

40 | 服务端的业务架构建议

2019-09-10 许式伟

许式伟的架构课

[进入课程 >](#)



讲述：姚迪迈

时长 12:10 大小 11.15M

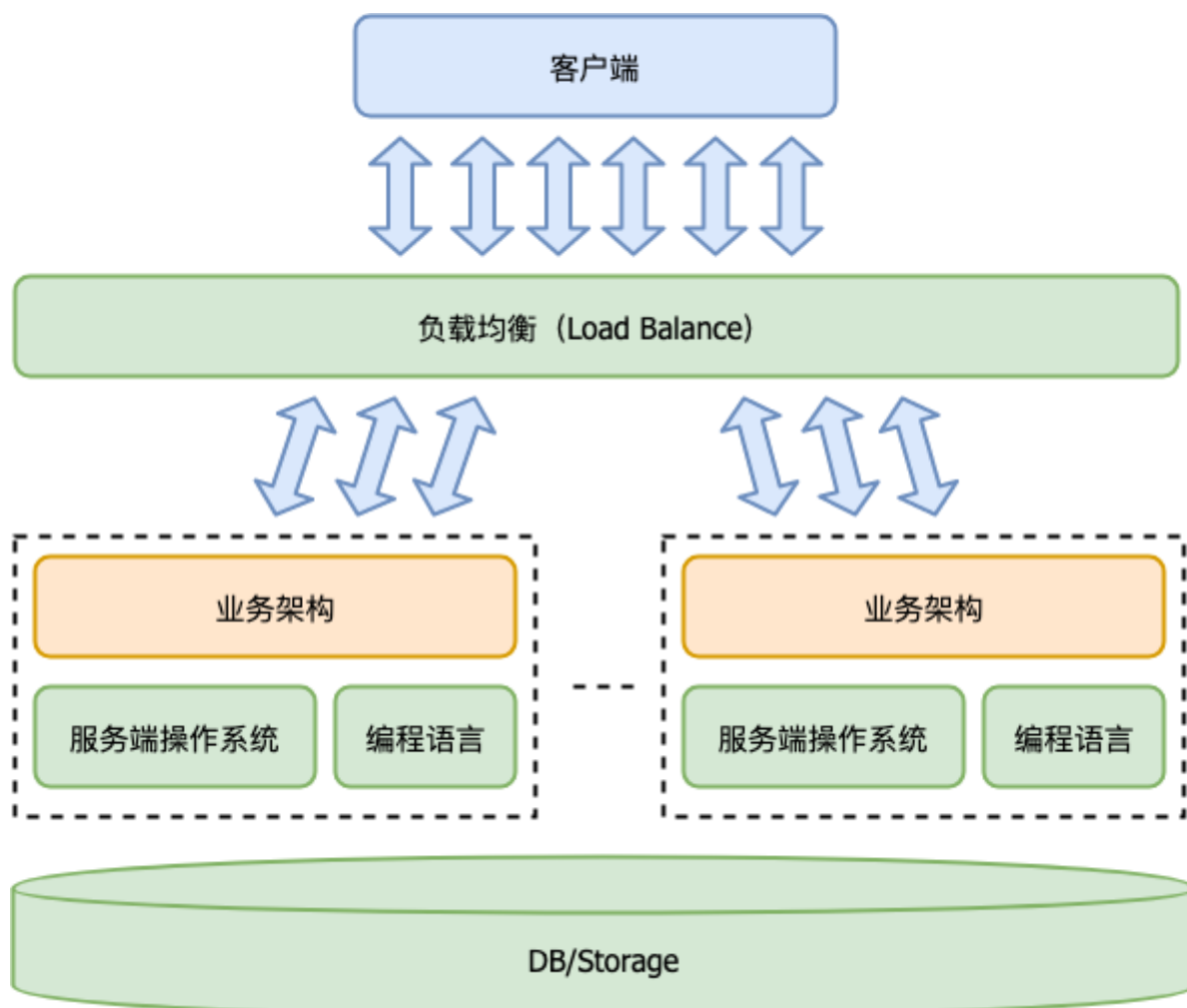


你好，我是七牛云许式伟。

相比桌面程序而言，服务端程序依赖的基础软件不只是操作系统和编程语言，还多了两类：

负载均衡（Load Balance）；

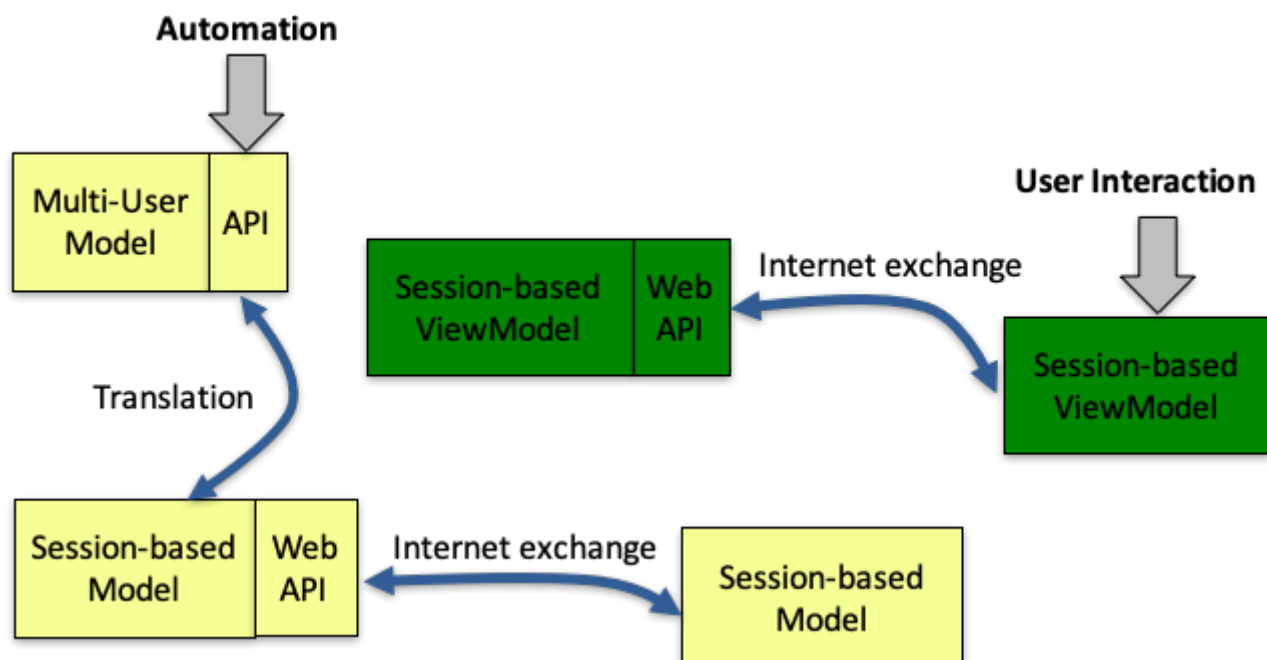
数据库或其他形式的存储（DB/Storage）。



我们前面几讲已经介绍了负载均衡和常见的存储中间件。今天，让我们就把焦点放在上图中的业务架构上。

大方向来说，业务架构必然是领域性的，与你所从事的行业息息相关。但就如同桌面程序会有自己的架构体系的套路一样，服务端的业务架构也会有自己的套路。

在第二章 [“24 | 跨平台与 Web 开发的建议”](#) 这一讲中，我们概要地画过服务端的体系架构，如下图所示。



在图中，我们把服务端分成了两层。底层是 Multi-User Model 层，一般情况下它对外提供了一套 RESTful API 接口。上层是 Web 层，对外提供 Web API。Web 层又分为 Session-based Model 层和 Session-based ViewModel 层。

一般来说，Session-based Model 是一个非常简单的转译层。而在胖前端的模式下，Session-based ViewModel 层也几乎没有任何后端的代码，就是一些托管的资源文件，包含一些 HTML + CSS + JavaScript 文件。

我个人会倾向于认为，Session-based ViewModel 层属于桌面开发的范畴，哪怕是胖后端的模式下也会这样去归类。只不过在胖后端的方式下，桌面程序的很多逻辑不再是由 JavaScript 完成，而是由类似 PHP 之类的语言完成。

故此，我们今天探讨的业务架构，主要谈的是 Multi-User Model 层。

网络协议

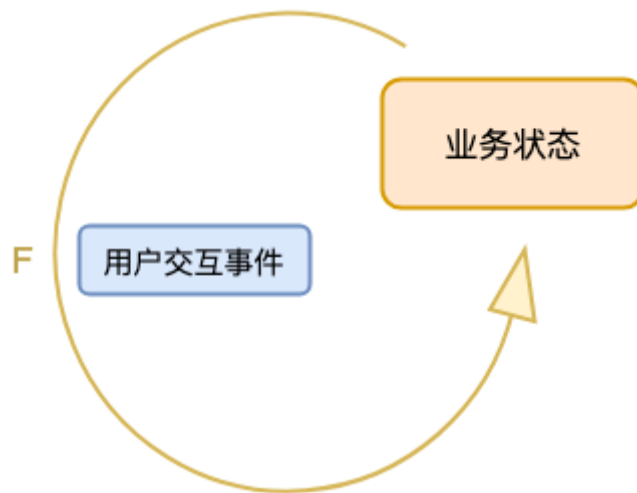
探讨 Multi-User Model 层，第一个重要话题是网络协议，它是服务端程序的使用界面（接口）。考虑到这一层网络协议往往提供的是 RESTful API，所以有时它也会被称为 RESTful API 层。

大家可能经常听到 RESTful，但它到底代表什么？

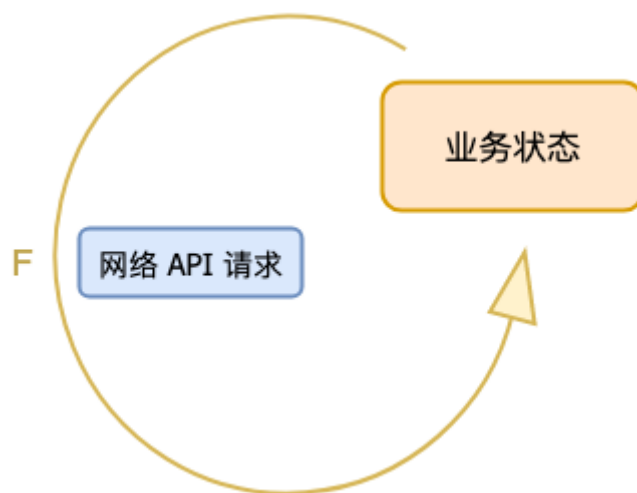
所谓 RESTful，是指符合 REST 原则。REST 的全称是 “Representational State Transfer” 。它强调的是：

第一，客户端和服务端之间的交互在请求之间是 “无状态” 的。这里的无状态更严谨的说法是 “无会话 (Session)” 的，从客户端到服务器的每个请求，都必须包含理解请求所必需的完整信息。服务器可以在请求之间的任何时间点重启，客户端不会得到通知。

在 “[36 | 业务状态与存储中间件](#)” 这一讲中，我们把桌面程序和服务端程序都看作一个状态机。桌面程序的状态转化由 “用户交互事件” 所驱动，如下图。



而服务端程序的状态转化由 “网络 API 请求” 所驱动，如下图。



但是从状态转化角度来说，桌面程序和服务端程序很不一样。桌面程序的状态转化往往存在中间的 “临时状态” ，这其实也是 Controller 层的价值所在。

在桌面程序的 MVC 架构中，Model 层提供核心业务，它不存在 “临时状态” ，每一个对外提供的接口 (API) 都完成一项完整的业务。View 层提供呈现，和我们的话题关联不

大，这里不展开来讲。Controller 层负责把 “用户交互事件” 翻译成 Model 层的业务 API。在 Controller 层往往存在 “临时状态” 的，它需要把多个连续的 “用户交互事件” 组装起来完成一项业务。我们第二章实战的 “画图” 程序，它的各类 Controllers，比如 FreePathCreator、RectCreator 等等，都是很好的例子。

服务端程序的状态转化，并不存在 “临时状态”。也就是说，它是 “无会话 (Session)” 的，每个 “网络 API 请求” 都包含了实现一个业务的完整参数。

而这，正是 REST 原则所强调的。

这也是我们把服务端程序看作是 Model 层的原因。如果存在会话 (Session)，这就意味着服务端也需要实现 Controllers，这样就太糟糕了。

REST 原则第二个强调的点，是统一的表现规范，也就是 Representational 一词传递的意思。它认为，所有网络 API 请求都应该统一抽象为对某种资源 URI 的 GET、PUT、POST、DELETE 操作。

由于 RESTful API 简单明了，易于理解和实施，今天已经基本成为事实上的网络 API 的定义规范。

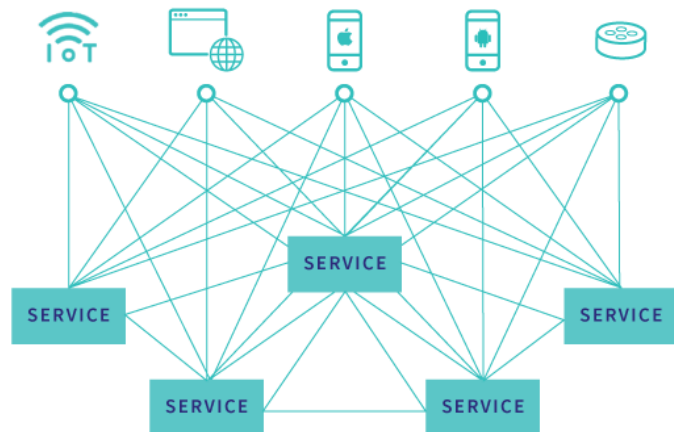
当然，RESTful API 显然并不是唯一选择。比如，基于 XML 的有 SOAP（简易对象访问协议）、WSDL（Web 服务描述语言）等。

还有一些人会觉得基于文本协议效率不够好，所以用二进制的协议。比如，Facebook 早年搞了个 thrift，不过 Facebook 自己应该不怎么用了。而 Google 也搞了个 protobuf 协议，并且基于 protobuf 搞了一个 grpc 框架。

还有一个选择是 GraphQL，它推崇企业在有多个业务的时候，不要发布很多套 RESTful API，而是基于一个统一的数据图，并通过 GraphQL 协议暴露给开发者。

BEFORE

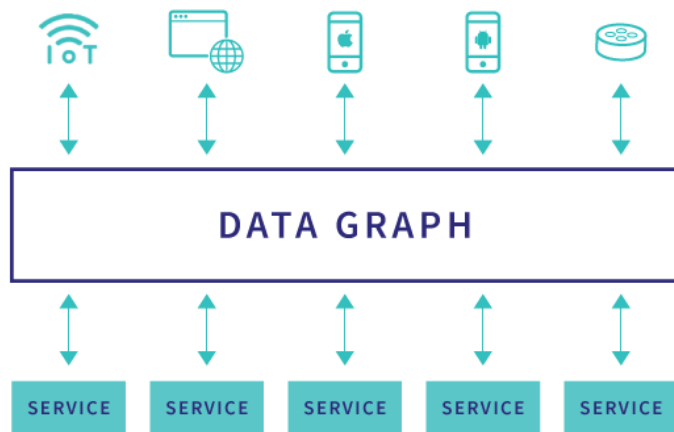
1ST AND 3RD PARTY APPS



APIS, MICROSERVICES AND DATABASES

AFTER

1ST AND 3RD PARTY APPS



APIS, MICROSERVICES AND DATABASES

目前来看，GraphQL 理念虽然先进，但是概念复杂，并不易于掌握，现在仍然处于不温不火状态。知乎甚至有一帖讨论 [GraphQL 为何没有火起来?](#)

这么多选择，应该怎么选？

我的答案大家已经知道了，我个人还是倾向于 RESTful API。虽然 GraphQL 值得关注，但是目前来看，它的投入产出比还远没有达到让人放弃简洁的 RESTful API 的地步。

至于二进制协议，虽然理论上效率更高，但是考虑到 HTTP 协议的江湖地位，各路豪杰纷纷贡献自己的智慧，提供支撑工具和效率优化，它实际的效率并不低。

只有 HTTP 协议，才有被广泛采纳的专门的应用层网关，比如 nginx 和 apache。这一点千万不要忘记。

就拿 Google 的 grpc 来说，它其实也是基于 HTTP 协议的，只不过它更推荐 HTTP 2.0，因为效率已经经过高度的优化。所以虽然 protobuf 是二进制的，但它取代的不是 HTTP 协议，而是 json、xml 或 Web 表单 (form)。

这可能也是 protobuf 还很活跃，而 thrift 已经半死不活的原因。凡是想对 HTTP 协议取而代之的，都会挂掉。

一旦确定我们要用 RESTful API，还是用 protobuf，剩下的就是如何定义具体的业务 API 了。这块是具体的领域相关内容，这里先略过。

授权 (Authorization)

确定好我们要选择什么样的网络协议，我们第二个要考虑的是授权 (Authorization)。

当前，主流的授权方式有两种：一种是基于 Token，一种是基于 AK/SK。这两种授权方式的场景非常不同。

基于 AK/SK 的授权，多数发生在面向企业用户提供 API，也就是说提供的是一个 To B 的云服务。如果大家经常使用各类云计算服务，对 AK/SK 这类授权应该并不陌生。

AK/SK 授权的背后是数字签名。

我们强调一下，AK/SK 并不是公私钥。实际上 AK 是密钥提示 (keyHint)，SK 是数字签名的密钥 (key)。

关于数字签名的原理，你可以回顾一下 [“16 | 安全管理：数字世界的守护”](#) 这一讲中的内容。

基于 Token 的授权，多数发生在面向终端用户的场景，也就是我要做一个 To C 的应用。

当前推荐的 Token 授权标准是 OAuth 2.0，它得到了广泛的支持，大家如果有在使用各类 C 端应用程序的开放接口，会发现他们往往都是基于 OAuth 2.0 的（有的还会同时支持 OAuth 1.x 版本）。

OAuth 2.0 的优势是对外提供 Open API，而不仅仅局限于自己的 App 用。OAuth 2.0 提供了一个很好的方式，能够让我们的客户不用向第三方应用去暴露自己的用户隐私（比如用户名和密码）的前提下，调用 API 来使用我们的服务。

所以总体来说，授权这块的选择是相对简单的。我们更多要考虑的，反而是如何构建业务无关的用户帐号体系和授权系统。它们隶属于通用的帐号与授权子系统，可以做到与业务无关。

后面在本章的实战案例中，我们会对这块内容进一步展开。

RPC 框架

明确了授权机制，确定了业务 API，那么下一步就是怎么实现的问题了。

如果业务 API 选择了基于 protobuf，那么 grpc 框架是个不错的选择。

对于 RESTful API，七牛云对外开源了一套非常精简的 restrpc 服务器框架，其 Github 主页为：

<https://github.com/qiniu/http>

这个 restrpc 框架主要的特点有：

URL 路由（URL Route）。支持用手工写 URL 路由表，也支持由 restrpc 框架自动实现路由。

参数的解析。可以支持 json、Web 表单（form）等格式的解释。对于其他格式对数据，可以由用户自己来解释。

返回值的序列化。默认序列化为 json，如果需要，用户也可自己做序列化。

授权（Authorization）。以开放框架的方式实现授权机制，以便用户可以选择自己的授权方式。

适度的开放机制。我们主要为了实现开放的授权机制而开放，但这个开放机制可以用来做各类扩展，而不只是局限于授权。

这里我们给了一个 restrpc 框架的使用样例：

<examples/authrestrpc>

为了简化，这个样例用的是一个 mock 的授权机制。这种 mock 授权非常适合用来做业务系统的单元测试。

这个样例我们采用由 restrpc 框架自动实现路由的方式。这样可以减少一些代码量，但是对路由 API 对应的实现方法的名字有要求，看起来不是那么美观。如果不喜欢可以采用手工路由方式。具体怎么做，后面我们的实战案例会有体现。

单元测试

另外，这个样例我们的单元测试采用了七牛开源的 httptest 框架。其 Github 主页为：

<https://github.com/qiniu/httptest>

这个 httptest 框架，最核心的逻辑是如何在不用写业务 API 的 Client SDK 的情况下，能够保持业务友好的方式来写测试案例。

它不只可以做单元测试，也可以做集成测试。

你可以通过下面这个演讲稿来了解它的核心思想：

<http://open.qiniudn.com/qiniutest.pdf>

这个 httptest 框架是非常通用的，所以它没有内建任何公司特有的授权机制。在七牛，我们会基于更贴近七牛自身业务的 qiniutest 进行测试。qiniutest 工具只是在 httptest 基础上作了少量的扩展，其 Github 主页为：

<https://github.com/qiniu/qiniutest>

你可以依葫芦画瓢，实现一个适合你们公司的授权机制下的 httptest 工具。

在本章的实战案例中，我们也会让大家看到如何基于 httptest 来进行业务的单元测试。

结语

我们总结一下今天的内容。

服务端业务架构，主要是怎么做一个多租户的 Model 层。Model 层本身最重要的是自然体现业务逻辑，它和具体的行业的领域问题相关，对此我们无法进一步展开。

但服务端程序还是有它很鲜明的特点。

今天我们重点讨论了服务端业务架构相关的通用问题。包括：网络协议、授权、RPC 框架、单元测试等等。

当然其实还有一个问题，就是选什么样的存储中间件。它和具体的业务特征更为相关，这一点在后面我们实战案例中再做探讨。

如果你对今天的内容有什么思考与解读，欢迎给我留言，我们一起讨论。我们服务端开发相关的内容就暂时告一段落，下一讲开始我们进入实战。结束实战后，我们会结合实战对服务端开发的架构做一个总结。然后我们进入服务端的另一半：如何做好服务的运维，甚至也会涉及少量的运营相关的话题。


如果你觉得有所收获，也欢迎把文章分享给你的朋友。感谢你的收听，我们下期再见。

许式伟的架构课

从源头出发, 带你重新理解架构设计

许式伟
七牛云 CEO



新版升级: 点击「 请朋友读」, 20位好友免费读, 邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有, 未经许可不得传播售卖。页面已增加防盗追踪, 如有侵权极客邦将依法追究其法律责任。

上一篇 39 | 存储与缓存

下一篇 41 | 实战 (一): “画图” 程序后端实战

精选留言 (7)

 写留言



leslie

2019-09-10

许老师对存储和架构的理解确实不一样: 开始学时还有些诧异为何只有此门功课的课程之间间隔相对较大; 一路学来一路梳理, 每次学完都会觉得有些疏漏需要自己去补; 最终发现其实讲的太快根本无法理解所学的知识。

最初的学习目的是因为当下的数据系统和中间件有问题: 之前自己在其它电商和金融业的经验无法移植, 虽然系统放在云上, 可是还是有性能问题, 刚好老师的课开课, 就...
展开

作者回复: 这本书没法承担太多职能, 我们的讨论重心始终会在架构, 包括基础架构和业务架构。从架构角度来说, 系统的瓶颈永远是存储中间件。但是从数据分析角度来说, CPU的确是我们的智能能够走得有多远的瓶颈。



Aaron Cheung

2019-09-10

楼上大佬留言很有分量 我就啥也不说了 老师教师节快乐



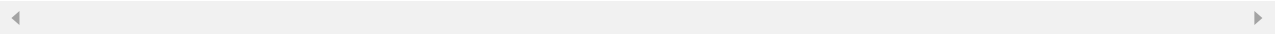
Sam

2019-09-15

老师，麻烦问下Session-base model作为web api，是否他是融合多个后端业务系统的RESTful api后，提供给前端使用的web api接口吗？

展开 ▾

作者回复: 是的



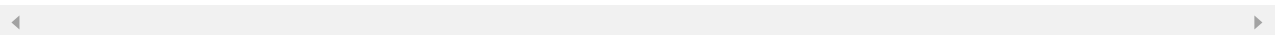
八哥

2019-09-12

还有很多前端工程师不知道在同域下，浏览器默认携带cookie，很多工程师理不清单点登录。认证这块可以多讲讲

展开 ▾

作者回复: 实战有一篇专门是认证篇

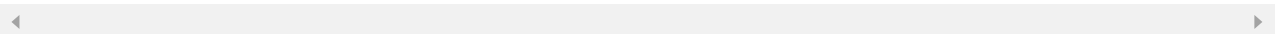


睡觉中

2019-09-11

如果一个服务需要c端用户为登录状态进行操作，例如淘宝或者京东，那么这个服务必然要维护session，所以这个服务就不符合Rest规则了吗？Rest仅仅是数据接口？

作者回复: 登录的session一般通过cookie来记录session id，在服务端记录session信息，这的确不那么无状态。不过authorization很特殊，通常不作为rest与否的判断依据。



Geek_88604f

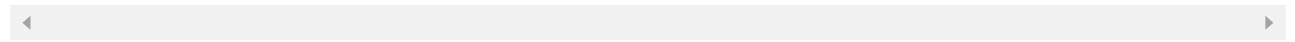
2019-09-11

但是从状态转化角度来说，桌面程序和服务端程序很不一样。桌面程序的状态转化往往存在中间的“临时状态”，这其实也是 Controller 层的价值所在。关于这段描述有如下疑问，望解答：

一，理论上应该存在四种选择：桌面程序有临时状态、桌面程序无临时状态、服务端程序有临时状态、服务端程序无临时状态，而实际上只有桌面程序有临时状态、服务端程...

展开 ▾

作者回复：内部有临时状态那是内部的事，从交互接口来说没有临时状态。另外服务端内部临时状态一般通过事务来完成原子化，以避免出现不一致的状态。



Caffeine

2019-09-10

现在物联网这么火 不知道有没有针对嵌入式软件或单片机软件开发的架构设计呢

作者回复：和桌面开发差不多，由于交互少，整个体系只会比桌面简单。

