XNet：简单实验室仪器控制协议构建及其思考

Building simple protocols for controlling laboratory facilities

罗嘉宏 luojh@mail.ustc.edu.cn | luojh@ustclug.org

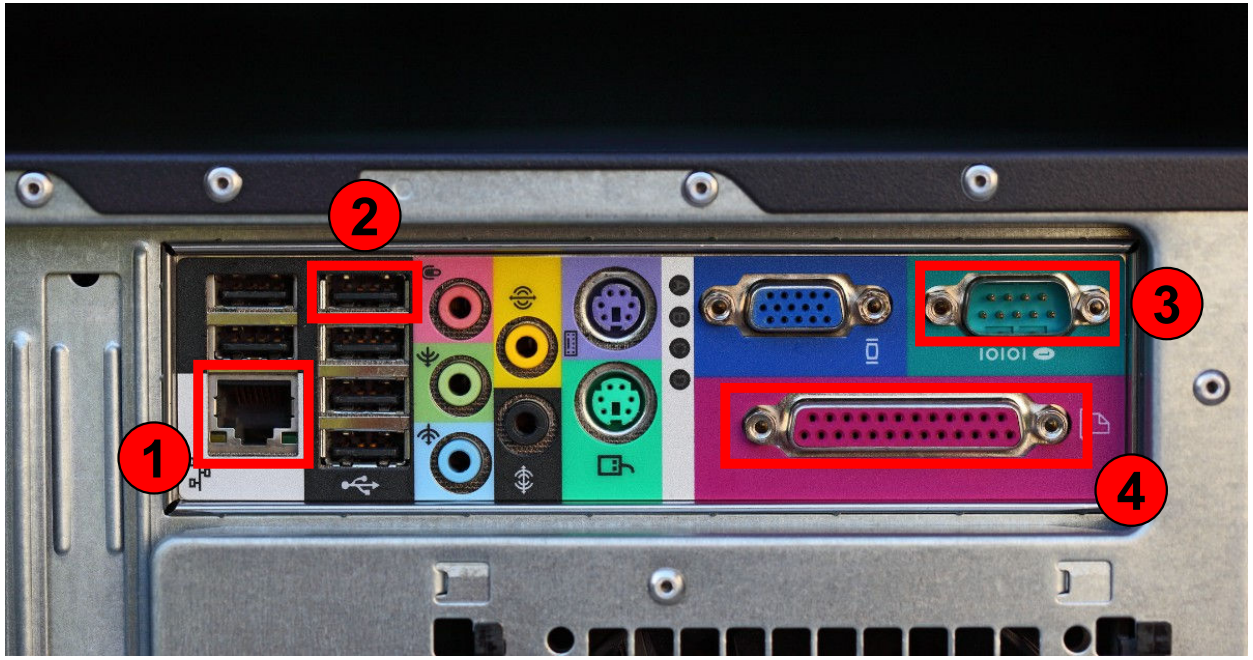
中国科学技术大学 化学与材料科学学院 scms.ustc.edu.cn

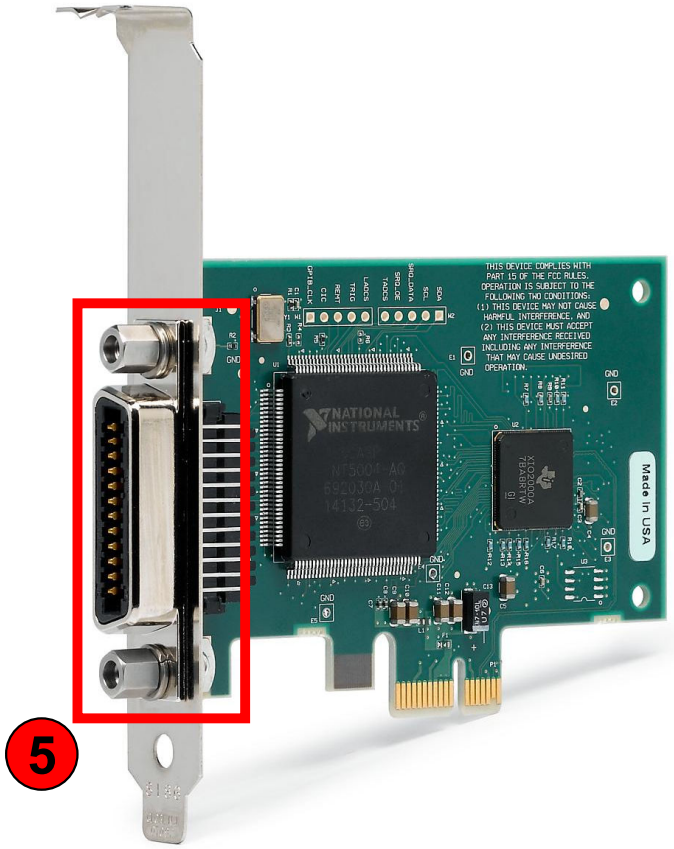
中国科学技术大学 学生 Linux 用户协会 lug.ustc.edu.cn

目录

1. 实验室仪器控制的现状

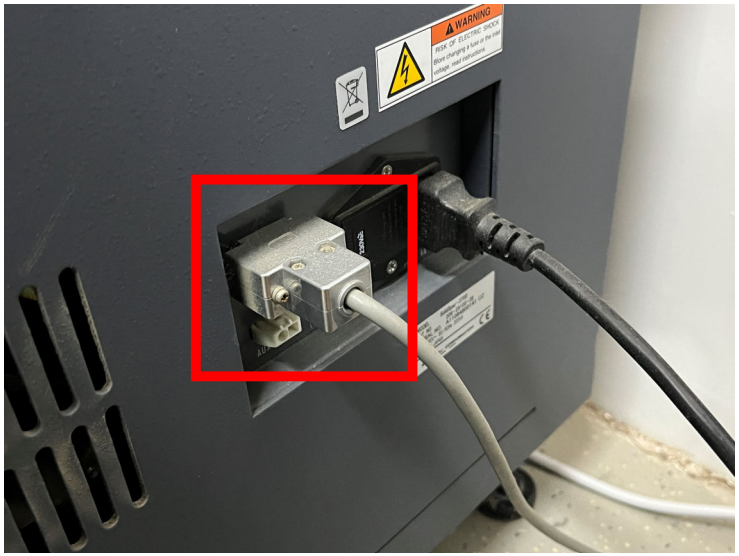
- 用几个关键词简单概括：
  - **新旧共存** 旧式控制接口（例如，GPIB 和 RS232）和新式控制接口（例如，USB 、以太网）并存，两者都有仪器厂商选用。而且，由于某些接口比较特殊，还需要为仪器专门配备有那种接口的计算机。
  - **高速和低速共用** 由于上面新旧式接口共存的原因，以及某些仪器上使用高速接口传输低速信号（例如串口转 USB），高速信号和低速信号往往是不分开的。



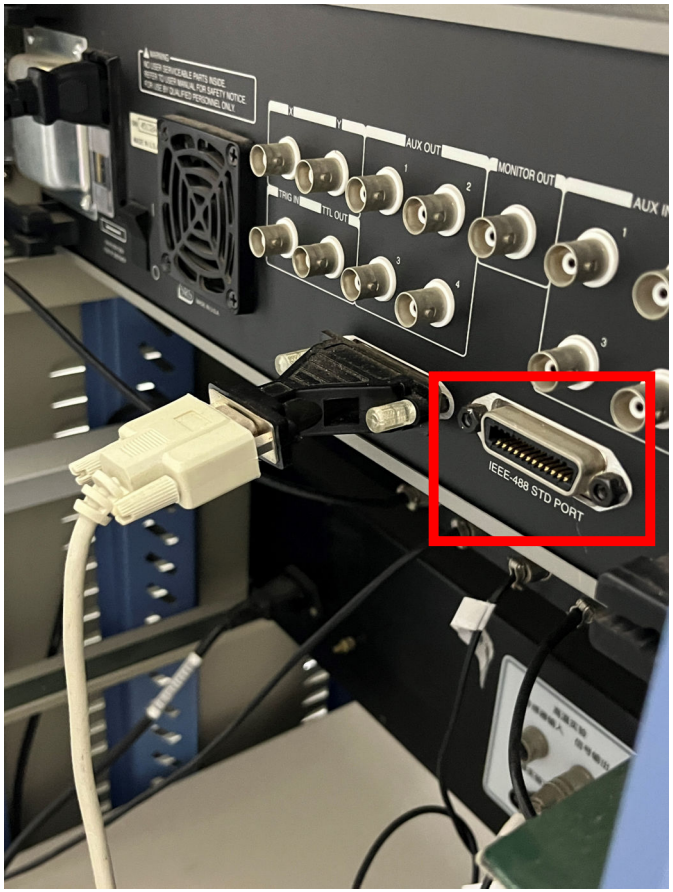


市面上常见的仪器接口

1. 有线网络端口（电口）
2. USB
3. 串行端口（RS232）
4. 并行端口
5. GPIB（IEEE488.2）



RS232



IEEE-488 (GPIB)

RS232 to USB

Ethernet

1. 实验室仪器控制的现状

- 用几个关键词简单概括：
  - **协议专有** 仪器厂商通常会提供专有的 SDK 简化控制程序的开发，但是很多时候这种 SDK 会只有编译好的二进制版本（例如提供 DLL），因此限制了运行的机器架构和 OS 选择。SDK 包装了计算机到仪器的控制协议，且这种协议通常是专有的，因此无法进行迁移。
  - **协议不一致** 即使是同类的仪器（例如光谱仪或者数字多用表），不同厂家甚至同一厂家的不同仪器也会有不同的控制协议。这使得更换不同的仪器变得极为困难。

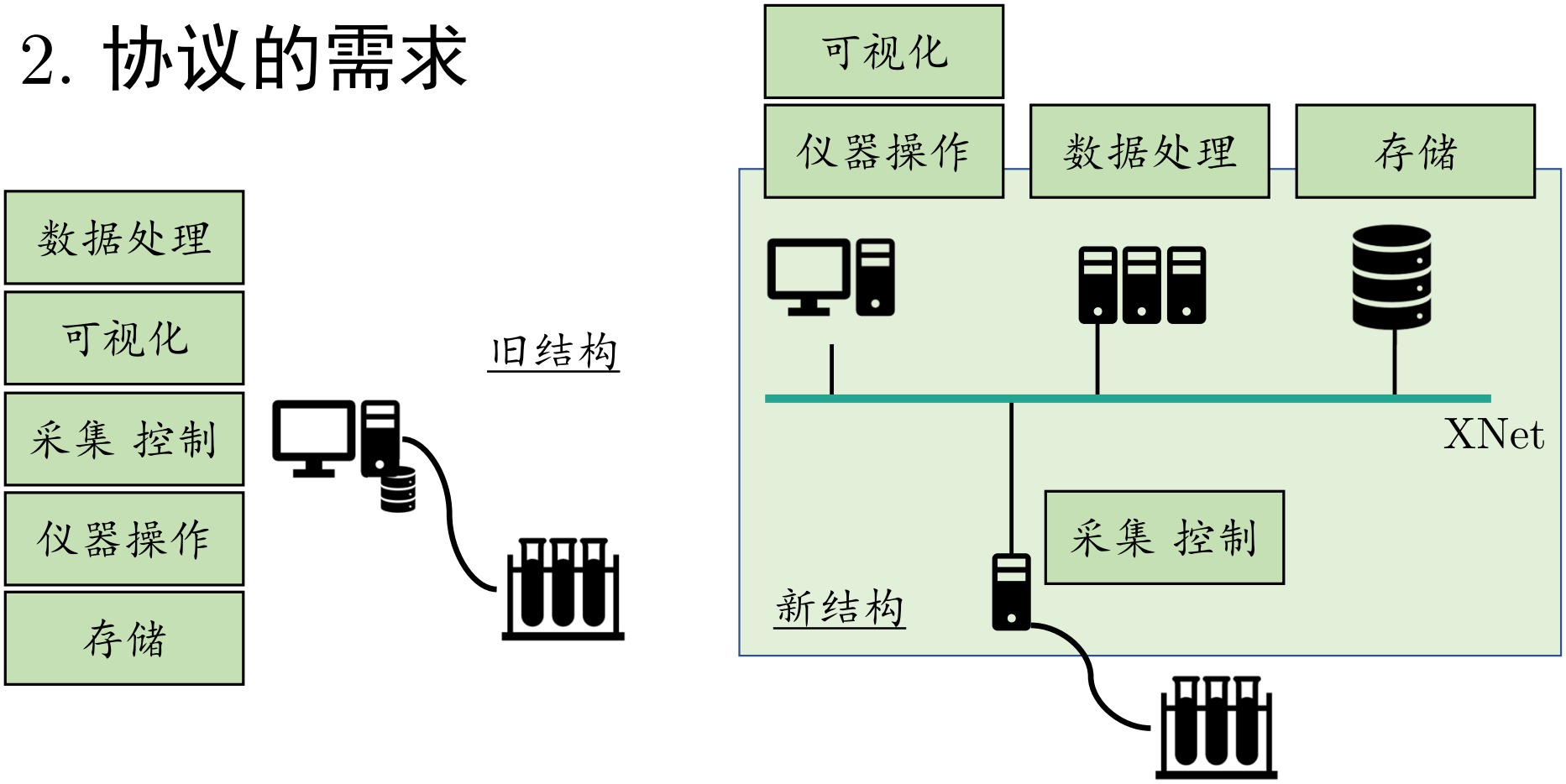
1. 实验室仪器控制的现状

- 此外，在控制用的软件方面也存在一些问题：
  - 为每套仪器单独编写一个控制程序，这个控制程序中包含了大量的与具体设备相关的参数且难以修改。故更换仪器时无法做小范围的更改，而是要大量地重写。
  - 数据采集和数据处理在同一个程序中，导致每个实验使用的计算机设备性能要求都较高。
  - 每人使用的编程语言和编码习惯不一致，使得程序的迁移变得困难，也使得全流程的自动化变得困难。

1. 实验室仪器控制的现状

- 此外，在控制用的软件方面也存在一些问题：
  - 操作界面和监控界面以图形界面为主，且直接运行在与仪器连接的计算机上，故每台设备都需要配置显示器、鼠标键盘等，不利于改造为自动化系统；
  - 数据传输靠 U 盘甚至微信文件传输，传输效率低。

2. 协议的需求



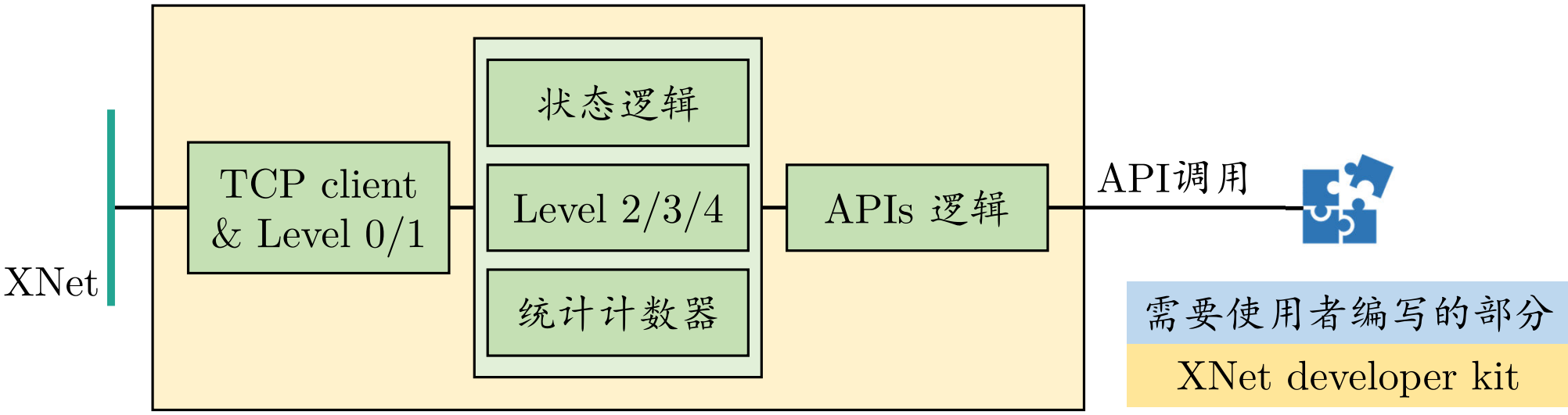
2. 协议的需求

- 作为实验室级的协议，特别是非计算机专业实验室中使用的协议，我们要求其具有一定的特点：
  - 与现有的编程偏好一致，即对各种实验室研究常见的编程语言（例如 NI LabVIEW 和 Matlab）有良好的操作性；
  - 实现控制、采集、数据处理、数据存储以及监控的分离，允许远程访问；
  - 可扩展性良好（例如著名的 `finger` 程序可乐机），能以同一套框架适配不同的仪器类型，以及适配后续仪器的升级和调整/变化；
  - 提供访问控制、统计、资源组等功能，允许多个应用同时连接仪器（复用），也允许仪器独占、资源调配/资源组等功能；

2. 协议的需求

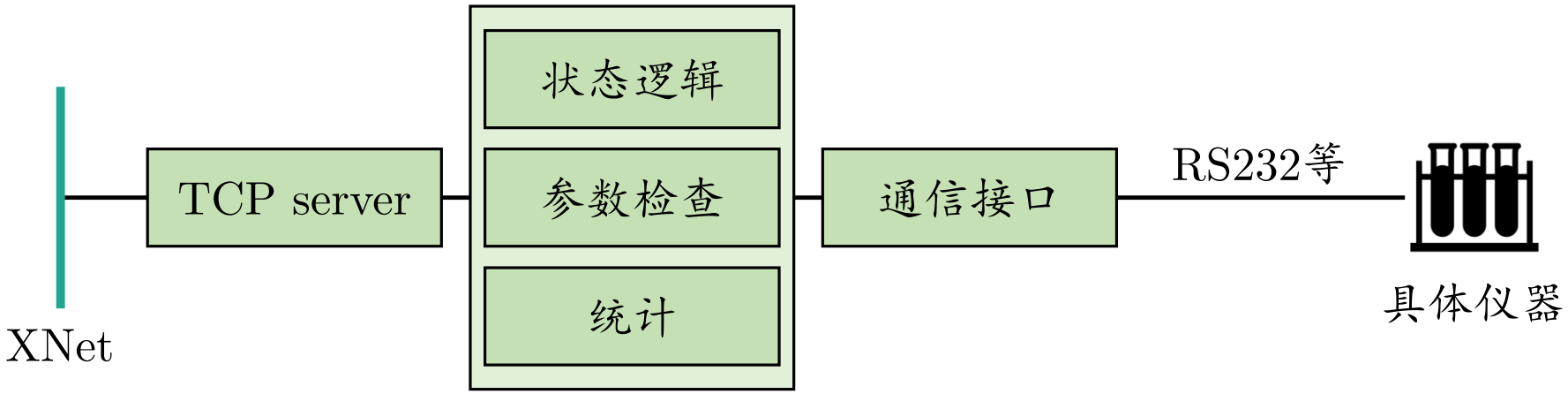
- 几个组件：
  - **用户程序**：由研究者自己编写的程序，其中包含了控制设备和数据收集的顶层逻辑。
  - XNet developer kit：作为外加库接入用户程序，用户程序通过 APIs 调用，完成 XNet 低层次的操作细节处理。
  - **仪器适配器**：XNet 到仪器具体的通信方式的转换器，附加仪器的状态逻辑和参数检查等功能。
  - **其他附加组件**：xnet-router 、简单 dashboard 等。

2. 协议的需求



用户控制程序的结构

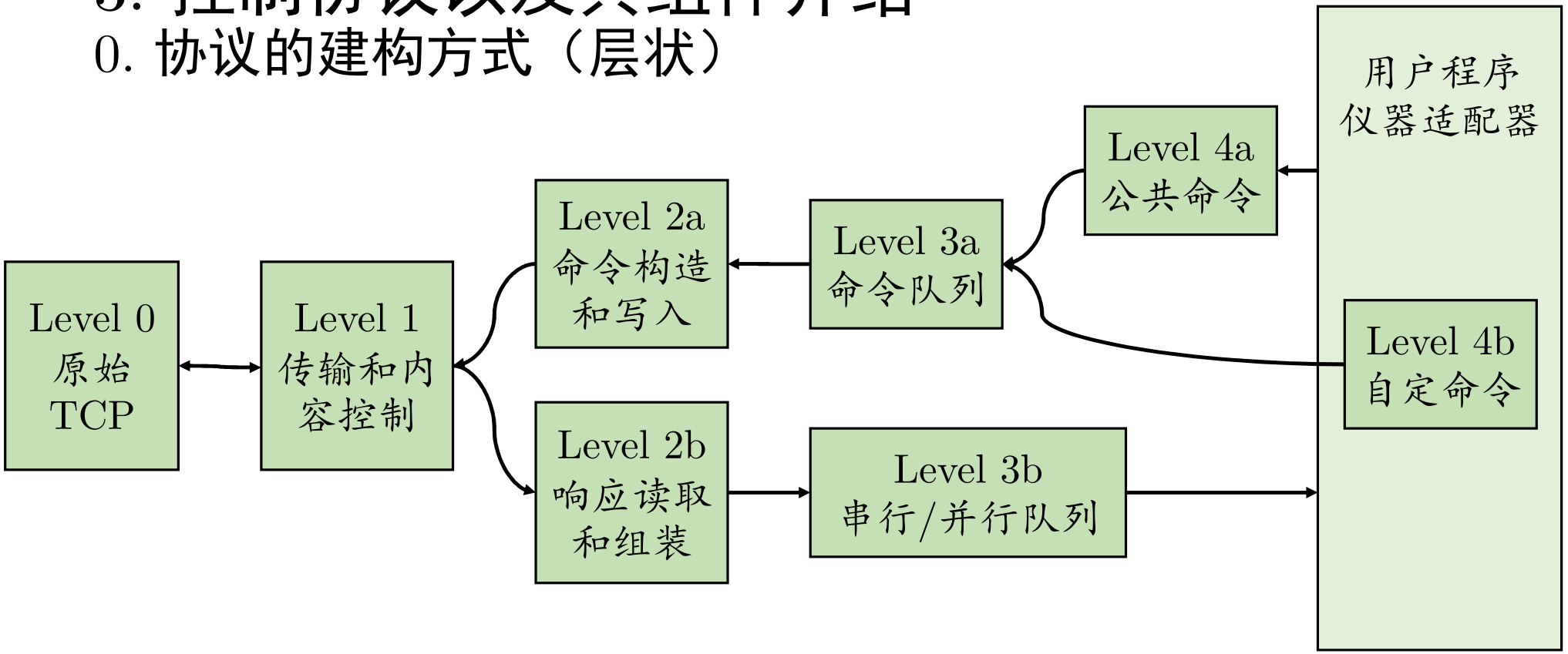
2. 协议的需求



仪器适配器的结构

3. 控制协议以及其组件介绍

0. 协议的建构方式（层状）



3. 控制协议以及其组件介绍
文本字符集

- 字符集：
  - 为了确保协议在各类平台上都能正常工作，规定下面的字符选用方式：
  - set0：使用标准 ASCII 字符；
  - set1：使用 ASCII 可打印字符，`\n` 和 `\r\n` 等价；
  - set2：进一步地，使用 `0-9` `A-Z` `a-z` `:` `/` `-` `_` `=` `+` ；
  - set3：进一步地，不区分大小写；

3. 控制协议以及其组件介绍
参与通信的设备

- 参与通信的设备需要：
  - 0、1、多个 TCP server / TCP client，用来完成底层数据的传输。
  - 一个 name ，表示该设备的友好名称。
  - 一个 addr ，表示该设备的地址。与 IP 地址不同的是，可以使用 set2 字符集的字符。
  - 一个 haddr ，表示设备的底层地址，是安装时生成的一个随机字符串，范围为 set3。

3. 控制协议以及其组件介绍

Level 0. 数据传输

- 注：下面的讨论都是基于去除加密和校验等安全措施后的简化版本。
- 数据通过简单 TCP 传输。传输使用的连接在会话期间一直保持。
  - 由于各种编程语言都对简单 sockets （例如 C 上提供的 Berkeley sockets）有良好的支持，确保绝大部分编程语言以及系统平台上能用较短的代码完成控制。
  - 此外还有一个优点是所有的程序在通信上都没有显著的 OS 平台限制。（Python：25 行，C：73 行，Matlab：37 行）

3. 控制协议以及其组件介绍
Level 1 & 2. 文本内容的格式

- 使用 JSON 作为传输的文本结构。
  - 被各种编程语言广泛支持，解析非常方便。
  - 具有很好的可扩展性，可以添加必须字段外的其他数据项。
  - 不需要担心（二进制数据）可能存在的字节序/浮点处理等等问题。
  - 方便调试和前后向兼容。
- JSON 是文本，中间允许换行，而一个请求/响应的结尾用双换行符（等价于一个空行）进行分隔。
  - 这是模仿 HTTP 的 header 和 body 的分隔方案。

3. 控制协议以及其组件介绍
Level 2. 传输控制

- 下列字段自动被添加，用来控制和统计数据的基本传输。
  - version: string：命令的版本。设备收到不能适配版本的命令时应丢弃，避免被错误控制。
  - time: int64：响应发出的时间，按 UTC 为时区，单位为 100ns。
  - serial: int64：一个整数，从 0 开始，每发出一条请求/响应就加 1。
  - from: string：发出数据的设备的 haddr。
- 可以指定的：
  - to{*h*|*a*|*n*}: string：发送到设备的 {haddr | addr | name}。可以使用正则表达式，且留空/不指定表示发送到任何设备。

3. 控制协议以及其组件介绍
Level 2a. 请求公共字段

- 参考了 SCPI (Standard Commands for Programmable Instrument) 的命令结构。字段分为公共字段和自定字段。
- 请求中应该至少包含下列公共字段：
  - command: string：所请求的命令/方法名称。如果有多个层次（例如子命令）则推荐使用下列符号分层：`:` `/` 。

3. 控制协议以及其组件介绍
Level 2b. 响应公共字段

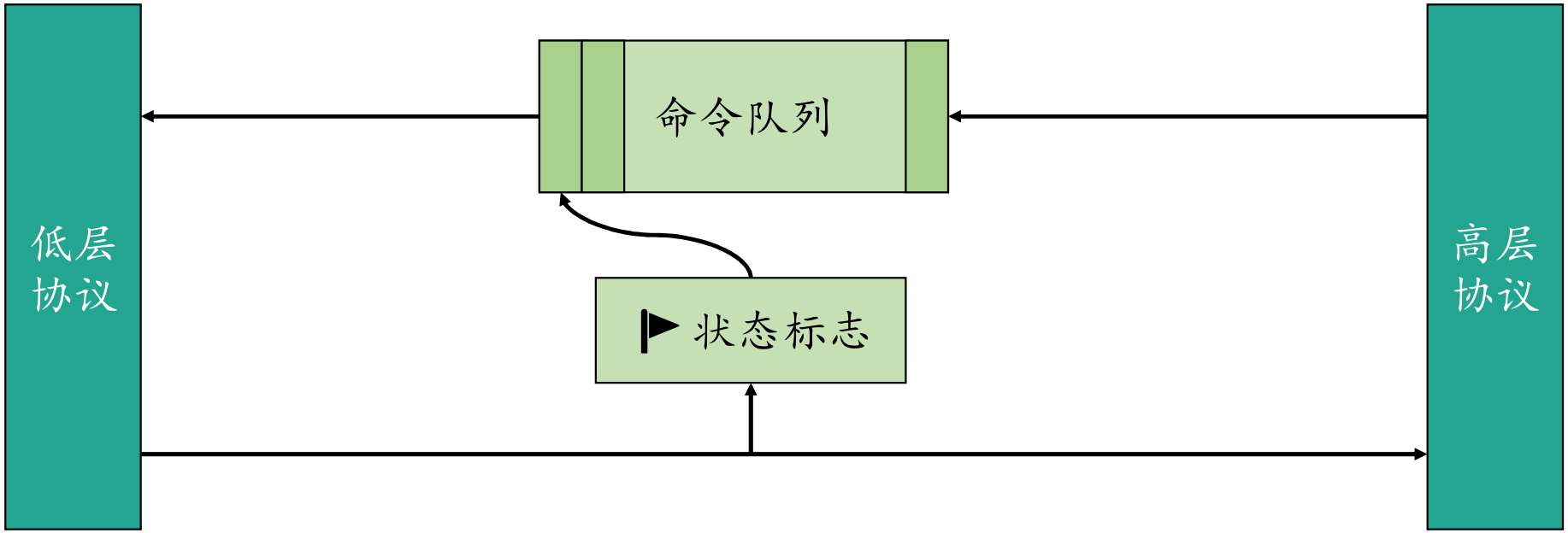
- 响应字段同样分为公共字段和自定字段。
- 响应中应该至少包含下列字段：
  - code: int：一个整数，用来表示命令的执行状态。目前采用的方案受到了 HTTP 状态代码的启发。
- 此外，响应一般还可以包含下列字段：
  - message: string：除了 code 提供的信息外还可以提供的额外信息。

3. 控制协议以及其组件介绍
Level 3. 串行和并行的考虑

- 有些仪器需要（也只能）串行执行命令，而有些则可以并行执行命令。如何解决这个问题？
  - 串行命令不需要做修改，发出后等待回应，确认执行完成后再发出即可。如果在上一条命令未完成时提交了新命令，可能会无响应/回应 busy 状态。
  - 并行命令需要一个并行控制器的参与。并行控制器简化了这些命令的调度复杂度，为上层应用提供了一个清晰简单的接口。并行控制器也可以被配置为串行执行，此时其相当于一个命令队列。

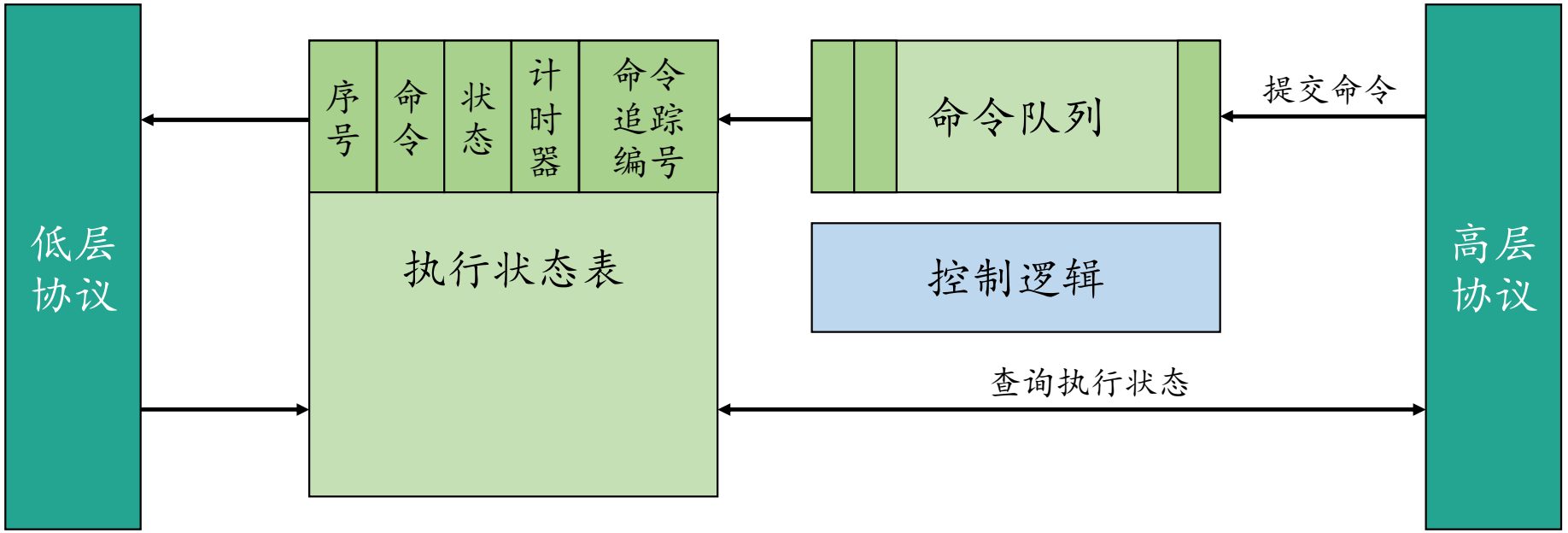
3. 控制协议以及其组件介绍

Level 3. 启用了串行本地队列时：



3. 控制协议以及其组件介绍

Level 3. 启用了并行执行控制器时：



3. 控制协议以及其组件介绍
Level 3. 串行和并行的考虑

- 并行命令需要额外的追踪用字段，因为命令和返回的结果未必是成对出现的。需要的字段：
  - trace: string：追踪代码，一般使用一个随机或组合顺序的字符串。命令指定的 trace 会被附加在所有与这一命令有关的数据上。并行控制器通过这一标签来分类和存储每个并行命令的状态。
- 如果需要高可靠性的并行执行，还需要下面的字段：
  - traces: string：追踪代码，以发出的命令为 0，该命令的第一个响应为 1，第二个响应为 2，等等。

3. 控制协议以及其组件介绍
Level 3. 串行和并行的考虑

- 值得注意的是，由于使用了单一的 TCP 连接，故其具有数据流的特性。即使使用了并行执行，每个命令或请求的数据收发（传输）时间内为了保证收发到的 JSON 数据的完整性，TCP 连接是被独占的。

3. 控制协议以及其组件介绍
Level 4a. 公共命令

- 为了满足此协议的要求，约定一些命令作为公共的、应普遍支持的命令。为了保证任意设备都可以在不加额外条件的情况下执行，这些命令均约定为串行的，且没有版本要求。
  - who 命令：返回对方的 name 、addr 、haddr。
  - version 命令：返回对方的版本和可接受的版本。
  - host 命令：返回对方的主机名称。
  - status 命令：为了收集工作过程的数据和监控状态，该命令用于返回对方的目前状态和统计数据。可以使用 filter 项目来指定需要那些数据。

3. 控制协议以及其组件介绍
自定命令：以一个位移台举例

- 控制：
  - move 命令，操作位移台移动到某个位置。完整的例子如下：

```
{
    "command": "move",
    "axis": "x",
    "type": "relative",
    "distance": "5.5"
}
```

- 注意这里对仪器的细节做了抽象：不同的具体位移台有不同的数据单位，例如 μm、mm、步进电机步数等。而位移台类的设备命令统一为 mm 单位，单位转换是在适配器中完成的。
- 同时，适配器也会对请求涉及的参数进行校验，确保不会因为错误的参数损坏仪器。

3. 控制协议以及其组件介绍
自定命令：以一个位移台举例

- 控制：
  - speed 命令，调整位移台的速度，包括加减速加速度、运行速度：

```
{
    "command": "speed",
    "axis": "x",
    "accel": {
        "begin": "1",
        "end": "-2"
    },
    "speed": "5"
}
```

同样也是统一了单位：mm/s 和 mm/s^2。

3. 控制协议以及其组件介绍
自定命令：以一个位移台举例

- 查询：
  - pos 命令，查询位移台目前的位置，任何时候都可以执行。
  - home 命令，指出位移台的滑块是否在零位。
- 以及一些辅助的命令，例如可以：
  - 读取温度传感器的读数
  - 读取滑台的总使用距离
  - 读取最近占用时间比例
  - 维护计数器（维护计时器）的查询和重置

3. 控制协议以及其组件介绍

额外组件：router

- 有时候会有将多路连接合为一路的需求，以及利用跨网络的设备完成数据转发、流量控制、流量统计等等的功能。此时，有一组件，称为 `xnet-router` ，可以在恰当的配置下完成这些工作。
- 包含一个 TCP server 和多个 TCP client，以及内部的控制逻辑和数据收集/统计功能。

3. 控制协议以及其组件介绍

额外组件：router

- router 组件将在任何流经它的数据包上增加（或修改）一个字段：
  - `route`: `string`：表示该命令到此时经过的转发路径，内容使用 | 分割为一个个项目，且从尾部进行添加。如果这个数据是一个响应，那么还会有 `fwroute` 项目，该项目是请求时经过的完整路径（由路径终端添加）。
- router 组件需要一些配置：
  - 简单的包括 TCP server 的监听地址和端口，TCP client 需要连接到的 `server:port` 等。如果需要流量控制，还可以设置相关的最大连接量等。
  - 此外，为了使得系统正常工作，router 需要一个配置项目，即其转发规则表，其中说明了符合哪些名称条件的包应该转发到哪里。

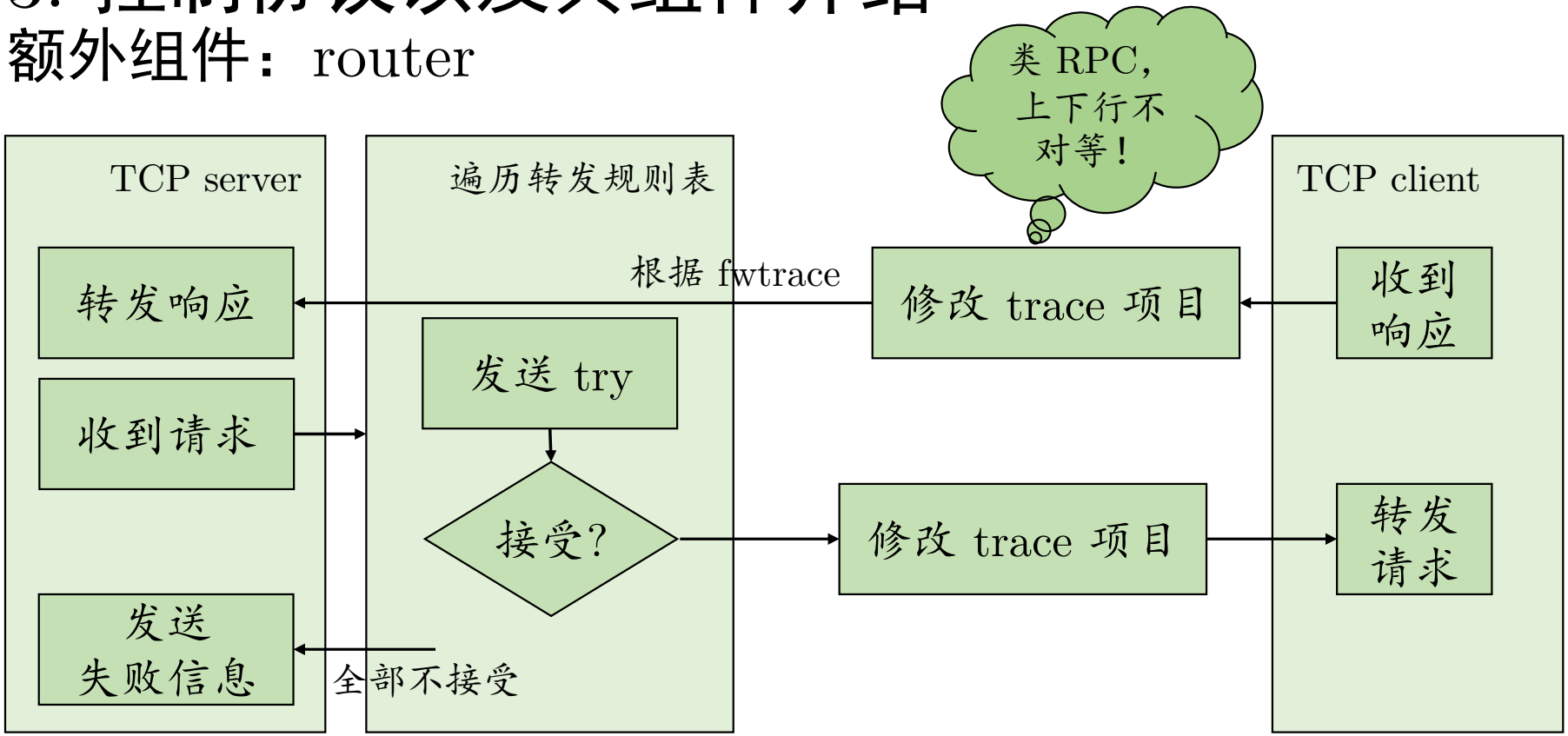
3. 控制协议以及其组件介绍

额外组件：router

- router 还增加了 2 个命令：
  - `table`：列出其转发规则表的所有入口，即所有可以接受的数据的条件。
  - `try`：测试指定的条件能否被接受转发。

3. 控制协议以及其组件介绍

额外组件：router



4. 兼容性和使用简易性的考虑

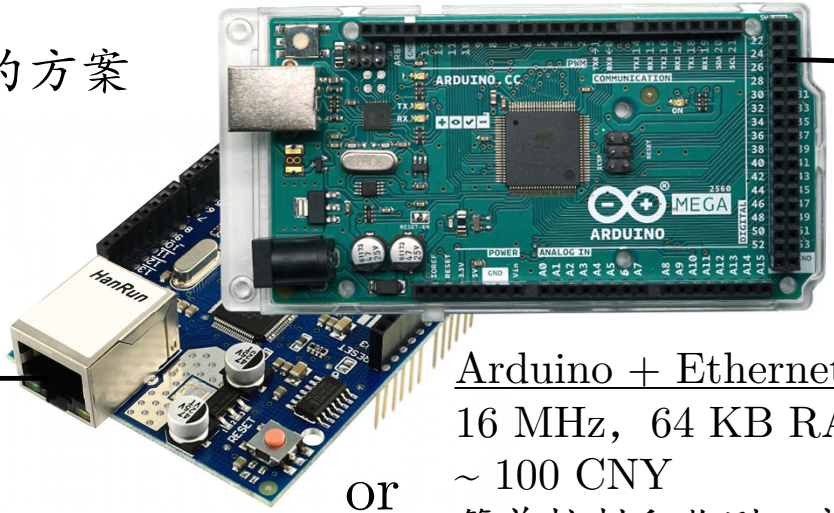
- 兼容性：
  - OS **平台**：非平台特定部分使用 .NET Core 编写，实现良好的跨平台特性。平台特定部分使用可更换的底层组件，方便迁移。
  - **仪器设备**：仪器设备的适配器（XNet 到具体的仪器控制协议的转换）有框架，只需要编写命令的处理程序即可。
  - **编程语言**：基本上只要有标准 TCP 通信的编程语言就可以使用，数十行代码就可以将自己的仪器控制程序基本运行起来。
- 目标用户群体并非专业计算机程序员。所以，应尽可能隐藏协议的实现细节。
  - 并行、调度方式以及转发等背后的通信细节一般不需要过多的人工调节。

5. 构建过程中的一些思考

- 除了可以用于仪器控制，还可以用来做什么？
  - 理论上可以控制任何的设备，也不仅是实验仪器。例如，目前有计划接入的：各类网络设备的流量监测（switch 等），以及做自动化的计算任务提交。
  - 也可以作为只读的（监控用途），例如读取实验室温度传感器的读数（这点充分利用开源硬件，例如 Arduino + Ethernet shield 接入）、UPS 系统的状态、实验室主要计算机的状态等等。
  - 这样 dashboard 就可以方便地接入。

低性能 & 低成本的方案

实验室
网络



Arduino + Ethernet shield
16 MHz，64 KB RAM
~ 100 CNY
简单控制和监测，实时性能好

or

Raspberry Pi: ~ 1 GHz，2 GB RAM
~ 300 CNY
仪器控制，中型数据采集（via USB）

某种传感器

5. 构建过程中的一些思考

- 如何方便非计算机背景的同学学习使用？
  - 编写了一个测试用的虚拟仪器程序，运行起来后就像一个真实的适配器一样可以接收命令，并将所有收到的命令解析好之后显示在一个窗口中。
  - 还可以模拟一些简单的常用设备的控制命令。
  - 有助于在实验程序编写的阶段进行测试，而不用担心长时间的调试占用/错误的命令导致仪器损坏。

5. 构建过程中的一些思考

- 大块数据（例如相机拍到的图片，大约 3 到 5 Mbytes 每张，从 XNet 上传过去的话，会占用较多的资源，这不是 XNet 的基本目标。
- 两个方式：
  - 定期通过 rsync 这种工具从仪器连接到的计算机上收取文件
  - 仪器连接到的计算机采集到数据后主动通过 ftp/rsync 上传到存储服务器

5. 构建过程中的一些思考

- 现代化高通量的实验，需要大量的实验室设备（10台，甚至上百台）的协同控制、高性能计算的处理能力支撑以及大容量存储 —— 这对于传统的单一计算机而言是不现实的。
- 因此，将实验中的各个角色拆开，并通过网络连接起来是一个良好的选择。
- 同时，将同类仪器设计成统一的接口，有助于简化仪器的扩容、替换和迁移过程。—— 仪器国产化

5. 构建过程中的一些思考

- 实验仪器的通信方式和接口从不统一走向现在的统一（主流为 USB 和网口）；
- 实验室工作自动化和“无人化”的重要意义：大量人力和时间花费在简单机械性的工作上，提高了实验的失误概率，而这正是程序控制的机器擅长的。
- 理学类学科缺乏的知识：程序结构和程序逻辑清晰、模块化？

谢谢大家！

XNet 正在修改和完善的过程中，希望得到更多的建议