LẬP TRÌNH DI ĐỘNG

Bài 12: networking với android

Nhắc lại bài trước

- Sensor là cảm biến giúp thiết bị android ghi nhận được các thông số về môi trường bên ngoài
  - Hệ thống các sensor làm cho app trở nên thông minh hơn nếu biết khai thác và phản hồi phù hợp tình huống
- Nguyên tắc làm việc chung
  - Chú ý tiết kiệm năng lượng, tắt sensor khi không dùng
  - Chú ý xử lý số liệu và độ chính xác của từng loại sensor
  - Mỗi loại sensor có cách làm việc khác nhau cần tìm hiểu chi tiết và có phương pháp xử lý số liệu phù hợp
  - Cần làm trên thiết bị thật và tinh chỉnh dần dần sai số
  - Nên kết hợp nhiều loại sensor nếu có thể

Nội dung

1. Giới thiệu chung về networking
2. Giao thức kiểu TCP
   - Nguyên tắc hoạt động
   - Lập trình
   - Quá trình giao tiếp
3. Giao thức kiểu UDP
   - Nguyên tắc hoạt động
   - Gửi gói tin
   - Nhận gói tin
4. HttpClient và web services

Phần 1

Giới thiệu chung về networking

Giới thiệu chung về networking

- Android làm việc với mạng dựa trên chuẩn IP
- Ở mức độ thiết bị, Android OS hỗ trợ nhiều cách kết nối và truyền dữ liệu
  - HttpClient để giao tiếp với server qua giao thức HTTP
  - Socket và ServerSocket để thực hiện truyền dữ liệu theo kiểu TCP
  - DatagramSocket để thực hiện truyền dữ liệu kiểu UDP
  - BluetoothSocket và BluetoothServerSocket để giao tiếp qua Bluetooth (TCP)
  - Dùng NfcManager để thực hiện giao tiếp NFC

Nguyên tắc dùng network cho app

- Không làm việc với network trên UI thread
- Mã chịu lỗi: lỗi có thể xảy ra bất kì lúc nào
  - Mạng bị ngắt, chập chờn
  - Gói tin bị mất trên đường truyền
  - I/O stream bị block
- Luôn nghĩ tới tiết kiệm năng lượng: ứng dụng càng dùng network nhiều càng hao pin (mức tiêu thụ pin của network chỉ sau màn hình)
- Hỗ trợ nhiều giao thức: có nhiều kiểu kết nối, mỗi kiểu kết nối có những ưu/nhược điểm riêng

Phần 2

Giao thức kiểu TCP

TCP – nguyên tắc hoạt động

- TCP là họ các giao thức IP làm việc theo nguyên lý "nghe và gọi"
  - Server (máy chủ): luôn ở trạng thái chờ phục vụ
  - Client (máy khách): chủ động yêu cầu kết nối và gửi yêu cầu phục vụ cho máy server
  - Khi có kết nối giữa client và server:
    - Hai bên giữ đường truyền và trao đổi dữ liệu liên tục
    - Dữ liệu gửi đi được đảm bảo chất lượng truyền
  - Một server phục vụ cùng lúc nhiều client
  - Một kết nối chiếm một port (cổng) trên cả server và client, một IP có 65536 port (một số port dùng riêng)

TCP – lập trình

- ServerSocket: class phía server
  - Tạo server: new ServerSocket(SERVERPORT);
  - Nhận kết nối: serverSocket.accept();
  - Gửi và Nhận dữ liệu thông qua I/O stream
- Socket: class phía client
  - Kết nối tới server: new Socket(server_ip, port);
  - Gửi và Nhận dữ liệu thông qua I/O stream
- Giao thức: ngôn ngữ để nói chuyện với nhau
  - Tùy vào loại dịch vụ: HTTP, FTP, SMTP, TELNET, IRC, …
  - Tự tạo giao thức dựa trên nhu cầu thực tế

TCP – quá trình giao tiếp

SERVER	CLIENT
Mở cổng dịch vụ	-
-	Yêu cầu kết nối tới server
Chấp nhận kết nối	Chấp nhận kết nối
Tạo thread riêng	-
Bắt đầu trao đổi dữ liệu	Bắt đầu trao đổi dữ liệu
…	…
Kết thúc trao đổi dữ liệu	Kết thúc trao đổi dữ liệu
Đóng kết nối	Đóng kết nối

TCP – ví dụ về remote control

- Mục tiêu: xây dựng ứng dụng cho phép dùng thiết bị cầm tay điều khiển máy tính
- Triển khai:
  - Đơn giản hóa vấn đề: điều khiển một số thao tác cơ bản trong PowerPoint như lật trang, phóng to, trở về màn hình soạn thảo,…
  - Phía server (PC): một ứng dụng nhỏ viết bằng Java
  - Phía client (mobile): ứng dụng gửi các lệnh
  - Sử dụng command pattern: đây là nguyên mẫu phù hợp với việc xây dựng server tổng quát, có thể mở rộng bởi các plug-ins phù hợp

TCP – ví dụ về remote control

- Phía PC:
  - Tạo một ServerSocket qua cổng 5555 (số này tùy chọn)
  - Nhận lệnh gửi từ xa ở dạng chuỗi (next, back, end, home, F5, ECS,…)
  - Ứng với mỗi chuỗi lệnh, tạo ra các sự kiện bàn phím tương ứng thông qua class java.awt.Robot
- Phía android device:
  - Kết nối tới server qua cổng 5555
  - Mỗi khi có sự kiện bấm phím, thì gửi chuỗi tương ứng cho PC

Phần 3

Giao thức kiểu UDP

Giao thức kiểu UDP

- UDP là họ các giao thức IP làm việc theo nguyên lý "gửi và quên"
  - Không có khái niệm server/client
  - Máy gửi:
    - Chuẩn bị dữ liệu, đóng gói vào DatagramPacket
    - Mở cổng gửi thông qua DatagramSocket
  - Máy nhận:
    - Mở cổng và nhận dữ liệu
  - Dữ liệu có thể bị mất trên đường truyền
  - Gói dữ liệu có thể rất lớn, phù hợp với các ứng dụng media, realtime hoặc game

UPD – ví dụ gửi gói tin

```
public class MainActivity extends Activity {
    EditText ip, port, message;
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ip = (EditText) findViewById(R.id.editText1);
        port = (EditText) findViewById(R.id.editText2);
        message = (EditText) findViewById(R.id.editText3);
    }
    public void btnSend(View v) {
        Client c = new Client(ip.getText().toString(),
            port.getText().toString(), message.getText().toString());
        c.start();
    }
}
```

UPD – ví dụ gửi gói tin

```
class Client extends Thread {
    String ip, port, text;
    public Client(String i, String p, String t) {
        ip = i; port = p; text = t;
    }
    public void run() {
        byte[] data = text.getBytes();
        DatagramSocket s = new DatagramSocket();
        DatagramPacket p = new DatagramPacket(data, data.length,
            InetAddress.getByName(ip), Integer.parseInt(port));
        s.send(p);
        s.close();
    }
}
```

UDP – ví dụ nhận gói tin

```
public class MainActivity extends Activity {
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ReadData s = new ReadData((TextView) findViewById(R.id.tv1));
        new Thread(s).start();
    }

    class ReadData implements Runnable {
        Handler x = new Handler();
        TextView tv;
        String text;
        public ReadData(TextView abc) {
            tv = abc;
        }
```

UDP – ví dụ nhận gói tin

```
        public void run() {
            byte[] msg = new byte[1000];
            DatagramSocket s = new DatagramSocket(12345);
            DatagramPacket p = new DatagramPacket(msg, msg.length);
            for (int i = 0; i < 10; i++) {
                s.receive(p);
                text = new String(msg, 0, p.getLength());
                x.post(new Runnable() {
                    public void run() { tv.append(text + "\n"); }
                });
            }
            s.close();
        }
    }
}
```

Phần 4

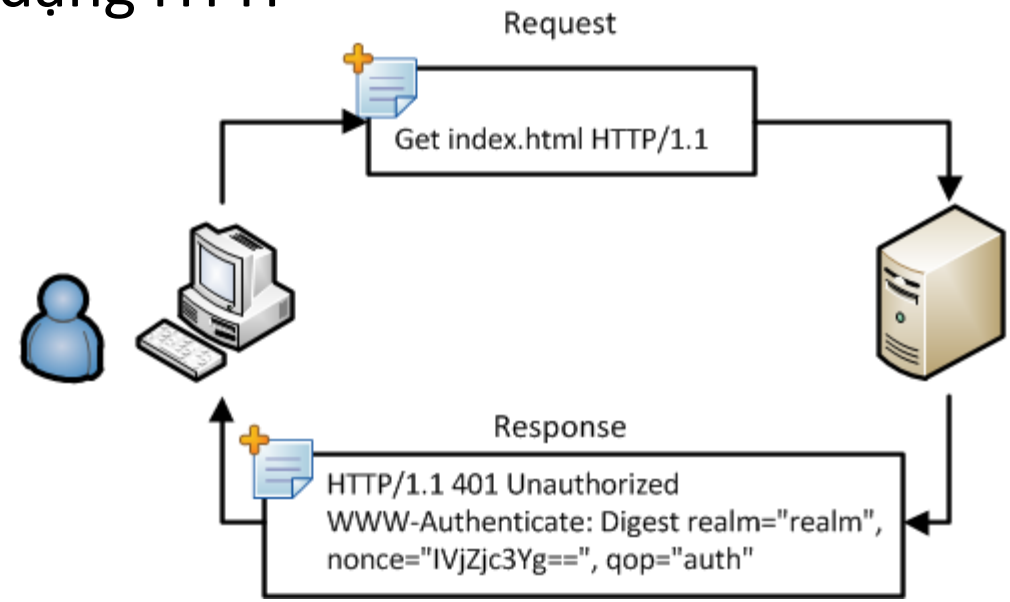
HttpClient và web services

HttpClient

```
String url = "http://www.google.com/search?q=httpClient";

HttpClient client = new DefaultHttpClient();
HttpGet request = new HttpGet(url);
HttpResponse response = client.execute(request);

System.out.println("Response Code : " +
    response.getStatusLine().getStatusCode());

BufferedReader rd = new BufferedReader(new
    InputStreamReader(response.getEntity().getContent()));

StringBuffer result = new StringBuffer();
String line = "";
while ((line = rd.readLine()) != null) result.append(line);
```

HttpClient

- Giao thức HTTP: là một loại TCP
  - http://www.w3.org/Protocols/rfc2616/rfc2616.html
  - Chuẩn trao đổi dữ liệu HTML giữa client và server
  - Dữ liệu trao đổi ở dạng HTML
  - Ngôn ngữ trao đổi ở dạng HTTP
- Ưu điểm:
  - Sử dụng rộng rãi
  - Ít bị chặn bởi proxy
- Nhược điểm:
  - Tốc độ chậm
  - Dễ bị dò (do thám)



Web services

- Class quan trọng (cùng với HttpPos/HttpGet) giúp client giao tiếp với HTTP server
  - Không có những class này ta vẫn làm việc được với web server bằng cách trực tiếp gửi gói tin TCP và phân tích kết quả nhận được
- Dùng giao thức HTTP để hiện thực hóa web service
  - Web services = các dịch vụ trên server có thể triệu gọi (yêu cầu) bằng cách gửi lệnh theo chuẩn HTML (và nhận kết quả cũng vậy)
  - Client mã hóa lời gọi thành dạng http request
  - Server xử lý và trả về kết quả dạng HTML
  - Client đọc kết quả trả về và tiếp tục tương tác với server

Web services

- Ví dụ về remote service: dịch chuỗi ở ngôn ngữ A sang ngôn ngữ B
- Thiết kế giao thức:
  - Request: chuỗi cần dịch + A + B
  - Response: kết quả + thông tin thêm
- Chọn giao thức phù hợp: tùy vào yêu cầu của khách hàng và sở trường của nhóm dev
  - Tự thiết kế giao thức riêng: giống như thiết kế giao thức HTTP (thường là đơn giản hơn)
  - Sử dụng các giao thức thuộc loại universal (có tính tổng quát hóa cao, chẳng hạn như HTTP)

Web services

- Tự thiết kế giao thức
  - Ưu điểm: tối ưu nhất về kiến trúc, tốc độ, băng thông, cách thức giao tiếp và chi phí vận hành
  - Nhược điểm: phải tự xử lý mọi thứ (sửa lỗi, đồng bộ, cân bằng tải, tương thích mã với các loại server,…)
- Sử dụng các giao thức phổ quát
  - Ưu điểm: được hỗ trợ bởi các hệ thống đã có, chi phí tối thiểu về code và maintains, thời gian phát triển nhanh
  - Nhược điểm: chậm hơn, tốn băng thông hơn, chi phí vận hành cao hơn

JSON vs XML

- Dữ liệu trao đổi giữa client và server để ở dạng nào?
  - XML là lựa chọn hiển nhiên do sự hỗ trợ của ngôn ngữ
  - JSON gần đây được ưa thích hơn do được hỗ trợ tốt từ server và tính đơn giản của định dạng
  - www.json.org cung cấp sẵn các cách xử lý JSON bằng nhiều ngôn ngữ lập trình khác nhau
- Khai thác remote service:
  - Biết về dịch vụ: làm việc với nhà cung cấp (microsoft, google, facebook, dropbox,… đều có remote services)
  - Biết cách thức giao tiếp với server: dựa trên tài liệu do nhà cung cấp phát hành

JSON vs XML

```
{"menu": { "id": "file", "value": "File",
    "popup": {
        "menuitem": [
            {"value": "New", "onclick": "CreateNewDoc()"},
            {"value": "Open", "onclick": "OpenDoc()"},
        ]
    }
}}
```

```
<menu id="file" value="File">
    <popup>
        <menuitem value="New" onclick="CreateNewDoc()" />
        <menuitem value="Open" onclick="OpenDoc()" />
    </popup>
</menu>
```