Networking: Generic Connection Framework

Back | Next

MIDP Profile provides networking support for various connection protocols. Package javax.microedition.io provides us classes and interfaces for following protocols.

- Http Connection / Https Connection
- UDP Connection
- TCP/IP Connection
- Comm Connection

Most of the classes here are similar to J2SE/J2EE classes except that here client aspect of each protocol is available to us and server part is omitted from the javax.microedition.io package. However basic Server Socket API's are provided for TCP/IP and UDP protocols.

HTTP Networking

Interface javax.microedition.io.HttpConnection defines methods related to HTTP connection

Get HTTP Connection Object

A HttpConnection object is obtained when Connector.open(String url) is called. The HttpConnection provides support for GET and POST requests/responses and respective Headers.

Set the Request Method

HttpConnection.setRequestMethod(HttpConnection.POST). By default the Request Method is GET.

Set Header Information

To set request header properties use setRequestProperty(String key, String value). Example: setRequestProperty(User-Agent, Profile/MIDP-1.0 Configuration/CLDC-1.0)

Send Request and get a Response

To send request call con.openInputStream();
To receive response call con.openInputStream();

Listing for HTTP Request Response Routine

```
void postRequest(String url) throws IOException {
        HttpConnection con = null;
        InputStream is = null;
        OutputStream os = null;

        try {
                con = (HttpConnection)Connector.open(url);

                // Set request method
```

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

                // Get the output stream
                os = con.openOutputStream();

                String sentData = J2ME-ed = J2ME NERD,
                        J2ME BUSTERS =  J2ME-ed Moderators;
                os.write(sentData.getBytes());
                os.flush();

                // Always get the Response code first .
                int responseCode = con.getResponseCode();
                if (responseCode == HttpConnection.HTTP_OK) {
                        // You have successfully connected.
                        is = con.openInputStream();
                        // Now Process your request.
                }
                else {
                        // Problem with your connection
                }
        } catch (ClassCastException e) {
                // Do your exception handling here
        } finally {
                if (is != null)
                        is.close();
                if (os != null)
                        os.close();
                if (con != null)
                        con.close();
        }
}
```


Session Handling and Cookies

Applications use session handling for User tracking while talking to a HTTP Server. The Browser based applications use Cookies for session handling and Cookie repository is not available in a Java Phone. There is a work around to this problem. You can get the cookie information from the Response headers and store it throughout the sessions and send this cookie information every time your MIDP application makes a request to the server.

Create HTTP Connection and receive data

HttpConnection con= (HttpConnection)Connector.open(url); InputStream in = con.openInputStream();

This received response data has following parameters. The Cookie parameter is "Set-Cookie" and this parameter can be parsed for session ID

HTTP/1.1 200 OK
Content-Type: text/plain
Content-Length: 100
Date: Wed, 20 Nov 2007
Server: Weblogic
Set-Cookie: JSESSIONID=35E2621570C3B1D052E86285D1A002D6;Path=/midp

To access Set-Cookie header use getHeaderField() Method

```
String cookie = con.getHeaderField("Set-Cookie");
```

To set this value to the Request header use setRequestProperty() method.

HTTPS - Secure HTTP Connections

HTTPS has been added since MIDP 2.0. Secure interfaces are provided by HTTPS and SSL/TLS protocol access over the IP network.

A HttpsConnection object is returned on calling Connector.open() with URL Schema"https://", example: Connector.open("https://www.yoursecuresite.com")

A SecureConnection is returned on calling calling Connector.open() with URL Schema"https://", example: Connector.open("ssl://host:port").

Following classes/interfaces have been added since MIDP2.0 for Secure connections.

- javax.microedition.io.HttpsConnection
- javax.microedition.io.SecureConnection
- javax.microedition.io.SecurityInfo
- javax.microedition.pki.Certificate
- javax.microedition.pki.CertificateException

TCP/IP and UDP Networking

TCP stands for Transmission Control Protocol. TCP is a connection-oriented protocol that is responsible for reliable communication between two end processes. The data transferred is transferred as a stream.

A SocketConnection is returned on calling Connector.open() with URL Schema " socket://", example: Connector.open("socket://host:port").

Server Socket can be also created using Connector.open() with URL Schema " socket://", example: Connector.open("socket://:port"). Here we need to specify only port number as the connection string. ServerSocketConnection object is returned in this case.

UDP stands for User Datagram Protocol. UDP is a connectionless protocol and is not a reliable communication between two end processes. The successive connections are not related to one another. This protocol is usually used for broadcasting.

A UDPDatagramConnection is returned on calling Connector.open() with URL Schema "datagram://", example: Connector.open("datagram://host:port").

UDP server connection can be created using the same URL Schema "datagram://" but the connection string does not include host IP address, example: Connector.open("datagram://host:port").

Following classes/interfaces have been added since MIDP2.0 for TCP and UDP networking.

- javax.microedition.io.SocketConnection
- javax.microedition.io.ServerSocketConnection
- javax.microedition.io.DatagramConnection
- javax.microedition.io.Datagram

- javax.microedition.io.UDPDatagramConnection

Serial Port Connections

Logical serial port connections or comm connections can be established to communicate with IR Ports etc. CommConnection class is available for this purpose.

A CommConnection is returned on calling Connector.open() with URL Schema " comm://", example: Connector.open("datagram://port identifier;<optional parameters>").

Optional parameters can be used to control your connection's parameters like baud rate.

Push Technology meets J2ME

Push Registry is a welcome addition since MIDP2.0. Push Registry maintains a list of inbound connections and associated applications. The event is triggered by the server and delegated to the mobile device where an application is registered for this event. The registered application comes alive on the device does the required task.

Push Registry can also be set to be triggered at regular intervals without any intervention from server or user. This is called timer based push registry.With the advent of push registry your application can now be launched in three ways

- Select the application from Application Launcher screen.
- Trigger by inbound network request
- Timer based activation.

Registration for Push Registry is done in two ways

1. Static Registration: MIDlets are registered at installation time. This registration is done in JAD or Manifest file using attribute "MIDlet-Push". Only Network Based Push Registry is supported.
2. Dynamic Registration: This is done by the application when the MIDlet is first activated manually. Here the registration has to be done in your code using PushRegistry class. Both Network Based and Alarm Based Push Registry can be done by this approach.

Connection / Network Based Push Registry

You can register your application to be activated to following connections

- SMS
- TCP
- UDP

Static Registration

In static registration you don't need to do any coding. Any existing application can be configured to be triggered by Network connection by adding "MIDlet-Push" attribute in the JAD file or manifest file. In this section we will convert our MIDlet Suite Demo to listen to Push Registry.

Usage

MIDlet-Push-<n>: <ConnectionURL>, <MIDletClassName>, <AllowedSender>

Parameter Description:

- MIDlet-Push-<n> = the Push registration attribute name. Multiple push registrations can be provided in a MIDlet suite. The numeric value for <n> starts from 1. Example: MIDlet-Push-1, MIDlet-Push-2 ……
- ConnectionURL = The URL at which MIDlet has to be registered. Example: socket://:79. Your server should use this portnumber to trigger the MIDlet
- MIDletClassName = the MIDlet which should be triggered or activated. Example: com.jme.midletsuite.FirstMIDlet
- AllowedSender = a filter that restricts which senders are valid for launching the requested MIDlet. Here you will have to specify the IP addresses of your server. Wild card characters (*, ?) can be also used here. Example: 200.200.50.1, 200.200.50.*, *(Any IP address).

Listing to convert MIDlet Suite Demo to listen to Push Registry

MIDlet-1: FirstMIDlet, FirstMIDlet.png, com.j2me.salsa.midletsuite.FirstMIDlet
MIDlet-2: SecondMIDlet, SecondMIDlet.png, com.j2me.salsa.midletsuite.SecondMIDlet
MIDlet-3: ThirdMIDlet, ThirdMIDlet.png, com.j2me.salsa.midletsuite.ThirdMIDlet
MIDlet-Jar-Size: 100
MIDlet-Jar-URL: MidletSuiteDemo.jar
MIDlet-Name: MidletSuiteDemo
MIDlet-Vendor: Unknown
MIDlet-Version: 1.0
MicroEdition-Configuration: CLDC-1.0
MicroEdition-Profile: MIDP-2.0
MIDlet-Push-1: socket://:79, com.j2me.salsa.midletsuite.FirstMIDlet, 200.200.50.1
MIDlet-Push-2: datagram://:50000, com.j2me.salsa.midletsuite.SecondMIDlet, *

Dynamic Registration

In dynamic registration you will have to add following line of code to the constructors of following MIDlets

FirstMIDlet

```
PushRegistry.registerConnection("socket://:79","com.j2me.salsa.midletsuite.FirstMIDlet", "200.200.50.1");
```

SecondMIDlet

```
PushRegistry.registerConnection("datagram://:50000,com.j2me.salsa.midletsuite.SecondMIDlet", "*");
```

To unregistered dynamically use unregisterConnection(String connection) Method

Alarm Based Push Registry

Only Dynamic Push Registry can be implemented in Alarm Based Push Registry. Use registerAlarm(String midlet, long time) Method to register your MIDLet where midlet is the MIDlet to be triggered and time is the time at which the MIDlet is to be triggered. Let us again convert our MIDlet Suite Demo to listen to Alarm Push Registry.

FirstMIDlet

```
registerAlarm ("com.j2me.salsa.midletsuite.FirstMIDlet", "200.200.50.1");
```

SecondMIDlet

```
registerAlarm ("com.j2me.salsa.midletsuite.SecondMIDlet", "*");
```

Other useful Push Registry Methods

- getFilter(String connection) : Retrieve the registered filter for a requested connection. Example: PushRegistry.getFilter("socket://:79") will return "200.200.50.1".
- getMIDlet(String connection) : Retrieve the registered MIDlet for a requested connection. Example: PushRegistry. getMIDlet ("socket://:79") will return "com.j2me.salsa.midletsuite.FirstMIDlet".
- listConnections(boolean available) : Returns a list of registered connections for the current MIDlet suite. Example: PushRegistry.listConnections(false) will return String Array {"socket://:79", "datagram://:50000"}

This brings us to the end of this beginner's tutorial for J2ME. Take your last test.

Back | Next

5 comments

4

Leave a message...

Best Community Share

wachdog • 3 years ago

good for nothing

2 Reply Share ›

Nilesh • 2 years ago

Nice Explaination....
But, How we can handle multiple request on same connection.....??????????????????????????

1 Reply Share ›

venkata reddy • 3 years ago

material just excellent.....

1 Reply Share ›

Chowdary05 • 2 years ago

didnt like the notes boring j2me

Reply Share ›

Vinay • 2 years ago

Does personal profile also support Http and Https URL Connection

Reply Share ›