型云计算网站。

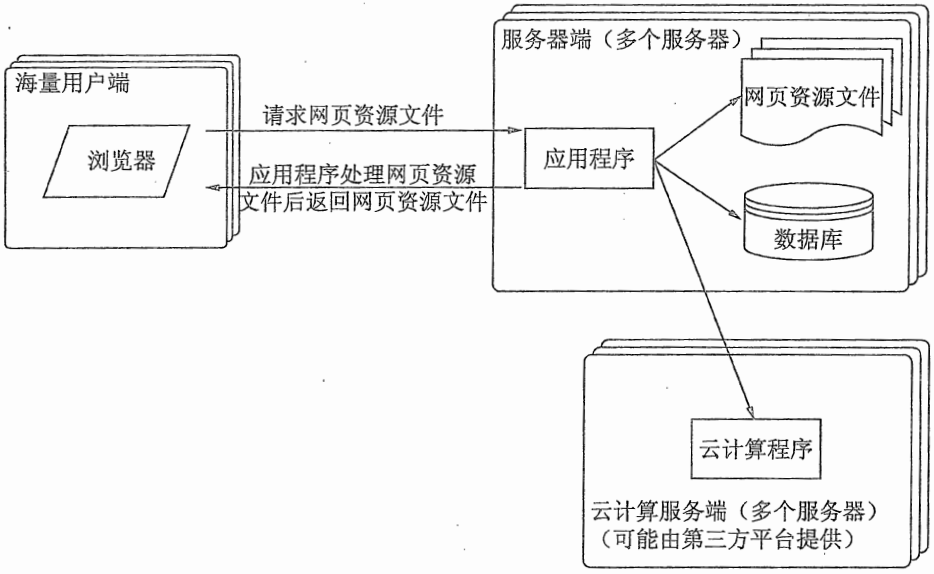

图 1.4　大型云计算网站系统的工作原理

1.1.5　大型网站的未来

2009 年，谷歌发布了一款网络笔记本计算机 Chromebook，这款笔记本计算机号称“完全在线”（可以认为笔记本中只有一个浏览器），用户所需的计算和存储都由网络服务提供。近几年，任天堂和索尼都推出了云游戏服务，游戏的画面渲染交给云计算服务器完成，而游戏机只负责显示画面。诸如此类，还有很多应用场景正在被云计算改变。

目前，上述网络应用产品都不算特别完善，这主要是网络的原因造成的。随着 5G 网络的建设和发展，网络问题会得到解决，到时候一定会有更多的云计算场景出现，也会有更多的网络应用出现。

但是，即使出现了足够多的网络应用，以现在各自运营、各自宣传的现状来说，离实现完全云计算化的未来还是有距离的，这需要一个网络应用市场来做统一的管理。网络应用市场除了可以整合零散的网络应用外，还可以提供权威的安全认证，消除用户对陌生网络应用私隐安全方面的顾虑。

综上，笔者认为，大型网站的未来应该是网络应用市场。到时候，出现更规范化约束的同时，也一定会有更成熟和更统一的技术出现。

1.2　大型网站架构的发展

1.1 节介绍了大型网站的业务需求和大致的工作原理，但是不能简单地理解为只要增加服务器，就能把一个网站变成一个能应对大量用户的网站。通过增加服务器来达到支持更多的用户是大型网站架构的目的。本节简要介绍大型网站架构的发展，并介绍大型网站架构如何有效地增加服务器。本节介绍的技术点只要了解即可，后续章节会有更详细的说明。

大型网站系统的内部是复杂的，一般是多种网站架构的混合（包括静态网站、动态网站和 B/S 架构网站等）。本节介绍的内容会忽略一些细节，另外，除了 1.2.1 小节所讲的动态网页以外，其他都是以 B/S 架构网站作为基础的。

说明：软件架构是有关软件整体结构与组件的抽象描述，是一个软件的基本思想。简单地说，架构就是以宏观的角度思考软件如何解决问题。

1.2.1　动态网页时代

在 1.1.2 小节动态网站的出现中提到了动态网站的工作原理，服务器在接到浏览器的请求后，应用程序处理网页资源文件后才返回文件。在这里进一步说明一下，经过处理后，返回的只是 HTML 格式的文件，如 JSP 和 PHP 文件等。对于不需要处理的资源文件，如 JavaScript 脚本文件、CSS 样式文件、图片文件、视频文件等，服务器在接到请求后，会直接返回。当然，动态网站除了可以操作数据库，同样也可以调度云计算服务。动态网站的技术架构如图 1.5 所示。

1.2.2　B/S 架构网站的崛起

不可避免的是，动态网页需要在每一次请求网页时都处理一遍所有的 HTML 格式的文件（如 JSP 和 PHP 文件）。这样无疑会有不少的资源浪费。如果需要更新网页中的内容，就必须重新加载整个网页，使用户体验不是那么友好，特别是在网速不好的情况下。

因此，网站更好的方式应该是类似于 C/S 架构模式（客户端-服务器模式，如桌面软件等），服务器只需要处理客户端关心的数据即可，无须做多余的处理。出于这样的考虑，B/S 架构模式（浏览器-服务器模式）出现了，服务器在接受网页请求的时候，还是像静态网站一样不经处理地直接返回 HTML 文件。浏览器需要更新部分网页数据的时候，只需要通过 JavaScript 脚本向服务器请求其关心的数据，然后对网页的某部分进行更新即可。B/S 架构的网站也常常称为伪静态网站。大型网站架构虽然内部复杂，可能会包含动态网

站和静态网站，但一般还是以 B/S 架构网站为主。

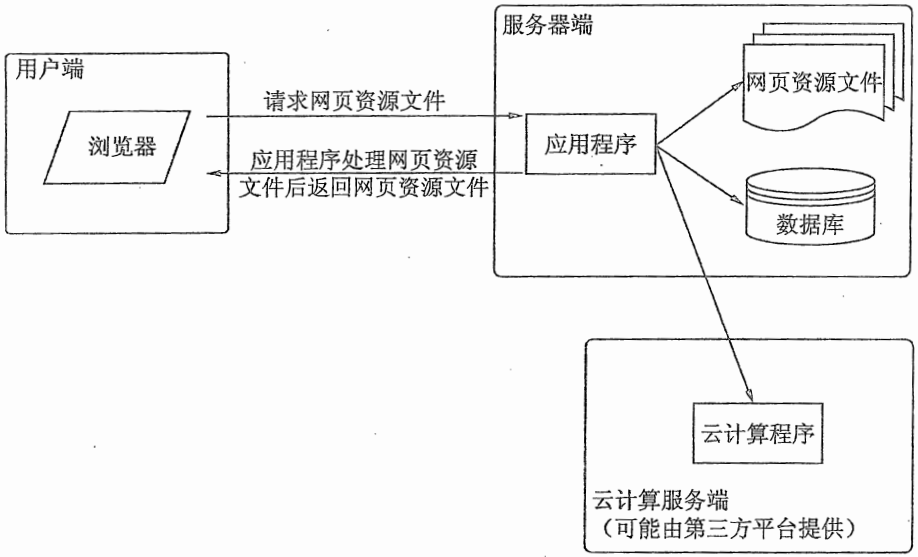

图 1.5　动态网站的技术架构

随着 B/S 架构的应用，浏览器运行的网页和服务器处理请求的接口也分别被称为前端和后端。随着 Ajax 技术的出现，进一步简化了前端与后端的请求方式，B/S 架构也逐渐崛起。B/S 网站技术架构如图 1.6 所示。

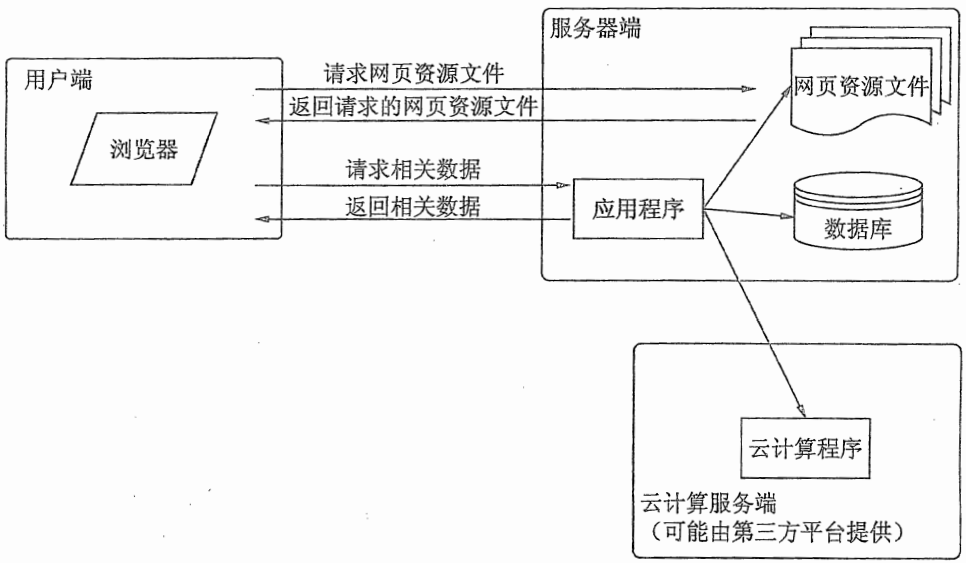

图 1.6　B/S 网站技术架构

说明：Ajax 即 Asynchronous JavaScript and XML 的缩写，意为异步 JavaScript 与 XML。Ajax 可以仅向服务器发送并取回必需的数据，并在浏览器中处理来自服务器的数据。

1.2.3　CDN 加速网站响应

虽然说网页不需要传统意义上的软件安装就可以使用，但其实每次打开网页时，都需要从服务器端下载网页文件（浏览器会有部分文件缓存，但大多数情况下都是重新下载）。而这些网页文件（如 HTML 超文本文件、JavaScript 脚本文件、CSS 样式文件、图片文件和视频文件等）大多数都是不需要服务器处理的。如果每次都从服务器返回网页文件，显然在大量用户访问的情况下，服务器的压力是很大的。加上复杂的网络环境，不同地区的用户访问网站时速度差别极大。

因此，针对这些服务器不需要做处理的文件，在面对大量用户的访问时，“怎么能让用户迅速打开网站”是一个很重要的问题。

为了解决这个问题，我们可以增加足够多的服务器，并且把服务器根据用户的地区分布合理分配。这确实是解决问题的思路，但如果完全由网站服务商提供这些服务器的话，成本是很高的。不过，已经有很多第三方供应商（如阿里云、腾讯云等）提供这些服务器，我们只需要在第三方平台上配置网站域名和缓存策略，就可以解决问题。配置完成后，第三方服务器会自动缓存网页文件，用户便可以从就近的服务器上获取网页文件，而不是每个请求都被积压到网站服务器上。这样的服务被称为 CDN（Content Delivery Network，内容分发网络）加速，这种网站的技术架构如图 1.7 所示。

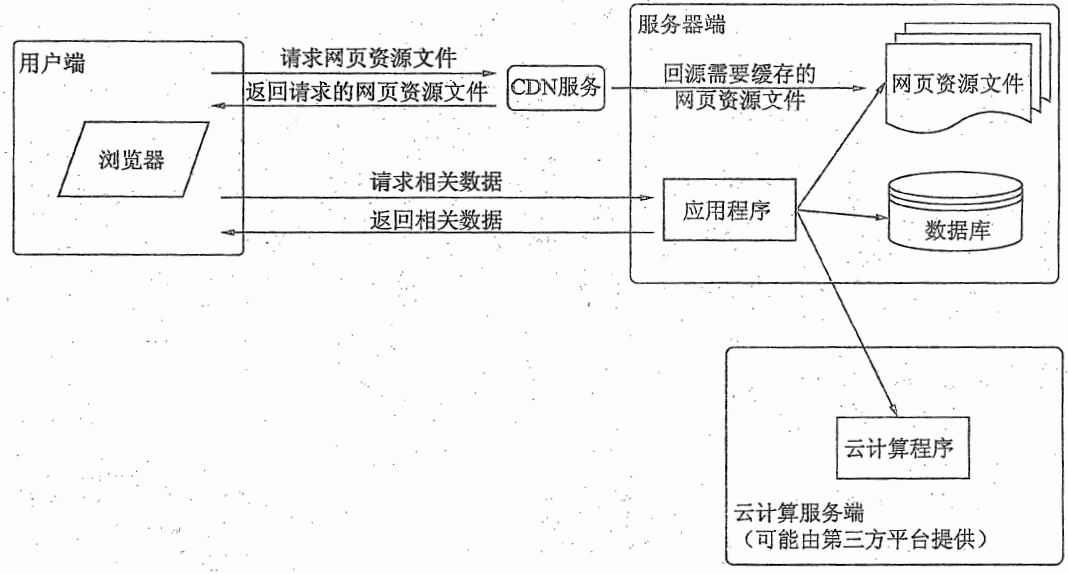

图 1.7　CDN 加速后的网站技术架构

说明：CDN 可以把静态文件缓存在多个服务器上，使用户就近获取所需内容，从而降低源服务器的网络拥塞，提高用户访问的响应速度。

1.2.4　应用和数据分离

网站服务器中有应用程序、资源文件、数据库和云计算服务，它们对计算机的物理性能要求都不一样，如资源文件需要更好的硬盘性能，数据库除了硬盘性能外还要求更好的 CPU 性能。如果把应用程序、资源文件、数据库和云计算服务都放在一台服务器上的话，是不能有针对性地对网站进行扩展的。

因此我们需要分离应用和数据，把应用程序、资源文件、数据库和云计算服务分别部署到专门的服务器上。当用户量越来越大时，就可以有针对性地增加存在性能瓶颈的服务器。应用和数据分离的网站技术架构如图 1.8 所示。

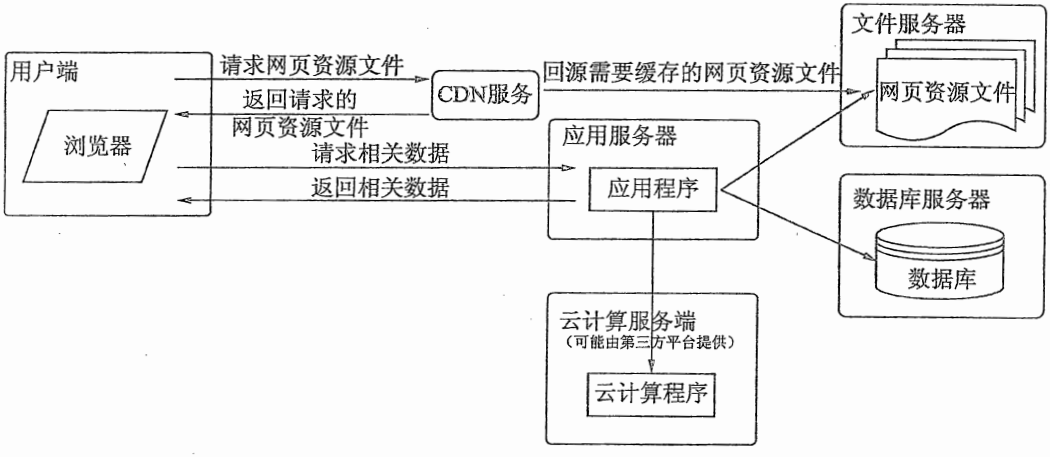

图 1.8　应用和数据分离的网站技术架构

1.2.5　非关系型数据库和关系型数据库并存

在一定层面上，网站是围绕数据工作的，网站的业务实际上是对数据的管理，数据库一般是整个网站系统的核心。因此大型网站架构对数据库的运用是非常重要的。数据库一般指的是像 MySQL 和 Oracle 等类似于表格的关系型数据库。当然，仅仅使用关系型数据库也可以满足所有的业务需求，但是存在两个问题：第一，针对查询而言，很多查询请求都是相同的，一段时间内查询的结果都是一样的，而数据库则基本上需要每次都重新检索一次；第二，表格形式的关系型数据库由于形式的限制，应对某些业务场景是乏力的，非关系型数据库的出现，就是为了应对大规模数据集合及多种数据类型等挑战。

针对问题一，虽然关系型数据库也有自己的缓存机制以达到减少检索的目的，但是它并不能根据具体业务定制缓存策略。引用 Redis 等键值存储的非关系型数据库可以对“预期访问量很大并且更新概率较小”的数据进行缓存，这样可以大大减少关系型数据库的压力。

针对问题二，应对海量数据时，采用 HBase 等列存储非关系型数据库比较省力；应对大量不限制结构的数据时，采用 MongoDB 等文档型非关系型数据库比较省力；应对社交关系等复杂的数据时，采用 Neo4J 等图形非关系型数据库比较省力，需要注意的是，这里的图形不是图片，是图形结构的意思。

根据实际情况选择对应的非关系型数据库，非关系型数据库和关系型数据库并存，无疑是大型网站架构设计中必要的一环。非关系型数据库和关系型数据库并存的网站技术架构如图 1.9 所示。

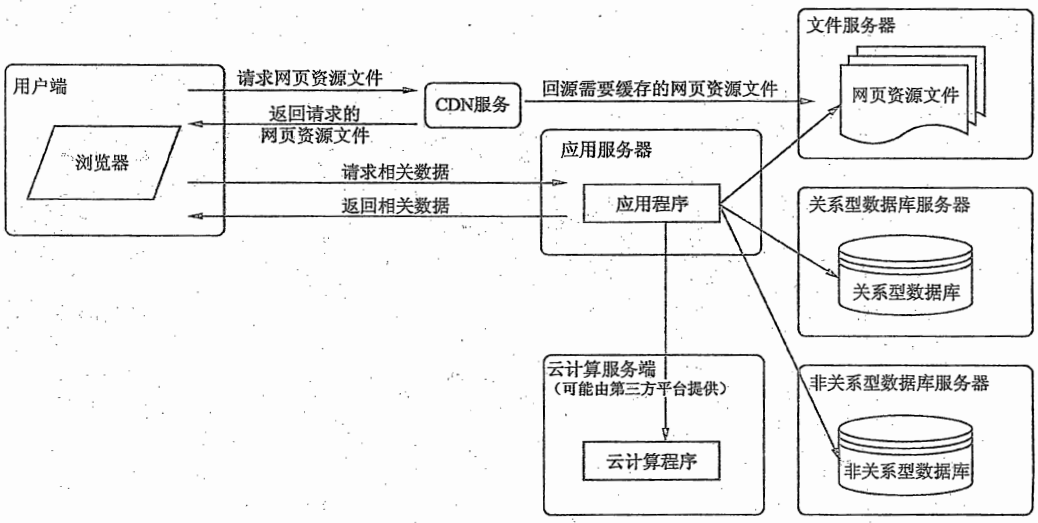

图 1.9　非关系型数据库和关系型数据库并存的网站技术架构

说明：非关系型数据库一般也被称为 NoSQL。NoSQL 的说法是相对使用 SQL 语言作为交互的关系型数据库而言的。

1.2.6　集群化

集群化实际上就是我们前面一直提到的增加服务器个数。但是，单纯地增加服务器最多是从一个网站变成多个网站，而不是让一个网站变成一个能接纳更多用户访问的网站。为了更好地增加服务器，需要增加一些软件为这些相同功能的服务器进行协调。下面根据 1.2.4 小节中提到的大型网站的四部分（应用程序、资源文件、数据库和云计算服务）分

别介绍集群化的相关技术。

- 应用程序的集群化：应用程序在 B/S 架构中，一般就是指后端接口。应用程序的集群化需要添加一个负载均衡的服务，让前端网页请求后端接口时，均衡地调度这些应用程序服务器。
- 资源文件服务器的集群化：资源文件服务器的集群化，主要是为了应对前端的下载请求。虽然 CDN 加速解决了大部分资源文件服务器的压力，但是 CDN 缓存一段时间后也会重新向资源文件服务器回源。当用户地区分布足够广的时候，这个压力也是不容忽视的。资源文件服务器的集群化同样需要添加一个负载均衡的服务。
- 数据库服务器的集群化：一般数据库都会提供集群化的部署方案，根据该方案部署即可。
- 云计算服务的集群化：如果使用的是第三方云计算服务，则集群化是第三方平台提供的；如果是自身的云计算服务器的话，则需要使用 RabbitMQ 等消息队列作为任务调度中心。更多细节可以参考第 5 章云计算服务架构的介绍。

说明：负载均衡（Load Balance）的作用是接受请求并把请求分发到多个服务器上（如 FTP 服务器和 Web 服务器等）。关于负载均衡的详细说明，请参考 6.3.13 小节中关于集群与分布式部署的介绍。

集群化的网站技术架构如图 1.10 所示。

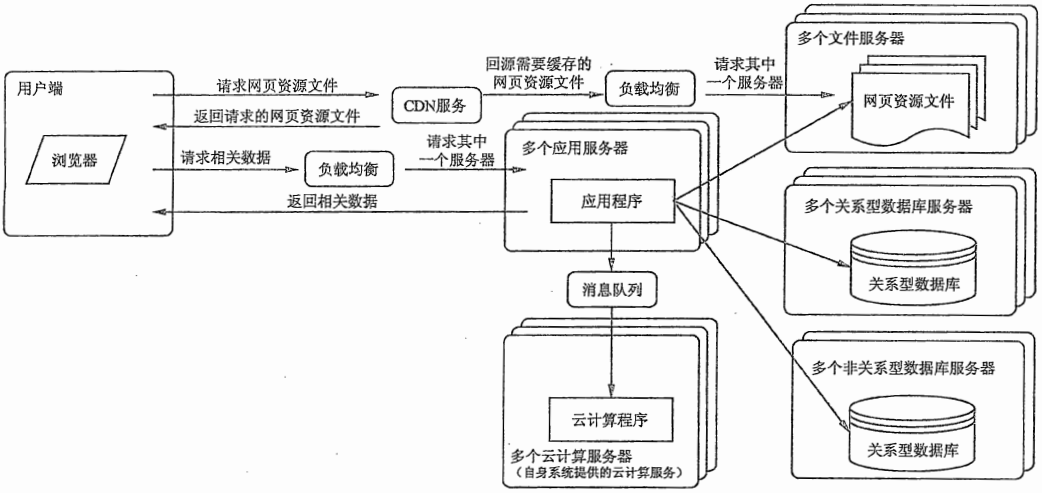

图 1.10　集群化的网站技术架构

1.2.7　分布式趋势

集群化后，大型网站系统能很好地通过增加服务器有效应对大量用户访问的压力。但

是，大型网站系统除了要应对大量的用户访问以外，还需要不断地扩展业务。而不断扩展业务功能后，应用程序部分会变得非常复杂和混乱。为了缓和这种应用程序部分复杂和混乱的情况，大型网站架构出现了分布式的趋势。

分布式的大型网站架构，简单地说就是把庞大的大型网站系统分割成多个独立的子系统和子模块，这些系统和模块通过互相协助的方式完成任务。在物理意义上，分布式的大型网站系统把这些子系统分别部署在不同的服务器上，并且能让这些应用程序协同完成任务。

例如，上传视频可以赢取积分，分布式网站系统会由视频管理子系统接收视频上传，然后由积分管理子系统增加该用户的积分。而视频管理子系统和积分管理子系统部署在不同的服务器集群中。

分布式的网站系统会越来越流行，虽然其开发成本相对较高，但是独立的子系统可以独立发布，发现问题时也可以单独测试，这为后续的运维提供了保障。另外，独立的子系统如果通用性足够的话是可以复用的，即该子系统可以为其他网站服务，缩减新网站系统的开发成本。分布式的网站技术架构如图 1.11 所示。

说明：分布式大型网站系统，不单单指的是把应用程序分割成相互协作的独立子系统，还会拆分出多组数据库服务器和云计算服务器等。为了简化说明，这里只介绍了应用程序部分的分布式协作。另外，前面提到的应用和数据分离的网站系统其实也是一种分布式架构。

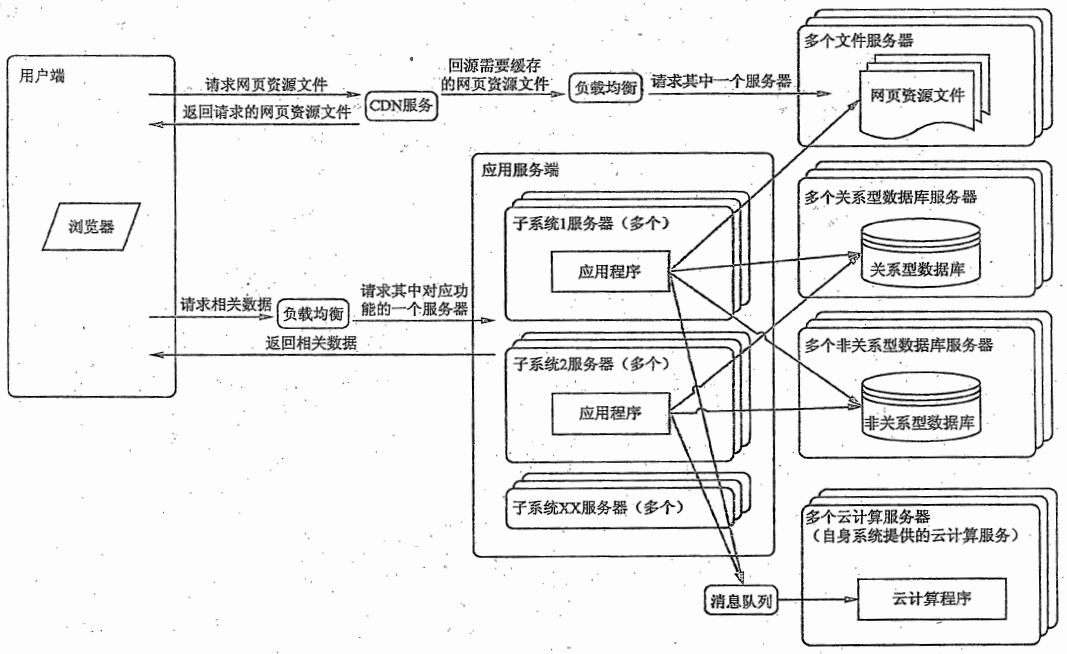

图 1.11　分布式的网站技术架构

1.2.8　微服务

微服务也是近年来比较热门的话题。2014 年，Martin Fowler 与 James Lewis 共同提出了微服务的概念，定义了微服务是由单一应用程序构成的小服务，拥有轻量化的处理程序。多个微服务共同提供网站系统所需要的功能。

微服务是分布式网站系统的进一步优化。简单地说，微服务希望一个大型网站可以通过很多个完全独立的小服务组成。这样可以更清晰地运维网站系统，更快速地进行开发，更精准地定位问题。

不过，微服务也是存在争议的，在笔者经历过的两个采用微服务的项目中，最后的结果都不太好。除了微服务框架的中间件增加了网站结构的复杂性以外，更关键的是，微服务的颗粒度需要项目自己定义，这个颗粒度的权衡很难拿捏。因此，大多数采用微服务的项目其结果都不太理想，应用程序部分变得十分臃肿，微服务间的调用也十分混乱。

笔者认为，微服务的概念会给大型网站架构带来新的思考，但目前的状态下，盲目地使用微服务框架，在大多数情况下只会弄巧成拙。微服务的网站技术架构如图 1.12 所示。

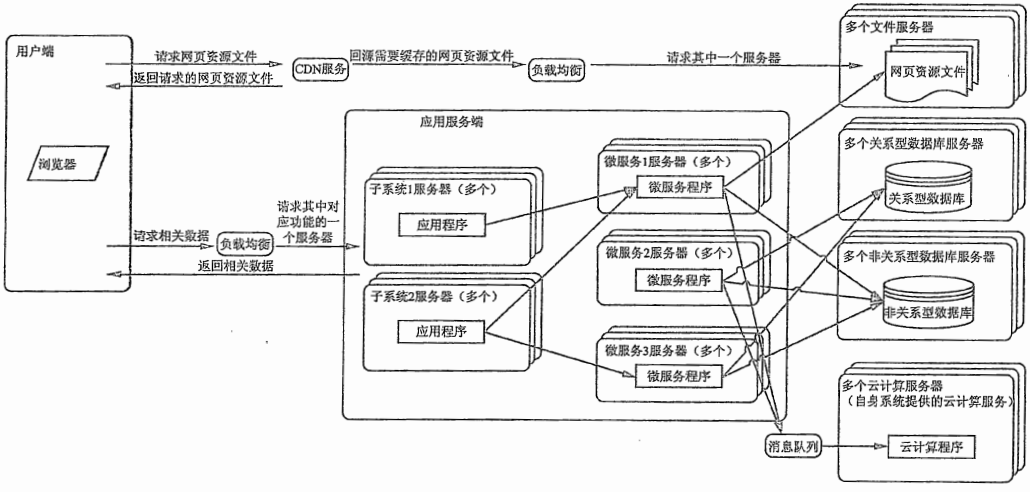

图 1.12　微服务的网站技术架构

1.2.9　大型网站架构的未来

目前大型网站架构的各种技术都是相对成熟的，第三方云服务平台（如腾讯云和阿里云）也提供了各式各样的基础服务和云计算服务。不过，如果要从零开始构造一个大型网站还是很困难的。

这种构造的难度恰恰证明大型网站的架构还没有完全成熟。微服务概念的提出，尽管其实际的应用效果不尽人意，但它也确实给原本以为已经成熟的大型网站架构带来了新的思考。更加成熟的大型网站架构应该是由很多独立的模块合并起来的，就好像一个庞大的机械设备是由很多现成的零件组装成的一样。

大型网站架构还在发展，更加标准化的架构将会出现。到时候，大型网站架构将变成一个标准化的生态环境，开发大型网站时将不需要考虑这么多的技术点，网站架构主要考虑组合哪些已有的子系统模块。

1.3 小　结

在了解大型网站的业务演变时，需要明白多种网站类型的应用场景；在了解大型网站架构的发展时，需要知道大型网站架构可能遇到的问题。通过了解这些发展历史，相信读者能对大型网站架构有大概的了解。

当然，仅仅从宏观上了解是不够的，毕竟我们忽略了很多复杂的细节。不过，我们可以站在巨人的肩膀上，前人发现了很多问题，解决了很多问题，也产生了很多种技术，我们可以学习别人的经验和前人的技术来解决遇到的问题。有些技术等需要用的时候再仔细学习也不晚。

随着发展，大型网站架构变得越来越复杂，想要在技术选型和架构决策时直击要害，确实需要一些经验。我们在虚心学习前辈经验的同时，也不要盲目跟风，而需要根据实际项目的情况做清醒的选择和冷静的判断。

第 2 章　大型网站架构面临的挑战

第 1 章介绍了大型网站业务和架构的发展，相信读者已经对大型网站架构有了大致的了解。本章我们将再深入一步，介绍大型网站架构的挑战及其基本的应对思路。

注意：为了方便讲解，本章把架构细分为业务架构和技术架构。业务架构是软件开发的目标，其作用是梳理需求和规划功能结构；技术架构的作用是规划软件结构并制定开发规则，是为了指导和约束开发过程而存在的，技术架构的好坏会直接影响软件质量。另外，除本章以外，本书提及的架构都是技术架构。

2.1　大型网站架构的基本问题

从所有大型网站的共性来讲，大型网站架构的最终目的是可以通过简单地增减服务器来适应当前的用户数量。另外，网站系统的开发终归是量体裁衣的过程，每个网站系统根据不同的运营目的和规模会有不同的功能需求，而大型的网站系统，往往也会有庞大的功能集合。

因此，大型网站架构的基本问题主要有两个：

- 如何应对大量的用户操作；
- 如何规划庞大的功能集合。

2.1.1　业务架构面临的挑战

业务指的是需要处理的事务。笔者对于业务的理解，就是某个特定场景需要处理的需求，比如电商网站系统有电商业务（如订单管理、商品管理、商家管理和用户管理等），直播网站系统有直播业务（如直播间管理和直播审核流程等）。简单地说，业务就是网站系统的功能。

对于大型网站而言，庞大的功能群是不可避免的，毕竟功能是一个网站的运营资本。但是很多时候，正因为功能群的庞大，导致很难整理出清晰的需求列表和功能结构（没有设计出一个清晰的业务架构）。没有清晰的业务架构的网站系统如图 2.1 所示，在这种情况下，看起来是整理完了所有的功能，但是这种没有任何逻辑的列举方式往往会造成需求

遗漏，导致项目实施过程中频繁变动需求，最后导致项目严重超支。

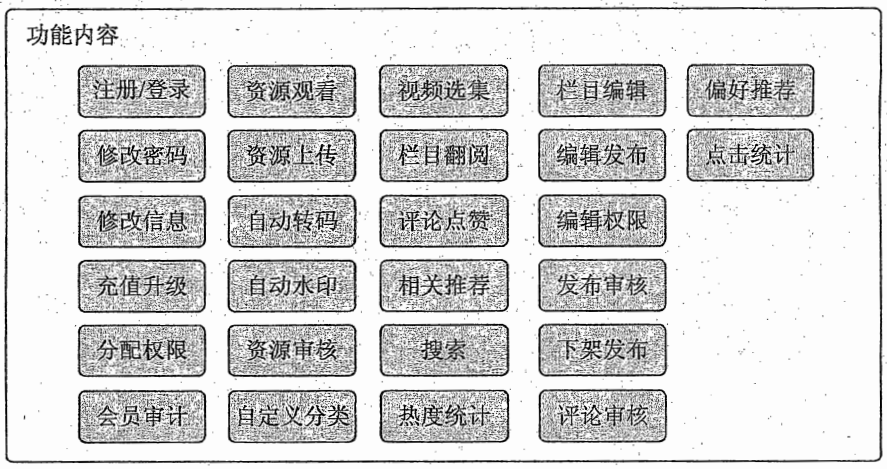

图 2.1　没有清晰的业务架构的网站系统

业务架构是必要的，它能帮助我们顺着业务逻辑思考功能点是否完备，避免需求遗漏。除此之外，业务架构对后续的架构设计和迭代计划都有一定的指导意义。业务架构主要包括两部分，即功能模块结构和核心业务逻辑。

功能模块结构是业务架构的主要部分。这部分需要划分出功能模块（业务模块），规整混乱的功能点。除此之外，用户角色、展示端和平台等信息也需要在此体现。以一个视频网站的业务架构为例，其功能模块结构如图 2.2 所示。一般而言，图 2.2 也被称为业务架构图。

核心业务逻辑是对功能模块结构的补充。一般而言，针对每一个功能模块，都需要梳理其主要的业务逻辑。另外，如果功能模块之间存在关联的话，也最好梳理这部分的业务逻辑。例如，一个视频网站的业务主逻辑如图 2.3 所示。其中，椭圆形代表的是用户角色，矩形代表的是平台入口，圆形代表的是某个资源的抽象。另外，业务逻辑图不需要把所有功能都表达出来，只需要把主要的业务逻辑表达清楚即可。

完善的业务架构确实能让整个项目的开发过程更加顺畅。但是，无论在项目实施过程中还是在运营过程中，都会不可避免地产生很多新的想法，自然也会产生很多需求变更（有时候甚至会推翻之前设计的业务流程）。因此，在大型网站项目中，始终如一的业务架构是很难做到的。大型网站项目的需求变更是很难约束的，而我们能做的就是控制需求变更的实施节奏，避免由于来回改动一些小问题而影响整个项目的进度。

综上，业务架构主要有以下两个挑战：

- 设计清晰的业务架构，在大量需求中规整功能模块，指定功能边界，厘清产品业务逻辑。一般来说，按照本小节提到的“业务架构图”和“业务主逻辑图”来做即可解决这个问题。

- 把控需求变更的实施节奏，避免由于来回改动一些小问题而影响整个项目的进度。关于这个挑战，在 2.2 节“业务架构的基本思路”中会详细说明。

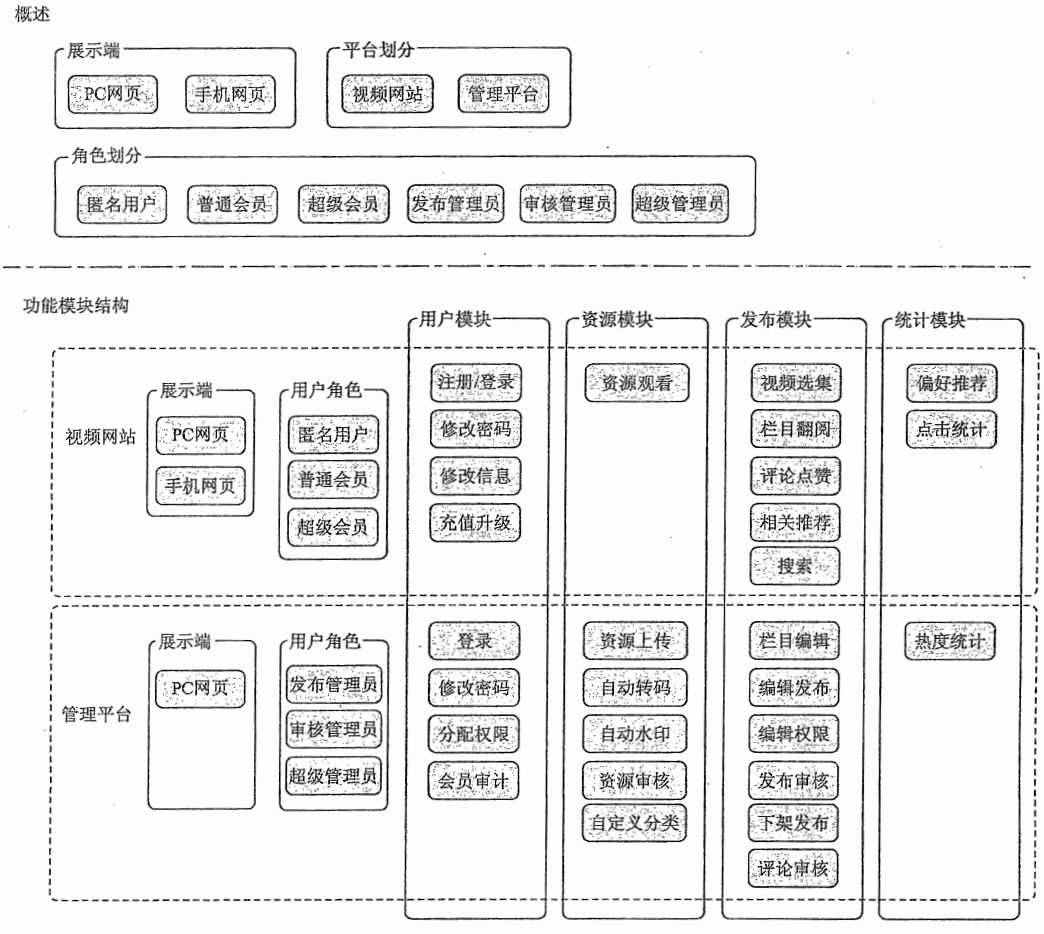

图 2.2　功能模块结构（业务架构图）

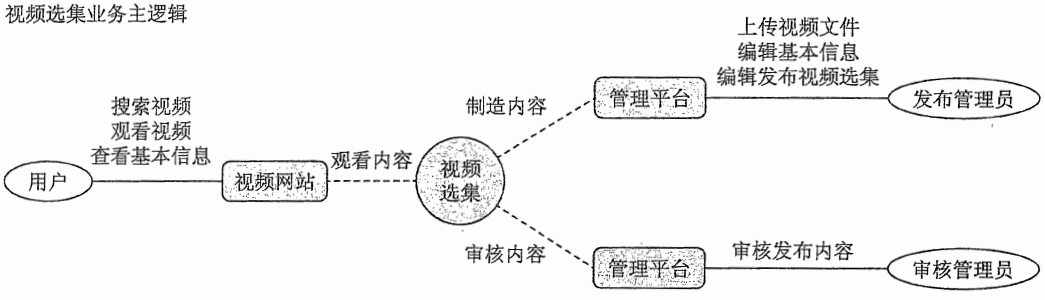

图 2.3　业务主逻辑

注意：在业务架构的设计过程中，可以先忽略“大量用户访问”这一问题，而只关注“如何规整大量的功能集合”这个问题即可。业务架构的设计不需要关心每个功能的细节和具体的实现形式，只需要清楚“做什么”即可。功能细节是“原型设计”阶段才应该关心的事情。

2.1.2　技术架构面临的挑战

技术架构指的是软件架构，是一个软件系统的骨架。技术架构的作用是规划软件结构并明确开发规则，是为了指导和约束整个开发过程而存在的，技术架构的好坏会直接影响软件质量。一个没经过技术架构设计的网站系统如图 2.4 所示。

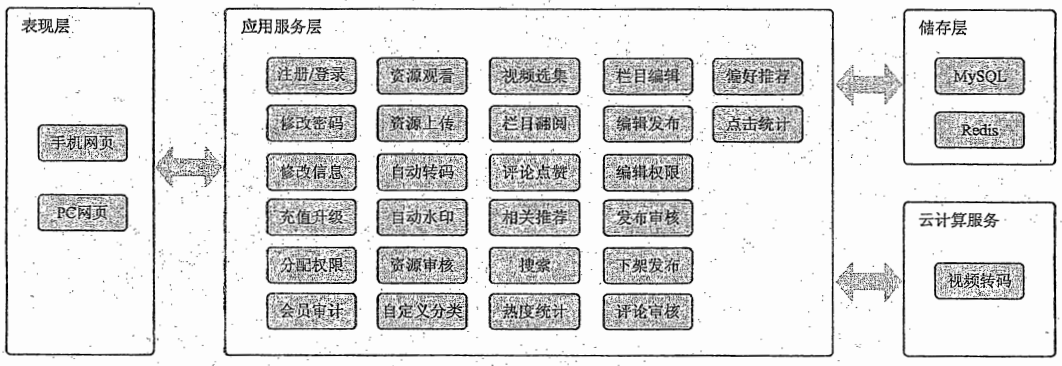

图 2.4　没经过技术架构设计的网站系统

图 2.4 看上去虽然有点结构感，但是这样的结构图对开发过程是没有指导意义的。它的问题在于：第一，业务功能是罗列的，没有划分功能模块或子系统；第二，忽略了太多技术细节。在这样的项目里，开发人员的编码自由度过大，开发人员会按照自己的喜好编码，随着功能的增多和开发人员的更替，网站系统的内部会变得非常混乱，从而导致在维护或者升级网站系统的时候举步维艰。

这些没经过技术架构设计或者技术架构做得不好的软件，笔者称它们为一次性软件，它们像一次性塑料袋一样基本不可再升级维护，使用起来也会存在各种问题。技术架构设计的作用除了能明确软件结构以外，还能让开发团队井然有序地协同工作，让网站系统维护和升级更容易一些。但是要做好一个大型网站的技术架构是困难的，主要有以下几个挑战：

- 清晰地描述系统逻辑，并具备适度的技术细节描述。
- 清晰地划分功能模块或子系统。
- 根据开发团队的水平制定开发规则。

注意：根据开发团队的水平制定开发规则，是指需要制定如编码规范、第三方技术使用规范和通用场景解决方案等规则，以约束整个开发过程，从而避免代码过度混乱。开发规则本章不展开介绍，在后续涉及代码的章节中会提及。

2.1.3　业务架构和技术架构的相互成全

有人说，做项目就是不断妥协的过程。这可能是因为没做好业务架构，开发过程中无原则地增加或改动功能，时间节点一拖再拖，导致最后上线的网站变成一个什么都有但又哪里都有问题的怪物；也有可能是因为技术架构没做好，软件结构模糊，开发过程纪律松散，导致网站系统内部凌乱不堪，用户使用过程中发现一堆 Bug。

除此之外，还可能是业务架构和技术架构没有相互成全。业务架构一般掌握在项目管理者手里，项目管理者往往是希望功能多而全，开发周期更短一些；而技术架构一般掌握在架构师手里，架构师往往希望软件质量过硬，开发周期再长一些。这么看来，业务架构和技术架构本身是存在矛盾点的，这个矛盾点在于成本（时间成本和人员成本等）。

偏倚业务架构的项目往往会造成工作量过大、工期过短的状况，导致每个开发人员都会不断冲破技术架构的既定规则，随心所欲地编写代码以求开发速度最快，最后交付的网站往往就像是一个山寨的电子产品，外表看上去还可以，一旦开始使用就会发现各种问题。这样的网站系统后续升级是困难的，很多时候只能重新再做一个。

偏倚技术架构的项目往往会制定严苛的开发规则并选用最前沿的技术，造成团队遵守规则和学习前沿技术的成本过高，很多工期都没有花在生产上，导致工期一拖再拖，最后的结果往往是成本严重超标，而且迫于成本的压力，最后交付的软件质量也一定与预期差距很大，反而弄巧成拙。

业务架构和技术架构的相互成全，其实是找到成本的平衡点。对于业务架构而言，适当地砍掉一些开发起来比较费劲的非主要功能是有必要的；对于技术架构而言，不能只追求软件质量，也应该考虑业务功能带来的工作量、开发团队的水平和前沿技术的风险等客观问题，适当降低标准。

2.2　业务架构的基本思路

大型网站系统有很多功能，一次性明确所有的功能需求并设计出一个庞大的业务架构是一件费力不讨好的事情。因为在项目前期，难免会忽视一些琐碎功能，而随着开发的进行，也会有很多新的想法产生，基本上不会存在完全按照最初的业务架构设计完成的软件产品。因此，业务架构不仅要做到“规整功能模块，厘清产品业务逻辑”，更重要的是如何做到“有规划性地应对项目过程中的需求变更”。

说明：关于如何做到“规整功能模块，厘清产品业务逻辑”，可以参考 2.1.1 小节中介绍的业务架构图和业务主逻辑图。

2.2.1　递进思想

传统的软件开发基本上都遵循瀑布开发模型，即项目开发过程必须严格按照需求分析、设计、编码、测试和维护等步骤执行。瀑布开发模型的流程和产出物如图 2.5 所示。在一般的瀑布开发模型中，每个阶段都需要有明确的产出物，通过严格的评审后才能进入下一阶段。瀑布开发模型的理想状态是每个阶段只执行一次，一次性完成整个项目。瀑布开发模型很大程度上依赖需求分析阶段的明确性，因为在瀑布开发模型中默认需求是充分和明确的，需求几乎不存在被改动的情况，所以设计和编码阶段都是完全以需求分析为依据的。如果在项目后期才发现有重要的需求变更或者有其他遗漏，往往就会导致项目失败或者项目重新启动。

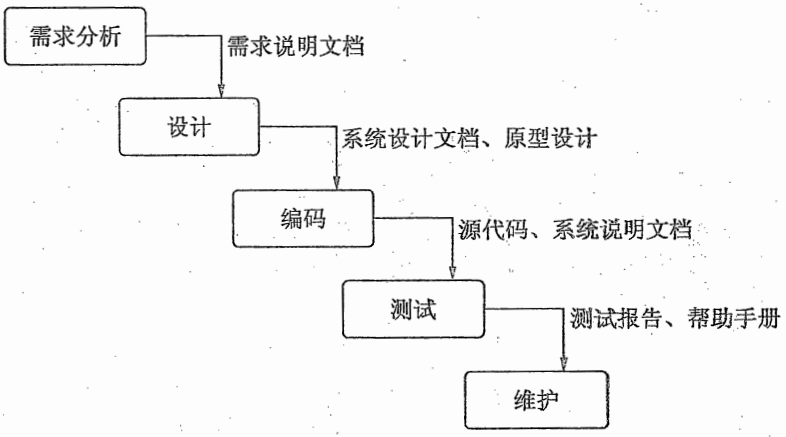

图 2.5　瀑布开发模型的流程和产出物

在 2.1.1 小节中曾提到，大型网站系统的需求很难做到完全明确，在项目开发过程中往往会有更好的想法产生。如果我们完全采用瀑布开发模型的思维方式开发大型网站项目，并且想一次性确定需求并交付项目，那么很可能在项目的后期才会发现需求有遗漏，或者很可能在网站基本定型后才发现大部分功能使用起来并没有预想的好。这些情况都会在很大程度上导致项目失败。

瀑布开发模型的弊端很明显，即需求必须是完全明确的，对需求的变化适应性差。瀑布开发模型一般就是我们执行项目时的惯性思维，正是因为这种惯性思维，使得开发者在项目开发过程中对需求的变更是恐惧的。针对瀑布开发模型的弊端，敏捷开发模型被提出来而且逐渐流行。敏捷开发模型不是确切的项目管理框架，它是一套软件开发的原则。敏捷开发主张适度的计划、迭代开发、提前交付和持续改进，并且提倡快速与灵活地看待开

发与变更。简单地说，就是在开发过程中保持沟通，不断交付完成的部分，并持续地改进，而不是一次性交付项目。遵循敏捷开发原则的项目流程如图 2.6 所示。

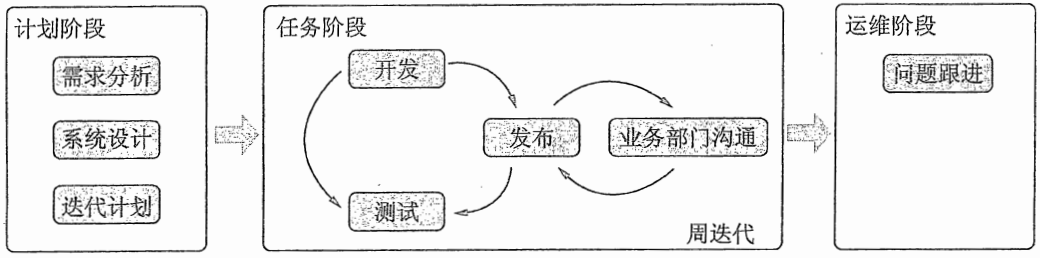

图 2.6　遵循敏捷开发原则的项目流程

不过，软件开发终归逃不出需求分析、设计、编码、测试和维护等步骤，没有一个明确的主体需求也会导致开发无法进行，盲目地持续改进更会造成很大的成本浪费。因此笔者认为瀑布开发模型的流程也不是不能借鉴。敏捷开发提倡的是沟通，通过持续的沟通和改进最终得到一个满意的结果，而非以一开始想象中的全部需求来指导整个项目开发。

因此，不用在意项目开发是遵循瀑布开发模型还是敏捷开发模型，关键是如何在明确需求的同时适应需求变化。笔者认为，项目开发需要有递进思想。遵循递进思想的项目流程如图 2.7 所示。可以看到，应先完成主体功能，然后再添砖加瓦，需求不用一次性完全明确，而是持续地进行沟通和改进，每个部分开始编码前其需求必须是明确的，这样通过多个递进阶段完成整个项目。

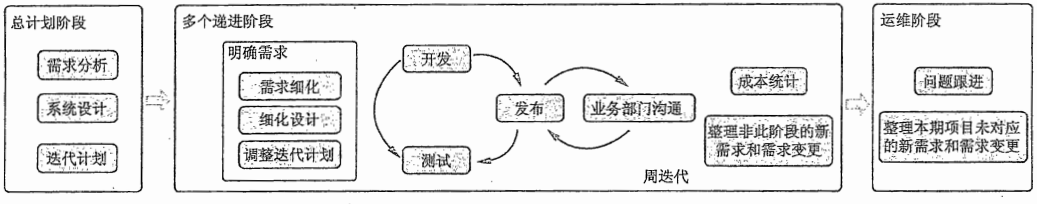

图 2.7　遵循递进思想的项目流程

项目流程只有遵循递进思想，才能更好地应对需求变化。同样，业务架构也需要遵循递进思想，才能有规划地应对项目开发过程中的需求变更。在明确主体需求的前提下，可以适当省略一些次要或琐碎功能的细节，但不能完全忽视这些次要或琐碎功能，完全忽略这些需求会导致工期失控。等主体功能开发完成后，再对次要功能的细节进行明确，等次要功能完成后，再对一些琐碎的功能进行明确，以递进的方式逐步细化和修改业务架构。

说明：除了瀑布开发模型和敏捷开发模型，常用的开发模型还有迭代式开发和螺旋开发等，这里不展开介绍。我们不用在意选用哪种开发模型，因为没有一个模型能确保项目开发过程是顺利的，也没有一个模型能直接导致项目失败。

2.2.2　版本计划逐渐完善

在 2.2.1 小节中曾提到，项目流程只有遵循递进思想，才能更好地应对需求变化。那么，项目应该规划为多个版本，以方便逐步完成。一般来说，项目版本可以划分为主功能阶段、次要功能阶段和优化阶段。版本计划逐步完善的项目流程如图 2.8 所示。

说明：这里的版本计划与项目管理的里程碑是类似的。另外，针对功能集特别庞大的项目，需要划分出独立的几个版本，并对每个版本划分主功能阶段、次要功能阶段和优化阶段等。

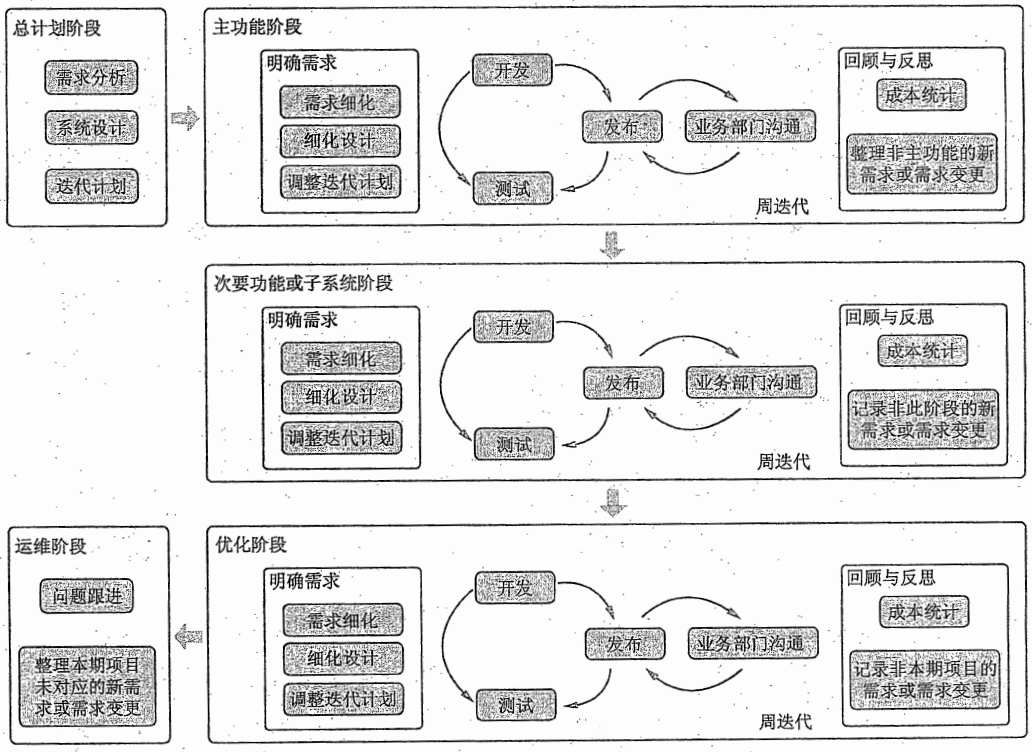

图 2.8　版本计划逐步完善的项目流程

1. 主功能阶段

主功能阶段主要实现整个主体功能，这一阶段的业务架构需要完全明确主体功能需

求，次要功能和琐碎功能的细节可以先忽略，但次要功能和琐碎功能也要体现，因为它们会影响技术架构和迭代计划。项目开发过程中可能会对主体功能进行调整，但是偏离程度不能太大，不能说一开始只想要一个网页，而项目开发过程中却想加入 App 客户端。

2．次要功能阶段

次要功能阶段的目标是实现之前忽略的次要功能。这一阶段可以根据实际情况进一步细分成多个次要功能阶段。在这个阶段中，不需要一次性明确全部的次要功能，只需要明确当前阶段的次要功能即可。次要功能阶段可能会出现频繁的需求变更，因为次要功能一般是一些用户体验方面的功能，经常会发生改动。但是，最好能按照过往经验和精品网站的要求做尽量好的方案，这样能在一定程度上减少需求变更的发生。

3．优化阶段

因为项目开发过程是不停地让业务部门或者客户使用网站系统的过程，所以其间难免会有很多新的想法，有一些甚至是从来没有提及的需求。这些新需求一定不能在主功能阶段和次要功能阶段添加（除非这些需求所需的工作量非常小，或者这些功能是主要功能或次要功能中必不可少的部分），因为这样会打乱这两个些阶段的开发节奏和既定计划，导致进度失控。要想把进度失控的项目重新拉到正常的状态是很困难的，很多时候失控只会越来越严重。因此，这些需求可以先记录下来，在优化阶段再仔细评估和处理这些需求。在优化阶段再处理这些“突发奇想”的需求，既可以保证前面的项目进度，又可以集中控制这些需求带来的风险。

注意：虽然项目开发过程大致分为主功能阶段、次要功能阶段和优化阶段，但是具体开发计划是灵活的，可以根据不同的子系统制定独立的详细计划。有些开发团队有 2 周或 4 周作为一个迭代周期的习惯，具体开发计划也可以按照这样的周期制定。

2.2.3　持续优化，推陈出新

很多项目的失败与团队经验、技术水平或项目管理水平没有直接的关系。大部分项目失败的原因是妄想网站第一次上线就具备市场上所有的好功能，同时还要具备特色功能。这个想法如果占据主动，则会添加过多其实并不需要的伪功能，也会习惯性地修改需求，最后在不知不觉中导致时间成本和人力成本的严重超支，以致项目崩塌。对于大型网站而言，由于功能繁多，所以值得斟酌的地方也会有很多，加之项目工期较长，开发人员看上去也充足，因此需求经常被改变，导致项目在不知不觉中失控。

罗马非一日建成。时间成本和人力成本是有限的，好的功能也会随着市场竞争被更好的功能所取代。而且从用户使用的角度讲，通过多个迭代版本持续学习新功能往往会比一次性地接受过多功能更有吸引力。因此，大型网站项目应该持续优化，且要不断推陈出新，

而非一次性地完成全部功能，即通过一期项目、二期项目、三期项目等有规划地逐步构建大型网站才是合理的。

对于业务架构而言，如果出现一些好的想法或功能，但是其工作量很大，则需要考虑是否将其放在下一期的项目中实现。

注意：网站系统很少像桌面软件或者操作系统一样有明确的版本区分和发布日期，往往是几天一更新，看起来像没有版本计划一样。实际上网站经常更新是由于网站系统升级非常便利，而且这种更新仅仅是对一些 Bug 进行修复或者对小功能进行升级，大的功能其实还是会按照内部版本的规划进行开发和发布的。

2.3　技术架构的基本思路

在 2.1.2 小节中曾提出，技术架构既要清晰地划分功能模块或子系统，又要对整个网站系统的技术逻辑有清晰的认知。庞大的技术架构确实会让人望而却步，架构设计也变得无从入手。如果把一个庞大的技术架构分成独立的几部分，然后再逐一深入的话，那么一个庞大的技术架构也不是不可理解的。

2.3.1　分层思想

架构设计一般被认为是普通编码的进阶，因此我们先从熟悉的编程思想讲起。想必读者一定了解过面向过程和面向对象编程，无论是面向过程编程还是面向对象编程，都需要关心整个业务流程。很多时候，一些重复、通用的步骤会影响我们对整体业务结构的理解，会不自然地产生一种结构混乱的感觉。

除了面向过程和面向对象的编程思想外，当然还有很多其他编程思想，其中面向切面的编程思想也逐渐在网站技术中得到重用。这种编程思想是想通过抽离重复可用的代码（如日志、用户权限等），让开发人员只关注核心代码，而忽略相同且琐碎的部分。不同编程思想下的业务代码对比如图 2.9 所示。

从图 2.9 中可以看出，在面向过程和面向对象的编程思想下，我们需要关心整个业务流程，因此不得不用“认证用户-修改信息-日志记录”和“认证用户-修改密码-日志记录”来描述业务结构；而在面向切面的编程思想下，可以清晰地看到一种层次感，由于这种层次感，我们可以把业务结构简单地归纳为“认证用户-具体业务-日志记录”。

面向切面的编程思想能让我们更轻松地理解业务结构，就是因为这种“层次感”。那么，对于大型网站技术架构而言，如果把庞大的网站系统架构分成独立的几层，用分层的思维去理解庞大的架构，那么设计大型网站技术架构就不再无从下手了。因此，对于大型网站技术架构的设计而言，首先需要有分层思想，然后再对其分而治之。

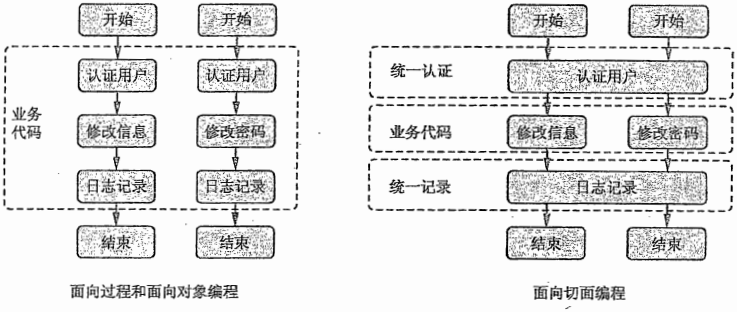

图 2.9　不同编程思想下的业务代码对比

注意：面向切面的编程思想不等同于分层思想，这里只是借现今网站系统开发中比较流行的面向切面的编程思想来引出分层思想。

2.3.2　IaaS、PaaS 和 SaaS 分层管理

一般来说，用分层思想来理解大型网站系统，目前比较公认的架构分层是基础设施服务层（IaaS，Infrastructure as a Service）、平台服务层（PaaS，Platform as a Service）和软件服务层（SaaS，Software as a Service）。例如，视频网站在 IaaS、PaaS 和 SaaS 分层下的技术架构如图 2.10 所示。需要注意的是，这里省略了应用程序部分和很多技术细节。

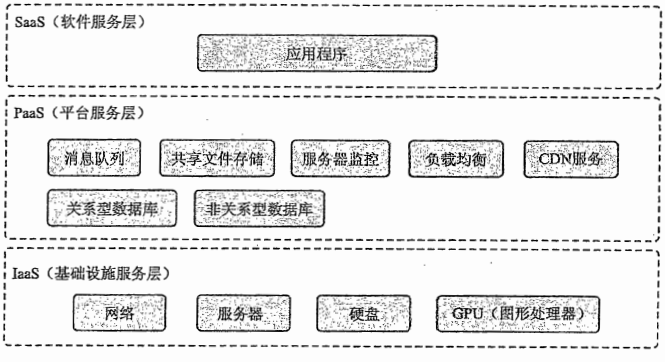

图 2.10　视频网站在 IaaS、PaaS 和 SaaS 分层下的技术架构

IaaS 层主要包括服务器、存储设备和网络设备等，即所有物理硬件都属于这一层。大型网站系统的物理设备可以是公有云的服务器（如阿里云服务器），也可以是购买的本地服务器。但无论是何种形式，对这一层的管理，并不是安排线路或者电源，而是估算每一个独立虚拟机的物理配置、磁盘大小及网络带宽等。

PaaS 层主要包括一些公共软件，也可以说是大型网站系统的运行环境，如操作系统、分布式数据库和分布式文件系统等。我们对这一层的管理主要是选择需要的软件服务，如

数据库和操作系统等。

IaaS 和 PaaS 层已经决定了网站系统的运行环境，SaaS 层才是需要开发的部分。这里先将 SaaS 层的内部称为应用程序，具体内容在 2.3.3 小节中会详细说明。

注意：很多时候 IaaS、PaaS 和 SaaS 也指云计算服务商提供的服务（如直播服务和智能审核服务被称为 SaaS 服务），这里不展开介绍。

2.3.3　前端、后端和云计算服务分层开发

本小节继续讨论 2.3.2 小节中提到的 SaaS 层中的应用程序。在这一部分，我们不仅要对其分层，而且要对其划分功能模块或子系统。

在第 1 章中讲过，大型网站一般以 B/S 架构为主，因此我们可以把应用程序进一步细分成前端、后端和云计算服务。而子系统的划分一般是根据业务架构而定的。以一个视频网站为例，前端、后端和云计算服务分层的技术架构如图 2.11 所示，其中前端部分的页面可能会使用多个子系统的功能。

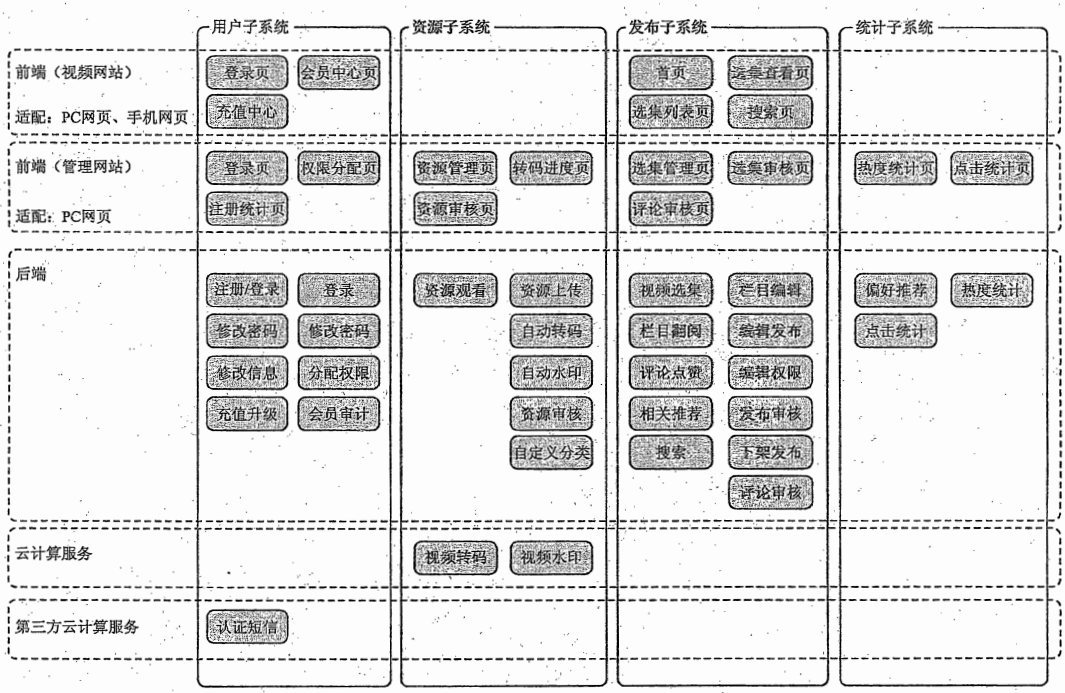

图 2.11　前端、后端和云计算服务分层的技术架构

前端指的是视图层，其作用是交互和展示，一般指的是网页。一些网站系统也有 App 或者 PC 软件，如果按照本小节的分层思路，其实它们也算是前端的一部分，不过很少有

人这么说。

后端指的是业务处理层，其作用是处理前端发送的请求，并且在处理后返回给前端。一些时候，后端也会接收非前端的请求（开放的 API 接口），不过后端处理的都是一些业务请求，如数据库操作和云计算任务调度等。

云计算服务受后端软件调度。这一部分一般是指运行时间较长或者需要持续运行的软件服务，如视频转码服务和爬虫服务等。在图 2.11 中，云计算服务被分成两部分，一部分是自身系统提供的云计算服务，另一部分是第三方云计算服务，它们是由第三方平台提供的。这两部分不一定同时存在，需要根据具体项目情况而定。

我们把 IaaS、PaaS 和 SaaS 分层架构与前端、后端和云计算服务分层架构合并，并加上一些技术说明（技术说明是参照 1.2.7 小节中的图 1.11 添加的），就可以得到一个完整的技术架构，如图 2.12 所示。

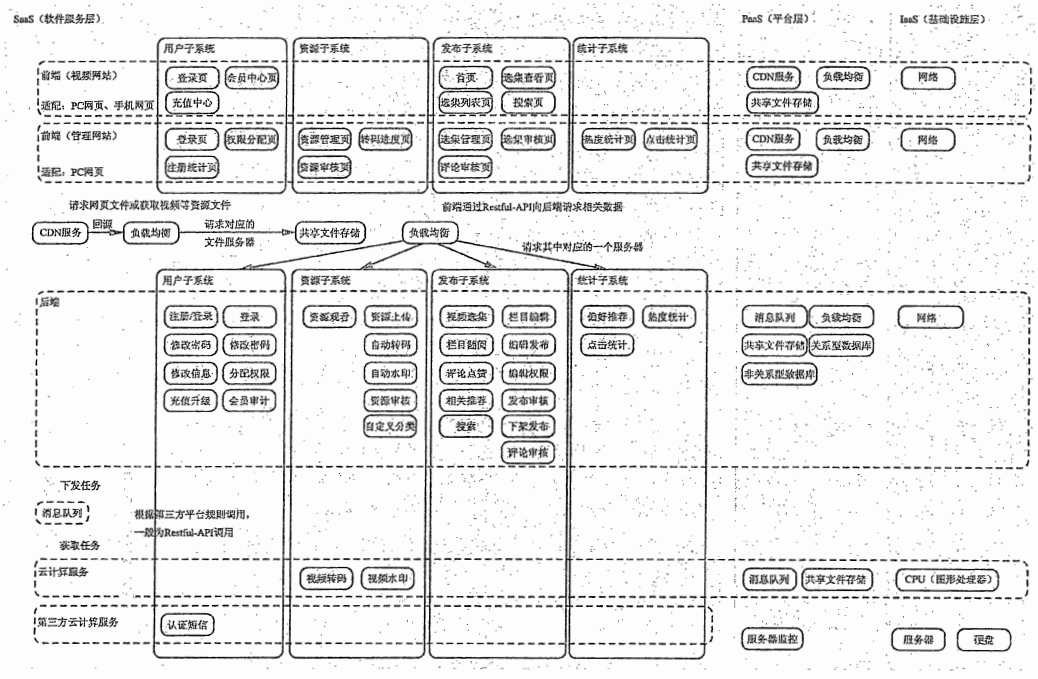

图 2.12　完整的技术架构

图 2.12 基本能表述清楚一些必要的技术细节、系统逻辑和子系统划分，具体项目可以根据实际需要补充或省略一些细节。但这样的架构图只是架构设计的一部分，一般还需要使用其他辅助文档进一步明确细节。不过对架构有了清晰的理解后，其他文档的编写并不复杂。

注意：平常听到的前台和后台实际上都是前端部分，前台一般指的是普通用户使用的前端网页，后台一般指的是管理员用户使用的前端网页。

2.4　大型网站技术架构的核心问题

即使我们以分层思想解决了大型网站技术架构的结构和主要细节，但是还不够全面。本节笔者会把大型网站技术架构的核心问题提出，让读者对其有足够全面的理解。

注意：本节只是提出这些问题，这些问题的具体解决方法将在第 6 章“整体架构”中进行详细说明。

2.4.1　性能问题

性能是所有软件的一个重要指标，网站系统也不例外。网站系统的性能问题可以分为两个，即“网站系统的响应速度是否足够迅速”和“网站系统是否能支撑足够多的在线用户”。一个响应速度很慢或者不能支撑足够多在线用户的网站，一定会被大量用户所诟病。

衡量网站性能有一系列指标，如响应时间、TPS（Transactions Per Second，每秒的事务数）及并发量等。通过测试这些性能指标，可较为客观地评估网站系统的性能。在上线前，可以通过压力测试模拟预期的用户量，从而测试网站性能是否达标。

性能问题无处不在，前端部分需要对浏览器的访问进行优化，后端部分需要考虑缓存、读写分离、代码和数据结构优化，整体部署上需要考虑 CDN 加速、负载均衡、集群和服务器配置调优等问题。

性能调优是网站上线后比较重要的工作，因为在上线前即使做了充分的模拟测试，也很难把正式上线后的所有性能问题都解决。但是也并不意味着网站的性能问题全部都要等到网站上线后才能解决。如集群化、读写分离和缓存等一些优化问题涉及大量编码工作，如果在开发过程中不重视的话，那么很可能会由于网站系统的性能过差且优化工作量过大而推迟上线和运营。笔者曾参与过一个开发过程中没有考虑缓存问题的项目，网站上线后由于数据库压力过大而导致网站响应速度非常慢，而加入缓存会涉及大量的编码工作，最后那个网站的上线时间推迟了几个月。

2.4.2　可用性问题

可用性（也可理解为稳定性）是网站系统的另一个重要指标。对于绝大多数大型网站而言，保证每天 24 小时正常运行是最基本的要求。但事实上，网站系统总会出现一些程序错误或服务器故障，也就是说，服务器宕机本身是难以避免的。高可用性设计指的是，

当一部分服务器宕机时，网站系统仍可正常使用。

网站的可用性指标一般是正常运行时间占总时间的百分比，网站上线后要密切监控网站的健康状况。

高可用性有几个基本的处理手段：第一是冗余，如热备（确保主服务器宕机后备用服务器可以马上取代主服务器）和数据备份（在一定程度上规避硬件故障带来的风险）等；第二是监控，如完整的日志机制（发生故障时有迹可循）和服务器监控等；第三是软件质量；第四是定期维护，如手动重启或清理服务器等，这样能防止很多奇怪的问题发生（如硬盘读写问题或程序崩溃等）。

2.4.3　伸缩性问题

伸缩性指的是网站系统对当前用户使用量的适应性。简单地说，具备伸缩性的网站系统可以通过简单地添加或者减掉服务器来适应当前的用户量。网站在运营中，其用户数量会不断地攀升或持续地下降，也有可能由于营销活动使得某几天的用户量激增。因此，网站系统的用户量是阶段性变化的，这就需要通过添加或者减少服务器来适应当前的用户量。

另外，一些大型网站（如电商网站和视频网站等）不可能在一天之中的每个时段的用户量都是稳定的。在一般情况下，其用户量会在某几个时段激增，而在其他时段会下降。如果时刻都保持满足峰值用户量的服务器数量，一定会造成大量的资源浪费。因此，大部分大型网站需要实现自动弹性伸缩服务器以动态适应用户量。

实现自动弹性伸缩服务器有两种手段：第一是通过监控服务器的 CPU 和内存等基本指标来伸缩服务器；第二是根据业务编写专门的弹性伸缩策略软件，根据业务指标来弹性伸缩服务器。

需要注意的是，增减服务器的手段不是关键，关键是能做到新增的服务器可以马上协同工作（无须进行多余的调控），而减掉的服务器也不影响网站系统的正常运行。

注意：公有云和私有云都提供弹性伸缩服务器的服务。一般来说，我们不需要关心弹性伸缩服务器的实现过程，而只需要制定好弹性伸缩的策略即可。如果按照监控服务器的基本指标进行弹性伸缩，则只需要配置弹性伸缩的性能指标就可以；如果根据业务定制弹性伸缩策略，则需要根据实际业务策略向公有云或私有云发送请求。

2.4.4　扩展性问题

扩展性评价的是，在新增功能时，能否对已有功能做到无影响或少影响，以及新功能能否快速上线。如何响应网站快速发展所带来的需求变化是一定需要考虑的。

扩展性跟其他几个核心问题有所区别，其好坏更多与应用软件部分（前端、后端和云计算服务）相关，也就是与团队编写的代码有很大的关系。好的扩展性要求代码质量过关、业务功能模块划分清晰等。因此，在一开始规划业务功能模块或子系统时就需要仔细斟酌。

分布式服务是解决扩展性的很好方法，通过使用一些分布式框架，便可以增加独立程序来添加新的功能模块。

2.4.5　安全性问题

安全性指的是网站对恶意访问和恶意攻击等的抵抗性。网站系统的安全性，一方面是保证网站的正常运行，另一方面是保证数据不被泄露。近年来，用户数据被泄露的事件对知名网站的影响很大，因此网站的安全性也越来越受到平台的重视。

安全性大多是一些琐碎或者经常被忽略的问题，保证网站的安全性需要做大量的安全性测试，如请第三方公司做渗透测试，或请相关机构做等保（信息系统安全等级保护）测评等。

2.5　小　　结

本章把大型网站架构细分成业务架构和技术架构，比较全面地论述其基本问题和设计思路。其中，提及的业务架构图和技术架构图不是唯一的标准，应根据实际情况进行设计。在本章的最后提出了大型网站技术架构的核心问题，解决了这几个问题，基本就勾勒出了大型网站技术架构的全貌。

从宏观上讲，大型网站架构其实就是这些内容。读者在有了清晰认知的同时，应该还会有很多疑问，那是因为我们还不知道怎么去做。所以从第 3 章开始，会对大型网站架构的细节展开讲解，后面的章节也会更侧重做法。

第2篇
大型网站架构的技术细节

第3章 前端架构

前面的章节从宏观的角度介绍了大型网站架构。从本章开始，我们将着眼于细节，逐一展开介绍前面章节中提到的前端、后端、云计算服务分层的技术架构。其中，前端部分是直接影响用户体验的，因此本章先介绍前端部分的架构。需要注意的是，这里的前端指的是B/S架构网站中的前端网页，是静态网页。

注意：本章不讨论前端的交互设计和UI设计，而只讨论前端架构应该如何应对大型网站前端的复杂性。本章提到的具体方法都不是唯一的，读者需要根据实际情况斟酌参考。

3.1 前端的工作原理

在讨论前端架构之前，我们先搭建一个前端 Web 服务器，再通过构造一个简单的网页来了解其工作原理。在了解前端网页的工作原理之后，我们才能更好地理解前端架构需要注重的细节。

注意：如果在3.1.1小节中对一些配置项感觉概念模糊的话，可以先暂时跳过，在看完3.1.3小节后，再回头琢磨这些配置项应该就清晰了。

3.1.1 Web服务器搭建

网页想要被非本机的浏览器访问，便需要搭建一个Web服务器。目前比较主流的Web服务器软件有三个，即Apache、IIS和Nginx。

Apache是目前使用最多的Web服务器软件，它拥有极其稳定的性能，扩展模块全面、多样，且可以运行在Windows和Linux等多个平台上；IIS是微软提供的互联网基本服务，它除了提供Web服务外，还提供FTP、NNTP和SMTP等服务，不过IIS只能运行在Windows操作系统上；Nginx是轻量级的Web服务器软件，它支持负载均衡和反向代理等功能，扩展模块虽然没有Apache全面，但比Apache占用的内存和资源少，在高并发处理上表现更好。

Web服务器的选取需要根据具体情况而定。对于一般的大型网站而言，因为服务器操

作系统一般以更为稳定的 Linux 为主，并且服务器需要应对大量的网页请求（高并发），所以本书选用 Nginx 作为 Web 服务器软件。

Nginx 的安装以 Windows 系统和 CentOS 系统为例。选择这两个系统进行介绍是因为 Windows 一般是开发人员在开发时使用的操作系统，而 CentOS 一般是网站服务器操作系统。

1．在Windows系统中安装Nginx

在 Windows 系统中安装 Nginx 的具体操作步骤如下：

（1）从 Nginx 官网（http://nginx.org/en/download.html）下载 Nginx，一般选择下载稳定版本。官网上的 Nginx 版本划分如图 3.1 所示，这里我们下载 Windows 系统上的稳定版本。

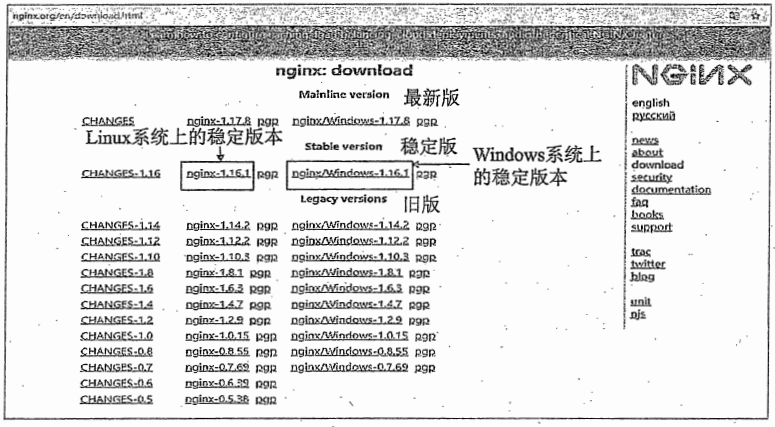

图 3.1　Nginx 官网上的版本划分

（2）下载完 Nginx 压缩包后，将其解压到一个目录下，其目录结构如图 3.2 所示。其中，html 文件夹是默认存放网页资源的地方，conf 文件夹存放的是 Nginx 的相关配置文件，logs 文件夹存放的是 Nginx 的运行日志。

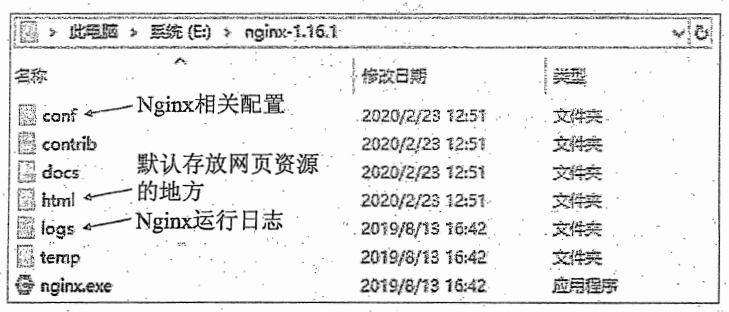

图 3.2　解压后的 Nginx 目录结构

（3）修改 Nginx 配置，Nginx 的配置文件是 conf/nginx.conf。默认的配置文件如代码 3.1 所示，其中，“...”是省略的意思，“#”后面是对属性的注释。如果是本地调试，保持默认配置即可。

代码 3.1　默认的 Nginx 配置

```
...
server {
    listen       80;                          #端口，80 为 HTTP 的默认端口
    server_name  default_server;               #服务名称
...
    location / {
        root   html;       #网页资源文件夹，这里的 html 指的是图 3.2 中的 html 文件夹
        index  index.html;                     #网站默认网页
    }
...
}
...
```

注意：由于本章的重点是前端的代码架构，所以这里对 Nginx 的配置只需要关注端口、服务名称、存放网页资源的路径和默认网页等基本配置即可，其他配置，如负载均衡、反向代理和高并发配置等，会在第 6 章中讲解。

（4）启动服务，双击图 3.2 中的 nginx.exe 文件即可，运行窗口会一闪而过。在不修改默认配置且正常启动的情况下，在浏览器的地址中输入 http://localhost 会打开 Nginx 的默认网页，如图 3.3 所示。这个默认网页是图 3.2 中 html 文件夹里的 index.html。

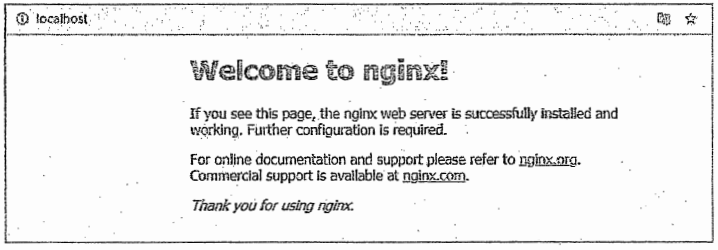

图 3.3　Nginx 的默认网页

如果不能正常启动，可以查看图 3.2 中 logs 文件夹里的运行日志。不能启动一般都是由于端口冲突造成的，此时可以修改步骤（3）中的 listen 配置项。

如果在步骤（3）中修改了 server_name 和 listen 配置项，则需要在浏览器的地址栏中输入 server_name:listen。例如，将 server_name 设置为 192.168.3.3，listen 设置为 8081，那么在浏览器的地址栏中应该输入 http://192.168.3.3:8081。一般情况下，将 server_name 设置成 default_server 即可。

另外，可以通过在任务管理器中结束所有 nginx.exe 任务来关闭 Nginx 服务，如图 3.4 所示。

（5）设置防火墙。如果非本机的浏览器想要访问网页，则需要设置防火墙端口的权限。对于大型网站而言，服务器系统一般为 Linux，Windows 系统下的 Nginx 安装一般只是为了方便本地开发，非本机浏览器访问的场景比较少，因此防火墙的设置不是必需的。如果开发时有非本机访问的情况，可以暂时关闭 Windows 防火墙。

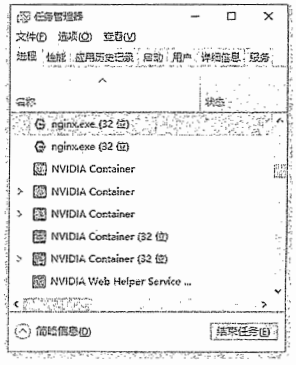

图 3.4　关闭 Nginx 服务

2．在CentOS系统中安装Nginx

在 Centos 系统中安装 Nginx 时，虽然也可以从官网上下载安装，但是操作比较复杂。因此，这里推荐使用 yum 安装，这种安装方式简单方便且不易出错，具体操作步骤如下：

（1）添加 Nginx 源。在默认情况下，CentOS 系统中不包含 Nginx 源，添加 Nginx 源的命令如代码 3.2 所示，其中“\”为换行符。添加成功后，会在/etc/yum.repos.d 目录下多出一个 nginx.repo 文件。

代码 3.2　添加 Nginx 源的命令

```
sudo rpm -ivh \
http://nginx.org/packages/centos/7/noarch/RPMS/nginx-release-centos-7-0.
el7.ngx.noarch.rpm
```

（2）安装 Nginx 的命令如代码 3.3 所示。在 CentOS 系统中，Nginx 默认安装在/etc/nginx 目录下，配置文件是/etc/nginx/nginx.conf，运行日志的路径是/var/log/nginx/，默认网页资源存放在/usr/share/nginx/html/目录下。

代码 3.3　安装 Nginx 的命令

```
sudo yum -y install nginx
```

（3）修改配置。Nginx 的配置文件是/etc/nginx/nginx.conf，默认的配置文件如代码 3.4 所示，其中，“…”是省略的意思，“#”后面是对属性的注释。如果是本地调试，保持默认配置即可。

代码 3.4　CentOS 系统中的 Nginx 配置

```
server {
    listen       80;                          #端口，80 为 HTTP 的默认端口
    server_name  default_server;               #服务名称
…
    location / {
        root   /usr/share/nginx/html;          #网页资源文件目录
        index  index.html;                     #网站默认网页
    }
…
}
…
```

（4）启动服务。设置 Nginx 开机自动启动、启动服务和停止服务的命令如代码 3.5 所