13

MIDP 網路程式設計

沒有好書和壞書的分別，只要任何一個人閱讀之後對他有所提昇，那麼對那個人來說，他就找到了好書。

前 言

在圖形使用者介面程式設計那章，我們曾經提到，雖然 Java 2 Standard Edition 提供 AWT 與 SWING 兩套圖形使用者介面類別函式庫讓標準的 Java 程式使用，但是在卻因為其設計理念和複雜度的關係，所以在 MIDP 之中特別又設計了一組 MIDP 專屬的圖形使用者介面類別函式庫。同樣的事情也發生在用於存取網路的類別函式庫身上。

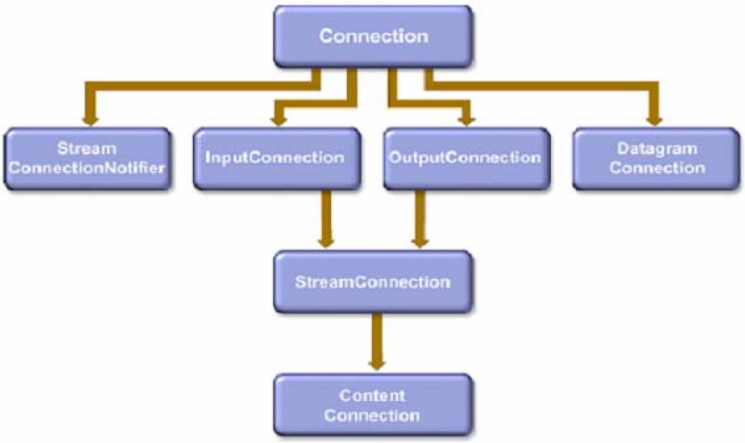
在傳統的 Java 程式之中，我們必須使用 java.io 與 java.net 這兩個套件來存取網路，但是基於兩個重要的因素，使得我們無法在 MIDP 之中使用這兩個套件:

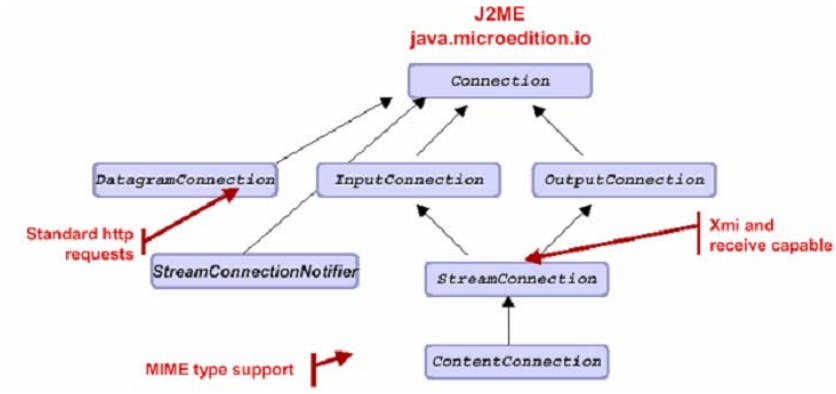
1. 太過龐大。這兩個套件合起來將近佔據了 200K 的空間，比起 KVM 來說要大的多，因此不適合型行動通訊裝置來使用。
2. 行動通訊裝置的對外溝通方式差異性太大。各家廠商的行動通訊裝置所使用的傳輸方式太過多樣，底層的通訊方法有 circuit-switched 或 packet-switched（GSM、GPRS、CDMA、PHS）等。

尤其是第二個因素，迫使希望能夠在各種不同裝置上執行的 Java 2 Micro Edition 必須在 CLDC 的規格之中設計一個全新的 Generic Connection Framework。

Generic Connection Framework

所有和 Generic Connection Framework 相關的類別、介面、以及例外全部都放置在 javax.microedition.io 這個套件中。在 CLDC 之中定義了七個介面，他們分別是 Connection、StreamConnection、DatagramConnection、InputConnection、OutputConnection、StreamConnection、ContentConnection。在 MIDP 增加了一個 HttpConnection 介面，且繼承自 ContentConnection，因此 HttpConnection 介面並非 Generic Connection Framework 的一部分。他們的繼承體系和相關功能如下圖所示：





在 Generic Connection Framework 之中，定義了幾種常見的對外溝通方式。不管我們打算用何種方式和外界溝通，所有的連線都是由 Connector 類別的 open()方法所開始：

使用 HTTP 和外界溝通：

```
Connector.open("http://www.sun.com.tw") ;
```

使用 Socket 和外界溝通:

```
Connector.open("socket://192.168.0.3:8080") ;
```

使用 Datagram 和外界溝通：

```
Connector.open("datagram://www.sun.com:9000") ;
```

使用檔案和外界溝通：

```
Connector.open("file:/input.txt") ;
```

使用序列阜和外界溝通：

```
Connector.open("coom:0;baudrate=9600") ;
```

雖然上述都是由 CLDC 之中所定義的對外溝通方式，可是因為我們撰寫的 MIDlet 是根據 MIDP 規格所建構，所以各家廠商的 MIDP 實做不見得會支援上述這幾種通訊方式。唯有 HttpConnection 明定在 MIDP 規格之中，所以唯有 http 傳輸協定才是 MIDlet 能夠在各家平台上使用的對外溝通方式。至於 http 之外的對外溝通方式，您必須參考各家廠商的文件，才能知道他們是否支援其他對外溝通方式，或者是他們額外增加的非標準溝通方式，比方說 MIDP for Palm 就很有可能特別增加紅外線的對外連線方式。

底下我們將特別為大家介紹利用 Socket 與 HTTP 這兩種方式與外界做溝通。

在正式在討論前，我們必須先做準備，首先我們假設您的電腦上安裝了 Apache Web Server（下載和安裝細節請參閱第四章）。

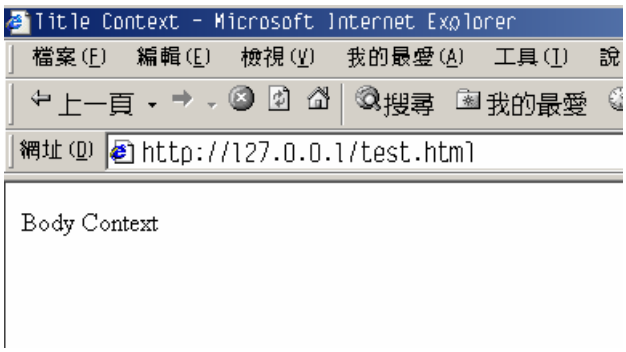
接著，我們製作一個簡單的 HTML 檔，內容如下：

Test.html

```
<html>
<title>
Title Context
</title>
<body>
Body Context
</body>
</html>
```

並將此 HTML 檔置放到 Apache 安裝目錄下的 htdocs 子目錄之中，然後啟動 Apache 伺服器。請在瀏覽器的 URL 欄位輸入 http://127.0.0.1/test.html，如果伺服器啟動成功，那麼瀏覽器

的畫面應該如下所示：



如此一來代表我們的前置工作完成。如果您的瀏覽器沒有出現正確的內容，請回頭參閱 Apache 的使用說明。

使用 HTTP 進行網路連線

根據 MIDP 1.0 的規格，HTTP 一定是所有廠商都要提供的連線方式，因此我們先讓大家了解如何利用 Generic Connection Framework 來完成與 Web 伺服器之間的溝通。

使用步驟如下：

1. 在 Connector.open()之中使用 http 當作通訊協定。如果 Connector.open()與 Web 伺服器連線成功，Connector.open()會傳回被向上轉型之後的 HttpConnection 物件。為了以後使用的方便我們必須將它轉型回 HttpConnection 類別。

本範例所使用的程式碼為：

```
String url = "http://127.0.0.1/test.html" ;
HttpConnection hc = (HttpConnection)
Connector.open(url) ;
```

2. 利用 HttpConnection 的 openInputStream 開啟輸入資料流，如此才能取得 Web 伺服器所回傳的資料。

   本範例使用的程式碼為：

   ```
   hc.openInputStream()
   ```

3. InputStream 只能用來讀取二進位資料，為了正確地讀取字串，我們必須將 InputStream 與 DataInputStream 串接在一起。

   本範例所使用的程式碼為：

   ```
   DataInputStream dis =
   new DataInputStream(hc.openInputStream()) ;
   ```

4. 當輸入資料流開啟之後，我們開始利用 DataInputStream 類別的 read()函式將所收到的字元一個一個讀進來。

   本範例所使用的程式碼為：

   ```
   dis.read()
   ```

因此一個完整的範例如下：

HTTPTest.java

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import javax.microedition.io.* ;
import java.io.*;

public class HTTPTest extends MIDlet
{
   private Display display;

   public HTTPTest()
   {
      display = Display.getDisplay(this);
   }
   public void startApp()
   {
    try
    {
        String url = "http://127.0.0.1/test.html" ;
        HttpConnection hc = (HttpConnection)
Connector.open(url) ;

        //取得伺服器輸出
        DataInputStream dis =
new DataInputStream(hc.openInputStream()) ;
        String content = "" ;
        int ic ;
        while( (ic = dis.read()) != -1 )
        {
            content = content + (char)ic ;
        }

        Form f = new Form("HTTP Test");
        f.append(content) ;
        display.setCurrent(f) ;
    }catch(Exception e)
    {
```

```
            System.out.println(e.getMessage()) ;
            notifyDestroyed() ;
        }
       }
       public void pauseApp()
       {
       }
       public void destroyApp(boolean unconditional)
       {
       }
}
```

【執行結果】

在本範例之中，我們將 Web 伺服器所傳送過來的資料讀取完畢之後，再將所有資料一次加到 Form 元件之中，再一次顯示在螢幕上。

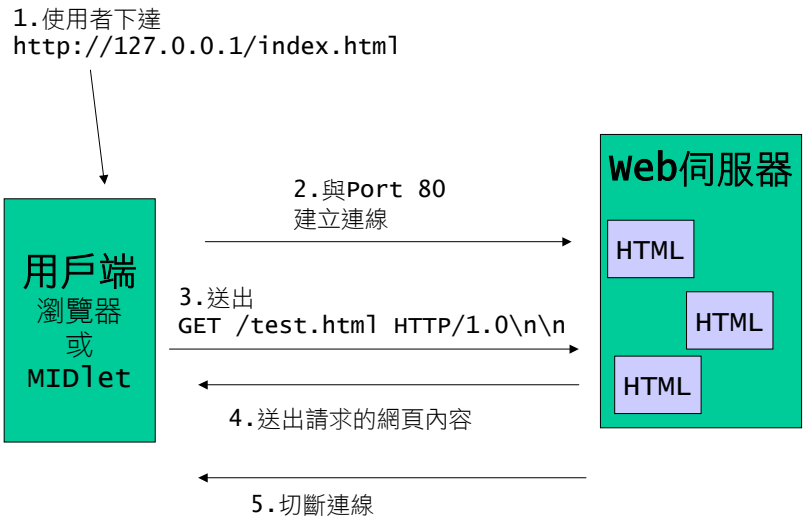
大家一定覺得很奇怪，為何我們只要指定了 URL，Web 伺服器就可以立刻幫我們把網頁的內容傳回來呢？這是因為，當我們下達：

http:/127.0.0.1/test.html

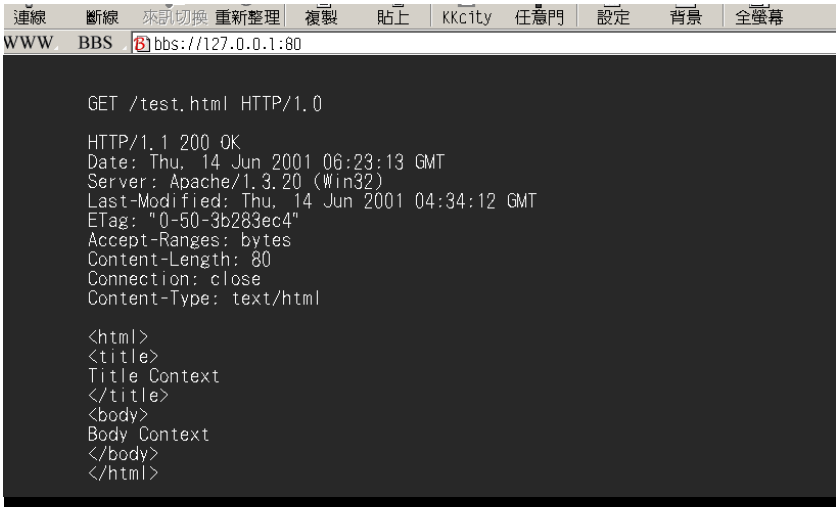
的時候，程式其實當我們把這個指令分成兩個步驟：

1. 與 127.0.0.1 上的 port 80 連線。
2. 送出 GET /test.html HTTP/1.0\n\n 指令。

我們可以用下圖表示：



我們可以使用任何一個終端機程式來證明這件事。以 KKMAN 來說，請輸入 bbs://127.0.0.1:80，讓終端機程式和 Web 伺服器連線。接著請在螢幕上輸入 GET /test.html HTTP/1.0，然後連按兩下 Enter，就可以看到底下畫面：



您可能會問，我們在範例中似乎只有看到最後那一段 HTML 檔的內容而已，應沒有看到前面的標頭訊息呀？這是因為 HttpConnection 幫我們把標頭隱藏起來了，雖然從輸入資料流之中只能看到 HTML 檔的內容，但是我們仍然可以利用 HttpConnection 所提供的函式取得標頭中的訊息。

在我們了解 HTTP 的細部運作過程之後。接下來，我們將重新用 Socket 製作一個與上述使用 HTTP 有相同結果的範例。

使用 **Socket** 進行網路連線 ▼

比起使用 HTTP 進行連線，使用 Socket 算是比較複雜的。因為 Socket 屬於比較低階的連線方式，我們必須熟知連線指令。但是，使用 Socket 也是最具彈性的連線方式。

使用步驟如下：

1. 在 Connector.open()之中使用 Socket 當作通訊協定。如果 Connector.open()與伺服器連線成功，Connector.open()會傳回被向上轉型之後的 StreamConnection 物件。為了以後使用的方便我們必須將它轉型回 StreamConnection 類別。

   本範例所使用的程式碼為：

```
String url = "socket://127.0.0.1:80" ;
StreamConnection sc = (StreamConnection) Connector.
open(url) ;
```

2. 利用 StreamConnection 的 openOutputStream 開啟輸出資料流，如此才能傳送命令給伺服器。然而 OutputStream 只能用來傳送二進位資料，為了正確地送出指令，我們必須將 OutputStream 與 DataOutputStream 串接在一起。

   本範例所使用的程式碼為：

```
DataOutputStream dos =
new DataOutputStream(sc.openOutputStream()) ;
```

3. 輸出資料流建立完成之後，我們開始對伺服器下命令。

   本範例所使用的程式碼為：

```
String command = "GET /test.html HTTP/1.0 \n\n" ;
byte cmd[] = command.getBytes() ;
    dos.write(cmd,0,cmd.length) ;
dos.flush() ;
```

4. 送出指令之後，我們要利用 StreamConnection 的 openInput Stream 開啟輸出資料流，如此才能接收伺服器傳送過來的資料。然而 InputStream 只能用來接收二進位資料，為了正確地送出指令，我們必須將 InputStream 與 DataInputStream 串接在一起。

   本範例所使用的程式碼為：

   ```
   DataInputStream dis =
   new DataInputStream(sc.openInputStream()) ;
   ```

5. 當輸入資料流開啟之後，我們開始利用 DataInputStream 類別的 read()函式將所收到的字元一個一個讀進來。

   本範例所使用的程式碼為：

   ```
   dis.read()
   ```

   當讀取完畢之後，read()函式會傳回 -1，代表讀取結束。

SocketTest.java

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import javax.microedition.io.* ;
import java.io.*;

public class SocketTest extends MIDlet
{
   private Display display;

   public SocketTest()
   {
      display = Display.getDisplay(this);
   }
   public void startApp()
```

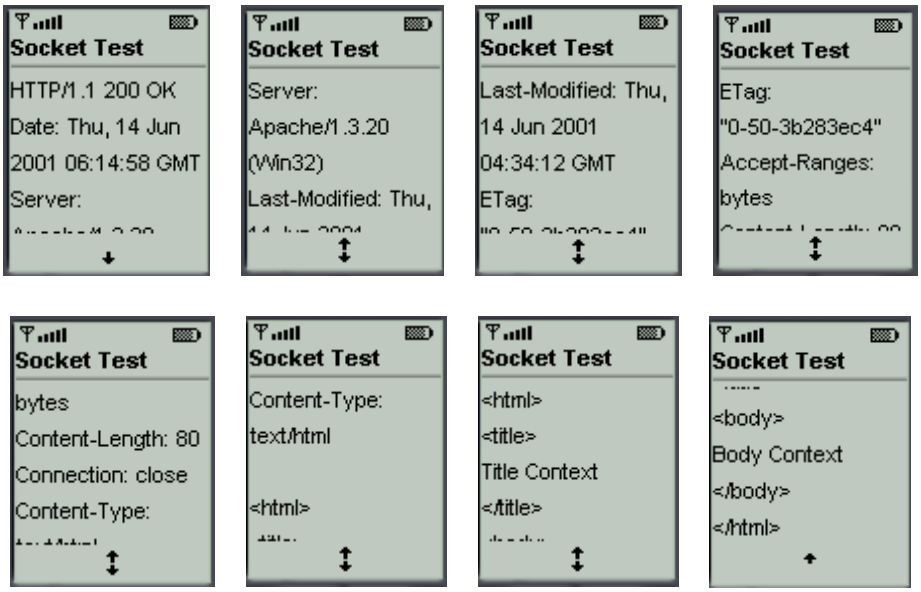
```
    {
     try
     {
          String url = "socket://127.0.0.1:80" ;
          String command = "GET /test.html HTTP/1.0 \n\n" ;
          StreamConnection sc = (StreamConnection)
Connector.open(url) ;

          DataOutputStream dos =
new DataOutputStream(sc.openOutputStream()) ;
          byte cmd[] = command.getBytes() ;
          dos.write(cmd,0,cmd.length) ;
          //dos.writeChars(command) ;
          dos.flush() ;

          //取得伺服器輸出
          DataInputStream dis =
new DataInputStream(sc.openInputStream()) ;
          String content = "" ;
          int ic ;
          while( (ic = dis.read()) != -1 )
          {
               content = content + (char)ic ;
          }

          Form f = new Form("Socket Test");
          f.append(content) ;
          display.setCurrent(f) ;
     }catch(Exception e)
     {
          System.out.println(e.getMessage()) ;
          notifyDestroyed() ;
     }
    }
    public void pauseApp()
    {
    }
    public void destroyApp(boolean unconditional)
    {
    }
}
```

【執行結果】



注　意

我們送出命令的程式碼為：

```
String command = "GET /test.html HTTP/1.0 \n\n" ;
byte cmd[] = command.getBytes() ;
dos.write(cmd,0,cmd.length) ;
dos.flush() ;
```

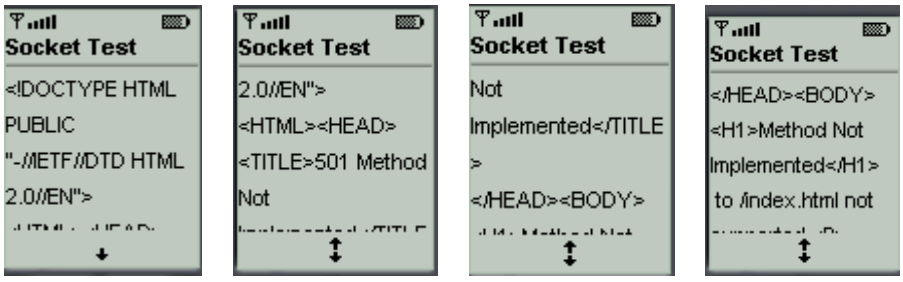
如果您查詢 DataOutputStream 的使用說明，您會看到有另外一個名為 writeChars()的函式，可以直接將字串寫入到輸出資料流，用法如下：

```
String command = "GET /test.html HTTP/1.0 \n\n" ;
        dos.writeChars(command) ;
        dos.flush() ;
```

請不要使用這個方法來傳送命令，因為這個字串會把命令轉換成

□G□E□T□□/□t□e□s□t□.□h□t□m□l□□H□T□T□P□/□1□.□0□□\n□\n

方式送出（□表示空白），這是從 Unicode 轉換成 ASCII 時所發生的問題，導致這麼一來伺服器會認為這是一個錯誤的命令，而 MIDlet 的輸出變成以下結果：



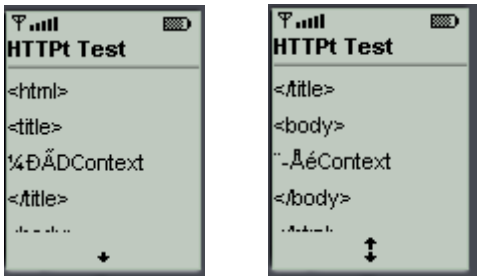
網路連線的中文問題

前面的範例用的都是英文作測試，如果遇到中文的話呢？首先，請將 test.html 的內容修改成：

Test.html

```
<html>
<title>
標題 Context
</title>
<body>
身體 Context
</body>
</html>
```

您會看到 HTTPtest 的輸出結果如下：



中文的部分全部變成了亂碼!!

之所以會有這種結果，原因在於我們的模擬器支援 Unicode 的緣故。由於網頁上的中文是採用 Big5 編碼，所以我們所撰寫的 MIDlet 從輸入資料流所讀進的中文皆是以 Big5 編碼，因此模擬器無法正確地顯示中文。(換句話說，架構於 Palm OS 上的 MIDP for Palm OS 因爲不支援 Unicode，中文系統只支援 Big5 編碼，所以 MIDlet 在 Palm OS 上執行的時候並不會發生上述的中文問題，一切運作正常)。

但是，如果您要在模擬器上顯示正確的中文，而今後可以執行 MIDlet 的裝置如果也完整地的支援 Unicode，那麼我們勢必採取一些措施。解決方案非常多，但是底下筆者只為各位提供三個較為方便的解決方案：

1. 讀入 Big5 之後，利用 Big5 ➡ Unicode 的對照表將 Big5 編碼轉換成 Unicode 編碼。

如此一來中文就可以正確地顯示在模擬器上。本方法看起來很簡單，但是實際上卻有執行上的難度。因為這張對照表太大（原始碼佔了將近 300K），所以我們不可能將它內建在 MIDlet 之中，因此除非系統提供，否則有其困難。當然，我們也可以只取這張表中常用的子集合，但是如何歸納出常用的子集合...就有待您的努力。

2. 將網頁改以 UTF8 的形式寫入 ➞ MIDlet 讀入之後，以要使用邏輯運算式將 UTF8 轉回 Unicode 即可立刻使用，可行。
3. 使用 ASCII 型態的 Unicode ➞ 這是一個稍微複雜，但是所發花費的資源較少的處理方式，可行。

底下我們為大家講解第三種方法。

首先，我們要借助 native2ascii.exe 這個工具幫我們將以 Big5 編碼的中文字轉換成 ASCII 型態的 Unicode。使用指令如下：

```
native2ascii -encoding Cp950 test.html test.htm
```

轉換之後，

Test.htm

```
<html>
<title>
\u6a19\u984cContext
</title>
<body>
\u8eab\u9ad4Context
</body>
</html>
```

這些\u6a19\u984c 與\u8eab\u9ad4 就是我們所謂"ASCII 型態的 Unicode"。

接著，我們撰寫工具函式如下：

```
private String unexpandString(String s)
{
    if(s == null) return null;
    StringBuffer result = new StringBuffer();
    int savedI, i, j, ch;
    for(i=0; i<s.length(); i++)
     {
          if((ch = s.charAt(i)) == '\\')
          {
               if(s.length() > i + 1 && s.charAt(i+1) == 'u')
               {
                    savedI = i;
                    i+=2;
                    while(s.length() > i && s.charAt(i) == 'u')
                    {
                         i++;
                    }
                    if(s.length() >= i + 4)
                    {
                         ch =
Integer.parseInt(s.substring(i, i+4), 16);
                         i+=3;
                    }else{
                         i = savedI;
                    }
               }
          }
          result.append((char)ch);
    }
    return result.toString();
}
```

這個工具函式可以幫我們把 ASCII 型態的 Unicode 轉換成二進位型態的 Unicode。

完整的使用範例如下：

SocketTest.java

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import javax.microedition.io.* ;
import java.io.*;

public class Big5OK extends MIDlet
{
   private Display display;

   public Big5OK()
   {
      display = Display.getDisplay(this);
   }
   public void startApp()
   {
    try
    {
        //注意，使用 ASCII 型態 Unicode 的版本是 test.htm，不是
test.html
String url = "http://127.0.0.1/test.htm" ;
        HttpConnection hc = (HttpConnection)
Connector.open(url) ;

        //取得伺服器輸出
        DataInputStream dis =
new DataInputStream(hc.openInputStream()) ;
        String content = "" ;
        int ic ;
        while( (ic = dis.read()) != -1 )
        {
            content = content + (char)ic ;
        }
        String trans = unexpandString(content) ;
        Form f = new Form("HTTPt Test");
        f.append(trans) ;
```

```
        display.setCurrent(f) ;
    }catch(Exception e)
    {
        System.out.println(e.getMessage()) ;
        notifyDestroyed() ;
    }
   }
   private String unexpandString(String s)
   {

    if(s == null) return null;

    StringBuffer result = new StringBuffer();

    int savedI, i, j, ch;

    for(i=0; i<s.length(); i++)
    {
            if((ch = s.charAt(i)) == '\\')
        {
            if(s.length() > i + 1 && s.charAt(i+1) == 'u')
            {
                savedI = i;
                i+=2;
                while(s.length() > i && s.charAt(i) == 'u')
                {
                    i++;
                }
                if(s.length() >= i + 4)
                {
                    ch = Integer.parseInt(s.substring(i, i+4),
16);
                    i+=3;
                }else{
                    i = savedI;
                }
            }
            }
            result.append((char)ch);
    }
    return result.toString();
   }
   public void pauseApp()
   {
   }
```

```
    public void destroyApp(boolean unconditional)
    {
    }
}
```

【執行結果】

我們可以看到，經過 unexpandString()函式的轉換之後，就可以顯示正常的中文。

最後重複強調，如果您使用 MIDP for Palm OS 撰寫 Palm OS 上的 MIDlet，因為 Palm OS 並不支援 Unicode，而且我們所使用的中文系統（如掌龍、CJKOS）都是使用 Big5 編碼。因此我們的網頁和 MIDlet 全都不需要任何處理，就可以讀入一般正常的中文網頁，因為從網頁中讀進來的就已經是 Big5 編碼的資料。

在上述範例之中，我們可以利用函式將 ASCII 型態的 Unicode 轉換成二進位的 Unicode 型態。可是這是因為我們的網頁是靜態的，所以在 MIDlet 接收前我們可以利用 native2ascii.exe 幫我們做轉換。可是萬一網頁內容是動態的呢？我們總不能每次網頁產生之後再借助 native2ascii.exe 吧！如果您是使用 JSP/Servlet 開發動態網頁，底下提供您一段 Java 程式碼，可以幫助您將 2 進位的

Unicode 轉換成 ASCII 型態的 Unicode，這麼一來，您的 JSP/Servlet 只要在最後輸出的時候利用這個函式將中文做好處理，就可以和 MIDlet 搭配的天衣無縫了 ^^”。

```
private String expandString(String s)
{
   StringBuffer result = new StringBuffer();
   int i, j, ch;
   // native to ascii
   for (i = 0; i < s.length(); i++)
   {
     if (s.charAt(i) > 0x007f)
   {
        // write \udddd
       result.append('\\');
       result.append('u');
       String hex =
         Integer.toHexString(s.charAt(i));
       StringBuffer hex4 = new StringBuffer(hex);
       hex4.reverse();
       int len = 4 - hex4.length();
       for (j = 0; j < len; j++)
      {
         hex4.append('0');
       }
       for (j = 0; j < 4; j++)
       {
          result.append(hex4.charAt(3 - j));
       }
     }else
     {
        result.append(s.charAt(i));
     }
   }
   return result.toString();
}
```

總　結

或許本篇所探討的 MIDP 網路程式設計內容讓您覺得不過癮。這是因為我們只單純地討論利用 MIDP/CLDC 所提供的類別函式庫完成連線、傳送、接收這三種簡單的工作。網路是一個很大的議題。因此我們將在下一本進階篇之中結合伺服器端的 JSP/Servlet 作深入的說明。