1. Chapter1: Basic Concept
2. Network

* Là một tập hợp các máy tính hoặc các thiết bị đầu cuối có thể gửi và nhận dữ liệu cho các thiết bị khác theo thời gian thực.
* Mỗi thiết bị trong mạn được goi là node. Node với đầy đủ chức năng của máy tính được gọi là *host*.
* Mỗi thiết node mạng đều có một địa chỉ là một chuỗi các byte không giống nhau, các địa chỉ này cũng có thể được đặt tên.
* Các mạng máy tính hiện đại hầu hết là mạng chuyển mạch gói: dữ liệu được chia thành các gói nhỏ và được xử lý khác nhau, các gói này đều có thông tin người gửi và người nhận. Điểm quan trọng nhất của cơ chế này là dữ liệu từ nhiều địa chỉ , nhiều luồng dữ liệu có thể dùng chung đường dây làm giảm chi phí để xây dựng. Lợi ích thứ 2 của chuyển mạch gói là có thể dùng hàm check sum để kiểm tra dữ liệu có bị
* *Protocol:* là một tập các luật để định nghĩa việc giao tiếp giữa các máy tính

1. Các lớp mạng

* Để dấu sự phức tạo của các ứng dụng và người dùng cuối, các khía cạnh giao tiếp mạng được phan chia thành nhiều lớp. Mỗi lớp chỉ giao tiếp trực tiếp với các lớn trên hoặc dưới nó
* Cac lớp mạng theo mô hình TCP/IP bao gồm:
  + Application Layer
  + Transport Layer (TCP, UDP)
  + Internet Layer (IP)
  + The host-to-network Layer(Ethenet, Wifi, LTE)
* *The Host-to-network Layer:* liên quan đến việc truyển tài dữ liệu (biên các tín hiệu số thành tín hiệu tương tự và truyền đi qua các hình thức: Wifi, cáp quang…)
* *Internet Layer* ( network layer): Giao thức tầng network định nghĩa cách mà các bit và byte dữ liệu được tổ chức thành các tập lớn gọi là cá gói và cơ chế đánh địa chỉ để các máy tính khác nhau có thể tìm thấy nhau. Giao thức phổ biến nhất là IP (Internet Protocol)
  + IP gồm 2 loại: Ipv4 dùng 32-bit để đánh địa chỉ và Ipv6 dùng 128-bit để đánh địa chỉ
* *Transport Layer:* Chịu trách nhiệm cho việc đảm bảo các package đến đúng địa chỉ và không bị mất dữ liệu.
  + Nếu gói tin bị mất sẽ yêu cầu sender gửi lại các gói tin này
  + Việc này được thực hiện bằng cách thêm header và các gói tin
  + Giao thức TCP(Transmission Control Protocol): là giao thức “chi phí cao” có thể thực hiện gửi lại các gói tin bị mất và đảm bảo việc sắp xếp các gói theo thứ tự đến người nhận và được gọi là giao thức tin cậy
  + Giao thức UDP( User Diagram Protocol) cho phép người nhận phát hiện các gói bị mất nhưng không đảm bảo các gói được sắp xếp theo đúng thứ tự và được gọi là giao thức không tin cậy. Tuy nhiên giao thức này có tốc độ nhanh hơn với TCP.
* *Appication Layer:* là tầng vận chuyển dữ liệu đến người dùng và phụ trách việc xử lý với các dữ liệu đó.

1. IP, TCP, UDP

* IP : được thiết kế để cho phép đa định tuyến giữa bất kỳ 2 điểm và định tuyển các gói dữ liệu các các điểm hư hại
* TCP: được thiết là là lớp trên của IP cho phép khả năng truyển tải lại các gói dữ liệu bị mất, ngoài ra giao thức còn có nhiệm vụ sắp xếp lại các gói tin theo thứ tự mà chúng được gửi đi.
* UPD: là một giao thức không tin cậy, nó không đảm bảo dữ liệu sẽ đến được người nhận và cũng không đảm bảo dữ liệu được sắp xếp theo thứ tự mà chúng được gửi đi. Giao thức này chỉ sử dụng trong trường hợp dữ liệu không thực sự quan trọng và việc mất dữ liệu không ảnh hưởng gì
* Địa chỉ IP và Domain Name:

1. STREAM

* I/O trong java được xây dựng thành các luông. Input Stream đọc dữ liệu, output stream ghi dữ liệu
* Stream là đồng bộ, khi muốn thực hiện đọc hoặc ghi 1 phần dữ liệu nào đó thì cần phải đợi các dữ liệu trước đó được thực hiện
* JAVA đề xuất nonblocking I/O sử dụng các kênh và bộ đệm (channel and buffer) giúp cải thiện hiệu năng trong một số trường hợp

1. Output Stream

* Class nền tảng là java.io.OutputStream
* Trong giao thức TCP/IP thường sẽ không gửi từng byte một mà sẽ sử dụng bộ đệm dữ liệu bằng cách thu thập các byte trên bộ nhớ và chỉ gửi chúng đi đến đích khi đủ số lượng hoặc đủ thời gian.
* Tuy nhiên khi số lượng buy nhỏ hơn độ lớ của bộ đệm sẽ xảy ra trường hợp dữ liệu không được gửi đi (do chưa đủ số lượng byte ) khi đó ta cần phải đẩy (flush) dữ liệu đi, cần thực hiện flush stream trước khi đóng nó lại
* Việc đóng stream bằng cách gọi method close () và cần thiết và quan trọng đẻ giải phóng tài nguyên mà stream đang nắm giữ (file, port, connection…). Java7 giới thiệu *try with resource* để thực hiện khai báo các biến stream trong khối try(), khi đó java sẽ tự call đến hàm clode của AutoCloseable

1. Input Stream

* Class cơ bản của java là java.io.InputStream, Các class con sử dụng các hàm để đọc dữ liệu. hàm cơ bản nhất là *read()* thực hiện đọc 1 byte từ nguồn và trả về giá trị int có giá trị từ 0-255, kết thúc stream sẽ trả về giá trị -1
* Input và Output có thể chậm vì thế các service liên quan đến I/O nên được thực hiện ở các thread riêng

1. THREAD

* Vấn đề xảy ra với các hệ thống client-server cũ là các thiết bị không đủ mạnh và mạng không đủ nhanh, Khi càng ngày số lượng người dùng tăng lên thì số lượng các request cũng tăng lên theo và khi đó cần có giải pháp để đáp ứng vấn đề này:
* Giải pháp 1: tái sử dụng các processes thay vì sinh ra mới, số lượng processer sẽ là cố định, Các request đến sẽ được gửi và queue và các tiến trình sẽ lấy từ queue ra xử lý
* Giải pháp 2: sử dụng các thread thay vị các tiến trình:
  + Các thread nhẹ hơn các tiến trình vì chúng dùng chung bộ nhớ và cho hiệu năng cao hơn
  + Ảnh hưởng của việc chạy nhiều các thread khác nhau đối với phần cứng bộ nhớ là tương đối nhỏ vì tất cả để chạy chung trong 1 tiến trình. Sử dụng threadpool để tái sử dụng các thread sẽ mang đến hiệu quả hơn.
  + Tuy nhiên sử dụng phương pháp này yêu cầu người lập trình cần viết các đoạn code phức tạp hơn đặc biệt là vấn đề safety và liveness.

1. INTERNET ADDRESS
2. URLs and URIs
3. HTTP

* HTTP(HyperText Transfer Protocol) định nghĩa cách mà client giao tiếp với server và cách mà server trả dữ liệu cho client. Nó đước sử dụng để truyển tải bất kỳ dữ liệu gì có thể chuyển hóa thành byte.

1. The protocol

* HTTP là giao thức tiêu chuẩn cho việc giao tiếp giữa web client và web server. Nó xác định:
  + Cách client và server thành lập kết nối
  + Cách client gửi yêu cầu đến server
  + Cách server trả lai dữ liệu cho client
  + Cách đóng kết nối
* Các kết nối HTTP sử dụng giao thức TCP/IP, với mỗi request bao gồm 4 bước:
  + Client mở kết nối TCP đến server với port mặc định 80 (port có thể cấu hình trong URL)
  + Client gửi thông điệp đến server để yêu cầu tài nguyên ở một đường dẫn xác định.
  + Server gửi phản hồi cho client. Phản hồi bao gồm response code cùng với đó là các thông tin khác.
  + Server đóng kết nối
* Ở phiên bản HTTP 1.1 nhiều req và res có thể được thực hiện trên một connection
* Mỗi req và res đều có form chung: header line, HTTP header, blank line và message body:

GET /index.html HTTP/1.1  
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.8; rv:20.0)  
Gecko/20100101 Firefox/20.0  
Host: en.wikipedia.org  
Connection: keep-alive  
Accept-Language: en-US,en;q=0.5  
Accept-Encoding: gzip, deflate  
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q

* Dòng đầu tiên goi là request line bao gồm method (GET), đường dẫn tới tài nguyên và HTTP version, các req client bao gồm các thông tin dưới dạng *Key: Value*
* Response bắt đầu với status line và sau đó là các header mô tả khác sử dụng cấu trúc “name: value”
* Keep Alive: Với các phiên bản HTTP1.1 thì hoàn toàn có thể giữ các kết nối TCP để gửi req và nhận res khi client kết nối đến cùng server.

1. HTTP Methods

* Các giao tiếp với HTTP server tuân theo kiết kế request-response: 1 stateless request sẽ đi cùng với 1 stateless ressponse. Mỗi request sẽ bao gồm 3 phần:
  + Dòng đầu tiên chứa method và thông tin đường dẫn đến resource
  + Header chưa các trường có dang *name:value*
  + Request body chưa các thông tin của tài nguyên (chỉ có ở method POST và PUT)
* Có 4 HTTP method chính bao gồm:
  + GET: Thực hiện lấy các thể hiện của tài nguyên, method này có hiệu ứng phụ (side-effect) có thể lặp lại mà không cần quan tâm đên thành công hay thất bại. Thông thường dữ liệu ở method này được *cache.*
  + PUT: thực hiện upload tài nguyên lên server theo URL. Method này không có tác dụng phụ tuy nhiên vẫn có thể thực hiện lặp lại mà không cần quan tâm đến kết quả lỗi vì upload cùng tài nguyên lên cùng một URL sẽ được ghi đè.
  + DELETE: Loại bỏ resource từ một URL xác định. Method này cũng có thể thực hiện lặp lại mà không cần quan tâm kết quả thất bại hay thành công.
  + POST: Thực hiện upload một tài nguyên lên server tuy nhiên không thể xác định được server sẽ làm gì với tài nguyên đó. Vì vậy cần sử dụng method này một cách an toàn, không nên thực hiện lặp
* Các hành động an toàn mà không cần commit dữ liệu thì nên sử dụng GET , còn các hành động cần commit dữ liệu thì nên sử dụng POST.
* Với trường hợp dữ liệu đầu vào request lớn (hơn 2000 ký tự) thì mới nên sử dụng POST, trường hợp còn lại thì nên sử dụng GET

1. REQUEST BODY

* Với các method POST và PUT thì các request sẽ có thêm body tuy nhiên trong header sẽ có thêm thông tin về body:
  + *Content-Type:* Xác định kiểu MIME media của các byte
  + *Content-Length:* Số lượng byte trong body

1. Cookies

* Cookies lưu trạng thái của client-site giữa các connection, được gửi từ server lên client và ngược lại trong các HTTP header của request và response
* Cookies được sử dụng ở server để lưu các thông tin như sessionID, thông tin xác thực login, có thể set nhiêu cookies trong header

1. URLConnections

* Là một abstract class biểu diễn cho môt kết nối tới một tài nguyên được xác định qua URL
* Class này bao gồm 2 mục đích:
  + Cung cấp khả năng tương tác với serve (Đặc biệt là HTTP server), nó có thể kiểm tra và thêm dữ liệu vào header với các req và res

1. Sockets for Client

* Dữ liệu được truyển tải quả Internet thông qua các gói được gọi là Datagram. Mỗi datagram bao gồm header và payload:
  + Header bao gồm các thông tin: Địa chỉ và port nguồn và đích của các gói, checksum để phát hiện thiếu dữ liệu và nhiều thông tin khác để đảm bảo việc truyền tải là đáng tin cậy
  + Payload chỉ chứa dữ liệu
* Việc đảm bảo toàn vẹn dữ liệu là khá phức tạp, Socket giúp lập trình viên tránh được những phức tạp này

1. Sử dụng Socket

* Socket là một kết nối giữa 2 máy chủ (host). Nó bao gồm 7 hành động cơ bản sau:
  + Kết nối đến máy tính từ xa
  + Gửi dữ liệu
  + Nhận dữ liệu
  + Đóng kết nối
  + Trói buộc với một port
  + Lắng nghe dữ liệu tới
  + Chấp nhận một kết nối từ một máy khác
* Java Socket cung cấp các hàm tương ứng với 4 hành động ban đầu
* Đầu tiên kết nối được thành lập, máy chủ local và remote thực hiện luồng dữ liệu đầu vào và đẩu ra từ socket và sử dụng luông đó để nhận và gửi dữ liệu
* Các kết nối này là song công (full-duplex), cả 2 máy chủ gửi và nhận dữ liệu một cách đồng thời,
* Khi gửi nhận dữ liệu hoàn thành một trong hai bên sẽ đóng kết nối.