Systemy operacyjne na platformach mobilnych

Wykład 4

Grzegorz Jabłoński, Piotr Perek
Katedra Mikroelektroniki i Technik Informatycznych

Zagadnienia wykładu

- Menadżer połączeń
- Usługi HTTP
  - Klient HTTP
  - Żądanie
  - Odpowiedź
  - Realizacja żądania (GET/POST)
  - Obsługa wyjątków
  - Klienci
- Zadania drugoplanowe
- Obsługa plików XML

Menadżer połączeń

- Klasa ConnnectivityManager służy do zarządzania łączami sieciowymi
- Podstawowe zadania tej klasy to:
  - Monitorowanie połączeń sieciowych (Wi-Fi, GPRS, UMTS itp.)
  - Wysyłanie intencji broadcastowych, gdy nastąpią zmiany z połączeniami sieciowymi
  - Próba "fail-over" do innej sieci, gdy połączenie z siecią jest stracone
  - Dostarczenie interfejsu API, który pozwoli aplikacjom na sprawdzenie stanu dostępnych sieci

Sprawdzenie stanu sieci

```
public void onStart() {
     super.onStart();
     ConnectivityManager cMgr = (ConnectivityManager)
                     this.getSystemService(Context.CONNECTIVITY_SERVICE);
     NetworkInfo netInfo = cMgr.getActiveNetworkInfo();
     if(info != null && info.isConnected())
     {
        return true;
     }
}
```

Klient HTTP

- SDK Androida dostarcza zmodyfikowaną wersję klienta HTTP fundacji Apache: org.apache.http.client.HttpClient (API podobne do używanego w aplikacjach JEE)
- Ogólny wzorzec wykorzystania klienta HTTP:
  - utworzenie/uzyskanie instancji HttpClient
  - utworzenie instancji metody HTTP (GetMethod lub PostMethod)
  - ustawienie parametrów HTTP
  - wysłanie żądania HTTP (z użyciem HttpClient)
  - przetworzenie odpowiedzi HTTP

AndroidManifest.xml

- Aby aplikacja mogła uzyskać dostęp do informacji o połączeniach sieciowych należy dodać do pliku AndroidManifest.xml pozwolenie *android.permission.ACCESS_NETWORK_STATE*

- Korzystanie z Internetu wymaga dodania do pliku AndroidManifest.xml pozwolenia *android.permission.INTERNET*

Gniazdo serwera TCP

```
ServerSocket server = new ServerSocket(PORT, 1);
while(true)
{
     Socket client = server.accept();
     System.out.println("Client connected");
     while(true)
     {
          BufferedReader reader = new BufferedReader(new InputStreamReader(
                                                  client.getInputStream()));
          System.out.println("Read from client");
          String textLine = reader.readLine() + "\n";
```

Gniazdo serwera TCP

```
        if(textLine.equalsIgnoreCase("EXIT\n"))
        {
            System.out.println("EXIT invoked, closing client");
            break;
        }
        BufferedWriter writer = new BufferedWriter(
        new OutputStreamWriter(client.getOutputStream()));
        System.out.println("Echo input to client");
        writer.write("ECHO from server: "+ textLine,0,textLine.length()+18);
        writer.flush();
    }
    client.close();
}
```

Gniazdo klienta

```
String output = callSocket(
         ipAddress.getText().toString(),
         port.getText().toString(),
         socketInput.getText().toString() );
private String callSocket(String ip, String port, String socketData)
{
     Socket socket = null;
     BufferedWriter writer = null;
     BufferedReader reader = null;
     String output = null;
     try{
          socket = new Socket(ip, Integer.parseInt(port));
          writer = new BufferedWriter(new OutputStreamWriter(
                                        socket.getOutputStream()));
```

Gniazdo klienta

```
    reader = new BufferedReader(new InputStreamReader(
                                            socket.getInputStream()));
    String input = socketData;
    writer.write(input + "\n", 0, input.length() + 1);
    writer.flush();
    output = reader.readLine();
    socketOutput.setText(output);

    // send EXIT and close
    writer.write("EXIT\n", 0, 5);
    writer.flush();

    // catches and reader, writer, and socket closes omitted for brevity
    // onCreate omitted for brevity

    return output;
}
```

Proste żądanie HTTP

```
String output = getHttpResponse(”www.google.pl”);
private String getHttpResponse(String location)
{
    String result = null;
    URL url = null;
    try
    {
        url = new URL(location);
    }
    catch(MalformedURLException e)
    {
        // log and or handle
    }
```

Proste żądanie HTTP

```
if(url != null)
{
    try
    {
        HttpURLConnection urlConn = (HttpURLConnection)url.openConnection();
        BufferedReader in = new BufferedReader(
                            new InputStreamReader(urlConn.getInputStream()));
        String inputLine;
        int lineCount = 0; // limit lines for example
        while((lineCount < 10) && ((inputLine = in.readLine()) != null)){
            lineCount++;
            result += "\n" + inputLine;
        }
        in.close();
        urlConn.disconnect();
    }
    catch(IOException e)
    { /* log and or handle */ }
    }
    else
    { /* log and or handle */ }
    return result;
}
```

Klient HTTP

- Domyślną implementacją interfejsu HttpClient jest klasa org.apache.http.impl.client.DefaultHttpClient
- Utworzenie instancji klienta:

```
HttpClient client = new DefaultHttpClient();
```

- Poprzez konstruktor można także podać:
  - menedżera połączenia org.apache.http.conn.Client-ConnectionManager
  - nagłówki HTTP org.apache.http.params.HttpParams

Żądanie HTTP

- Żądanie HTTP jest reprezentowane przez interfejs *org.apache.http.HttpRequest*
- Interfejs *org.apache.http.client.methods.HttpUriRequest* dziedziczy po interfejsie HttpRequest dostarczając metod użytkowych, m.in.:

```
public String getMethod() //nazwa metody, której używa żądanie
public URI getURI() //URI, którego żądanie używa
```

- Implementacjami tego interfejsu są m.in. klasy (odpowiadające metodom protokołu HTTP): HttpGet, HttpHead, HttpPost, HttpDelete, HttpPut, HttpTrace, HttpOptions

Odpowiedź HTTP

- Odpowiedź HTTP jest reprezentowana przez interfejs *org.apache.http.HttpResponse*
- Metoda *getEntity()* umożliwia dostęp do treści odpowiedzi przesłanej przez serwer
- Obiekt typu HttpResponse jest zwracany jako wynik metody *execute()* klienta HTTP.

Żądanie typu GET

- Przeznaczone głównie do odbierania danych
- Parametry żądania podawane jawnie w adresie
- Długość zapytania ograniczona do 2048 znaków

Przykład wysłania żądania typu GET:

```
HttpClient client = new DefaultHttpClient();
HttpGet request = new HttpGet("http://www.abc.pl/?sub=addd");
HttpResponse response = client.execute(request);
```

Żądanie typu POST

- Umożliwia przekazywanie danych do serwera zarówno w postaci parametrów tekstowych, jak i plików binarnych
- Parametry żądania przesyłane w formie niejawnej
- Brak ograniczeń w długości żądania

Przykład wysłania żądania typu POST:

```
HttpClient client = new DefaultHttpClient();
HttpPost request = new HttpPost("http://www.abc.pl/android/");
List<NameValuePair> postParams = new ArrayList<NameValuePair>();
postParams.add(new BasicNameValuePair("param1", "val1"));
postParams.add(new BasicNameValuePair("param2", "val2"));
UrlEncodedFormEntity formEntity =
                        new UrlEncodedFormEntity(postParams);
request.setEntity(formEntity);
HttpResponse response = client.execute(request);
```

AndroidHttpClient

- Klasa android.net.http.AndroidHttpClient implementuje
- interfejs HttpClient i ułatwia tworzenie klienta HTTP
- Klasa dostarcza ustawień domyślnych oraz dodatkowej logiki dostosowanej do aplikacji Androida, m.in.:
  - timeout'y (dla connection i socket) ustawione na 20 s
  - domyślny menedżer połączenia: ThreadSafeClient-ConnManager
- Uzyskanie klienta metodą fabrykującą:

```
AndroidHttpClient httpClient
        = AndroidHttpClient.newInstance("my-http-agent-string");
```

Użycie klienta AndroidHttpClient

- Wywołanie metody *execute()* musi nastąpić w wątku innym niż wątek główny UI
- Po zakończeniu połączenia należy na kliencie wywołać metodę *close()*, w celu poprawnego zwolnienia pamięci
- Klient dostarcza metody umożliwiające kompresję danych przed ich wysłaniem:

```
static AbstractHttpEntity getCompressedEntity(byte[] data,
                                              ContentResolver res)
static longgetMinGzipSize(ContentResolver resolver)
static InputStream getUngzippedContent(HttpEntity entity)
static void modifyRequestToAcceptGzipResponse(HttpRequest request)
```

Zadania drugoplanowe

- Czasochłonne zadania (np. czysto obliczeniowe) należy wykonywać w osobnym wątku – można wykorzystać do tego celu klasę Thread
- To podejście nie sprawdzi się, jeżeli w trakcie lub po zakończeniu zadania trzeba będzie uaktualnić UI
- Aktualizacja interfejsu użytkownika nie jest bezpieczna wielowątkowo i musi być wykonywana tylko z wątku głównego

AsyncTask

- Abstrakcyjna klasa generyczna *android.os.AsyncTask*<*Params*, *Progress*, *Result*> ułatwia realizację zadań asynchronicznych poprzez wyraźną separację zadania wykonywanego w tle (w wątku roboczym) i wyników, które aktualizują interfejs użytkownika (w wątku głównym UI)
- Typy generyczne wykorzystywane w klasie
  - Params – typ parametrów przekazywanych do zadania
  - Progress – typ wyników pośrednich publikowanych w trakcie pracy w tle
  - Result – typ wyniku zadań wykonywanych w tle
  - jeśli dany typ nie jest używany, należy użyć typu void

Cykl życia zadań drugoplanowych

Cykl życia zadań drugoplanowych jest ściśle związany z wywoływaniem odpowiednich metod klasy AsyncTask

```
protected void onPreExecute();
```

- Wywoływana w wątku głównym UI, bezpośrednio po uruchomieniu zadania
- Wykorzystywana zwykle do inicjalizacji zadania, np. wyświetlenia paska postępu

Cykl życia zadań drugoplanowych

```
protected abstract Result doInBackground(Params... params);
protected final void publishProgress(Progress... values);
```

- Metoda *doInBackground()* jest wywoływana w wątku roboczym, po zakończeniu metody *onPreExecute()*
- Wykorzystywana do wykonywania czasochłonnych obliczeń w tle
- Może przyjąć parametry wejściowe i zwrócić wynik (przekazywany do następnej fazy)
- W metodzie *doInBackground()* można wywołać metodę *publishProgress()* – spowoduje to wywołanie zwrotne do UI umożliwiające publikację wyników pośrednich

Cykl życia zadań drugoplanowych

```
protected void onProgressUpdate(Progress... values);
```

- Jest wywoływana w wątku głównym UI, w odpowiedzi na wywołanie metody *publishProgress()*
- Wykorzystywana do aktualizacji stanu interfejsu użytkownika bez przerywania zadania trwającego w tle
- Dokładny moment wywołania metody nie jest określony

Cykl życia zadań drugoplanowych

```
protected void onPostExecute(Result result);
```

- Jest wywoływana w wątku głównym UI, po zakończeniu obliczeń w tle
- Wynik obliczeń jest przekazywany jako parametr

Uruchamianie zadań drugoplanowych

- Instancję zadania (typu AsyncTask) należy utworzyć w wątku głównym UI
- Metodę *execute()* trzeba wywołać w wątku głównym UI
- Nie należy samodzielnie wywoływać metod: *onPreExecute(), onPostExecute(), doInBackground(), onProgressUpdate()*
- Zadanie można uruchomić tylko raz (kolejne uruchomienie spowoduje zgłoszenie wyjątku)

```
DownloadTask task = new DownloadTask(DownloadActivity.this);
String url = urlField.getText().toString();
String filename = filenameField.getText().toString();
task.execute(url, filename, "GET");
```

Przerywanie zadań drugoplanowych

- Zadanie może być przerwane w każdym momencie poprzez wywołanie metody *cancel()*
- Wywołanie metody *cancel()* spowoduje wywołanie metody *onCancelled()* zamiast *onPostExecute()* po wyjściu z *doInBackground()*
- Informację o tym czy została wywołana metoda *cancel()* i wątek powinien być anulowany można pobrać korzystając z metody *isCancelled()*
- Metoda *isCancelled()* powinna być periodycznie wywoływana z *doInBackground()* w celu sprawdzenia, czy zadanie ma być zakończone

AsyncTask - przykład

```
public class DownloadTask extends AsyncTask<String, Integer, String>
{
    private Context ctx;
    private ProgressDialog dialog;
    private TextView message;
    DownloadTask(Context ctx) {
        this.ctx = ctx;
    }

protected void onPreExecute()
{
    super.onPreExecute();
    message = (TextView) ((Activity) ctx).findViewById(R.id.result);
    message.setText("");
    dialog = new ProgressDialog(ctx);
    dialog.setMessage("Downloading file...");
    dialog.setCancelable(false);
    dialog.setIndeterminate(false);
    dialog.setMax(100);
    dialog.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
    dialog.show();
}
```

AsyncTask - przykład

```
protected String doInBackground(String... params) {
    return downloadFile(params);
}

protected void onProgressUpdate(Integer... values) {
    super.onProgressUpdate(values);
    dialog.setProgress(values[0]);
}

protected void onPostExecute(String result) {
    super.onPostExecute(result);
    dialog.dismiss();
    message.setText(result);
}
```

AsyncTask - przykład

```
public String downloadFile(String... params)
{
    String path = params[0];
    String filename = params[1];
    String method = params[2];
    int count;
    String result = "";
    try{
        OutputStream output = new FileOutputStream("/sdcard/" + filename);
        HttpClient client = new DefaultHttpClient();
        HttpResponse response = null;
        if("GET".equals(method)) {
            HttpGet request = new HttpGet(path + "?file=" + filename);
            response = client.execute(request);
        } else if("POST".equals(method)) {
            HttpPost request = new HttpPost(path);
            List<NameValuePair> postParams = new ArrayList<NameValuePair>();
            postParams.add(new BasicNameValuePair("file", filename));
            UrlEncodedFormEntity formEntity = new UrlEncodedFormEntity(
                                                        postParams);
            request.setEntity(formEntity);
            response = client.execute(request);
        }
```

AsyncTask - przykład

```
        HttpEntity entity = response.getEntity();
        InputStream input = entity.getContent();
        int dataLength = (int) entity.getContentLength();
        byte data[] = new byte[1024];
        long total = 0;
        while((count = input.read(data)) != -1)
        {
            total += count;
            int currentProgress = (int) (total * 100 / dataLength);
            publishProgress(currentProgress);
            output.write(data, 0, count);
        }
        output.flush();
        output.close();
        input.close();
        result = "Download of " + filename + " finished... \n(" + total
                                              + "B of " + dataLength + "B)";
    } catch(Exception e) {
        result = "Download of " + filename + " failed";
    }
return result;
}
};
```

Obsługa plików XML

- W Androidzie istnieje kilka różnych parserów dla plików XML np.:
  - XmlPullParser
  - DOM
  - SAX

XmlPullParser

- Rekomendowany jako najwydajniesze rozwiązanie do parsowania plików XML
- Sposoby tworzenia parsera:

```
xmlpullparser parser = Xml.newPullParser();
```

```
XmlPullParserFactory pullParserFactory;
    try {
        pullParserFactory = XmlPullParserFactory.newInstance();
        XmlPullParser parser = pullParserFactory.newPullParser();
    } catch (XmlPullParserException e) {

        e.printStackTrace();
    }
```

Przykład

```
<?xml version="1.0" encoding="UTF-8"?>
   <products>
      <product>
         <productname>Jeans</productname>
         <productcolor>red</productcolor>
         <productquantity>5</productquantity>
      </product>
      <product>
         <productname>Tshirt</productname>
         <productcolor>blue</productcolor>
         <productquantity>3</productquantity>
      </product>
      <product>
         <productname>shorts</productname>
         <productcolor>green</productcolor>
         <productquantity>4</productquantity>
      </product>
   </products>
```

XmlPullParser - parsowanie

```
private void parseXML(XmlPullParser parser) throws
                                    XmlPullParserException,IOException
{
    ArrayList<product> products = null;
   int eventType = parser.getEventType();
   Product currentProduct = null;

   while(eventType != XmlPullParser.END_DOCUMENT){
       String name = null;
       switch (eventType){
          case XmlPullParser.START_DOCUMENT:
             products = new ArrayList();
             break;
```

XmlPullParser - parsowanie

```
case XmlPullParser.START_TAG:
    name = parser.getName();
   if (name == "product"){
       currentProduct = new Product();
   } else if (currentProduct != null){
       if (name == "productname"){
           currentProduct.name = parser.nextText();
       } else if (name == "productcolor"){
           currentProduct.color = parser.nextText();
       } else if (name == "productquantity"){
           currentProduct.quantity= parser.nextText();
       }
    }
    break;
```

XmlPullParser - parsowanie

```
            case XmlPullParser.END_TAG:
                name = parser.getName();
                if(name.equalsIgnoreCase("product") && currentProduct != null)
                {
                    products.add(currentProduct);
                }
            }

        eventType = parser.next();
        }

    printProducts(products);
}
```

DOM parser

- Zgodny z modelem DOM (Document Object Model)
- Działanie identyczne jak w przypadku Javy
- Tworzenie parsera:

```
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
DocumentBuilder db = dbf.newDocumentBuilder();

InputSource is = new InputSource();
is.setCharacterStream(new StringReader(xml));
Document doc = db.parse(is);
```

DOM - parsowanie

```
NodeList nl = doc.getElementsByTagName("product");
// looping through all item nodes <"product">
for (int i = 0; i < nl.getLength(); i++) {
    currentProduct = new Product();

    Element e = (Element) nl.item(i);
    currentProduct.name = parser.getValue(e, "productname");
    currentProduct.color = parser.getValue(e, "productcolor");
    currentProduct.quantity = parser.getValue(e, "productquantity");

    products.add(currentProduct);
}
```

DOM - parsowanie

```
public String getValue(Element item, String str) {
    NodeList n = item.getElementsByTagName(str);
    return this.getElementValue(n.item(0));
}

public final String getElementValue( Node elem ) {
    Node child;
    if( elem != null){
        if (elem.hasChildNodes()){
            for( child = elem.getFirstChild(); child != null; child =
                                                child.getNextSibling() ){
                if( child.getNodeType() == Node.TEXT_NODE  ){
                    return child.getNodeValue();
                }
            }
        }
    }
    return "";
}
```

SAX parser

- Implementacja identyczna jak w języku Java
- Alternatywa dla DOM, działa w oparciu o mechanizm SAX (Simple API for XML)
- Tworzenie parsera:

```
// create a XMLReader from SAXParser
XMLReader xmlReader = SAXParserFactory.newInstance().newSAXParser()
                                                    .getXMLReader();
// create a SAXXMLHandler
SAXXMLHandler saxHandler = new SAXXMLHandler();
// store handler in XMLReader
xmlReader.setContentHandler(saxHandler);
// the process starts
xmlReader.parse(new InputSource(is));
```

SAX Handler

```
public class SAXXMLHandler extends DefaultHandler {

    private List<Product> products;
    private String tempVal;
    private Product tempProduct;

    public SAXXMLHandler() {
        products = new ArrayList<Product>();
    }

    public List<Product> getProducts() {
        return products;
    }

    // Event Handlers
    public void startElement(String uri, String localName, String qName,
            Attributes attributes) throws SAXException {
        // reset
        tempVal = "";
        if (qName.equalsIgnoreCase(„product")) {
            // create a new instance of product
            tempProduct = new Product();
        }
    }
```

SAX Handler

```
    public void characters(char[] ch, int start, int length)
            throws SAXException {
        tempVal = new String(ch, start, length);
    }

    public void endElement(String uri, String localName, String qName)
            throws SAXException {
        if (qName.equalsIgnoreCase("product")) {
            // add it to the list
            product.add(tempProduct);
        } else if (qName.equalsIgnoreCase("productname")) {
            currentProduct.name = tempVal;
        } else if (qName.equalsIgnoreCase("productcolor")) {
            currentProduct.color = tempVal;
        } else if (qName.equalsIgnoreCase("productquantity")) {
            currentProduct.quantity = tempVal;
        }
    }
}
```