

02-18 算法10.4 计算机系统-进程状态的切换

02-20 算法 16

02-21 算法 23

02-22 算法 25

02-25 算法 33

03-05 算法 34

03-06 算法36 38(还未理解)

涉及递归的题：24、25、26、和树相关的算法、38

递归算法有四个特性：

必须有可达到的终止条件，否则程序陷入死循环；(必须存在至少一个语句块，不用执行递归)

子问题在规模上比原问题小；

子问题可通过再次递归调用求解；

子问题的解能组合成整个问题的解。

适合直接看面试总结的：

三星：

--算法

--剑指Offer题解()：面试常见题

--Leetcode题解()：面试常见题

--算法(7晚上)：一些常见的算法，比如排序，树的查找

--操作系统：

--Linux(12小时)

--计算机操作系统(13小时)

--网络

--计算机网络(12小时)

--HTTP(8小时)

--Socket(4小时)

--系统设计

--系统设计基础(1小时)

--分布式(12小时)

--集群(12小时)

--缓存(1晚上)

--消息队列(1小时)

--攻击技术(1小时，可不看)

c/c++学习：

https://github.com/Ewenwan/ShiYanLou

面试roadmap

面试知识点：

https://github.com/CyC2018/Backend-Interview-Guide/blob/master/doc/%E4%B8%80%E6%96%87%E5%B8%AE%E4%BD%A0%E7%90%86%E6%B8%85%E9%9D%A2%E8%AF%95%E7%9F%A5%E8%AF%86%E7%82%B9.md

剑指offer：

https://github.com/CyC2018/CS-Notes/blob/master/docs/notes/%E5%89%91%E6%8C%87%20offer%20%E9%A2%98%E8%A7%A3.md

计算机操作系统：

https://github.com/CyC2018/CS-Notes/blob/master/docs/notes/%E8%AE%A1%E7%AE%97%E6%9C%BA%E6%93%8D%E4%BD%9C%E7%B3%BB%E7%BB%9F.md

**汇总：**

**https://github.com/CyC2018/CS-Notes**

数据库：

数据库系统原理🡪SQL🡪Leetcode-Database题解🡪SQL Server书籍🡪Redis

算法：

时间/空间复杂度🡪剑指offer题解(68题)🡪Leetcode题解

操作系统：

计算机操作系统🡪Linux

网络：

计算机网络🡪HTTP🡪Socket

面向对象：

设计模式🡪面向对象思想

面试经验：

<https://github.com/FangWW/Document>

**时间空间复杂度**

时间复杂度：反映了程序执行时间随输入规模增长而增长的量级。

时间消耗决定因素：

1算法采用的策略、方法

2编译产生的代码质量

3问题的输入规模

4机器执行指令的速度

从算法中选取一种对于所研究的问题(或算法类型)来说是基本操作的原操作，以该操作的重复执行的次数作为算法的时间量度。

时间频度：一个算法中语句执行的次数称为时间频度或语句频度。记为T(n)

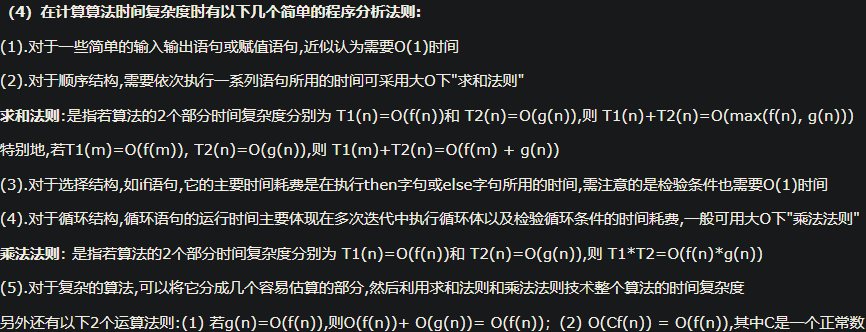
时间复杂度：n称为问题的规模，一般情况下，算法中基本操作重复执行的次数是问题规模n的某个函数，用T(n)表示，若有某个辅助函数f(n),使得当n趋近于无穷大时，T(n)/f(n)的极限值为不等于零的常数，则称f(n)是T(n)的同数量级函数。记作T(n)=Ｏ(f(n)),称Ｏ(f(n)) 为算法的渐进时间复杂度，简称时间复杂度。

计算时间复杂度：

1找出算法中的基本语句。算法中执行次数最多的那条语句，通常是最内层循环的循环体。

2计算基本语句的执行次数的数量级。保证基本语句执行次数的函数中最高次幂正确即可，可以忽略低次幂及最高次幂系数.

3用大O表示算法的时间性能。



空间复杂度：该算法所耗费的存储空间。包括存储算法本身所占用的存储空间，算法的输入输出数据所占用的存储空间和算法在运行过程中临时占用的存储空间。

算法：

1设置到数组，关于寻值，判断。一般在进行查找的同时，需要对数组进行排序，降

低算法的时间复杂度。

2递归和动态规划将一个问题划分成多个子问题求解，但是动态规划会把子问题的解

缓存起来，从而避免重复求解子问题(如：斐波那契数列)

3首先要想出一个完整可行的算法，如果条件允许的话，再考虑有无更好的算法。

4一眼就能知道一种解决算法时，往往是需要利用其他算法降低时间、空间复杂度。

5递归和迭代，递归代码量少，但是难理解；迭代代码多，需要用到额外的指针变量，

但是容易理解；

**系统设计基础**

一：性能

性能优化

集群：使用负载均衡将请求转发到集群中。

缓存：

缓存数据通常位于内存中，读操作快；

缓存数据可以位于靠近用户的地理位置上；

可以将计算结果进行缓存，从而避免重复计算。

异步：

二：伸缩性

指不断向集群中添加服务器来缓解不断上升的用户并发访问压力和不断增长的数据存储需求。

三：扩展性

指添加新功能时对现有系统的其他应用无影响，这就要求不用应用具备低耦合的特点。

实现可扩展的方式：

1使用消息队列进行解耦，应用之间通过消息传递进行通信；

2使用分布式服务将业务和可复用的服务分离开来，业务使用分布式服务框架调用可复用的服务。新增的产品可以通过调用可复用的服务来实现业务逻辑，对其他产品没有影响。

四：可用性(和伸缩性的区别)

冗余

保证高可用的主要手段是使用冗余，当某个服务器故障时就请求其他服务器。冗余通过主从复制实现。

监控

服务降级

系统为了应对大量的请求，主动关闭部分功能，从而保证核心功能可用。

五：安全性

**消息队列**(因为内容较少，后续复习可看GitHub)

一：消息模型

二：使用场景

三：可靠性

幂等性：多次执行的效果与一次执行的影响相同。

**Socket**

一：I/O模型

套接字上的输入操作步骤：

等待数据从网络中到达，到达时，数据被复制到内核中的某个缓冲区；

把数据从内核缓冲区复制到应用进程缓冲区。

Unix 的I/O模型

阻塞式I/O

非阻塞式I/O

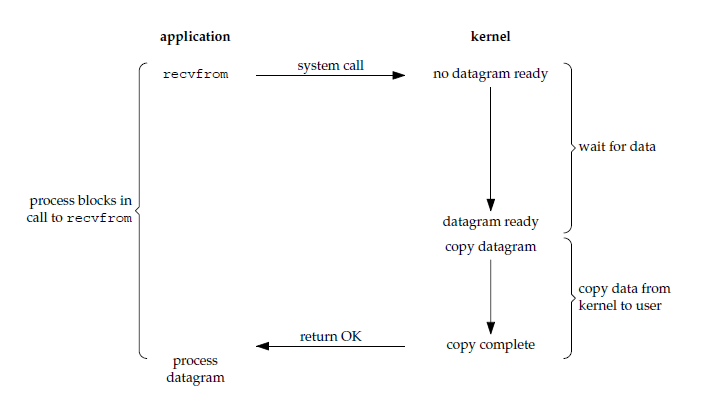
I/O复用(select和poll)

信号驱动式I/O(SIGIO)

异步I/O(AIO)

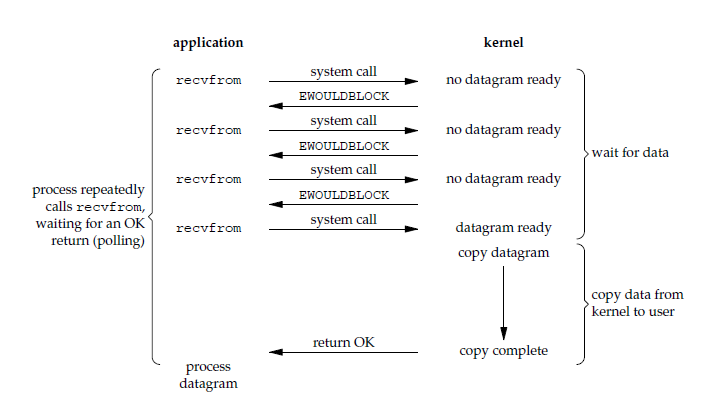
1阻塞式I/O

应用程序被阻塞，直到数据从内核缓冲区复制到应用进程缓冲区才返回。在阻塞的过程中，其他应用进程还可以执行，因此阻塞不意味着整个操作系统都被阻塞。因为其他应用进程还可以执行，所以不消耗CPU时间，这种模型的CPU利用率较高。



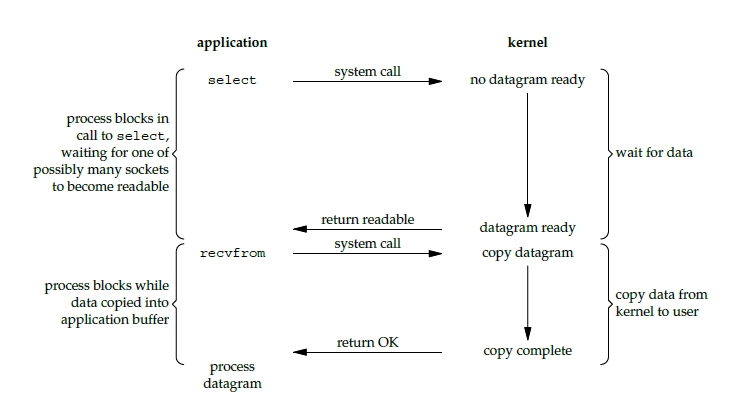
2非阻塞式I/O

应用进程执行系统调用之后，内核会返回一个错误码。应用进程可以继续执行，但是需要不断地执行系统调用来获知I/O是否完成，这种方式称为轮询(polling)。由于CPU要处理更多的系统调用，因此CPU利用率较低。



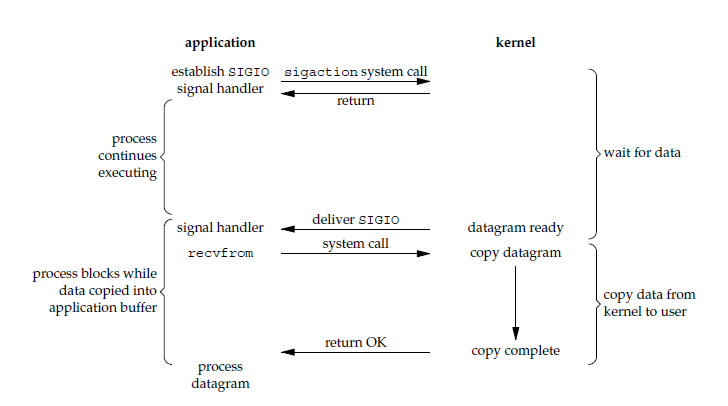
3 I/O复用(select和poll)

使用 select 或者 poll 等待数据，并且可以等待多个套接字中的任何一个变为可读。这一过程会被阻塞，当某一个套接字可读时返回，之后再使用 recvfrom 把数据从内核复制到进程中。它可以让单个进程具有处理多个 I/O 事件的能力。



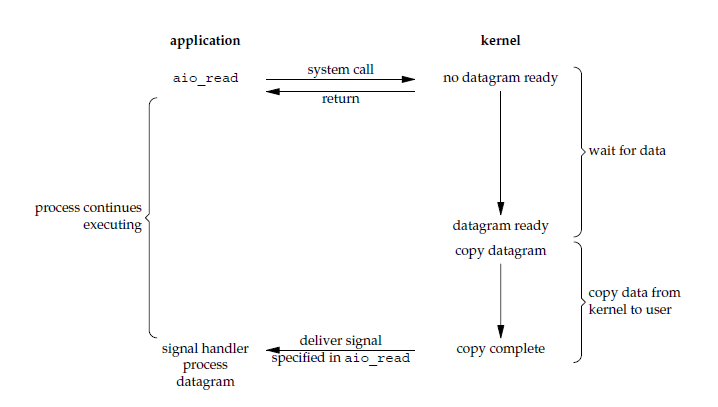
4信号驱动式I/O(SIGIO)

应用进程使用sigaction进行系统调用，内核立即返回，应用进程可以继续执行，等待数据阶段是非阻塞的。内核在数据到达内核时，向应用进程发送sigio信号，应用进程收到之后，调用recvfrom将数据从内核复制到应用进程



5异步I/O(AIO)

异步I/O和信号驱动的区别是，异步在将数据从内核复制到应用进程后才调整应用进程，也就是异步 I/O 的信号是通知应用进程 I/O 完成，而信号驱动 I/O 的信号是通知应用进程可以开始 I/O。



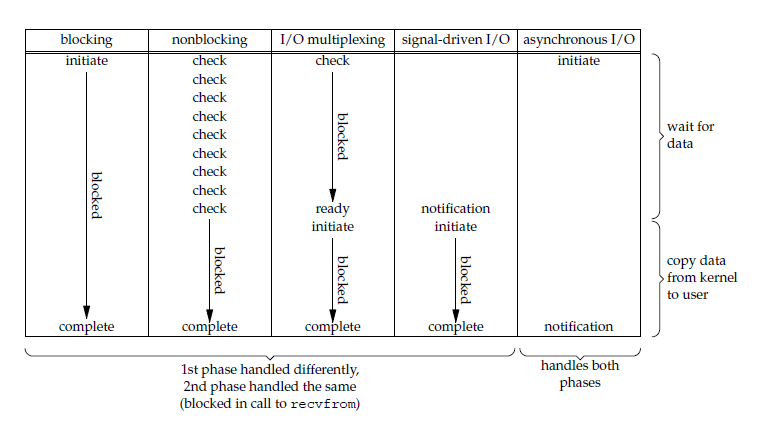
五大I/O模型比较

同步 I/O：将数据从内核缓冲区复制到应用进程缓冲区的阶段，应用进程会阻塞。

异步 I/O：不会阻塞。

阻塞式 I/O、非阻塞式 I/O、I/O 复用和信号驱动 I/O 都是同步 I/O，它们的主要区别在第一个阶段。

非阻塞式 I/O 、信号驱动 I/O 和异步 I/O 在第一阶段不会阻塞。



五大I/O模型比较

二 I/O复用

select，poll，epoll比较

**缓存**

**一：缓存特征**

命中率：当某个请求能够通过访问缓存而得到响应时，称为缓存命中。

最大空间：缓存通常位于内存中，当存放的数据量超过最大空间时，需要淘汰部分数据来存放新到达的数据。

淘汰策略：

FIFO(First in First Out)：先进先出

LRU(Least Recently Used)：优先淘汰最久未使用的数据，保证缓存命中率。

**二：LRU**

LRU可以基于双向链表和HashMap实现：双向链表用以保证最新的数据放在头部，最旧的数据在尾部，这样删除时，直接删除尾部节点；哈希表用以保存Key到节点的映射，通过Key就能以O(1)的时间得到节点，然后再以O(1)的时间将其从双向链表中删除。

**三：缓存的位置**

浏览器：当HTTP响应允许缓存时，浏览器会将html，css，js，图片等静态资源进行缓存；

ISP：

反向代理：反向代理位于服务器之前，请求与响应都需要经过反向代理。通过将数据缓存在反向代理，在用户请求时就可以直接使用缓存进行响应；

本地缓存：

分布式缓存：使用Redis、Memcache将数据缓存在分布式缓存系统中。相对于本地缓存，分布式缓存单独部署，可以根据需求分配硬件资源。另外，服务器集群都可以访问分布式缓存，而本地缓存需要在服务器集群之间进行同步，实现难度和性能开销都非常大。

数据库缓存：

**四：CDN**

**五：缓存问题**

缓存穿透：指对一个一定不存在的数据进行请求，该请求会穿透缓存到达数据库。

解决方案：

对这些不存在的数据缓存一个空数据；

对这类请求进行过滤。

缓存雪崩：指由于数据没有被加载到缓存中，或缓存数据在同一时间大面积失效(过期)，或缓存服务器宕机，导致大量的请求都到达数据库。当发生缓存雪崩时，数据库无法处理这么大的请求，导致数据库崩溃。

解决方案：

合理设置缓存过期时间；

使用分布式缓存，分布式缓存中每一个节点只缓存部分的数据，当某个节点宕机时可以保证其他节点的缓存仍然可用；

使用缓存预热，避免在系统刚启动不久由于还未将大量数据进行缓存而导致缓存雪崩。

缓存一致性：指数据更新的同时能实时更新缓存。保证缓存一致性需要付出很大的代价，缓存数据最好是一些对一致性要求不高的数据，允许缓存数据存在一些脏数据。

**六：数据分布**

哈希分布

顺序分布

**七：一致性哈希**

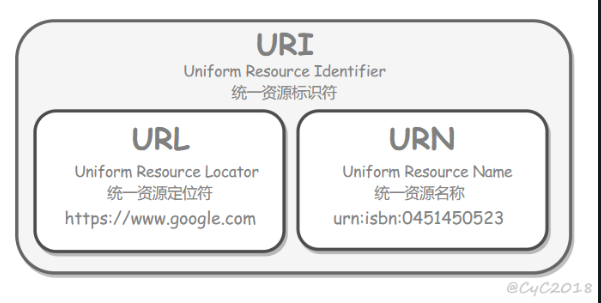
这个没有看懂、

**HTTP**

**一：基础概念**

统一资源标识符URI(Uniform Resource Identifier)：

包含统一资源定位符URL(Uniform Resource Locator)和统一资源名称URN(Uniform Resource Name)



**二：HTTP方法**

GET

HEAD

和GET方法类似，但是不返回报文实体主体部分。主要用于确认URL的有效性以及资

源更新的日期时间等。

POST

POST主要用于传输数据，GET主要用于获取数据。

PUT

上传文件。由于自身不带验证机制，任何人都可以上传文件。

PATCH

对资源进行部分修改。PUT也可以用于修改资源，但是只能整体替换，Patch允许部分

修改。

DELETE

删除文件。与PUT相反，并同样无验证机制。

OPTIONS

查询支持的方法。查询指定的URL能够支持的方法，会返回Allow:GET,POST这样的内容。

CONNECT

要求在于代理服务器通信时，建立隧道。使用 SSL（Secure Sockets Layer，安全套接层）

和 TLS（Transport Layer Security，传输层安全）协议把通信内容加密后经网络隧道传输。

TRACE

追踪路径。服务器会将通信路径返回给客户端。发送请求时，在 Max-Forwards 首部字

段中填入数值，每经过一个服务器就会减 1，当数值为 0 时就停止传输。通常不会使

用 TRACE，并且它容易受到 XST 攻击（Cross-Site Tracing，跨站追踪）。

**三：具体应用**

连接管理

1 短连接和长连接

当浏览器访问一个包含多个图片的html页面时，除了请求html页面资源，还会请求图片资源。如果每进行一次HTTP通信就要新建一个TCP连接，那么开销会很大。

长连接只需要建立一次TCP连接就能进行多次HTTP通信。

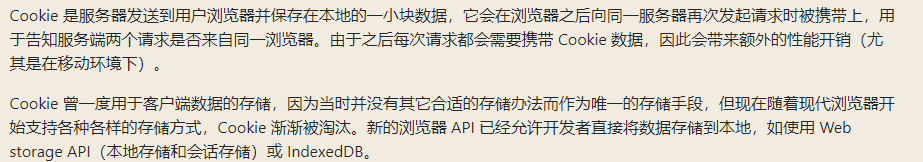


2 流水线

默认情况，HTTP请求只有当接受到当前请求的响应后，才会发出下一个请求。由于会受到网络延迟和带宽的限制，在下一个请求被发送到服务器之前，可能需要等待很长的时间。流水线是在同一条长连接上发出连续的请求，而不用等待响应返回，这样可以避免连接延迟。

**Cookie**

HTTP协议是无状态的，主要是为了让HTTP协议尽可能简单，使得它能够处理大量事务。HTTP/1.1引入Cookie来保存状态信息。



1 用途

会话状态管理

个性化设置

浏览器行为跟踪

2 创建过程

服务器发送的响应报文包含Set-Cookie首部字段，客户端得到响应报文后把Cookie内容保存在浏览器中。

客户端之后对同一个服务器发送请求时，会从浏览器中取出Cookie信息，并通过Cookie请求首部字段发送给服务器。

3 分类

会话期Cookie：浏览器关闭后会被删除，仅在会话期有效；

持久性Cookie：有一个过期时间(Expires)或有效期(max-age)。

4 作用域

5 JavaScript

document.cookie可创建Cookie，也可通过该属性访问非HttpOnly标记的Cookie。

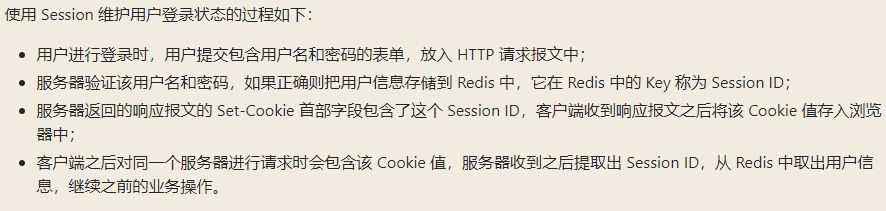
6 HttpOnly

7 Secure

标记为 Secure 的 Cookie 只能通过被 HTTPS 协议加密过的请求发送给服务端。但即便设置了 Secure 标记，敏感信息也不应该通过 Cookie 传输，因为 Cookie 有其固有的不安全性，Secure 标记也无法提供确实的安全保障。

8 Session

除了将用户信息通过Cookie存储在用户浏览器中，也可以利用Session存储在服务器端，更安全。可以存在服务器端的文件、数据库或内存中，也可以存在Redis这种内存数据库中，效率更高。



9 浏览器禁用Cookie

此时无法使用Cookie保存用户信息，只能使用Session。除此之外，不能将Session ID放在Cookie中，而是使用URL重写技术，将Session ID作为URL的参数进行传递。

10 Cookie和Session的选择

Cookie只能存储ASCII码字符串，Session可以存取任何类型数据；

Cookie 存储在浏览器中，容易被恶意查看。如果非要将一些隐私数据存在 Cookie 中，可以将 Cookie 值进行加密，然后在服务器进行解密；

对于大型网站，如果用户所有的信息都存储在 Session 中，那么开销是非常大的，因此不建议将所有的用户信息都存储到 Session 中。

**缓存**

1 优点

2 实现方法

3 Cache-Control

4 缓存验证

**内容协商**

通过内容协商返回最合适的内容，例如根据浏览器的默认语言选择返回中文界面还是英文界面。

1 类型

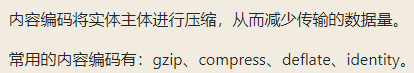
服务端驱动型：客户端设置特定的HTTP首部，服务器根据这些字段返回特定的资源。

代理驱动型：

2 Vary

在使用内容协商的情况下，只有当缓存服务器中的缓存满足内容协商条件时，才能使用该缓存，否则应该向源服务器请求该资源。

**内容编码**

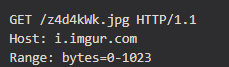


**范围请求**

如果网络出现中断，服务器只发送了一部分数据，范围请求可以使得客户端只请求服务端未发送的那部分数据，从而避免服务器重新发送所有数据。

1 Range

在请求的报文中添加Range首部字段指定请求的范围。



2 Accept-Ranges

响应首部字段Accept-Ranges用于告知客户端是否能处理范围请求，可以处理使用 bytes，否则使用 none。

3 响应状态码

**分块传输编码**

Chunked Transfer Coding，可以把数据分割成多块，让浏览器逐步显示页面。

**多部分对象集合**

一份报文主体内可含有多种类型的实体同时发送，每个部分之间用 boundary 字段定义的分隔符进行分隔，每个部分都可以有首部字段。

**虚拟主机**

HTTP/1.1使用虚拟主机技术，使得一台服务器拥有多个域名，并且在逻辑上可以看成多个服务器。

**通信数据转发**

1 代理

代理服务器接受客户端的请求，并且转发给其他服务器。

目的：

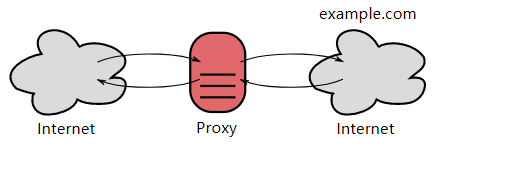
缓存

负载均衡

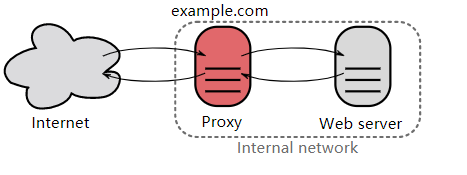
网络访问控制

访问日志记录

正向代理服务器：用户察觉得到正向代理的存在



反向代理服务器：位于内部网络中，用户察觉不到



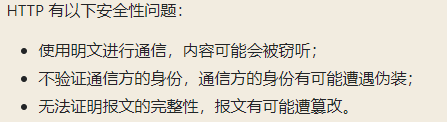
2 网关

与代理服务器不同，网管服务器会将HTTP转化为其他协议进行通信，从而请求其他非HTTP服务器的服务。

3 隧道

使用SSL等加密手段，在客户端和服务器之间建立一条安全的通信线路。

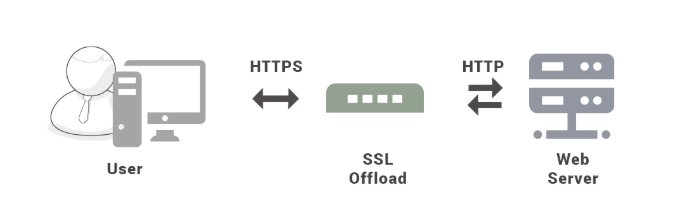
**六：HTTPS**



https不是新协议，而是让HTTP先和SSL通信，再由SSL和TCP通信，也就是说HTTPS使用

了隧道进行通信。

通过使用 SSL，HTTPS 具有了加密（防窃听）、认证（防伪装）和完整性保护（防篡改）



现阶段可以去学习的：Redis、深入学习C#、net core(eshoponcontainer可以学习了)