


Programación de Dispositivos Móviles




Sesión 17:
Conexiones de red

Java y Dispositivos Móviles

© 2007-2009 Depto. Ciencia Computación e IA

Conexiones de red-1

Índice




- Marco de conexiones genéricas
- Conexión HTTP
- Envío y recepción de datos
- Conexiones a bajo nivel
- Mensajes SMS

Java y Dispositivos Móviles

© 2007-2009 Depto. Ciencia Computación e IA

Conexiones de red-2

Conexiones de red



- Marco de conexiones genéricas
- Conexión HTTP
- Envío y recepción de datos
- Conexiones a bajo nivel
- Mensajes SMS

Java y Dispositivos Móviles

© 2007-2009 Depto. Ciencia Computación e IA

Conexiones de red-3

GCF



- **GCF = *Generic Connection Framework***
 - Marco de conexiones genéricas, en `javax.microedition.io`
 - Permite establecer conexiones de red independientemente del tipo de red del móvil (circuitos virtuales, paquetes, etc)
- **Cualquier tipo conexión se establece con un único método genérico**

```
Connection con = Connector.open(url);
```

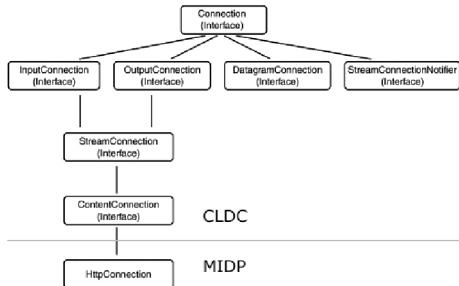
- **Según la URL podemos establecer distintos tipos de conexiones**

<code>http://jtech.ua.es/pdm</code>	HTTP
<code>datagram://192.168.0.4:6666</code>	Datagramas
<code>socket://192.168.0.4:4444</code>	Sockets
<code>comm:0;baudrate=9600</code>	Puerto serie
<code>file:/fichero.txt</code>	Ficheros

Tipos de conexiones



- En CLDC se implementan conexiones genéricas
- En MIDP y APIs opcionales se implementan los protocolos concretos



Conexiones de red

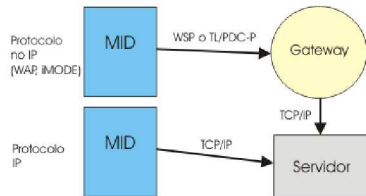


- Marco de conexiones genéricas
- **Conexión HTTP**
- Envío y recepción de datos
- Conexiones a bajo nivel
- Mensajes SMS

Conexión HTTP



- El único protocolo que se nos asegura que funcione en todos los móviles es HTTP
 - Funcionará siempre de la misma forma, independientemente del tipo de red que haya por debajo



Leer de una URL



- Abrimos una conexión con la URL

```
HttpConnection con = (HttpConnection)Connector.open(
    "http://jtech.ua.es/index.htm");
```

- Abrimos un flujo de entrada de la conexión

```
InputStream in = con.openInputStream();
```

- Podremos leer el contenido de la URL utilizando este flujo de entrada

- Por ejemplo, en caso de ser un documento HTML, leeremos su código HTML

- Cerramos la conexión

```
in.close();
con.close();
```

Mensaje de petición



- Podemos utilizar distintos métodos

```
HttpConnection.GET
HttpConnection.POST
HttpConnection.HEAD
```

- Para establecer el método utilizaremos:

```
con.setRequestMethod(HttpConnection.GET);
```

- Podemos añadir cabeceras HTTP a la petición

```
con.setRequestProperty(nombre, valor);
```

- Por ejemplo:

```
c.setRequestProperty("User-Agent",
    "Profile/MIDP-1.0 Configuration/CLDC-1.0");
```

Mensaje de respuesta



- A parte de leer el contenido de la respuesta, podemos obtener

- Código de estado

```
int cod = con.getResponseCode();  
String msg = con.getResponseMessage();
```

- Cabeceras de la respuesta

```
String valor = con.getHeaderField(nombre);
```

- Tenemos métodos específicos para cabeceras estándar

```
getLength()  
getType()  
getLastModified()
```

Conexiones de red



- Marco de conexiones genéricas
- Conexión HTTP
- Envío y recepción de datos
- Conexiones a bajo nivel
- Mensajes SMS

Enviar datos



- Utilizar parámetros

- GET o POST

- Parejas <nombre, valor>

```
HttpConnection con = (HttpConnection)Connector.open(  
    "http://jtech.ua.es/registra?nombre=Pedro&edad=23");
```

- No será útil para enviar estructuras complejas de datos

- Añadir los datos al bloque de contenido de la petición

- Deberemos decidir la codificación a utilizar

- Por ejemplo, podemos codificar en binario con
`DataOutputStream`

Tipos de contenido



- Para enviar datos en el bloque de contenido debemos especificar el tipo MIME de estos datos
 - Lo establecemos mediante la cabecera **Content-Type**

```
con.setRequestProperty("Content-Type", "text/plain");
```

- Por ejemplo, podemos usar los siguientes tipos:

application/x-www-form-urlencoded	Formulario POST
text/plain	Texto ASCII
application/octet-stream	Datos binarios

Codificación de los datos



- Podemos codificar los datos a enviar en binario
 - Establecemos el tipo MIME adecuado

```
con.setRequestProperty("Content-Type",  
    "application/octet-stream");
```

- Utilizaremos un objeto **DataOutputStream**

```
DataOutputStream dos = con.openDataOutputStream();  
dos.writeUTF(nombre);  
dos.writeInt(edad);  
dos.flush();
```

- Si hemos definido serialización para los objetos, podemos utilizarla para enviarlos por la red

Leer datos de la respuesta



- Contenido de la respuesta HTTP
 - No sólo se puede utilizar HTML
 - El servidor puede devolver contenido de cualquier tipo
 - Por ejemplo, XML, ASCII, binario, etc
- Si el servidor nos devuelve datos binarios, podemos descodificarlos mediante **DataInputStream**

```
DataInputStream dis = con.openDataInputStream();  
String nombre = dis.readUTF();  
int precio = dis.readInt();  
dis.close();
```

- Podría devolver objetos serializados
 - Deberíamos deserializarlos con el método adecuado

Conexiones de red



- Marco de conexiones genéricas
- Conexión HTTP
- Envío y recepción de datos
- **Conexiones a bajo nivel**
- Mensajes SMS

Conexiones a bajo nivel



- **A partir de MIDP 2.0 se incorporan a la especificación conexiones de bajo nivel**
 - Sockets
 - Datagramas
- Nos permitirán aprovechar las características de las nuevas redes de telefonía móvil
- **Podremos acceder a distintos servicios de Internet directamente**
 - Por ejemplo correo electrónico
- **Su implementación es optativa en los dispositivos MIDP 2.0**
 - Depende de cada fabricante

Sockets



- **Establecer una comunicación por sockets**

```
SocketConnection sc = (SocketConnection)
Connector.open("socket://host:puerto");
```
- **Abrir flujos de E/S para comunicarnos**

```
InputStream in = sc.openInputStream();
OutputStream out = sc.openOutputStream();
```
- **Podemos crear un socket servidor y recibir conexiones entrantes**

```
ServerSocketConnection ssc = (ServerSocketConnection)
Connector.open("socket://:puerto");
SocketConnection sc =
(SocketConnection) ssc.acceptAndOpen();
```

Datagramas



- **Crear conexión por datagramas**

```
DatagramConnection dc = (DatagramConnection)
Connector.open("datagram://host:puerto");
```

- **Crear un enviar paquete de datos**

```
Datagram dg = dc.newDatagram(datos, datos.length);
dc.send(dg);
```

- **Recibir paquete de datos**

```
Datagram dg = dc.newDatagram(longitud);
dc.receive(dg);
```

Conexiones de red



- Marco de conexiones genéricas
- Conexión HTTP
- Envío y recepción de datos
- Conexiones a bajo nivel
- **Mensajes SMS**

Conexión de mensajes



- **Con WMA podremos crear conexiones para enviar y recibir mensajes de texto SMS**

- **Utilizaremos una URL como**

```
sms://telefono:[puerto]
```

- **Creamos la conexión**

```
MessageConnection mc = (MessageConnection)
Connector.open("sms://+34555000000");
```

Envío de mensajes



▪ Componemos el mensaje

```
String texto =  
    "Este es un mensaje corto de texto";  
TextMessage msg = mc.newMessage(mc.TEXT_MESSAGE);  
msg.setPayloadText(texto);
```

▪ El mensaje no deberá pasar de 140 bytes

- Si se excede, podría ser fraccionado
- Si no puede ser fraccionado, obtendremos un error

▪ Enviamos el mensaje

```
mc.send(msg);
```

Recepción de mensajes



▪ Creamos conexión de mensajes entrantes

```
MessageConnection mc = (MessageConnection)  
    Connector.open("sms://:4444");
```

▪ Recibimos el mensaje

```
Message msg = mc.receive();
```

▪ Esto nos bloqueará hasta la recepción

- Para evitar estar bloqueados, podemos utilizar un listener
- Con un `MessageListener` se nos notificará de la llegada de mensajes
