性，可以不指定，机器人自己看着拿一个（目前只能拿一个位置，后续再扩展功能
6          "work_station_id": "fh_ccs_001", // 要架设的目标工作站点，enum，目前bic这边就一个过柱通风橱，因此目前只有这一个值，以后可能会更多，因此需要指定一下站点
7          "end_state": "wait_for_screen_manipulation" // 结束这个动作后机器人的终末状态，enum，由robot这边整理提供enum取值范围。通常机器人做完所有task都可以回到idle，但这里这个值表示让机器人停留在等待操作屏幕的一个状态，这样可以更方便地衔接后续操作屏幕，开始过柱的任务
8      }
9  }

```


其中：

- **end_state:** 是所有技能都有的一个入参，可以不提供，缺省值是idle。这里，由于设置完试管架后，我们通常衔接屏幕操作，开始过柱，因此这里不需要让机器人回到idle姿态，否则会有动作冗余，不够连续。



Robot Motion

- 机器人会根据location_id所在位置，前往对应站点，双手抓取试管架，然后根据work_station_id的位置，前往对应站点，完成试管架的架设。
- 由于指定了end_state是wait_for_screen_manipulation，因此机器人不会回到idle姿态，而是停留在一个能更快速开始后续屏幕操作的姿态。



Response

代码块

```
1  {
2      "code": 200,
3      "task_id": "xxx",
4      "msg": "success",
5      "updates": [
6          {
7              "type": "robot",
8              "id": "talos_001",
9              "properties": {
10                 "location": "work_station_id",
11                 "state": "wait_for_screen_manipulation"
12             }
13         },
14         {
15             "type": "tube_rack",
16             "id": "tube_rack_001",
17             "properties": {
18                 "location": "work_station_id",
19                 "state": "mounted"
20             }
21         }
22     ]
23 }
```

start_column_chromatography

Request

代码块

```
1  {
2      "task_id": "xxx",
3      "task_name": "start_column_chromatography", // 开始过柱的task，需要携带较多的过
        柱机设置的参数，机器人会根据这些参数，转换为屏幕操作动作序列，设定好对应参数，然后开始过柱
4      "params": {
5          "work_station_id": "fh_ccs_001", // 要开始过柱的工作站点，enum，同上，目前就
        这一个值
6          "device_id": "isco_combiflash_001", // 设备ID，由labrun分配提供，机器人会在
        后续告知这个设备的状态更新
7          "device_type": "isco_combiflash_nextgen_300+", // 过主机类型，目前我们家里
        的是300，客户这里的是300+，对应UI交互上有一定的差异，需要提供区分一下。未来可能会支持更
        多不同品牌的过柱机，对应不同的操作方式，因此也需要告知设备类型
8          "experiment_params": {
9              "silicone_column": "40g", // 实验选用的硅胶柱规格
10             "peak_gathering_mode": "peak", // 峰收集模式：all, peak, none
11             "air_purge_minutes": 3.0, // 空气吹扫时长，3分钟，float类型
12             "run_minutes": 30, // 运行时长，30分钟
13             "need_equilibration": true, // 是否需要润柱，目前能想见的情况下，通常都
        是true
14             "solvent_a": "pet_ether", // 这两个溶剂也是需要选择的，enum，总共四个
        值，详见代码：pet_ether, ethyl_acetate, dichloromethane, methanol
15             "solvent_b": "ethyl_acetate",
16             "left_rack": "16x150", // 左试管架规格，enum。我们目前只有这一种规格。我
        们家的isco 300型号需要选，bic这里的300+型号不需要选，由过柱机自动识别试管架底部的RFID来
        检测试管架的存在与否以及型号。为了保持二者兼容，这个参数我们还是先保留吧
17             "right_rack": null, // bic这边暂时也不需要这个参数。我们家暂时
18         },
19         "end_state": "watch_column_machine_screen" // enum，详见代码，由robot方提
        供
20     }
21 }
```

其中：

- **end_state:** 开始过柱后，机器人保持在观察屏幕的状态



Robot Motion

- 机器人会根据location_id所在位置，前往对应站点（如果已经在这里就不会动），完成屏幕的locate，然后根据指定参数操作屏幕，开始润柱、过柱，并直到结束，终止过柱，回到end_state。

Response

代码块

```
1  {
2      "code": 200,
3      "task_id": "xxx",
4      "msg": "success",
5      "updates": [
6          {
7              "type": "robot",
8              "id": "talos_001",
9              "properties": {
10                 "location": "work_station_id",
11                 "state": "watch_column_machine_screen"
12             }
13         },
14         {
15             "type": "isco_combiflash_nextgen_300",
16             "id": "isco_combiflash_001",
17             "properties": {
18                 "state": "running",
19                 "experiment_params": {
20                     "silicone_column": "40g",
21                     "peak_gathering_mode": "peak", // 峰收集模式: all, peak,
22                     none
23                     "air_purge_minutes": 3.0, // 空气吹扫3分钟, float类型
24                     "run_minutes": 30, // 运行时长30分钟
25                     "need_equilibration": true, // 是否需要润柱
26                     "left_rack": "16x150", // 左试管架规格为16mmx150mm
27                     "right_rack": null, // 不用右试管架
28                 },
29                 "start_timestamp": "2025-01-13_01-17-25.312"
30             }
31         },
32         {
33             "type": "silica_cartridge",
34             "id": "sepaflash_40g_001",
35             "properties": {
36                 "location": "work_station_id",
37                 "state": "using" // 这个硅胶柱正在被使用
38             }
39         }
40     ]
41 }
```

```

38     },
39     {
40         "type": "sample_cartridge",
41         "id": "ilok_40g_001",
42         "properties": {
43             "location": "work_station_id",
44             "state": "using" // 表示这个拌样柱正在被使用
45         }
46     },
47     {
48         "type": "tube_rack",
49         "id": "tube_rack_001",
50         "properties": {
51             "location": "work_station_id",
52             "state": "using" // 表示这个试管架正在被使用
53         }
54     },
55     {
56         "type": "ccs_ext_module", // 过柱机外置机构
57         "id": "ccs_ext_module_001",
58         "properties": {
59             "state": "using" // 表示这个外置机构正在被使用
60         }
61     },
62     {
63         "type": "column_chromatography_system", // 过柱机
64         "id": "isco_combiflash_001",
65         "properties": {
66             "state": "using" // 表示这个过柱机正在被使用
67         }
68     }
69 ]
70 }

```

take_photo

过柱过程中如果上层希望拍照做中途的分析，来决策是否提前停止过柱，或者调整参数，需要调用此接口进行拍照。

Request

代码块

```

1 // 值守过柱过程中的拍照观察
2 {

```

```

3     "task_id": "xxx",
4     "task_name": "take_photo", // 给指定设备拍照
5     "params": {
6         "work_station_id": "fh_ccs_001", // 要拍照的站点id, 过柱机通风橱, 目前bic就
          这一个
7         "device_id": "isco_combiflash_001", // 要拍照的设备对象id, 由labrun指定, 后
          续result/log加以区分。目前bic这边, 这个通风橱上需要拍照的过主机就这一个, 暂时从需求角度来
          说好像也没别的拍照需要, 但如果后续需要拍别的设备, 那就是对应的设备id
8         "device_type": "isco_combiflash_nextgen_300+", // 设备类型, enum, 由robot
          这边每种设备决定了robot对它的定位方式, 以及包含哪些可拍摄的components
9         "components": ["screen"], // 目前这款过主机只支持screen拍照, 因为暂时也没别
          的需求
10        "end_state": "watch_column_machine_screen" // enum, 机器人执行完这个任务后
          保持的状态, 通常所有task默认都可以回到idle姿态, 除此之外还支持哪些end_state就因task而异
          了
11    }
12 }
13 // 值守旋蒸过程中的拍照观察
14 {
15     "task_id": "xxx",
16     "task_name": "take_photo",
17     "params": {
18         "work_station_id": "fh_evaporate_001", // 要拍照的站点id, 旋蒸通风橱, 目前
          bic就这一个
19         "device_id": "evaporator_001", // 要拍照的设备对象id, 目前旋蒸通风橱上要拍照
          的对象信息还没从需求侧去整理更多, 但预计, 对着切形瓶拍摄后面应该是有这个需求的, 我们总要观
          察旋蒸的进展之类的
20         "device_type": "evaporator", // 要拍摄的设备对象, 旋蒸机。目前我们其实只支持
          这一款旋蒸机, 后续如果有更多, 需要做型号区分, 这里的device_type可能会改的更具体一点, 比如
          加上品牌名、型号名
21         "components": ["round_bottom_flask"], // 对着茄形瓶拍摄
22         "end_state": "idle" // 同上, task结束后的状态
23     }
24 }

```

这是一个面向事件语义的拍摄需求描述。机器人会根据自身状态, 自行处理到达位点、locate、拍摄等动作。其中:

- **device_id**: 要拍照对象的设备id
- **device_type**: 设备类型, 这里是这款过柱机的型号。
- **components**: 设备上的具体要拍摄的部件list。
- **end_state**: watch_column_machine_screen表示技能结束后, 继续保持拍摄时的姿态。idle则是回到idle姿态。

Robot Motion

- 机器人会根据location_id所在位置，前往对应站点（如果已经在这里就不会动），完成设备component的locate，然后完成对component的拍摄。
- 希望上层不需要关心机器人怎么拍，用哪只手拍，而只需要描述清楚要拍摄的对象是什么，拍它的哪里。

Response

代码块

```
1  {
2      "code": 200,
3      "task_id": "xxx",
4      "msg": "success",
5      "updates": [],
6      "images": [
7          {
8              "work_station_id": "fh_evaporate_001",
9              "device_id": "evaporator_001",
10             "device_type": "evaporator",
11             "component": "screen", // 对准旋蒸机屏幕拍照
12             "url": "http://xxxxx.jpg",
13             "create_time": "2026-02-04_16:18:39.234"
14         },
15         {
16             "work_station_id": "fh_evaporate_001",
17             "device_id": "evaporator_001",
18             "device_type": "evaporator",
19             "component": "round_bottom_flask", // 对准茄形瓶拍照
20             "url": "http://xxxxx.jpg",
21             "create_time": "2026-02-04_16:18:39.234"
22         }
23     ]
24 }
```

其中：

- image_url：机器人拍摄好后上传到边缘计算机上的对象存储的可访问地址。

terminate_column_chromatography

暂时不独立提供暂停/恢复过柱的接口，尽管实现上也比较容易做到。我们还是跟着眼下最紧迫的需求来，后面需要了再补。

Request

代码块

```
1  {
2      "task_id": "xxx",
3      "task_name": "terminate_column_chromatography", // 终止过柱的task
4      "params": {
5          "work_station_id": "fh_cc_001", // 同上，这是要执行任务的目标站点名，这个值
          // 目前就是bic这里的过柱通风橱的id，机器人会根据这个id，映射到内置地图的对应站点，然后前往对
          // 应站点执行该任务。机器人会做检查，比如这个站点是否有这个设备，能否执行这个task，如果不对，
          // 会报错（目前还没实现，这是个饼）
6          "device_id": "isco_combiflash_001", // 要执行这个操作的过柱机的设备id
7          "device_type": "isco_combiflash_nextgen_300+", // 要执行这个操作的设备类
          // 型，用作以后区分不同的过柱机型号，对应动作可能会不同，
8          "experiment_params": {
9              "air_purge_minutes": 1.2 // 300+型号需要在终止过柱后手动做空气吹扫，因此
          // 虽然前面开始过柱传了，但其实当时以目前的实操方案来说，不会设置，所以暂时，咱们终止的这个
          // task里再传一下先。目前而言，这里的experiment_params里也不会有其它参数，但还是先放在这一
          // 级dict里吧，万一以后还有别的参数
10         }
11         "end_state": "idle" // 同上，任务结束后的机器人状态
12     }
13 }
```

Robot Motion

- 机器人会根据location_id所在位置，前往对应站点（如果已经在这里就不会动），完成过柱机屏幕操作，终止正在进行的过柱任务，拍摄过柱结果界面的图片并返回。
- 目前默认情况下，机器人完成这次过柱结果的拍照后，过柱机界面保持在最终终止、呈现过柱结果的界面。机器人下次启动过柱时需要再行判断，是否要关闭当前界面，然后开始新实验。

Response

代码块

```
1  {
2      "code": 200,
```

```
3  "task_id": "xxx",
4  "msg": "success",
5  "updates": [
6      {
7          "type": "robot",
8          "id": "talos_001",
9          "properties": {
10             "location": "work_station_id",
11             "state": "idle"
12         }
13     },
14     {
15         "type": "isco_combiflash_nextgen_300",
16         "id": "isco_combiflash_001",
17         "properties": {
18             "state": "idle",
19         }
20     },
21     {
22         "type": "silica_cartridge",
23         "id": "sepaflash_40g_001",
24         "properties": {
25             "location": "work_station_id",
26             "state": "used"
27         }
28     },
29     {
30         "type": "sample_cartridge",
31         "id": "ilok_40g_001",
32         "properties": {
33             "location": "work_station_id",
34             "state": "used"
35         }
36     },
37     {
38         "type": "tube_rack",
39         "id": "tube_rack_001",
40         "properties": {
41             "location": "work_station_id",
42             "state": "used"
43         }
44     },
45     {
46         "type": "ccs_ext_module", // 过柱机外置机构
47         "id": "ccs_ext_module_001",
48         "properties": {
```



```

49         "state": "used" // 外置机构目前处于被使用状态，意为这一次过柱完成了，
           柱子还没拆
50     }
51 },
52 ]
53 }

```

collect_column_chromatography_fractions

包含机器人拉出试管架，收集指定洗脱管中的溶液，倒掉其余溶液，丢弃试管架，合上废液桶盖子，抓取茄形瓶的完整动作序列。

Request

代码块

```

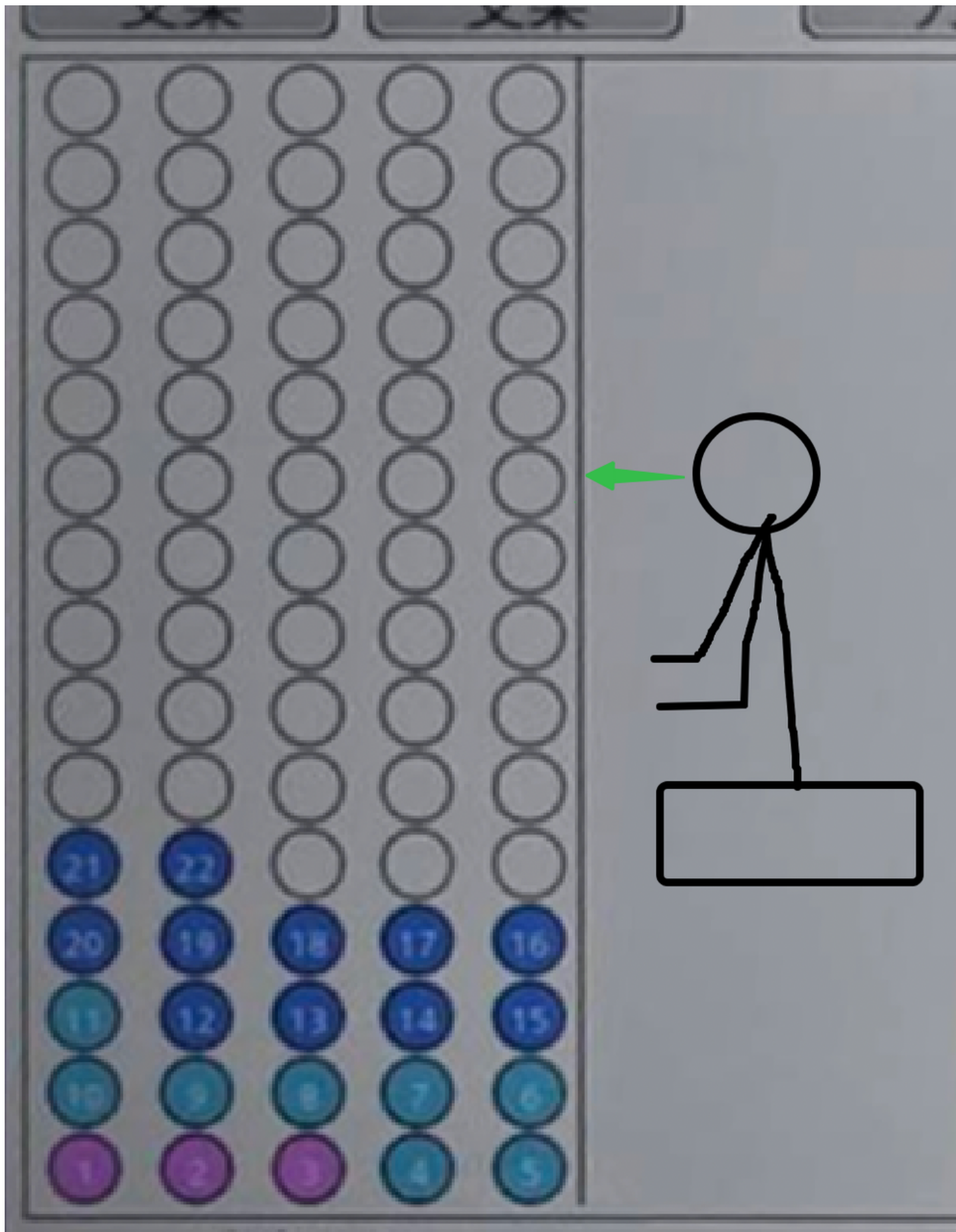
1  {
2      "task_id": "xxx",
3      "task_name": "collect_column_chromatography_fractions", // 过柱后处理环节，收
           集过柱后的容液到茄形瓶。具体而言，包括到指定通风橱，把试管架拉出来，拉出抽屉，然后一根一根
           把样品从试管里转移到茄形瓶或者废液桶，丢弃试管，丢弃茄形瓶上的漏斗，关上废液桶盖子，关上抽
           屉，抓住茄形瓶，回到一个moving姿态
4      "params": {
5          "work_station_id": "fh_cc_001", // 执行任务的站点，目前在bic就这一个能执行这
           个任务的过柱通风橱
6          "device_id": "isco_combiflash_001", // 要执行的设备id，目前就这一个过柱机
7          "device_type": "isco_combiflash_nextgen_300+", // 设备型号，目前就这一个型
           号。以后可能有不同的，对应动作可能也会有一定差异
8          "collect_config": [0, 0, 0, 1, 1, 1, 1, 0, 0, 0], // 试管架上要收集的试管
           编号映射关系，详情参见下面的文字说明
9          "end_state": "moving_with_round_bottom_flask"
10     }
11 }

```

其中：

- collect_config: 表示过柱机收集试管的顺序下，每个试管是否要收集，1表示要，0表示不要。list长度应当为所有用到的试管的长度，机器人呢也只会处理这些数量的试管。目前，我们只支持这一款是管家，是15*5的阵列，如下图所示，这里的list的index编号方式，我们就和这款过柱机自身默认的编号顺序保持一致，即按下图方式来说，最左下角是1号，从左往右是1-5，然后向上是6，即走一个之字形的编号。labrun以及AI算法团队对于这张图的解析应该也是按照这个编号来的，最终labrun的到这个编号方式下的collect_config，给到robot，robot来做后续的格式转换和处理。

- 比如，机器人实际上如图所示，是在右边面朝这个试管架的，机器人会它的左下角，即下图的右下角开始抓取，即机器人实际抓取顺序是5，4，3，2，1，6，7，8，9，10，15，14，13，12，11，以此类推，也并不是一个之字形。但这些labrun都不需要担心，机器人会自己处理好这些事情。





Robot Motion

- 机器人会根据location_id所在位置，前往对应站点（如果已经在这里就不会动），完成上述的所有动作序列。



Response

代码块

```
1  {
2      "code": 200,
3      "task_id": "xxx",
4      "msg": "success",
5      "updates": [
6          {
7              "type": "robot",
8              "id": "talos_001",
9              "properties": {
10                  "location": "work_station_id",
11                  "state": "moving_with_round_bottom_flask" // 机器人手持茄形瓶，可
以移动的姿态
12              }
13          },
14          {
15              "type": "tube_rack",
16              "id": "tube_rack_001",
17              "properties": {
18                  "location": "work_station_id",
19                  "state": "used,pulled_out,ready_for_recovery" // 这里我还没想的很
清楚，想表达的是，目前试管架已经用过了，拉出来了，已经可以被拿去回收处理了
20              }
21          },
22          {
23              "type": "round_bottom_flask",
24              "id": "rbf_001",
25              "properties": {
26                  "location": "work_station_id",
27                  "state": {
28                      "content_state": "fill",
29                      "has_lid": false,
30                      "lid_state": null,
31                      "substance": {
```

```

32         "name": "",
33         "zh_name": "",
34         "unit": "ml",
35         "amount": null
36     }
37     } // 已经收集完，可以用于旋蒸的状态
38 }
39 },
40 {
41     "type": "pcc_left_chute",
42     "id": "pcc_left_chute_001", // post-column-chromatography
43     "properties": {
44         "pulled_out_mm": 12, // 被拉出的距离（这里仅做一个状态记录，后续机器人
        人操作不会严格依赖的，会确保重新安全地观察这个抽屉的状态，来自行妥善处理）
45         "pulled_out_rate": 0.02, // 被拉出的比例
46         "closed": false,
47         "front_waste_bin": "close", // open表示已开盖，close表示已关盖，
        null表示不存在，即未架设
48         "back_waste_bin": "full" // 暂时拍的，表示后面的桶满了，后面大概率要
        重新想想，可能会改
49     }
50 },
51 {
52     "type": "pcc_right_chute",
53     "id": "pcc_right_chute_001", // post-column-chromatography
54     "properties": {
55         "pulled_out_mm": 0.464, // 被拉出的距离
56         "pulled_out_rate": 0.8, // 被拉出的比例
57         "closed": false,
58         "front_waste_bin": "full", // 暂时瞎拍的，留个空，用过的试管桶一律认
        为满了，后面再讨论再改
59         "back_waste_bin": "full"
60     }
61 }
62 ]
63 }

```

其中，这里涉及到了大量的设备状态更新，其中肯定有许多的描述方式是欠考虑的，这里先简单拍一版，后面整体考虑设计清楚了再调整。比如：

- 暂未考虑茄形瓶上漏斗的状态变化，目前机器人会和试管一起丢到右侧方桶里

start_evaporation

需要机器人当前正抓着茄形瓶（这个抓茄形瓶动作到底归到哪个技能，怎么切，以后可以再调整）。

包括机器人工作：前往旋蒸通风橱，架设茄形瓶，操作真空泵，设置旋蒸机参数，开始旋蒸。

- 旋蒸这里还有一点之前没考虑的是，水浴锅里的水位如何确保，水位不同，下降同样高度就不一定能满足预期浸没效果。理论上，感觉是需要机器人来为此负责，要能在实验进行中观察水位位置，是否满足需要，并且动态调整茄形瓶下降高度，以达到预期浸没程度。

Request

代码块

```
1  {
2    "task_id": "xxx",
3    "task_name": "start_evaporation",
4    "params": {
5      "work_station_id": "fh_evaporate_001",
6      "device_id": "evaporator_001",
7      "device_type": "evaporator",
8      "profiles": { // 可以设置套的参数，用于不同情况
9        "start": { // start为保留键，表示开始时的初始状态，不需要trigger
10          "lower_height": 60.5, // 茄形瓶要下降的高度，单位mm
11          "rpm": 60, // 转速，单位rpm
12          "target_temperature": 40, // 水浴加热温度，单位摄氏度
13          "target_pressure": 660, // 真空泵压力，单位mbar，初始参数会再压力到达后
    才松开茄形瓶
14        },
15        // stop我们先暂时去掉吧，暂时机器人也不具备完全终止这个旋蒸的能力，机器人现在
    不妨就做到开始旋蒸
16        // 这一步就好了，后续就交给化学家人工来判断合适结束，以及终止。我们后续再加一个
    terminate_evaporation的
17        // task好了
18        "stop": { // stop为保留键（可以不提供，则不预先指定停止动作），表示结束的条件，机器人点击stop一键停止，不用设置固定参数。这里的stop仅为停止旋蒸机，不涉及取下茄形瓶
19        "trigger": {
20        "type": "time_from_start",
21        "time_in_sec": 3600 // 旋蒸开始后一小时结束
22      }
23    },
24    "updates": [
25      {
26        // 我们暂时不支持后续修改这几个参数好了，对应机器人动作还没开发完，需要
    在目前基础上
27        // 略作修改和测试验证的。但是调整真空泵的压强这件事可以API调用，因此这个
    倒是不妨先加上
28        // 并且师兄也确实有这个需要，开始一会儿后，需要把压强进一步降低。要间隔
    一会儿再降低也是为了
29        // 防治爆沸
```

```

30         "lower_height": 60.5,
31         "rpm": 60,
32         "target_temperature": 40,
33         "target_pressure": 240,
34         "trigger": {
35             "type": "time_from_start", // 延时触发
36             "time_in_sec": 600 // 开始旋蒸后2分钟切为这个参数
37         }
38     ],
39     // 同上，我们暂时先去掉吧
40     "reduce_bumping": { // 爆沸应对参数，不期望发生，作为安全兜底
41         "lower_height": 59, // 爆沸时抬高一点茄形瓶，减小水浴接触面积
42         "rpm": 60,
43         "target_temperature": 40, // 温度不变
44         "target_pressure": 500, // 升高压力，缓解爆沸
45         "trigger": {
46             "type": "event", // 基于事件触发
47             "event_name": "bumping" // 需要机器人具备爆沸检测能力
48         }
49     }
50 },
51 "post_run_state": "observe_evaporation" // 如果profiles里只有start，这里是
idle，则机器人启动旋蒸后就解放了。如果需要机器人值守，持续观察，则这里要设置为观察state
52 }
53 }

```

其中：

- profiles：目前假想设计了比较多不同的profile，有些trigger比如event还是目前机器人不具备的能力。短期内可能只要支持start，stop，以及类似lower_pressure这样由time_from_start触发的就可以。



Robot Motion

- 机器人会根据location_id所在位置，前往对应站点（如果已经在这里就不会动），完成上述的所有动作序列。
- 机器人自己会对技能做一些最基本的前置条件检查，比如它会确认自己当前是不是拿着一个待旋蒸的茄形瓶，如果不是，那会拒绝这个技能，并返回相应报错信息。



Response

代码块


```

1  {
2      "code": 200,
3      "task_id": "xxx",
4      "msg": "success",
5      "updates": [
6          {
7              "type": "robot",
8              "id": "talos_001",
9              "properties": {
10                 "location": "work_station_id",
11                 "state": "observe_evaporation"
12             }
13         },
14         {
15             "type": "round_bottom_flask",
16             "id": "rbf_001",
17             "properties": {
18                 "location": "fh_evaporate_001",
19                 "state": "used,evaporating" // used表示不干净了，有液体，被使用过，
evaporating表示正在旋蒸
20             }
21         },
22         {
23             "type": "evaporator",
24             "id": "evaporator_001", // post-column-chromatography
25             "properties": {
26                 "running": true, // 表示已启动，启动其实主要是启动水浴加热
27                 "lower_height": 60.5, // 茄形瓶下降的高度，单位mm
28                 "rpm": 60, // 当前转速，单位rpm
29                 "target_temperature": 40, // 目标水浴加热温度
30                 "current_temperature": 36, // 当前水浴加热温度
31                 "target_pressure": 660,
32                 "current_pressure": 659
33             }
34         }
35     ]
36 }

```

其中，response中，目前把气压放到evaporator里了，尽管目前我们实际上物理意义上真空泵是一个独立的设备，但从配套使用上，我们暂时不妨把它视为一个完整设备，感觉会简单一点，否则，要额外维护一些设备关联关系之类的，技能传参也会变得更加复杂。

[TODO] stop_evaporation

结束旋蒸，把茄形瓶取下，拿到需要得地方去。

次优先级，先放一放。

[TODO] setup_ccs_bins

架设过柱机通风橱右侧的若干桶

[TODO] return_ccs_bins

把用过的过柱机通风橱右侧的桶给拿到废料区

[TODO] return_cartridges

把用过的柱子拆下来放回废料区

[TODO] return_tube_rack

把用过的试管架取下来拿回废料区

三、实现进展

3.1 lab_edge

边缘计算机上的服务：https://github.com/c12-ai/lab_edge

目前就放了个minio，以及rabbitmq的server。

3.2 mars_service

机器人上的服务：https://github.com/c12-ai/mars_service

还在开发中。