# 体系结构

分布式系统是由许多复杂的软件组成的。正确的将系统组织好是掌握这种复杂性的关键。有不同的方法可以识别出分布式系统的组织结构，但更显而易见的是，我们可以从软件的逻辑组成和具体硬件实现上来进行区分。

说起分布式系统的组成更多谈起的是软件组成。这些软件架构告知我们不同的软件是如何组织的和如何联系的。这一节我们将先来观察分布式系统的组织架构风格是什么样的。

分布式系统的一个重要目标是通过中间层来分离应用和底层平台。使用中间层是一个重要的架构决定，并且它的主要目标是提供分布式系统的透明程度。

然而，我们需要在分布式系统的透明性，和为了这种透明性需要进行对中间层的技术修改之间做权衡。我们将讨论这些应用在中间层组织结构的技术。

分布式系统的实现需要我们再真机上实例化软件。这里我们有很多种选择。最终的软件架构和系统架构是相关的。在这一节中，我们将深入讨论传统的中心式架构，即大部分的软件都在一个服务器上部署，同时远程客户可以通过简单的通信手段来访问这个服务器。另外，我们也将讨论去中心化的peer-to-peer架构，即所有节点都担当同样的角色。许多现实的分布式系统都采用了混合式的策略，即中心和非中心的混合。

我们将在这节结尾讨论两种应用广泛的分布式系统，NFS文件分享系统和WEB系统。

## 2.1 架构风格

关于分布式架构，我们首先讨论软件成分的逻辑组织，也即软件架构。软件架构的研究已经成熟了，并且通常认为一个好的架构设计是一个软件系统发展成功的关键。

对我们来讲，软件架构风格的讨论更为重要。这种风格是由各个组件之间的联系，组件之间的数据交换以及各个元素是如何加入并配置到这个系统中来决定的。一个组件是一个定义良好的模块化单元，并提供了在其环境中可被替代的接口。系统中的每个组件都是可被替换的。这是因为关闭系统并不是一个好的选择。最好的情况是，只有部分设备暂时出现了故障。只有当被替代的组件的接口已经不能访问了，这种替代才可以发生。一个比较难以理解的概念是连接器，它通常被描述为组件之间的沟通，协调和合作的中间机制。例如，有的连接器是由远程调用，消息传递和流数据组成的。换句话说，一个连接器可以让连接和数据在组件之间进行流式传输。

使用组件和连接器，我们会遇到多种不同的配置，这都将归类于架构风格。不同的风格都已经指出了，下面列几种对分布式系统最为重要的：

Layered architectures

Object-based architectures

Resource-centered architectures

Event-based architectures

在接下来，我们将以此讲解这些架构。在现实的分布式架构中，以上这些架构是被联合使用的。对系统进行细分的方法，已经成为了一个普遍的使用多种架构风格的原则。

**Layered architectures**

层次架构的风格很简单：高层的组件可以对低层的组件进行访问，并等待一个回应。只有在极其例外的情况下才会要低层的去调用高层的情况。这三种情况如下图所示。

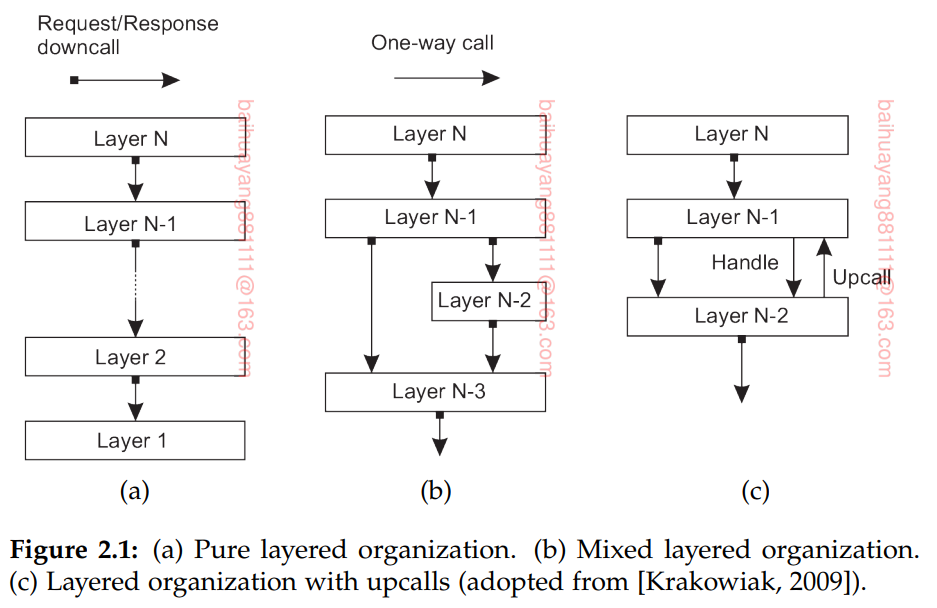


图2.1(a)展示了一个标准的组织结构，即只存在向下调用的情况。这种组织结构通常部署网络通信的情况下。

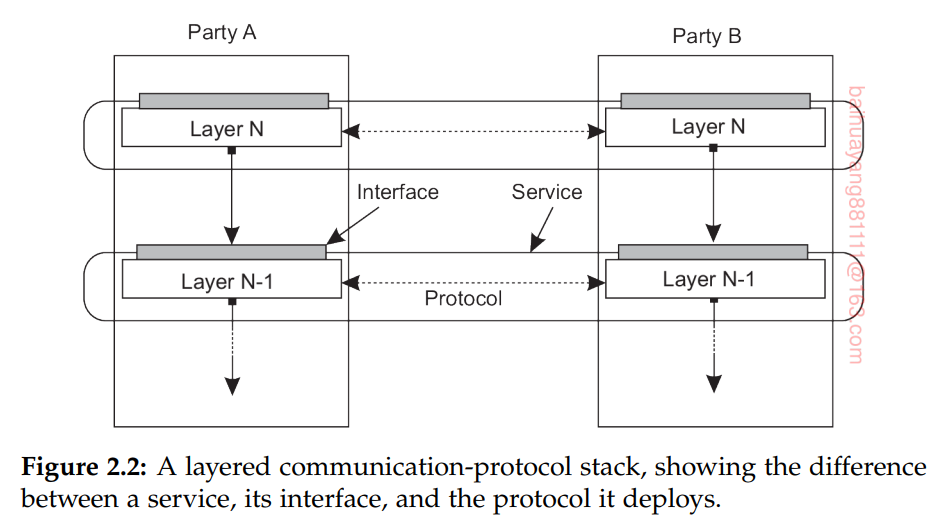
在很多情况下，我们也会遇到如图2.1(b)的情况。比如应用A需要调用操作系统底层接口，并且也需要调用实现了操作系统底层接口的另外一个接口，比如math接口，这样就如2b图所示的结构了。

最后是2.1c这种情况。有些情况下，会出现下层调用上层接口的情况。比如回调，应用发生时，传递给操作系统一个引用，当应用结束时，要求操作系统执行这个引用。

**层级通信协议**

一个比较有名的层级通信架构是通信协议栈。这里讲主要描述它的轮廓，细节放在4.1节讨论。

在通信协议栈里，每一层通过对通信服务的实现，可以将数据从一个地方传输到另一个地方。为此，每一次提供一个接口供其他层进行调用。原则上，这些接口的实现应当完全隐藏。另一个重要的通信协议的功能是，为各种信息交换提供遵循的规则。我们最好能区别出服务，接口和协议这三种的区别。如图2.2所示



为了更清楚的区分，我们考虑一个许多通信系统都提供的一个可靠地，面向连接的服务。在这种情况下，发送数据之前两边需要先建立连接。可靠性保证了数据一定能从一段传送到另一端。这些服务通常能保证了他们发送数据的和接收数据的顺序一致性。

这种网络服务有一种叫做TCP的协议实现了。这种协议保证了数据交换于一个连接上，它能够保存数据的传输顺序，并发现和纠正在数据传输中的错误。这种服务被封装成一种简单的可使用的编程接口。事实上，在不同的操作系统上有不同的可使用的接口。类似这种协议和接口的实现还有很多。

**应用层**

接下来讨论应用逻辑层。考虑到很大一部分的分布式应用都会支持数据库的访问，人们在层次架构上又区分了三种逻辑层，分别是

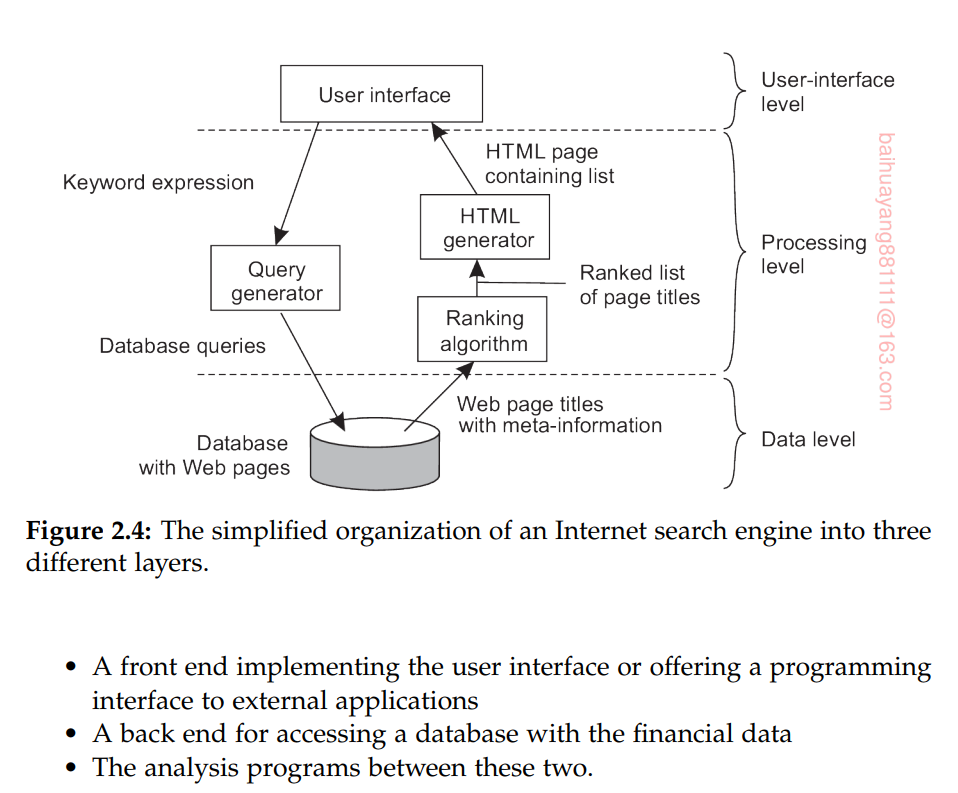
应用接口层

执行层

数据层

通过这么划分，我们可以了解应用程序通常可以有三种不同的方面组成：一方面处理与外部用户或系统的交互，一部分操作数据库和文件系统，中间那部分包括应用的核心方法。中间这部分就是处于执行层。不同于用户接口和数据库层，中间层的实现并没有太多类似的东西。我们将会用例子来说明。

第一个例子，网络搜索引擎。不考虑图片，banners 等一系列花哨复杂的东西，搜索引擎的用户接口非常简单：一个用户键入一个关键词，网页就返回一系列标题。后端系统是一个巨大的已经预处理的并建好索引的网页数据库。这个搜索引擎的核心是将用户的关键词传输到多个数据库中用于查询。然后将结果进行排名，并输出到html网页中。信息检索的部分位于执行层，如图2.4



金融数据分许则需要统计学和人工智能中的复杂方法和技术。在一些情况下，金融分析支持系统需要在高性能的机器上执行，用来满足用户需要的吞吐量和响应。

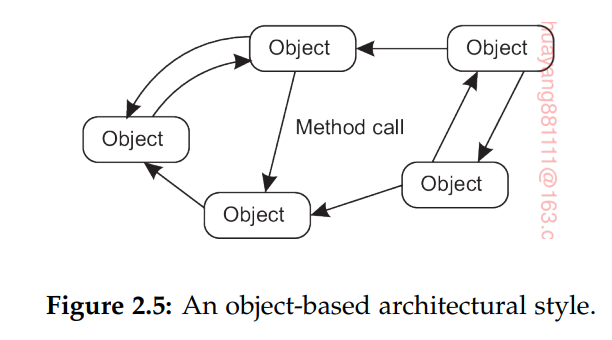
最后一个例子是，考虑一个台式机，由文字处理器，电子表格应用，通信工具组成。这样的办公套装通过支持文件管理和用户主目录操作的公共用户接口来集成。（在办公环境下，用户主目录通常在一个远程机器上。）在这个例子中，处理层会由相对大的程序集合组成，并且每一个都有简单的处理能力。

数据层则存储着应用需要访问和操作的数据。数据层一个重要的性质是，里面的数据都是持久化的，即便没有应用在运行，数据也是一样存储起来。在最简单的行驶中，数据层包括文件系统，但它也通常使用一个完整的数据库。

除了存储数据，数据层通常需要保持各个应用之间的数据一致性。当数据库被使用时，像一些数据元数据如表描述，建约束和应用特殊的元数据都被储存起来。例如，在银行中，我们想要生成一个用户的信用卡债务到达一个特定值的时间的备注。这种类型的信息可以通过数据库的触发器在特定是时候激活一个句柄来实现。

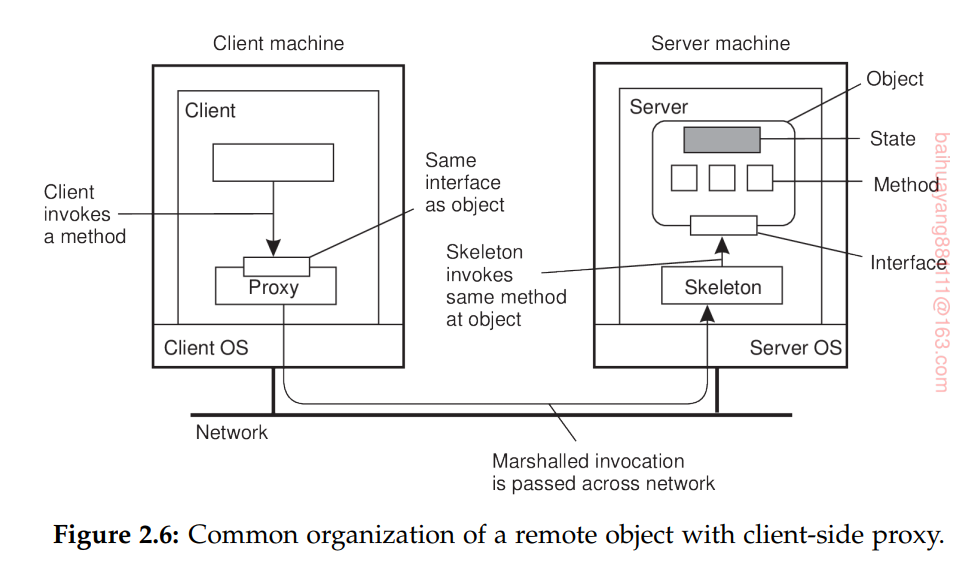
**Object-based and service-oriented architectures （面向对象架构）**

一个更加松散的组织结构是 object-based architectures，如下图所示。本质上，每一个object对应于一个组件，并且每一个组件通过过程调用机制相互连接。，在分布式系统中，一个过程调用通常发生在网络中，所以过程调用通常不是发生在本机器上。



Object-based 架构是很吸引人的，因为它用一种很自然的方式对数据进行了包装，并且这些包装的数据可以很容易的操作。通过Object提供的接口，它的实现细节会被隐藏，原则上我们可以认为 object可以完全独立于系统而存在。或者说，如果接口被清晰的定义然后被废弃使得其他应用无法调用，那么这个object就可以被替换。

这种接口和实现分离的方式允许我们将接口定义在一台机器上，而object依赖在另一台机器上（自我理解，就是类似RPC，一台机器负责所有接口定义，其余机器实现）。这种组织结构如下图2.6所示。



当一个哭护短和一个分布式object绑定时，object接口的实现称之为代理，会被加载到客户端的地址空间。这个代理和RPC系统的客户端stub是一个意思。它的唯一作用是封装方法调用和解封返回消息。实际的object驻留在服务器上，它和客户端机器提供同样的接口。首先进入服务器stub的是调用请求，stub将请求解封并执行方法调用。服务器stub同样负责封装回应并将相应数据返回给客户端的代理。

服务端的stub同城被称为股价，因为它提供了允许服务器中间件访问的裸方法。实际上，他常常包含不完整的代码，这些代码以特定的语言的形式出现，需要开发人员进一步特殊化。

大部分分布式系统的object的一个特性是，他们的状态不是分布式的：他们都取决于一个单一的机器。只有实现了接口的object才能够对其他机器使用。这种objects通常称作remote objects。通常的分布式object中，他们的状态都是在物理上分布在不同机器上的，但这种分布式也隐藏在object的接口中。

可以认为object为基础的架构是将服务封装到独立的单元中，在这里封装是关键词：服务作为一个整体被时限为一个自包装的实体，虽然它也会使用其他的服务。将各种各样服务分离使得他们可以独立操作，我们可以使用 **service-oriented architectures**（面向服务架构） 即，SOAs

在面向服务的架构中，一个分布式应用和系统本质上是有许多不同的服务组成的。并非所有的服务都属于同一个管理组织。我们讨论云计算的时候已经遇到了这个现象：很有可能一个结构使用云端提供的存储服务来运行业务应用。这些存储服务逻辑上被包装成一个单一的单元，并对用户开放接口。