分布式服务器

什么是分布式？

分布式系统是由多个相互独立的计算节点（计算机、服务器等）组成，这些节点通过网络通信协作，共同完成一个系统目标。分布式系统的关键特性是，整个系统对外表现为一个整体，而内部实际上由多个节点共同分担任务。

分布式的核心思想

1. 将任务拆分成多个部分，让多个节点分别处理。
2. 节点间协作并通过通信保证任务的一致性。

分布式 vs 集中式 vs 并行

集中式

所有任务由一个中心节点处理，比如传统的单机系统。

分布式

任务分布在多个节点上，这些节点可能位于不同的地理位置，通过网络通信完成目标。

并行计算

多个任务同时在多个处理器上执行，但通常是在同一个物理节点上，属于一种特殊的分布式形式。

分布式的应用

区块链

- **定义**：区块链是一种特殊的分布式账本技术。所有参与者（节点）共同维护一个账本，通过共识机制确保账本的一致性和不可篡改性。
- **特点**：
  - 数据分布在多个节点上，所有节点共同维护数据。
  - 数据通过共识机制保证一致性。
  - 常见例子：比特币、以太坊。

分布式的应用

联邦学习

- **定义**：联邦学习是一种分布式机器学习框架，允许多个参与方在**不共享数据的前提下**协作训练一个全局模型。**数据保存在各自的**设备中，仅共享训练的模型参数或梯度。
- **特点**：
  - **数据**隐私得到保护（数据不离开本地）。
  - **模型的**训练由多方参与，依赖分布式计算。

分布式服务器

- 什么是分布式服务器：**分布式服务器是由多个物理服**务器或虚拟服务器组成的系统，这些服务器通过网络协同工作，共同为用户提供服务。分布式服务器通常将任务和数据分散到不同的节点（服务器）上，**以提高系**统的性能、可靠性和可扩展性。

分布式服务器特点

高可用性

通过在多个服务器上复制数据和任务，可以保证即使部分服务器故障，整个系统仍可继续运行。

可扩展性

可以根据需要轻松添加更多服务器来处理增加的负载。

灵活性

分布式服务器可以部署在云环境或物理环境，适应各种应用场景。

分布式数据库服务器：

- Apache Cassandra：一种高度可扩展的NoSQL数据库，设计用于管理大量结构化数据，具有无中心节点、强一致性选项以及故障恢复机制。
- MongoDB：文档型数据库，支持水平扩展，适合大数据量和高并发场景下的应用。
- MySQL Cluster (NDB)：基于MySQL的集群解决方案，适用于要求高可用性的事务型应用场景。
- Amazon DynamoDB：完全托管的键值和文档数据库服务，提供了快速且可预测的性能。

分布式缓存服务器：

- Memcached 和 Redis：高速内存中的键值存储系统，广泛用于减轻数据库压力，加速Web应用程序响应速度。

分布式计算与消息队列服务器：

- Apache Hadoop：用于大规模分布式处理和分析海量数据的开源框架。
- Apache Spark：基于内存的数据处理框架，比Hadoop MapReduce更加快速，支持批处理、流处理和机器学习等。

分布式搜索引擎服务器：

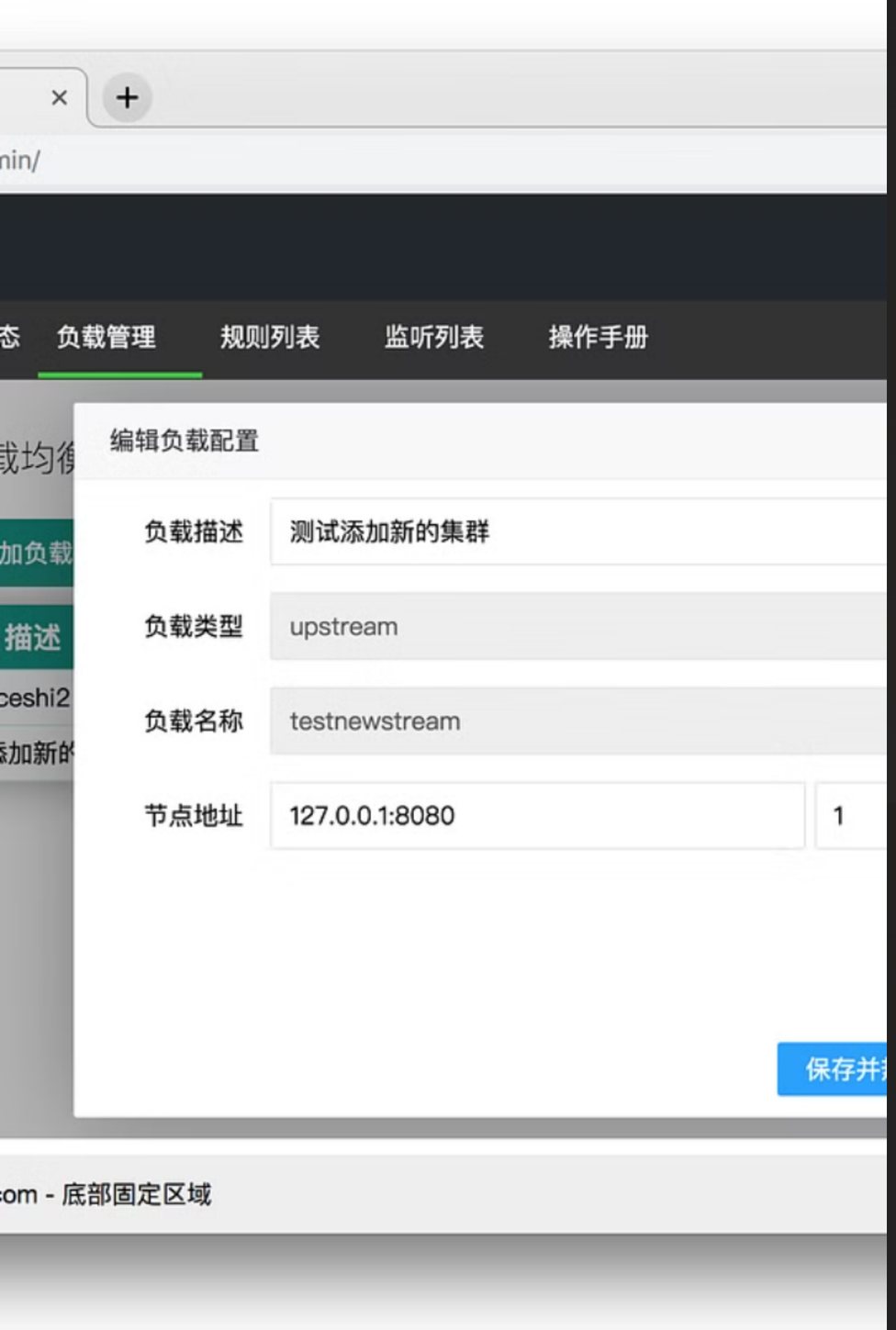
- Elasticsearch：分布式的全文搜索引擎，能够处理PB级的大数据，并能近乎实时地进行搜索和分析。

分布式负载均衡器：

- Nginx Plus 和 HAProxy：这些软件可以通过负载均衡算法将网络流量分发到后端的多个服务器，确保系统的高性能和高可用性。

分布式服务器的挑战

- **数据一致性**: 保持多个节点之间数据的一致性是关键挑战，需要解决数据同步和冲突的问题。
- **网络延迟与带宽**: 节点间的通信依赖网络，网络延迟和带宽限制会影响系统性能，跨区域节点通信可能导致响应时间增加。
- **分布式事务**: 保证跨多个节点的事务操作的原子性和一致性需要复杂的协议和机制，例如两阶段提交协议（2PC）。
- **负载均衡**: 合理分配请求和任务到各个节点，避免某些节点过载而其他节点空闲，例如 Web 服务器集群中的负载均衡器分发 HTTP 请求。
- **故障检测与恢复**: 及时检测节点故障并采取恢复措施，例如自动重启故障节点或重新分配任务到健康节点。



探索 Nginx

Nginx 是一个高性能的 HTTP 和反向代理服务器，同时也是一个 **IMAP/POP3/SMTP 邮件代理服务器**。它以高并发连接处理能力、低内存占用和灵活的配置著称，被广泛应用于网站服务、负载均衡和反向代理等场景

Nginx 的主要功能

- **HTTP 服务器:** 作为静态文件服务器，提供 HTML、CSS、JS、图片等文件服务
- **反向代理:** 将客户端请求转发给后端应用服务器，如 Node.js、Tomcat
- **负载均衡:** 支持多种负载均衡算法，例如轮询、IP 哈希、权重等
- **邮件代理:** 支持 IMAP、POP3 和 SMTP 协议，作为邮件服务器的代理
- **高并发支持:** 使用异步事件驱动架构，可以处理大量的并发连接

Nginx 作为反向代理的流程

1

客户端请求

请求首先到达 Nginx。

2

接收请求

Nginx 作为反向代理接收 HTTP 请求。

3

请求转发

Nginx 根据配置转发到后端服务器。

Nginx 的架构特点

- **事件驱动架构**：Nginx 使用异步非阻塞 I/O 和事件驱动模型，通过一个主进程和多个工作进程协作完成请求处理。每个工作进程使用单线程，同时可以处理成千上万的并发请求。
- **模块化设计**：Nginx 的功能由模块实现，可以根据需要选择性地加载模块，例如 HTTP 模块、邮件代理模块和负载均衡模块。
- **高性能与低资源消耗**：Nginx 的内存和 CPU 占用非常低，尤其在处理静态资源时性能表现优异，适合处理高并发请求。

Nginx 的优势

性能卓越

Nginx 利用系统资源更高效，处理大量并发请求。

可扩展性强

轻松扩展使用，例如作为负载均衡器。

资源消耗低

资源使用优于其他 Web 服务器。

负载均衡

负载均衡是指将接收到的网络请求或者计算任务分配到多个服务器或多个网络路径上，以提高总体处理速度、优化资源使用和减少响应时间。

负载均衡策略与 Nginx 实现

轮询	请求依次分配给每个服务器。
加权轮询	依权重分配请求量。
最少连接数	新请求到当前连接数最少的服务器。
权重最少连接数	结合权重考虑服务器连接数。
源地址哈希	根据请求源IP分配服务器。

. 轮询（Round Robin）

- **描述**：这是最简单的负载均衡算法。请求按顺序依次分配给每个服务器。当列表结束时，算法再从头开始。
- 优点：实现简单，适用于服务器性能相似的场景。
- **缺点**：不考虑服务器的实际负载和处理能力。

```
upstream myapp {
    server server1.example.com;
    server server2.example.com;
    server server3.example.com;
}
```

加权轮询（Weighted Round Robin）

- **描述**：在轮询的基础上，给每个服务器设置一个权重。服务器接收的请求数量按其权重比例分配。
- 优点：可以根据服务器的性能和处理能力调整权重，实现更合理的负载分配。
- **缺点**：权重设置需要根据服务器性能手动配置。

```
upstream myapp {
    server server1.example.com weight=3;
    server server2.example.com weight=2;
    server server3.example.com weight=1;
}
```

最少连接（Least Connections）

- 描述：新的请求会被分配给当前连接数最少的服务器。
- 优点：能较好地应对不同负载的情况，合理分配请求。
- 缺点：在高并发情况下，统计和更新连接数可能会成为性能瓶颈。

```
upstream myapp {
    least_conn;
    server server1.example.com;
    server server2.example.com;
    server server3.example.com;
}
```

加权最少连接（Weighted Least Connections）

- 描述：在最少连接算法的基础上，考虑服务器的权重，权重越高的服务器可以承担更多的连接。
- 优点：相比最少连接算法，更能考虑到服务器的处理能力。
- 缺点：算法相对复杂，需要维护更多的状态信息。

```
upstream myapp {
    ip_hash;
    server server1.example.com;
    server server2.example.com;
    server server3.example.com;
}
```

基于源地址哈希（Source IP Hashing）

- 描述：根据请求的源 IP 地址进行哈希，然后根据哈希结果分配给特定的服务器。
- 优点：保证来自同一源地址的请求总是被分配到同一服务器，有利于会话保持。
- 缺点：当服务器数量变化时，分配模式会发生变化，可能会打乱原有的分配规则。

```
upstream myapp {
    ip_hash;
    server server1.example.com;
    server server2.example.com;
    server server3.example.com;
}
```

负载均衡选项

1

DNS负载均衡

通过 DNS 分散请求到不同服务器。

2

硬件负载均衡器

使用专业硬件设备，如 F5 BIG-IP。

3

云服务负载均衡

云服务商如 AWS 提供的解决方案。

负载均衡的进阶实现

应用层负载均衡

根据应用逻辑调整负载。

集中式负载均衡

所有请求先到集中器再分发到服务器。

分布式负载均衡

多地点独立处理流量。

响应时间

请求分配给响应最快的服务器。

Nginx 配置解析

深入解析nginx.conf：如何定义服务器组，处理客户端请求，及使用 proxy_pass 指令转发请求到服务器组。

```
events {
    worker_connections  1024;  # 每个 worker 进程的最大连接数
}
http {
    upstream myapp {
        server localhost:8080;  # C++ 服务器实例1
        server localhost:8081;  # C++ 服务器实例2
    }
    server {
        listen 80;  # Nginx 监听端口
        location / {
            proxy_pass http://myapp;  # 转发到上游服务器组
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
        }
    }
}
```

- 配置解释：
  - upstream 定义了一个服务器组，用于后端的 C++ 服务器实例。
  - server 块定义了如何处理到来的客户端请求。在 location 块中，我们使用 proxy_pass 指令将请求转发到定义的上游服务器组。
  - proxy_set_header Host $host;：保留原始请求中的Host头信息。
  - proxy_set_header X-Real-IP $remote_addr;：向后端服务器传递真实的客户端IP地址。
  - proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;：添加一个X-Forwarded-For头，记录客户端请求经过的所有代理服务器的IP地址，$proxy_add_x_forwarded_for变量会自动添加当前Nginx服务器的IP地址。
  - proxy_set_header X-Forwarded-Proto $scheme;：传递原始请求使用的协议类型（http或https）给后端服务器，这对于后端服务器判断请求是否通过HTTPS加密非常重要。

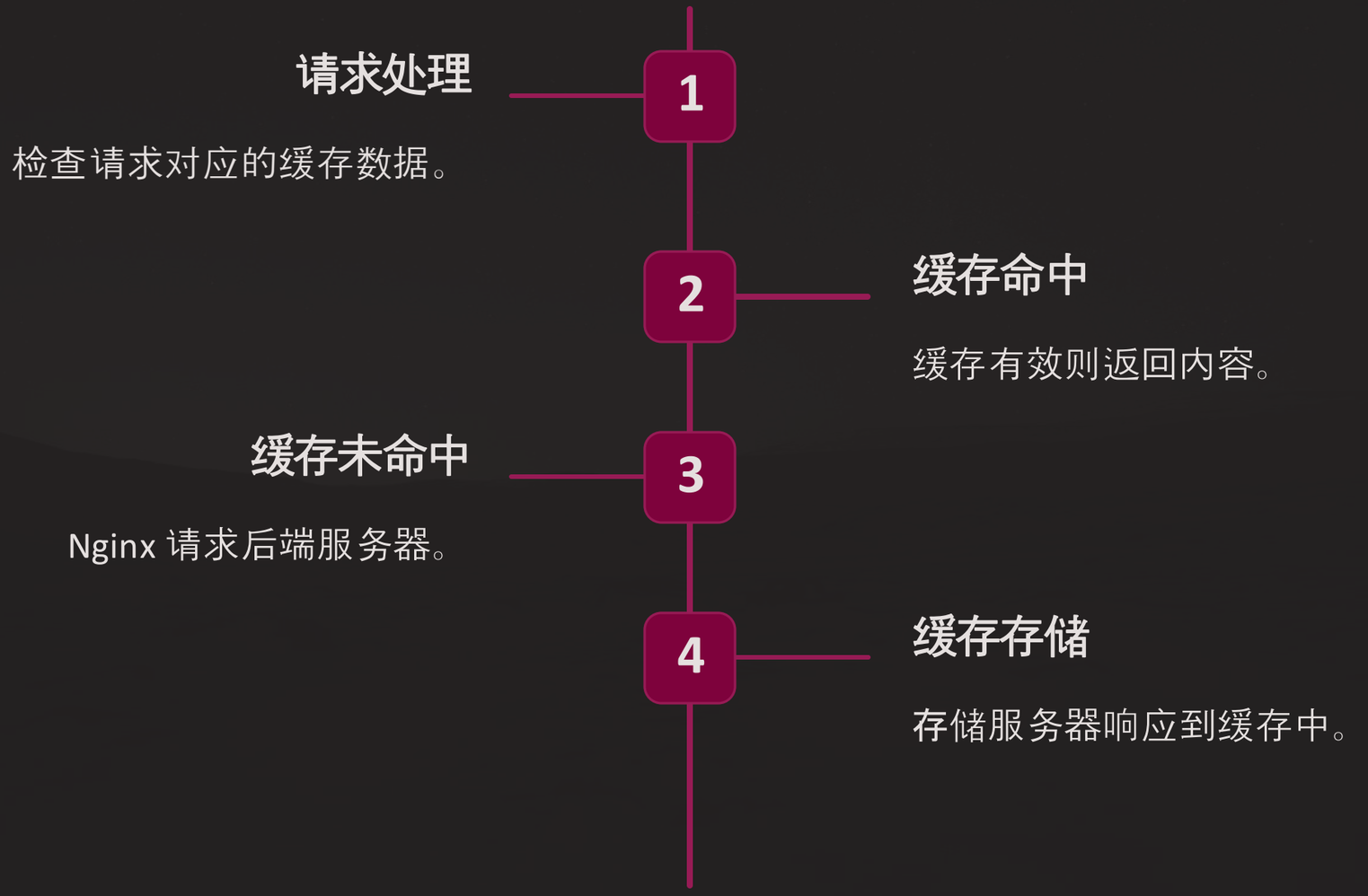
Nginx 缓存的意义

Nginx 缓存通过存储静态和动态资源，有效提升网站性能，减少后端服务器负载。

当客户端请求相同资源时，Nginx 直接从缓存中返回结果，无需再次访问后端服务器，从而提高网站响应速度。

可以通过在nginx.conf中配置proxy_cache指令来设置缓存的规则，包括缓存的有效期、缓存文件路径等。

Nginx 缓存原理



基本概念

缓存决策

- 何时缓存：根据配置来确定哪些响应应该被缓存。
- 缓存时长：可以配置不同类型的响应被缓存的时间长度。

缓存键

- 每个缓存项都由一个唯一的“缓存键”（Cache Key）标识，通常基于请求的URL和其他头部信息生成。

缓存存储

- 缓存数据存储在指定的文件系统路径中，通过高效的文件系统操作和索引机制管理这些缓存文件。

性能优化

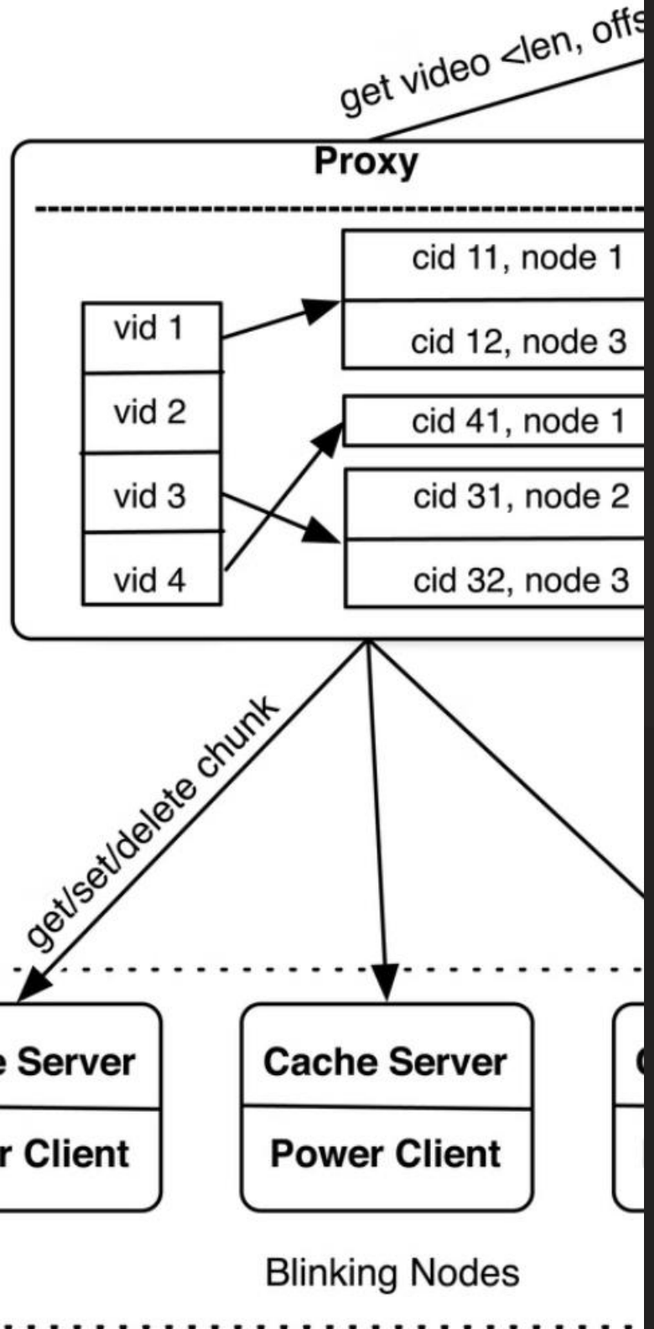
- Nginx 的缓存机制非常高效，能够处理大量的并发请求，同时减少对后端服务器的压力。
- 缓存可以配置在本地磁盘或者内存中，以提供更快的读写速度。

高级特性

- 条件请求：Nginx 支持根据客户端发送的条件请求头（如 If-Modified-Since）来判断是否需要返回完整内容。
- 缓存刷新和过期：管理员可以手动清理缓存或者配置自动过期机制。

优化的 Nginx 缓存配置

通过proxy_cache_path 和 keys_zone 指定缓存的存储路径，分配内存。设置缓存时间长度及策略，确保存取效率和数据更新。



缓存应用实践

1 **缓存键设置**

唯一标识请求的缓存键。

2 **缓存存储之道**

缓存文件有效管理。

3 **提升性能**

快速处理并发请求。

实操：Docker 中的 Nginx 与 C++ 服务器

配置 Dockerfile

设置 Nginx 和 C++ 环境，复制配置和代码。

容器端口

暴露 Nginx 和服务器所需端口。

启动脚本

使用启动脚本运行服务。

面试常问问题

分布式系统的主要特点有哪些？

回答：

- **可扩展性**：通过增加节点提升性能。

- **高可用性**：冗余设计确保系统持续运行。

- **容错性**：节点故障不会影响整体系统。

- **一致性**：保证数据在各节点间一致。

- **透明性**：用户无需了解系统内部结构。

CAP 定理是什么？它的三个要素分别代表什么？

回答：

CAP 定理指出，在分布式系统中，无法同时保证以下三项：

- **一致性（Consistency）**：所有节点在同一时间看到的数据是一致的。

- **可用性（Availability）**：每个请求都能在有限时间内得到响应。

- **分区容忍性（Partition Tolerance）**：系统在网络分区时仍能正常运行。

解释一下Nginx在分布式系统中的作用。

回答：

Nginx作为反向代理和负载均衡器，将客户端请求分发到多个后端服务器，提升系统的并发处理能力和可靠性。此外，Nginx还可以缓存静态资源，减少后端服务器负载。

谢谢大家