


Android

Rede com Sockets, Internet e Web Services




Softblue
cursos online

Tópicos Abordados



- O que são sockets
- Modelo cliente/servidor
- Sockets e o Android
- Sockets TCP/IP e UDP/IP
 - Funcionamento
 - Características
 - Implementação do cliente e do servidor
- Realizando requisições HTTP
- Manipulação de documentos XML e JSON
- Web Services e o Android
 - SOAP web services
 - RESTful web services
- Conectividade

O que são Sockets



- Mecanismo de comunicação entre dois programas que funcionam na mesma rede
- Modelo cliente/servidor
 - Uma aplicação servidor é executada num determinado dispositivo e tem um socket ligado a uma porta específica desse dispositivo
 - O servidor espera que um cliente faça um pedido de conexão através desse socket

Programação de Sockets



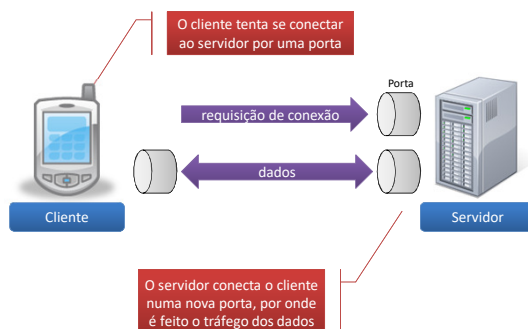
- A programação de sockets no Android é feita da mesma forma que no Java SE
 - O Android possui a Java I/O API
- Isto facilita o aprendizado para quem sabe como implementar sockets em Java SE

Sockets TCP/IP




- Existe uma conexão entre o cliente e o servidor
 - Permite utilização de fluxos de dados (streams)
- A comunicação é confiável
 - Sem perda de dados
 - Sem inversão de ordem dos pacotes
- Classes utilizadas
 - *ServerSocket*
 - *Socket*

Funcionamento dos Sockets TCP/IP



Implementando um Servidor TCP/IP



```

ServerSocket serverSocket = new ServerSocket(3000);
Socket clientSocket = serverSocket.accept();

InputStream is = clientSocket.getInputStream();
OutputStream os = clientSocket.getOutputStream();

DataInputStream dis = new DataInputStream(is);
DataOutputStream dos = new DataOutputStream(os);

int valor = dis.readInt();
dos.writeInt(valor);

clientSocket.close();
serverSocket.close();
    
```


Abre um socket TCP/IP

Aguarda uma conexão

Lê e escreve nas streams

É importante implementar o servidor em uma thread separada

Implementando um Cliente TCP/IP



```

Socket socket = new Socket("localhost", 3000);

InputStream is = socket.getInputStream();
OutputStream os = socket.getOutputStream();

DataInputStream dis = new DataInputStream(is);
DataOutputStream dos = new DataOutputStream(os);


dos.writeInt(5);
int valor = dis.readInt();

socket.close();
    
```

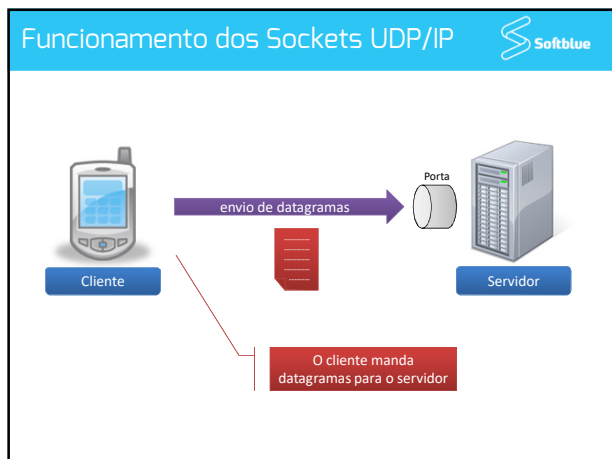
Faz uma conexão

Escreve e lê nas streams

Sockets UDP/IP



- Não existe uma conexão entre o cliente e o servidor
 - Envio de datagramas (remetente, receptor, conteúdo)
- A comunicação não é confiável
 - Dados podem ser perdidos
 - Datagramas podem chegar fora de ordem
- Muito mais veloz que sockets TCP/IP
- Classes utilizadas
 - *DatagramSocket*
 - *DatagramPacket*



Implementando um Servidor UDP/IP

```
DatagramSocket socket = new DatagramSocket(3000);
byte[] buff = new byte[1024];

DatagramPacket dp = new DatagramPacket(buff, buff.length);
socket.receive(dp);

String dado = new String(dp.getData(), 0, dp.getLength());
socket.close();
```

Abre um socket UDP/IP

Aguarda o recebimento de um datagrama

É importante implementar o servidor em uma thread separada

Implementando um Cliente UDP/IP

```
DatagramSocket socket = new DatagramSocket();

String valor = "isto é um texto";
byte[] bytes = valor.getBytes();

int porta = 3000;
InetAddress addr = InetAddress.getLocalHost();

DatagramPacket dp =
    new DatagramPacket(bytes, bytes.length, addr, porta);
socket.send(dp);

socket.close();
```

A mesma classe é utilizada

Envia um datagrama

Requisições HTTP



- O HTTP é o protocolo utilizado no tráfego de informações pela internet
 - **H**yper **t**ext **T**ransfer **P**rotocol
- Ele é baseado num modelo cliente-servidor
 - O cliente faz uma requisição ao servidor
 - O servidor processa a requisição e envia uma resposta
- Em um nível mais baixo, a comunicação é baseada em sockets TCP/IP

Trabalhando com XML



- XML
 - **E**xtensible **M**arkup **L**anguage
- Linguagem de marcação para descrever dados estruturados
- XML possibilitou fácil integração de sistemas multiplataforma
 - É um padrão aberto e independente de fornecedor e dispositivo
 - Bastante usado na internet
- O sucesso do XML também se deve à grande quantidade de código pronto
 - Facilita a manipulação de documentos

Manipulação de XML



- A manipulação de documentos pode ser feita pelas APIs DOM e SAX
 - O Android suporta as duas APIs de forma completa
 - A programação usando DOM ou SAX para o Android é a mesma do Java SE

A API DOM



- **D**ocument **O**bject **M**odel
- API fácil e intuitiva
- Cria uma árvore de objetos em memória
- Esta árvore representa a estrutura do documento XML
- Desvantagem
 - Ocupa muita memória, o que pode tornar seu uso proibitivo em dispositivos móveis se o XML for muito grande

A API SAX



- Não é tão intuitiva quanto a DOM
- É baseada em eventos
 - Funções de callback são invocadas durante o parsing do documento
 - API no estilo “push”
- Como não cria uma estrutura em memória, é aconselhada para documentos XML extensos

Variação do SAX: Pull Parser



- No SAX, o parser do XML vai “empurrando” os eventos para o objeto que faz a análise (modo *push*)
- Em um pull parser, é o código quem extrai os eventos do parser (modo *pull*)
- Deixa o código mais simplificado

Variação do SAX: Pull Parser



```
XmlPullParser parser = Xml.newPullParser();  
parser.setInput(reader);  
int eventType = parser.getEventType();  
while (eventType != XmlPullParser.END_DOCUMENT) {  
    if (eventType == XmlPullParser.START_TAG) {  
        String tagName = parser.getName();  
    }  
    eventType = parser.next();  
}
```

Cria o pull
parser

Define a fonte
do XML

Lê a primeira
informação

Lê a próxima
informação

Criando Documentos XML



- A forma mais fácil de criar um documento XML é através da classe **XMLSerializer**

```
XmlSerializer serializer = Xml.newSerializer();  
StringWriter writer = new StringWriter();  
serializer.setOutput(writer);  
serializer.startDocument("UTF-8", true);  
serializer.startTag("", "contatos");  
serializer.startTag("", "contato");  
serializer.text("Contato 1");  
serializer.endTag("", "contato");  
serializer.startTag("", "contato");  
serializer.text("Contato 2");  
serializer.endTag("", "contato");  
serializer.endTag("", "contatos");  
serializer.endDocument();  
String xml = writer.toString();
```

Criando Documentos XML



- XML gerado

```
<?xml version='1.0' encoding='UTF-8' standalone='yes' ?>  
<contatos>  
  <contato>Contato 1</contato>  
  <contato>Contato 2</contato>  
</contatos>
```

Trabalhando com o Formato JSON



- **J**ava**S**cript **O**bject **N**otation
- O formato JSON é bastante poderoso e é mais simplificado que o XML
 - É um formato texto, que independe de tecnologia
- As principais classes para trabalhar com JSON disponíveis no Android são
 - **JSONArray**: representa um array de dados
 - **JSONObject**: representa um objeto

Web Services com Android

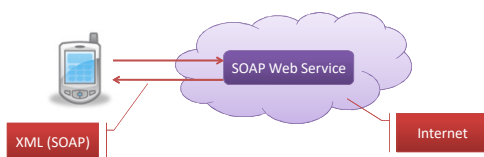


- É uma forma de integrar aplicações heterogêneas através da internet
- A integração pode ser feita com web services de dois tipos
 - SOAP Web Services
 - RESTful Web Services

SOAP Web Services



- Os dados trocados são no formato XML
- O protocolo de comunicação é o SOAP (Simple Object Access Protocol)



SOAP Web Services

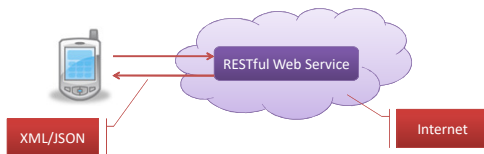


- O Android não possui suporte nativo à invocação de web services
 - É preciso utilizar alguma API externa
- Uma das APIs mais utilizadas pelos desenvolvedores é a **kSOAP 2**

RESTful Web Services



- Utiliza o próprio protocolo HTTP e operações já definidas por este protocolo
 - GET, POST, PUT, DELETE
- Os dados são normalmente trocados em XML ou JSON



Conectividade



- É interessante verificar se o acesso à internet está disponível antes do uso

```
ConnectivityManager cm = (ConnectivityManager)
    getSystemService(Context.CONNECTIVITY_SERVICE);
NetworkInfo activeNetwork = cm.getActiveNetworkInfo();
boolean connected = activeNetwork.isConnected();
```

Indica se uma conexão
está presente

- O tipo de conexão também pode ser obtido

```
int type = activeNetwork.getType();
```

TYPE_MOBILE,
TYPE_WIFI,
TYPE_BLUETOOTH,
etc.

Constante de
ConnectivityManager