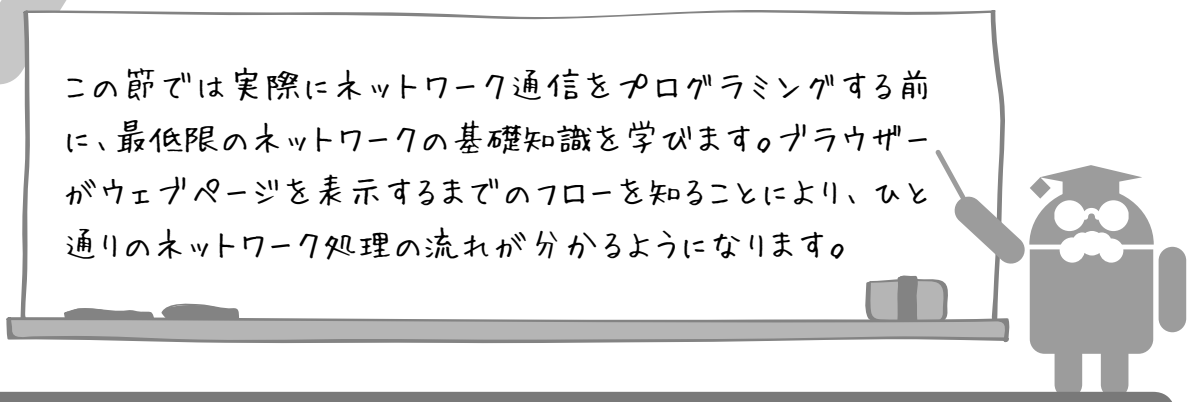
第14章

ネットワークプログラミング

著：山下智樹、橋爪香織

14-1 ネットワークプログラミング

著：山下智樹



この節で学ぶこと

- そもそもネットワークとは？
- IPアドレスとは何か
- 名前解決（DNS）について
- プロトコルというルール
- ポート番号とは？
- Telnetを使ったHTTPリクエストの送り方

この節で出てくるキーワード一覧

ネットワーク	HTTP
ホスト	ウェブサーバー
リクエスト	telnet
レスポンス	ヘッダー
クライアント	ボディ
サーバー	ステータスコード
IPアドレス	ポート番号
名前解決 / DNS	Well-Known Port
プロトコル	

14-1-1 ネットワークとは？

ネットワーク通信とは、2つ以上のコンピューターでデータのやりとりをするための手法です。データのやり取りの際、送受信を行うコンピューターのことを「ホストコンピューター（以下、ホスト）」と呼びます。

また、データの送り手と受け手についても呼び分けています。Androidの「ブラウザー」アプリを使って「http://tomorrowkey.jp/」というウェブページを開いたとします。このとき、Android端末は送り手の役割となりますね。この送り手のことを「クライアントコンピューター（以下、クライアント）」と呼んでいます。

また、インターネットの向こう側にある、データを返す側となるホストのことを「サーバー」と表現します（**図1**）。

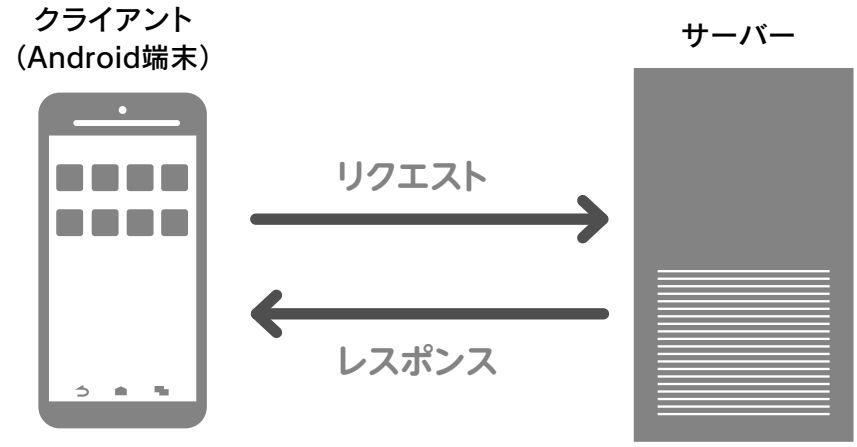


図1: Androidでウェブページを表示するとき

データのやり取りに関する動作についても、呼び方があります。ウェブページを要求する動作を「リクエスト」と呼びます。リクエストに対し、返信をすることを「レスポンス」と呼びます。リクエストが行われると、かならずレスポンスが返ってきます。ネットワーク通信では、クライアントからサーバーにリクエストを送り、サーバーからレスポンスを取得することをセットで考えます。

14-1-2 IPアドレスとは？

インターネットにはたくさんのホストが繋がっています。ネットワーク通信を行う際は、どのホストと通信するかを指定する必要があります。各ホストは異なる番号を持ち、それを「IPアドレス」と呼びます。IPアドレスは次のように4つの数字をドットで区切った形式で表します。

IPアドレスの例

192.168.12.10

ここで説明しているIPアドレスは、IPv4と呼ばれるものです。容量を大きくしたIPv6というものもあります。詳しくは"IPアドレス枯渇問題"というキーワードでウェブ検索してみてください。

みなさんが使っているパソコンにも、IPアドレスが割り当てられています。あなたのパソコンがどんなIPアドレスを使っているのか確認してみましょう。

IPアドレスを調べるには次のコマンドを実行します。使用しているOSがWindowsの場合はコマンドプロンプトで次のコマンドを実行します。

IPアドレスを調べるコマンド(Windows)

```
> ipconfig
```

使用しているOSがMac OSまたはLinux、Free BSDの場合はターミナルで次のコマンドを実行します。

リスト6.2: IPアドレスを調べるコマンド(OSX, Linuxなど)

```
> ifconfig
```

14-1-3 数字だけのアドレスではわかりにくい！

前項でIPアドレスについて説明しました。基本的には、インターネット上のサーバーなどはIPアドレスが割り振られており、これを入力することでアクセスできるようになります。しかし、企業のホームページやニュースサイトにアクセスする際、ブラウザーにIPアドレスを入力することはありません。代わりに「www.○○○.com」「○○.jp」といったURLを入力しますよね? 数字だけの覚えにくいアドレスを、わかりやすい名称にするために「名前解決」と呼ばれる技術を用いています。名前解決は「DNS(Domain Name System)」とも呼ばれます。

名前解決は、覚えやすい文字列の「ホスト名」を、数字の「IPアドレス」に変換する作業を行います。また、その逆の変換も可能なので、名前解決ではホスト名とIPアドレスを相互に変換することができるのです。

さきほどブラウザーでウェブページにアクセスした時を例に具体的な説明をします。ブラウザーでは「http://tomorrowkey.jp/」というURLを入力しました。URL中の「tomorrowkey.jp」という部分がホスト名です。ブラウザーはホスト名をDNSサーバーに問い合わせてIPアドレスを取得します。このとき、DNSサーバーは「tomorrowkey.jp」という値を「49.212.164.150」という値に変換してクライアントへ返します。その後、ようやくブラウザーはサーバーと通信を始めるのです。

名前解決がどのように行われるかは、コマンドプロンプトやターミナルから名前解決のコマンドを実行すると理解できることと思います。

名前解決のコマンド

```
nslookup <ホスト名>
```

実行すると次のようにサーバーのIPアドレスを取得できます。

nslookupの実行結果

```
$ nslookup tomorrowkey.jp
Server:   220.159.212.200
Address:  220.159.212.200#53

Non-authoritative answer:
Name: tomorrowkey.jp
Address: 49.212.164.150
```

ホスト名とIPアドレスは同じように扱われます。そのため、さきほど取得したIPアドレス「220.159.212.200」をブラウザーに入力すると「http://tomorrowkey.jp/」と同じページが表示されます。

14-1-4 プロトコルとは？

サーバーとクライアントの2つのホストが結ばれたとき、どのような内容のデータをやりとりするのでしょうか。

まずは、データをどのような順番で送るかをあらかじめ決めておかなければなりません。サーバーとクライアントとの間で、伝送方法のルールを取り決めておく必要があります。そのルールのことを「プロトコル」と呼びます。

さきほど説明した「DNS」も、プロトコルの一種です。DNSには、名前をDNSサーバーに渡すことでIPアドレスを返すというルールが決められています。

ウェブサーバーからウェブページを取得するときに使われるプロトコルを「HTTP（Hypertext Transfer Protocol）」と呼びます。ホストに対して特定のページがほしいとリクエストを送ると、そのページがレスポンスとして返されます。

ブラウザーで「http://tomorrowkey.jp/」にアクセスした場合を例に、どのような値がレスポンスとして返されるかを確認してみましょう。ブラウザーはサーバーに対して次のようなリクエストを送っています。

```
GET / HTTP/1.0
User-Agent: Mozilla/5.0 (Linux; Android 4.3; Build/LPV79)
 AppleWebKit/537.36 (KHTML, like Gecko)
 Chrome/36.0.1985.94 Mobile Safari/537.36
Host: tomorrowkey.jp
```

最初の「/」では通常「index.html」または「index.htm」が返されますが、別のファイルを参照させたい場合などは、サーバー管理者がウェブサーバー上の設定ファイル「.htaccess」に記述することで変更できるようになっています。クライアントからの参照先の変更はできません。よく、ウェブを見ていると「404 Not Found」といったページを開くことがありますね? これは、該当のフォルダーに「index.html」がないときに「404 Not Found」と書かれたページを返すようサーバー上の設定ファイルに記述されているのです。

まずは、1行目から説明します。この行はHTTPでアクセスする概要が指定されています。最初のパラメータでは、アクセスの種類を指定しています。前述の例では「GET」となっていますね。これはファイルを取得するという意味です。ほかに「POST」「PUT」「DELETE」といった種類の指定方法があります。スペースで区切り、2つ目のパラメータ「/」が指定されています。これは、とくにサーバー上での指定がない限りは「index.html」と解釈されます。さらにスペースで区切った後ろにHTTPのバージョンを書きます。

2行目以降は「ヘッダー」と呼ばれる、リクエストの付加情報が書かれています。レスポンスにもヘッダーがあるので、それと区別するために「リクエストヘッダー」とも呼ばれます。「User-Agent」にはどのようなソフトウェアを使ってアクセスしているかが指定されています。「Host」にはアクセスするホストの名前「tomorrowkey.jp」が入ります。

ヘッダーの下には「ボディ」を書きます。ボディには、サーバーに送りたいデータを設定します。GETリクエストではボディを設定することはできないので、このリクエストにはボディがありません。

なお、レスポンスにもボディがあるので、区別するために「リクエストボディ」とも呼びます。

続いてレスポンスを見てみましょう。

```
HTTP/1.1 200 OK
Date: Sun, 13 Jul 2014 06:20:10 GMT
Server: Apache/2.2.15 (CentOS)
Last-Modified: Sat, 07 Jun 2014 14:29:18 GMT
ETag: "26133e-f3-4fb3fcdaabf43"
Accept-Ranges: bytes
Content-Length: 243
Connection: close
Content-Type: text/html

<html>
<head>
<title>Hello, Tomorrow!</title>
</head>
<body>
<h1>Hello, Tomorrow!</h1>
</body>
</html>
```

1行目を「ステータスライン」といい、HTTPのバージョンとステータスコードが設定されます。ステータスコードはレスポンスの結果を表します。上記の例では「200 OK」になっていますね。これは正常にレスポンスを返していることを示します。ほかにリクエストがおかしい事を表す400番台のステータスコードや、サーバー側のエラーを表す500番台のエラーがあります。

2行目以降はヘッダーが続きます。リクエストする時に設定するヘッダーと区別するために、「レスポンスヘッダー」と呼ぶことがあります。「Content-Length」にはボディの長さが入り、「Content-Type」にはボディの種類を返します。「text/html」とは「HTML」形式のファイルを表します。この表現方法については「MIME」タイプで検索してみてください。ほかにもたくさんのヘッダーがありますが、ここでは説明しませんので、気になるヘッダーがあったらキーワード検索してみましょう。

空行までがヘッダーで、それ以下をボディと言います。ここにレスポンスの実際のデータが入っています。今回の例では「http://tomorrowkey.jp/」にアクセスし、その下にある「index.html」がボディに入っています。ブラウザーはこのボディを解釈してウェブページとして表示しています。

14-1-5 ポート番号とは？

1つのホストでは複数のソフトウェアが動いていています。ネットワーク通信をする際に、IPアドレスを指定しただけでは、そのホストのどのソフトウェアとやりとりをしたいのかが分かりません。リクエストを送るソフトウェアを特定するための番号を「ポート番号」と呼びます。

各プロトコルによって使われるポート番号は異なります。一部のプロトコルについては一般的に使われるポート番号が決まっています。それらを「Well-Known Port」と呼びます。例えばDNSであれば53で、HTTPであれば80です。すべてを覚えておく必要がありませんが、よく使われるポート番号は覚えておくと便利です。

ポート番号	プロトコル
20	FTP(データ)
21	FTP(制御用)
22	SSH
23	Telnet
25	SMTP
53	DNS

ポート番号	プロトコル
67	DHCP(サーバー)
68	DHCP(クライアント)
80	HTTP
110	POP3
143	IMAP
443	HTTPS

表1: 代表的なWell-Known Port

14-1-6 リクエストを送ろう

HTTPリクエストは、ブラウザーでなくてもコマンドとして実行することができます。LinuxやMacをご使用の方はターミナルやコンソールで実行できますが、Windowsでは、コマンドプロンプトからの実行ができません。

Windowsの場合は「PuTTY」などの「telnetクライアント」をダウンロードする必要があります。まずは、以下のURLからソフトを入手します(**図2**)。

http://www.chiark.greenend.org.uk/~sgtatham/putty/

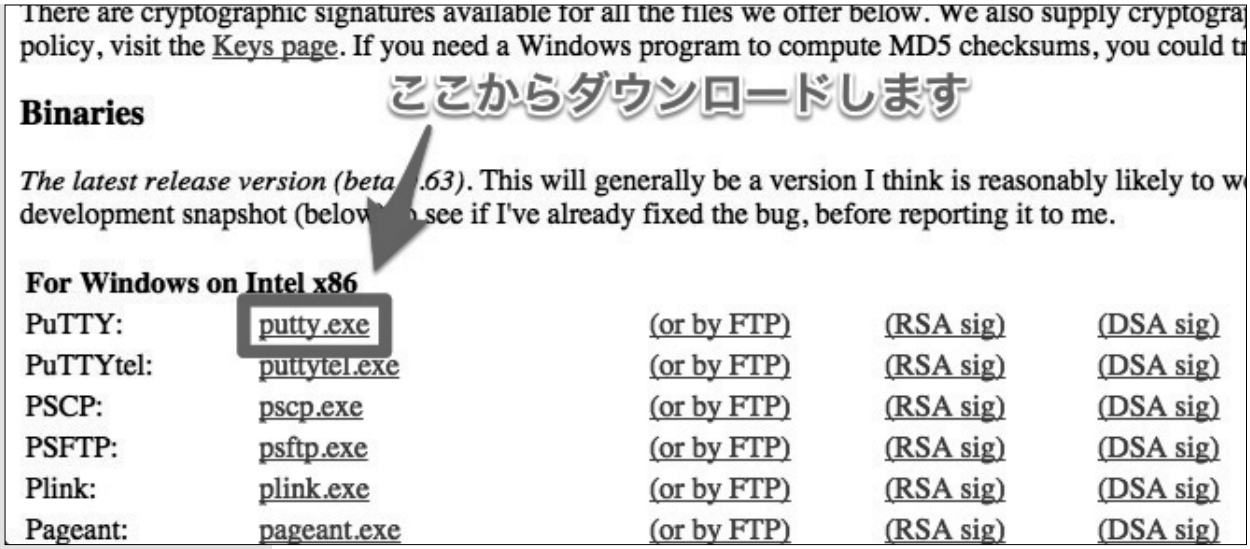


図2: PuTTYのダウンロード画面

PuTTYを起動すると設定画面が表示されます。**図3**を参考に「Category」ペインから「Session」を選択して、「Host Name」欄に「tomorrowkey.jp」を入力、「Port」欄に「80」を入力。「Connection type」欄を「Raw」に指定して、「close window on exit」で「Never」を選択します。

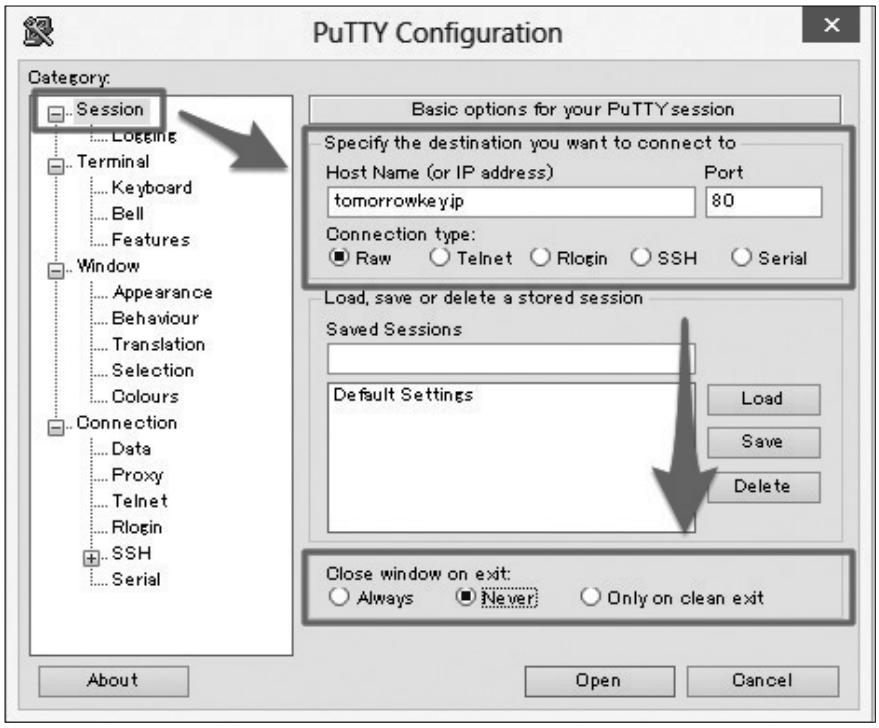


図3: PuTTYの設定1

次に「Category」ペインから「Terminal」を選択します。「Impilcit CR in every LF」にチェックを入れて「Open」をクリックします(**図4**)。

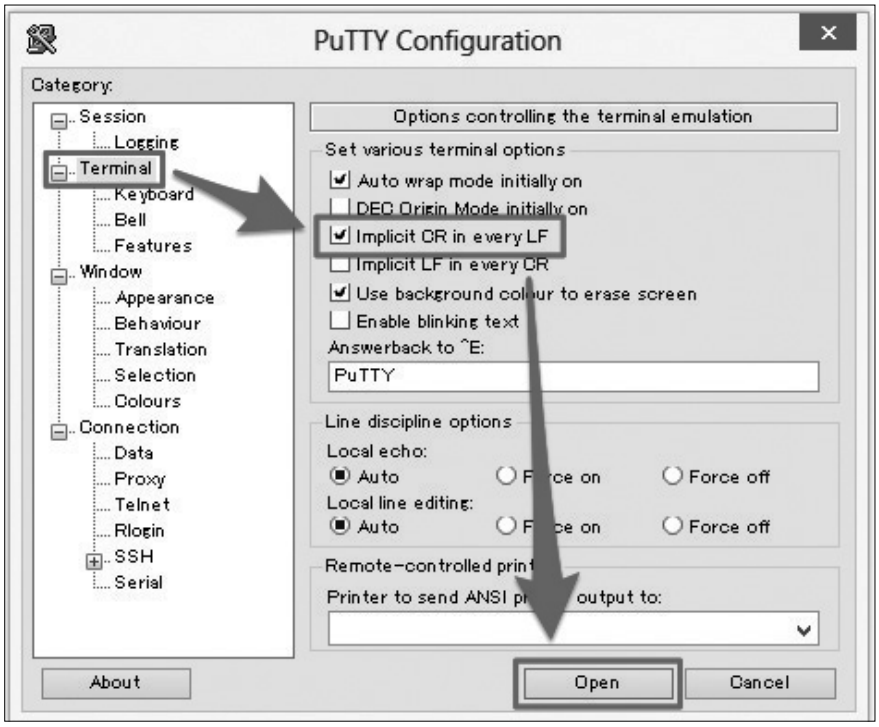


図4: PuTTYの設定2

すると、次の画面が開きます。以上でリクエストを送る準備は完了です(**図5**)。

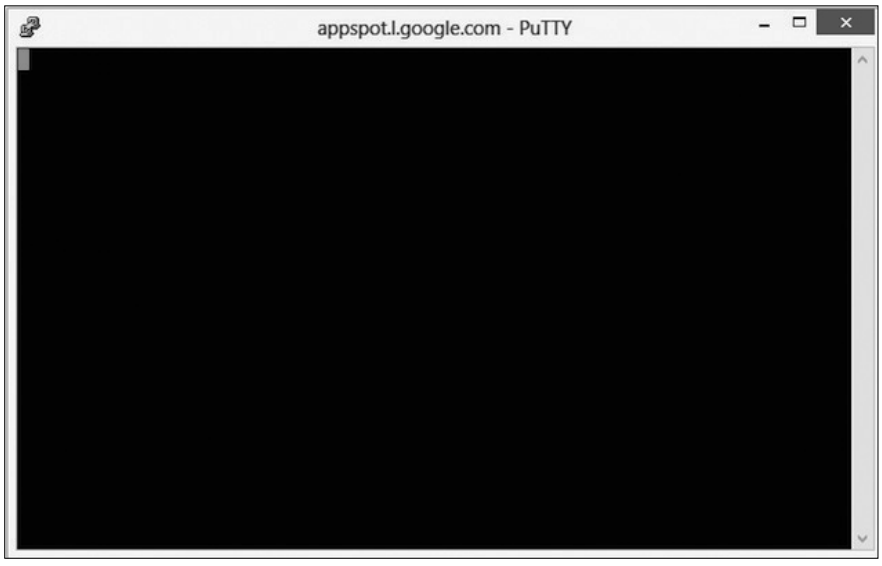


図5: PuTTYの起動

MacやLinuxでは、ターミナルやコンソールを起動して、そのままTelnetコマンドを使うことができます(「telnet tomorrowkey.jp 80」とタイプします)。

HTTPリクエストを送るための準備が完了したら、次のコマンドを実行してみましょう。

Telnetを使ったHTTP通信のリクエスト

```
GET / HTTP/1.1
Host: tomorrowkey.jp
User-Agent: telnet
```

最後に改行を2回することを忘れないようにしてください。実行してみると次のようなレスポンスが返ってきます。

Telnetを使ったHTTP通信のレスポンス

```
HTTP/1.1 200 OK
Date: Fri, 05 Sep 2014 05:24:41 GMT
Server: Apache/2.2.15 (CentOS)
Last-Modified: Sat, 07 Jun 2014 14:29:18 GMT
ETag: "26133e-f3-4fb3fcdaabf43"
Accept-Ranges: bytes
Content-Length: 243
Content-Type: text/html

<html>
<head>
<title>Hello, Tomorrow!</title>
</head>
<body>
<h1>Hello, Tomorrow!</h1>
</body>
</html>
```

以上で、リクエストの送信とレスポンスの内容が確認できましたね。

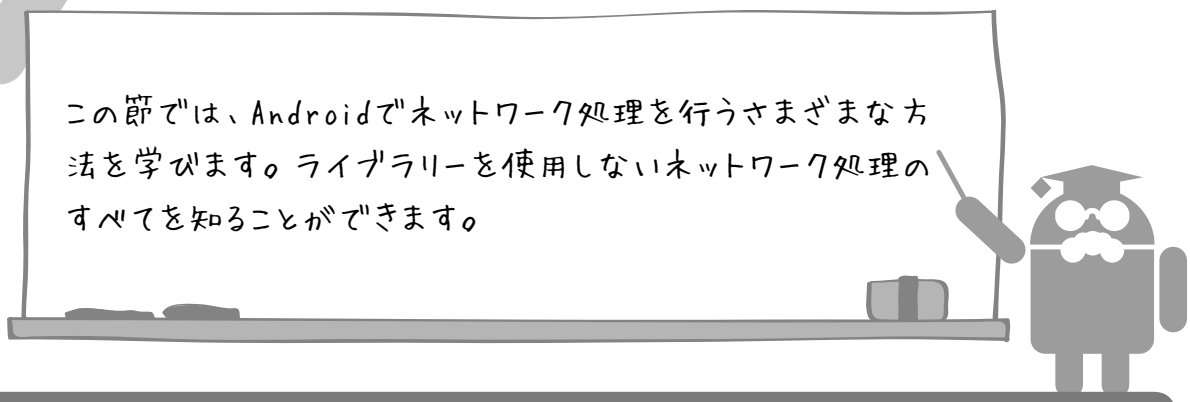
演習問題

・説明したところ以外へHTTPリクエストを送ってみましょう

・Jpegファイルを取得するHTTPを送ってみましょう

14-2 Androidでのネットワーク処理

著：山下智樹



この節で学ぶこと

- Androidの通信方式
- アプリでネットワーク処理を書く上で注意すべきこと
- Socketを使ったネットワーク通信
- HttpURLConnnectionを使ったネットワーク通信
- HttpClientを使ったネットワーク通信

この節で出てくるキーワード一覧

Permission
ストリーム
java.net.Socket
java.net.InetSocketAddress
java.io.OutputStream
java.io.InputStream
java.net.UnknownHostException
java.io.IOException
java.net.HttpURLConnection
java.net.URL
java.net.MalformedURLException
org.apache.http.client.methods.HttpGet
org.apache.http.client.HttpClient
org.apache.http.HttpResponse

14-2-1 Androidアプリでネットワークを使うために

Androidアプリでインターネットに接続するには、「android.permission.INTERNET」という「パーミッション」が必要です。

パーミッションとは、アプリで使用する特別な機能を宣言し、ユーザーの許可を得るもので、AndroidManifest.xmlに書き込んで設定します（下記のコードを参照）。

たとえば、バイブレーションを使う場合には「android.permission.VIBRATE」を設定します。GPSを使う場合には「android.permission.ACCESS_FINE_LOCATION」を設定します。

ここに設定したパーミッションは、アプリをインストールする直前にユーザーに表示されます（**図1**）。

インターネットパーミッションを追加する

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="com.example.sample.network">

  <uses-permission android:name="android.permission.INTERNET" />

  <application
```

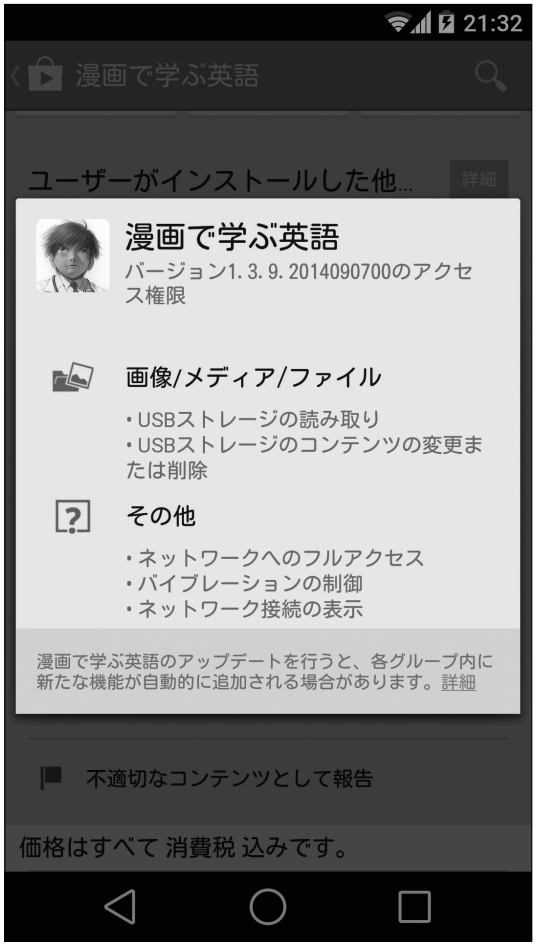


図1：アプリのインストールに際して、パーミッションが表示された例

14-2-2 Androidで使用できる通信方式

Android搭載端末のほとんどは、通信機能を持っています。通信機能にはさまざまな種類がありますが、現在多く利用されているのが、携帯電話基地局を利用したモバイル通信です。モバイル通信には「3G」「LTE」「WiMAX」「AXGP」といった種類があります。一方、自宅や職場、カフェなどの屋内ではWi-Fi通信を利用することがあります。Wi-Fi通信にも「802.11ac」「802.11g」といった、さまざまな方式が利用されています。他にも、NFCやBluetoothといった通信方式はありますが、インターネット接続の手段として用いられることは少ないので省略します。

この章では、モバイル通信とWi-Fi通信を用いてHTTPでのやり取りを行う方法を紹介します。

14-2-3 Androidでネットワーク処理をする上で注意すること

Androidのアプリケーションには、サーバーのアプリケーションやデスクトップ向けアプリケーションとくらべて、2つの大きな特徴があります。1つはネットワークが不安定ということと、もう1つは電池消費量に気をつけなくてはならないということです。

ネットワークは不安定

Androidを搭載した端末の多くは携帯電話です。さまざまな場所に持ち歩くものなので、必ずしもネットワークに恵まれた場所にいるとは限りません。アプリを作るにあたって、いつネットワークが切れても問題ないように作らなくてはなりません。また、ネットワークが切れてしまった場合にユーザーにストレスをかけないような作りを心がけることも重要です。

電池消費量

モバイル環境下では電池消費量にも気を配らなくてはなりません。モバイルで電池を消耗しやすい機能の1つがネットワーク通信です。できるだけ通信量を減らす、通信回数を減らす、一度にまとめて通信するといった事を心がけると、電池消費にも優しいアプリを作ることができます。

メインスレッドと非同期スレッド

理論値では100Mbpsを越えるネットワークでも環境によっては数百bpsしかでないため、小さいデータのやりとりであっても時間がかかるものだと考えるべきです。時

間のかかる処理はUIスレッドで行ってはいけません。必ず非同期処理ができる仕組み（AsyncTaskなど）を使うようにしましょう。

14-2-4 Androidで使えるネットワークAPI

ここからは、Androidでネットワーク通信をプログラミングをする具体的なコードを示します。Androidのネットワークプログラミングに用意されたAPIは、以下の3つです。

まず1つ目が「Socket」です。ソケットプログラミングとも呼ばれ、一番原始的な実装方法です。高い柔軟性がありますが、その分多くのコードを書かないといけなかったり、HTTPプロトコルに精通していなければ実装することはできません。

2つ目は「HttpURLConnection」です。HTTPプロトコルについて詳しく知らなくても、ある程度実装できるように用意されたJavaのAPIです。特にライブラリーを使用しなければ、たいていの人がこのAPIを使います。

最後が「HttpClient」です。実は、このAPIはJavaフレームワークに含まれていないものですが、Androidでは標準搭載しているため、ライブラリーの参照をしなくても使うことができます。HttpURLConnectionと比べてインターフェイスが分かりやすいのですが、Androidのバージョンによっては不具合を含んでいることがあり、あまり使われません。

ネットワーク通信全体の流れ

以上のように、ネットワークプログラミングのためのAPIは3種類ありますが、全体の流れはどれを使う場合でもほぼ同じになります。

具体的なコードに入る前に、全体の流れを説明しましょう。

・接続するサーバーを指定
・サーバーとの接続を確立
・リクエストの送信
・レスポンスの受信

接続するサーバーを指定

接続するサーバーを指定します。IPアドレスを指定すれば、それがそのまま使われます。ホスト名を指定した場合は自動的に名前解決されて、IPアドレスに変換されます。

サーバーとの接続を確立

クライアントとサーバーで通信を行うための、前処理をします。たいていの場合は

「open()」や「connect()」といったメソッドで、接続が確立されます。もしネットワークが安定しておらず、接続できなかった場合は、「IOException」が投げられます。

リクエストの送信

接続が確立できたら、サーバーにリクエストを送ります。

Javaに限った話ではないのですが、ネットワークプログラミングをする際には「Stream」という知識が必要です。

Streamとは、データを順次に送るパイプのようなイメージです。そのパイプに1つずつデータを送っていきます。

Javaでデータを送信するには、「OutputStream」を使います。OutputStream.write()メソッドにバイト配列を書き込むことで、サーバーにデータを送ることができます。なお、Streamは使い終わったら、「close()」メソッドを呼んで閉じなければならないというルールがあります。

レスポンスの受信

データを送りきったら、今度はレスポンスを受信します。

ここでもまた、Streamで順次データを受け取らなくてはなりません。

Javaでデータを受信するには「InputStream」を使います。InputStream.read()メソッドでバイト配列を読み込むことで、サーバーからデータを受け取ることができます。このStreamも、使い終わったらclose()メソッドを呼んで閉じなければなりません。

14-2-5 ソケットプログラミング

Androidで一番低レベルなネットワーク通信方法は、「Socket」を使ったやり方です。HTTPサーバーにアクセスするには少し貧弱ですが、HTTP以外のサーバーにもアクセスできるので便利です。たとえば「http://tomorrowkey.github.io」にアクセスしようとした場合、次のようなプログラムになります。「AsyncTask」の「doInBackground()」メソッドに書くといいでしょう。

Socketの実装方法の例

```
try {
  // リクエスト
  Socket socket = new Socket();                                    ❶
  socket.connect(new InetSocketAddress("tomorrowkey.github.io", 80));  ❷

  String request = "GET / HTTP/1.1\n" +
    "Host: tomorrowkey.github.io\n" +
    "\n\n";
  OutputStream outputStream = socket.getOutputStream();
  outputStream.write(request.getBytes());                          ❸
  outputStream.flush();

  // レスポンス
  InputStream inputStream = socket.getInputStream();
  byte[] buffer = new byte[1024];
  int length;
  while ((length = inputStream.read(buffer)) != -1) {              ❹
    Log.d("TEST", new String(buffer, 0, length));
  }

  outputStream.close();
  inputStream.close();
} catch (UnknownHostException e) {
  throw new RuntimeException(e);                                   ❺
} catch (IOException e) {
  throw new RuntimeException(e);
}
```

どのような処理をしているか、順番に見ていきましょう。

リクエストを送信する

❶では「Socketインスタンス」を生成しています。Socketインスタンスは、通信を行う役割を持っています。ホスト名からIPアドレスへの変換や、そこから先の通信も行います。先ほどはTelnetを使って通信しましたが、それと同等の機能を持ちます。

Socketインスタンスができたら、❷でホスト名とポート番号の接続先を指定します。今回はHTTPなので、ポート番号は80番になります。

❸は「Socket」でリクエストするために、直接「OutputStream」へHTTPリクエストを書いています。

リクエストの組み立て

Socketで通信を行う場合、プロトコルに従った通信内容を自分で組み立てる必要があります。今回はHTTP通信なので、HTTPのプロトコルに従ったリクエストを自分で組み立てます。

ウェブサーバーからページを取得する命令は「GET」です。一番単純なGETリ

クエストは次のようになります。

GETリクエスト

```
GET / HTTP/1.1
Host: tomorrowkey.github.io
```

1行目では大まかなリクエスト内容を送ります。"/"とリクエストすると、たいていのウェブサーバーでは"/index.html" と解釈されます（**表1**）。

GET	/	HTTP/1.1
ファイルを取得する	"/"以下にある「index.html」を取得する	HTTP/1.1というプロトコル（手続き方法）を使う

表1: GETリクエスト1行目

2行目からはリクエストヘッダー（リクエスト時の付属情報）を送信します（**表2**）。

Host:	tomorrowkey.github.io
ヘッダーのキー	Hostキーはリクエスト先のホスト名を指定

表2: GETリクエスト2行目

リクエストの最後に改行を2回送るとリクエスト完了となります。

レスポンスを受け取る

以下のように「socket」から「InputStream」を取得してHTTPレスポンスを取得します（❹）。

レスポンスの取得

```
InputStream inputStream = socket.getInputStream();
byte[] buffer = new byte[1024];
int length;
while ((length = inputStream.read(buffer)) != -1) {
  Log.d("TEST", new String(buffer, 0, length));
}
```

レスポンスの取得まで完了したら「InputStream」と「OutputStream」の「close」メソッドを呼んで「Stream」を閉じましょう（❺）。

実行結果

実行すると、logcatにレスポンスが出力されます。

Socketを使ったリクエストのレスポンス

```
D/TEST    ( 1371): HTTP/1.1 200 OK
D/TEST    ( 1371): Server: GitHub.com
D/TEST    ( 1371): Content-Type: text/html; charset=utf-8
D/TEST    ( 1371): Last-Modified: Mon, 02 Jan 2012 05:54:49 GMT
D/TEST    ( 1371): Expires: Mon, 30 Jun 2014 11:37:13 GMT
D/TEST    ( 1371): Cache-Control: max-age=600
D/TEST    ( 1371): Content-Length: 169
D/TEST    ( 1371): Accept-Ranges: bytes
D/TEST    ( 1371): Date: Mon, 30 Jun 2014 12:24:02 GMT
D/TEST    ( 1371): Via: 1.1 varnish
D/TEST    ( 1371): Age: 3409
D/TEST    ( 1371): Connection: keep-alive
D/TEST    ( 1371): X-Served-By: cache-ty66-TYO
D/TEST    ( 1371): X-Cache: MISS
D/TEST    ( 1371): X-Cache-Hits: 0
D/TEST    ( 1371): X-Timer: S1404131042.773465872,VS0,VE174
D/TEST    ( 1371): Vary: Accept-Encoding
D/TEST    ( 1371):
D/TEST    ( 1371): <html>
D/TEST    ( 1371): <!DOCTYPE html>
D/TEST    ( 1371): <html lang="ja">
D/TEST    ( 1371): <head>
D/TEST    ( 1371): <title>tomorrowkey GitHub page</title>
D/TEST    ( 1371): <meta charset="UTF-8" />
D/TEST    ( 1371): </head>
D/TEST    ( 1371): <body>
D/TEST    ( 1371): <h1>Hello, tomorrow!!</h1>
D/TEST    ( 1371): </body>
D/TEST    ( 1371): </html>
```

HTTPレスポンスの「header」と「body」両方が出力されました。「Socket」で通信する場合はこの文字列をJavaのオブジェクトに変換する処理が必要です。「Socket」通信はリクエストを自分で組み立てる必要があったり、レスポンスが生データのままであったりするので大変ですが、とても自由度が高いのが特徴です。

14-2-6 HttpURLConnectionの利用

「Socket」ではリクエストを自分で組み立てる必要がありましたが、HTTP／HTTPSにアクセスする場合もっと便利なクラスがあります。それが「HttpURLConnection」です。どれほど便利かを確認するために、先ほどと同様に「http://tomorrowkey.github.io」へアクセスしてみましょう。「AsyncTask」の「doInBackground()」メソッドに、以下のコードを実装します。

HttpURLConnectionの実装方法

```
try {
  URL url = new URL("http://tomorrowkey.github.io");                    ❶
  HttpURLConnection connection = (HttpURLConnection) url.openConnection();  ❷
  connection.setRequestMethod("GET");
  connection.setRequestProperty("Host", "tomorrowkey.github.io");       ❸
  connection.connect();                                                 ❹

  int responseCode = connection.getResponseCode();                      ❺
  Log.d("TEST", "responseCode=" + responseCode);
  String contentLength = connection.getHeaderField("Content-Length");
  Log.d("TEST", "Content-Length=" + contentLength);
  String contentType = connection.getHeaderField("Content-Type");       ❻
  Log.d("TEST", "contentType=" + contentType);

 InputStream inputStream = connection.getInputStream();
byte[] buffer = new byte[1024];
int length;
while ((length = inputStream.read(buffer)) != -1) {                     ❼
    Log.d("TEST", new String(buffer, 0, length)); }
}
  inputStream.close();
} catch (MalformedURLException e) {
  throw new RuntimeException(e);
} catch (IOException e) {
  throw new RuntimeException(e);
}
```

「HttpURLConnection」で、どのような処理をしているか、順番に見ていきましょう。

リクエストを送信する

❶では、アクセスするURLを使いURLオブジェクトを作っています。

❷では「openConnection()」メソッドを使い「HttpURLConnection」を取得します。ここで注意したいことがあります。ここで使いたいクラスはHttpURLConnectionなのですが、「URL.openConnection()」メソッドの戻り値は「URLConnection」なので「HttpURLConnection」に変換しなければなりません。

リクエストはファイルの取得なので❸の「setRequestMethod()」メソッドで「GET」を渡します。また「setRequestProperty()」メソッドを使うことでリクエストヘッダーを設定できます。「Host」の値は設定しなくても自動的に追加されるのですが、説明上、あえて追加しています。

❹ではサーバーと接続するために「connect」メソッドを利用しています。

レスポンスを受け取る

ステータスコードを取得するよう❺で「getResponseCode()」メソッドを呼び出しています。

❻は、レスポンスヘッダーを取得するために「getHeaderField()」メソッドを呼び出しています。

❼レスポンスボディを取得するために「getInputStream()」メソッドを呼び出します。

その後、「InputStream」を文字列に変換してログに出力しています。

実行結果は次のとおりです。

HttpURLConnectionを使ったリクエストのレスポンス

```
D/TEST    ( 1231): responseCode=200
D/TEST    ( 1231): Content-Length=null
D/TEST    ( 1231): contentType=text/html; charset=utf-8
D/TEST    ( 1231): body=<html>
D/TEST    ( 1231): <!DOCTYPE html>
D/TEST    ( 1231): <html lang="ja">
D/TEST    ( 1231): <head>
D/TEST    ( 1231): <title>tomorrowkey GitHub page</title>
D/TEST    ( 1231): <meta charset="UTF-8" />
D/TEST    ( 1231): </head>
D/TEST    ( 1231): <body>
D/TEST    ( 1231): <h1>Hello, tomorrow!!</h1>
D/TEST    ( 1231): </body>
D/TEST    ( 1231): </html>
```

「Socket」とは異なり、HTTPプロトコルを楽に使えるようなメソッドが用意されているため、簡単に取りあつかうことができます。

14-2-7 HttpClient (Apache Http)

GET / POSTInputStream / OutputStream

Androidには、Apacheソフトウェア財団の「HttpClient」が標準で入っています。 Apacheは、オープンソースソフトウェアのプロジェクトを支援している団体で、この団体によってさまざまなプログラミング言語の多様なライブラリーが公開されています。

「HttpClient」はそのライブラリーのうちの1つで、JavaでHTTPにアクセスするために作られたものです。JavaにはHttpURLConnectionとSocketを使った通信方法がありますが、HttpClientはより使いやすく作られています。

HttpClientの実装方法

```
try {
  HttpGet httpGet = new HttpGet("http://tomorrowkey.github.io");  ❶
  httpGet.addHeader("Host", "tomorrowkey.github.io");  ❷
  HttpClient httpClient = new DefaultHttpClient();  ❸

  HttpResponse httpResponse = httpClient.execute(httpGet);  ❹

  StatusLine statusLine = httpResponse.getStatusLine();
   Log.d("TEST", "Status-Code=" + statusLine.getStatusCode());

   Header contentLengthHeader = httpResponse.getFirstHeader("Content-Length");
   Log.d("TEST", "Content-Length=" + contentLengthHeader.getValue());
   Header contentTypeHeader = httpResponse.getFirstHeader("Content-Type");
   Log.d("TEST", "Content-Type=" + contentTypeHeader.getValue());

   InputStream inputStream = httpResponse.getEntity().getContent();
   String body = readToEnd(inputStream);
   Log.d("TEST", body);
   inputStream.close();
} catch (MalformedURLException e) {
   throw new RuntimeException(e);
} catch (IOException e) {
   throw new RuntimeException(e);
}
```

「HttpClient」ではどのような処理になるのか、順番に見ていきます。

リクエストを送信する

❶はGETリクエストを行うための「HttpGet」オブジェクトを作っています。引数にはアクセスするURLを指定しています。

❷はヘッダーの指定のために「addHeader」メソッドを使っています。

❸では、実際に通信を行う「HttpClient」を生成します。「HttpClient」はInterfaceなので、そのままではインスタンス化することはできません。今回は「DefaultHttpClient」を使って「HttpClient」を生成しています。「DefaultHttpClient」の代わりに、Androidにより最適化された「AndroidHttpClient」を使うこともできます。

❹では生成した「httpClient」の「execute()」メソッドにリクエストオブジェクト「(httpGet)」を渡して通信をしています。通信したレスポンスはHttpResponseというクラスで戻り値に返ってきます。

レスポンスを受け取る

ステータスコードを取得するために「getStatusLine()」メソッドでステータスラインを取得したあとで「getStatusCode()」メソッドを呼びます。

ステータスラインとは、得られたHTTPレスポンスの1行目を指します。

ステータスライン

```
HTTP/1.1 200 OK
```

レスポンスヘッダーを取得するには「getFirstHeader()」メソッドで「Header」オブジェクトを取得します。「Header」オブジェクトの値を取得するには「getValue()」メソッドを以下のようにして呼びます。

ヘッダーの取得

```
Header contentLengthHeader = httpResponse.getFirstHeader("Content-Length");
Log.d("TEST", "Content-Length=" + contentLengthHeader.getValue());
Header contentTypeHeader = httpResponse.getFirstHeader("Content-Type");
Log.d("TEST", "Content-Type=" + contentTypeHeader.getValue());
```

レスポンスボディを取得する場合、getInputStream()メソッドを呼びます。

レスポンスボディの取得

```
InputStream inputStream = httpResponse.getEntity().getContent();
String body = readToEnd(inputStream);
Log.d("TEST", body);
inputStream.close();
```

リクエストとレスポンスのオブジェクトが分かれているので、HttpURLConnectionと比べてイメージしやすいと思います。。実行結果は次のとおりです。

HttpClientを使ったリクエストのレスポンス

```
D/TEST    ( 1295): Status-Code=200
D/TEST    ( 1295): Content-Length=169
D/TEST    ( 1295): Content-Type=text/html; charset=utf-8
D/TEST    ( 1295): <html>
D/TEST    ( 1295): <!DOCTYPE html>
D/TEST    ( 1295): <html lang="ja">
D/TEST    ( 1295): <head>
D/TEST    ( 1295): <title>tomorrowkey GitHub page</title>
D/TEST    ( 1295): <meta charset="UTF-8" />
D/TEST    ( 1295): </head>
D/TEST    ( 1295): <body>
D/TEST    ( 1295): <h1>Hello, tomorrow!!</h1>
D/TEST    ( 1295): </body>
D/TEST    ( 1295): </html>
```

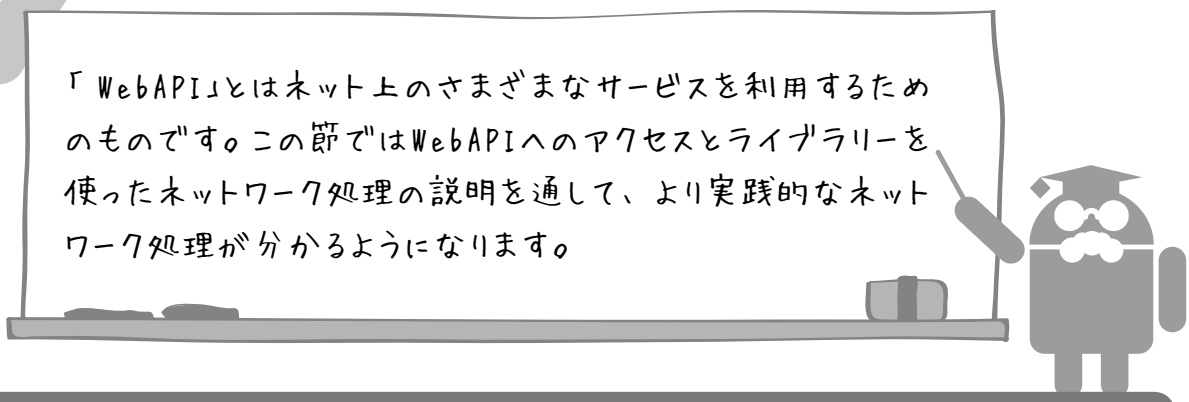
演習問題

・Hostヘッダーに実際のホストとは異なる値を入れてみましょう

・すべてのレスポンスヘッダーをログに出力してみましょう

14-3 WebAPIの利用

著：山下智樹



この節で学ぶこと

・Volleyを使ったネットワーク処理

・WebAPIへのアクセス

この節で出てくるキーワード一覧

JSON

XML

Volley

API

IME

文字コード

Volley

14-3-1 XMLとJSON

「WebAPI」をあつかう前にデータ形式について知っておく必要があります。ウェブサーバーとアプリとでデータをやりとりする際のデータフォーマットとして広く使われている「XML」と「JSON」について説明します。

リソースや設定ファイルなどに使われる「XML」

「XML（eXtensible Markup Language）」は、1998年に標準化されたデータ形式で、アプリの設定ファイルやオフィスアプリのファイル形式として広く利用されています。Androidでも「strings.xml」などのリソースファイルや「AndroidManifest.xml」で設定ファイルとして使われています。

XMLのサンプル

```
<?xml version="1.0" encoding="utf-8" ?>
<school>
  <count>2</count>  ❶
  <students>
    <student age="18" gender="male">Taro Yamada</student>  ❷
    <student age="19" gender="female">Hanako Tanaka</student>
  </students>
</school>
```

このコードには「Taro Yamada」「Hanako Tanaka」という2人の生徒がいる、というデータが入っています。

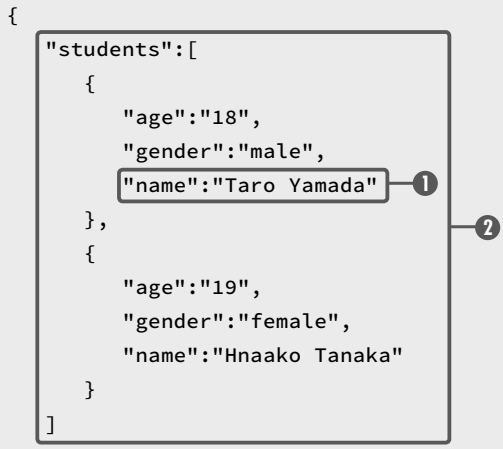
XMLでは、半角の「<」「>」に囲まれたタグで要素を囲います。たとえば、❶を見てください。ここには生徒の数を表す「count」が、行末の「/count」までの間に書かれている内容であることを定義しています。わかりやすく言い換えれば、「count」を「2」としているのです。タグを開始する場合は要素の前後に「<」「>」を付けて、<count>と記述します。続けて内容を定義し、末尾には半角スラッシュを付けた</count>で閉じます。

また、タグの中には、要素の付加的な情報を付けることができます。これを属性と呼びます。❷を見てください。タグの開始部分が「student」だけでなく、続けて「age="18" gender="male"」となっていますね。この部分が属性です。

もうひとつのデータ形式「JSON」

「JSON（JavaScript Object Notation）」は2006年に標準化された、比較的新しいデータ形式です。当初はJavaScriptであつかわれていたデータ形式ですが、XMLより見やすいため、今では幅広い用途に使われています。

JSONのサンプル



JSONを構成するのは、「KeyValue」の「JsonObject」と、配列の「JsonArray」です。具体的には、上記の例の❶の部分が「JsonObject」で、❷の部分が「JsonArray」です。WebAPIを使う時には、XMLやJSONなどのデータ形式をやりとりします。

14-3-2 WebAPIにアクセスする

それではWebAPIにアクセスしてみましょう。まず、そもそもWebAPIとはなにかという話から始めましょう。各種オンラインサービスはAPI（Application Programming Interface）を公開していることがあり、それを使うことでそのサービスが提供している情報をあつかうことができるようになります。それをWebAPIや、単にAPIと呼びます。

WebAPIを使うと、自分だけのオリジナルのTwitterアプリを作ることができます。また天気予報サービスのWebAPIを使えば、自分好みの天気予報アプリを作れます。

サービスから提供されている情報を簡単に使えるのでとても便利なのですが、WebAPIを使う前に気をつけなければならないことがあります。

料金体系

WebAPIを使うための料金体系はさまざまです。完全に無料のものもありますが、たいていは一定期間のアクセス数に応じて制限がかかったり、有料になったりします。

利用規約を読む

すべてのWeb APIには利用規約があり、使用するための条件があります。たとえば、オープンソースアプリや無料で公開するアプリについては無料でAPIを使うことができるが、広告を掲載したり有償でアプリを公開する場合はAPI使用料を払わないといけないケースがあります。

ほかにもAPIを使用するにあたって気をつけなければならないことが利用規約に書いてありますので、必ず使用する前に熟読しましょう。もしこの利用規約を守れなかった場合、予告のないサービスの提供停止や、場合によってはサービス提供元から法的手段をとられる場合も考えられるので注意しましょう。

14-3-3 ソーシャルIMEを使ってみよう

ここでは、「ソーシャルIME」というサービスを使って、入力したひらがなを変換してみましょう。

「Input Method Editor(IME)」は文字入力を補助するソフトウェアです。言葉で説明するとすごく分かりづらいですが、ざっくりとわかりやすく説明すると、キーボードを使って入力するときにひらがなを漢字に変換してくれるソフトウェアです。ソーシャルIMEでは、変換するための仕組みをWebAPIとして提供しています。

ソーシャルIMEは簡単に試すことができます。たとえば、次のURLにブラウザーでアクセスしてみてください。

```
http://www.social-ime.com/api/?string=けいたいでんわ
```

すると、次のように変換された結果がタブ区切りで表示されます。

```
携帯電話　けいたいでんわ  ケイタイデンワ
```

それではこれをAndroidプログラムに実装してみましょう。

ソーシャルIMEは下記を参照してください。
「Social IME ～みんなで育てる日本語入力～」
http://www.social-ime.com/

Social IME

```
try {
  String keyword = params[0];
  URL url = new URL("http://www.social-ime.com/api/?string=" + keyword);
  HttpURLConnection connection = (HttpURLConnection) url.openConnection();
  connection.connect();
  InputStream inputStream = connection.getInputStream();

  StringBuilder sb = new StringBuilder();
  int length;
  byte[] buffer = new byte[1024];
  while ((length = inputStream.read(buffer)) != -1) {
    sb.append(new String(buffer, 0, length, "EUC-JP"));
  }
  return sb.toString();
} catch (IOException e) {
  throw new RuntimeException(e);
}
```

基本的には前述した「HttpURLConnection」を使ったアクセスと同じです。

気をつけなければならないのは、指定する文字コードです。ソーシャルIMEでは特に指定しなければ、レスポンスは"EUC-JP"という文字コードでレスポンスを返すプロトコルを採用しています。受け取ったbyte配列からそのまま文字列を生成すると文字化けを起こしてしまうので、文字コードを指定します。WebAPIを使う際に文字化けを起こしてしまうことはよくあるので注意しましょう。

WebAPIごとにリクエスト、レスポンスのフォーマットが異なるため、仕様に合わせてリクエストを組み立て、レスポンスを分析してください。

わたしたちは、普段何気なくコンピューターを使ってテキストを入力していますが、コンピューターは数字を計算することしかできません。コンピューターでも処理できるようにするには、アルファベットやひらがな、漢字などの文字1つ1つを、対応する数字の羅列に置き換えなければなりません。その対応表のことを「文字コード」といいます。文字コードには、数字やアルファベット、一部の記号のみをサポートした「ASCII」、日本語をサポートするために作られた「Shift JIS」、全世界の言葉を網羅するために作られた「Unicode」など、いくつかの種類があります。

14-3-4 ライブラリを使ったネットワーク通信

ネットワーク通信をする度に「AsyncTask」を継承して、同じようなバックグラウンド処理を書くのは大変です。バックグラウンド処理を毎回書かなくてもいいようなライブラリーがグーグルから公開されています。名前は「Volley」と呼びます。

すでに「10.2.1 ネットワーク経由でデータを取得する」でVolleyをダウンロードしているので、それを使ってみましょう。

Volleyの参照方法を再掲します。

プロジェクトを作成し、プロジェクトを右クリックしてプロパティを開きます。左側のペインで「Android」をクリックし、右下の「Add...」をクリックします。

「Volley」は以下のURLを参照してください。
https://android.googlesource.com/platform/frameworks/volley/

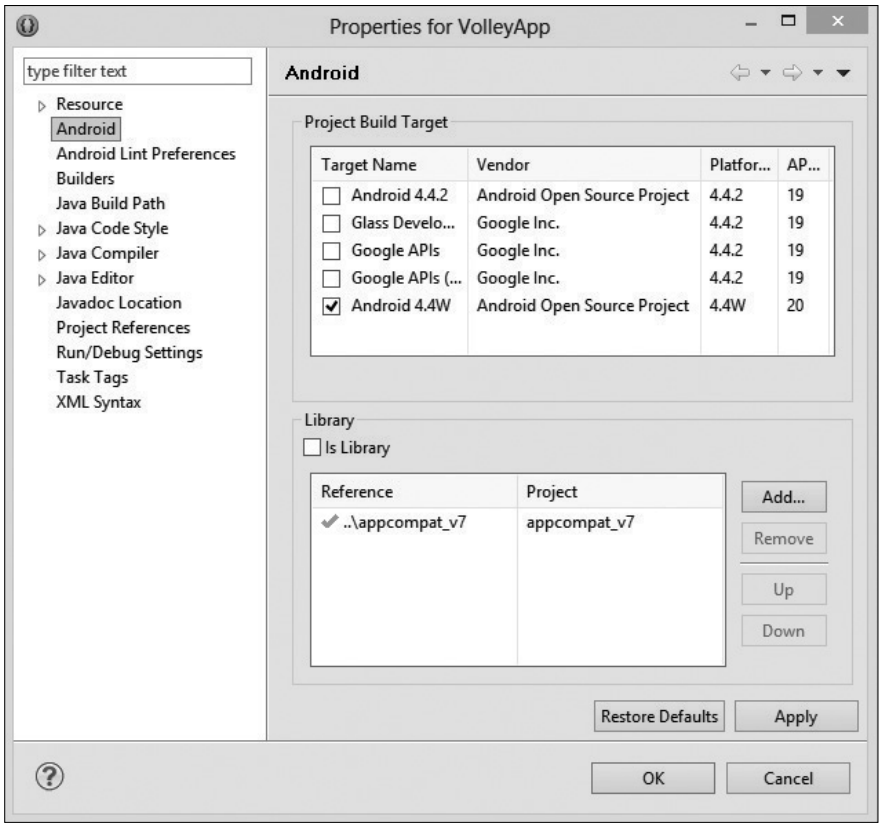


図1: 左ペインで「Android」を選択し、「Add...」をクリック

プロジェクトを選択するダイアログが表示されるので「Volley」を選択してOKをクリックします。

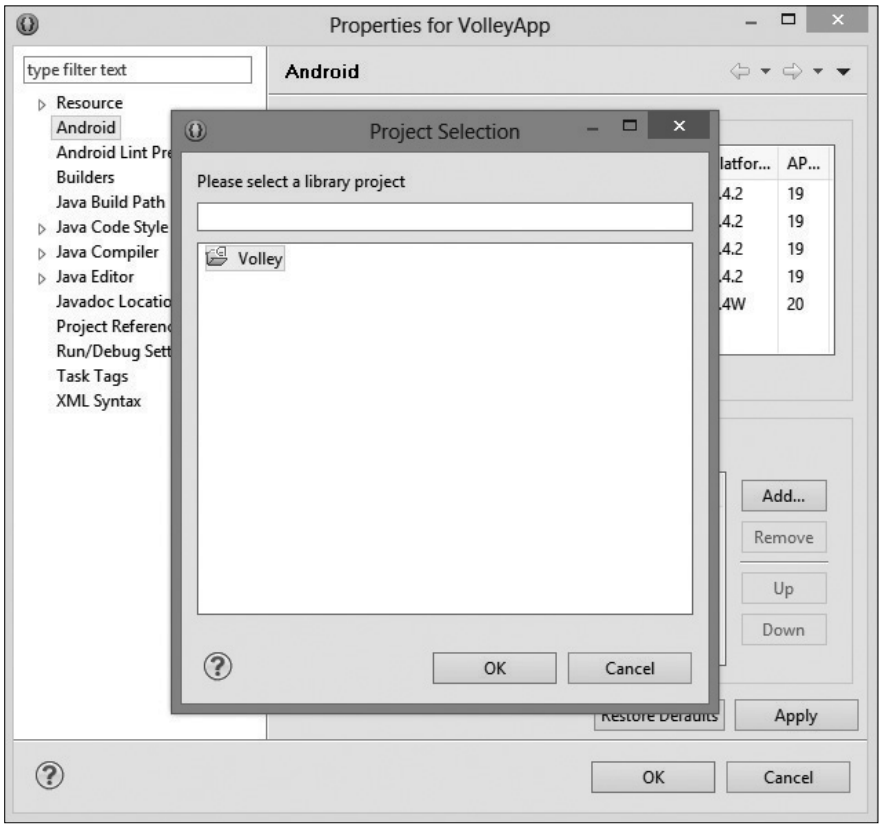


図2: 表示されるダイアログで「Volley」を選択

これでVolleyを使う準備は完了です。

次に「JSON」ファイルをダウンロードしてログに出力するプログラムも実装します。

APIアクセス

```
mRequestQueue = Volley.newRequestQueue(getApplicationContext());  ❶

int method = Request.Method.GET;
String url = "https://raw.githubusercontent.com/TechBooster/
    AndroidOpenTextbook/master/code/network/assets/sample.json";
JSONObject requestBody = null;
Response.Listener<JSONObject> listener = new Response.Listener<JSONObject>() {
  @Override
  public void onResponse(JSONObject jsonObject) {
    Log.d("TEST", jsonObject.toString());
  }
};                                                                  ❷
Response.ErrorListener errorListener = new Response.ErrorListener() {
  @Override
  public void onErrorResponse(VolleyError volleyError) {
    String message = volleyError.getMessage();
    Log.d("TEST", message);
  }
};                                                                  ❸

mRequestQueue.add(new JsonObjectRequest(method,
     url, requestBody, listener, errorListener));
```

リクエストを送信する

まずは❶のようにリクエストキューを作ります。この作成されたリクエストキューにリクエストを追加することで、自動的にバックグラウンドで随時リクエストを送ります。ネットワークが不安定で失敗したとしても、リトライ処理をある程度行ってくれます。

次に、リクエストを追加します。リクエストを追加する際には次のパラメータが必要です（**表1**）。

method	リクエストメソッドを指定する
url	アクセスするURLを指定する
requestBody	リクエスト時にボディに送るJSONObjectを指定する
listener	レスポンスリスナー
errorListener	エラーレスポンスリスナー

表1: GETリクエスト2行目

リクエストをリクエストキューに追加すると自動的にバックグラウンドでネットワーク通信が行われ、ネットワーク通信が完了すると引数に渡したコールバック（「listener」「errorListener」）が実行されます。

レスポンス

サーバーのレスポンスが正常系(200～299)だった場合、❷の「listener」のコールバックメソッドが実行されます。レスポンスボディは自動的に「JSONObject」にパースされ、引数に渡されます。

正常系レスポンスリスナー

```
@Override
public void onResponse(JSONObject jsonObject) {
  Log.d("TEST", jsonObject.toString());
}
```

正常系なレスポンス

```
D/TEST    ( 1699): {"users":[
    {"id":1,"gender":"female","name":"alice"},
    {"id":2,"gender":"male","name":"bob"}]}
```

サーバーのレスポンスが異常系(200～299以外)だった場合、❸のerrorListenerのコールバックメソッドが実行されます。エラーの内容は引数のVolleyErrorオブジェクトに入っています。

エラー系レスポンスリスナー

```
@Override
public void onErrorResponse(VolleyError volleyError) {
  NetworkResponse networkResponse = volleyError.networkResponse;
  int statusCode = networkResponse.statusCode;
  Log.d("TEST", "Status-Code=" + statusCode);

  String contentLength = networkResponse.headers.get("Content-Length");
  Log.d("TEST", "Content-Length=" + contentLength);

  String body = new String(networkResponse.data);
  Log.d("TEST", body);
}
```

エラーレスポンス

```
D/TEST    ( 1654): Status-Code=404
D/TEST    ( 1654): Content-Length=9
D/TEST    ( 1654): Content-Type=null
D/TEST    ( 1654): Not Found
```

「Volley」を使うと非同期処理を書かなくていいのでとても便利ですね。このようなライブラリーを使うと独自で実装するよりも手間が省けます。その一方、実装の自由度は低くなってしまうことを覚えておくとよいでしょう。とはいっても、一般的なネットワーク通信において、「Volley」のようなライブラリは非常に有効です。

まとめ

さまざまな方法を使ってサーバーからファイルを取得する方法を実装しました。やり方がたくさんあってどれを使っていいか迷うかもしれませんが、それぞれの実装方法の特徴とやりたいことを比較して、最適な方法を選択してください。

なにかアプリを作りたいけど、アイデアが浮かばない！というときにWebAPIを眺めていると、あのAPIとこのAPIを組み合わせれば面白いことができるぞ！なんて思いつくことがあります。たいていのWebAPIはHTTP通信ができれば使えるので、おもしろいWebAPIを見つけて奇抜なアプリを作りましょう！

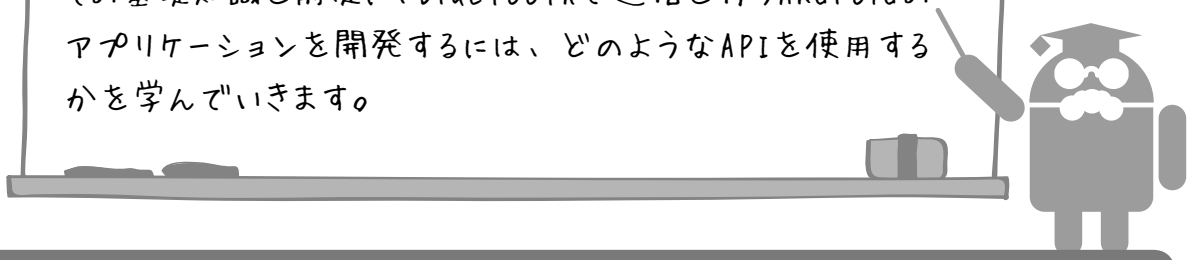
演習問題

- Mashup Award(http://mashupaward.jp/) を見て気になるAPIを見つけましょう。
- 気になるAPIを見つけたら利用規約を読んでみましょう。
- APIのサンプルを参考にしながら実際にアクセスしてみましょう。

14-4 Bluetooth通信

著：橋爪香織

この節では、まず、Bluetooth通信の基礎知識を学びます。その基礎知識を前提に、Bluetoothで通信を行うAndroidのアプリケーションを開発するには、どのようなAPIを使用するかを学んでいきます。



この節で学ぶこと

- Bluetooth通信の基礎知識（電波帯、バージョン、Class、プロファイル、ペアリング）
- AndroidのクラシックBluetooth用API概要と使用方法(SPP接続でのデータ通信)
- HSP/HFP/A2DP/HDP プロファイルサポートの概要
- Bluetooth Low Energy(BLE)

この節で出てくるキーワード一覧

BluetoothAdapter
BluetoothDevice
BluetoothSocket
BluetoothServerSocket
BluetoothProfile.ServiceListener
BluetoothHeadset
BluetoothA2dp
BluetoothHealth
BluetoothManager
LeScanCallback
BluetoothGattCallback
BluetoothGatt
BluetoothGattService
BluetoothGattCharacteristic
BluetoothGattDescriptor

14-4-1 Bluetooth 通信の基礎知識

Bluetoothは2.4GHzの電波帯を利用する通信速度が最大24Mbpsの近距離無線通信を行うための規格です。Bluetoothの規格の最新バージョンは4.0（2014年9月現在）です。

バージョンが3.0までと4.0では、Bluetoothの機能に大きな違いがあり、3.0以前は「クラシックBluetooth」と呼ばれています。クラシックといっても、現在流通している一般的なBluetooth機器は2.0から3.0のBluetoothです。

4.0は「Bluetooth Low Enagy」と呼ばれています。本節の前半部分はクラシックBluetoothが対象で、後半で「Bluetooth Low Enagy」を対象とした説明を掲載します。

また、電波強度によりBluetooth機器の種別がClass1～Class3と分かれており、Class1の電波の到達距離は100mとなっています（**表1**）。

クラス	出力	到達距離
Class1	100mW	100m
Class2	2.5mW	10m
Class3	1mW	1m

表1: Bluetoothのクラス別の到達距離

Android端末では主にBluetooth対応のキーボードやヘッドセットを使用する機会が多いですが、パソコンや他のAndroid端末と接続してデータのやり取りを行うこともできます。Bluetoothでは通信しようとする機器同士が同じ通信プロトコルのプロファイルを持っている場合に限り、そのプロファイルの機能に特化した通信ができます。一般的なBluetoothプロファイルの一部を紹介します（**表2**）。

プロファイル名	説明
SPP	Bluetooth機器を仮想シリアルポート化する
DUN	携帯電話・PHSを介してインターネットにダイアルアップ接続する
HID	マウスやキーボードなどの入力装置を無線化する
HSP	Bluetooth搭載ヘッドセットと通信する
HFP	車内やヘッドセットでハンズフリー通話を実現する
A2DP	音声をレシーバー付きヘッドフォンに伝送する
HDP	健康管理機器同士を接続する

表2: 一般的なBluetoothプロファイル

14-4-2 アプリケーションでBluetooth機能を使う

Androidのアプリケーションにbluetoothの機能を入れるには、「android.bluetooth」パッケージのAPIを使用します。使用するクラスは次のとおりです。

・BluetoothAdapter
・BluetoothDevice
・BluetoothSocket
・BluetoothServerSocket

リファレンスはhttp://developer.android.com/intl/ja/guide/topics/connectivity/bluetooth.htmlで確認できます。

これらのAPIはAndroid2.0（API Level 5）から使用できます。ただし一部の機能ではAndroid3.0（API Level 11）以降でだけ使えるAPIもあります。今後もAPIが増える可能性もありますので、使用する場合はリファレンスの確認もするようにしましょう。

Bluetooth APIをアプリケーションで使用することによって、次のような機能を実現できます。

RFCOMMはBluetoothでRS232Cによるシリアル通信をエミュレートするプロトコルです。

・他のBluetooth機器をスキャンする
・ペアリングされたBluetooth機器のBluetoothアダプターを照会する
・RFCOMMチャンネルを確立する
・サービスを検出して他の機器に接続する
・他の機器との間でデータをやりとりする
・複数の接続を管理する

Bluetoothを扱う際にはAndroidManifest.xmlの<uses-permission>に「android.permission.BLUETOOTH」のパーミッションの指定が必要です。また端末のBluetooth機能の有効/無効設定やリモートデバイスの検索を行う際は「android.permission.BLUETOOTH_ADMIN」のパーミッションも必要になります。

さらに、Bluetooth機能を用いたアプリケーションを開発する際には、注意すべき点があります。Bluetoothの接続処理やデータの送受信処理などでは処理が完了するまでに一定時間応答を待つことがあり、そのままではANR（Application Not Responding）となってしまう場合もあります。ANRを防ぐために、Bluetoothの接続処理やデータの送受信処理などは、ActivityのUIスレッドではなく別のスレッドで行うようにしましょう。

14-4-3 自端末のBluetooth設定制御（BluetoothAdapter）

アプリケーションからBluetoothの有効／無効を設定するサンプルプログラムを次に示します。

Bluetooth設定制御

```
BluetoothAdapter btAdapter = BluetoothAdapter.getDefaultAdapter();
  // btAdapter がnullの場合はBluetoothがサポートされていない
  if(btAdapter == null){
    // ユーザに通知するなど
    return;
  }
  if (!btAdapter.isEnabled()) {  // BluetoothがOFF状態
    // 有効にするリクエストを投げる
    Intent intent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
    startActivityForResult(intent, REQUEST_BT_ENABLE); // REQUEST_BT_ENABLEは"0"などの定数
  }
```

Bluetoothの有効/無効状態の取得にはBluetoothAdapterクラスのisEnabledメソッドを使用します。Bluetoothを有効にするにはインテントに「ACTION_REQUEST_ENABLE」を指定してstartAcitivityForResultメソッドを実行します。

アプリケーションの実行時、システムはユーザーに自動でBluetooth許可リクエストダイアログを表示します（**図1**）。ユーザーがダイアログのボタンを押し下げるとonActivityResultメソッドがコールバックされます。

Bluetoothが有効になった場合は、このメソッドの引数resultCodeにActivity.RESULT_OKが返ってきます。

図1: Bluetooth許可リクエストダイアログ（有効設定）

14-4-4 Bluetooth対応の外部デバイスの検索(BluetoothAdapter)

アプリケーションから、実行中のAndroid端末の近隣にあるBluetoothデバイスを検索するサンプルプログラムについて説明します。デバイス検索を行うにはBluetoothAdapterクラスのstartDiscoveryメソッドを使用し、次のように書きます。

デバイス検索

```
BluetoothAdapter  btAdapter = BluetoothAdapter.getDefaultAdapter();
if (btAdapter.isDiscovering()) {
  //  探索中だったら一度探索をキャンセル
  btAdapter.cancelDiscovery();
}
btAdapter.startDiscovery();
```

デバイス検索が終了したときに発行されるインテント「ACTION_DISCOVERY_FINISHED」と、デバイスが発見されたときに発行されるインテント「ACTION_FOUND」を受信できるように、BroadcastReceiverを作成し登録しておきます。

アクションインテントとBroadcastReceiverの登録

```
IntentFilter filter = new IntentFilter();
filter.addAction(BluetoothAdapter.ACTION_DISCOVERY_FINISHED);
filter.addAction(BluetoothDevice.ACTION_FOUND);
// mReceiverはBroadcastReceiverのインスタンス
registerReceiver(mReceiver, filter);
```

「ACTION_FOUND」のインテントには「BluetoothDevice.EXTRA_DEVICE」という名前でBluetoothDeviceクラスのオブジェクトが格納されているので、デバイス名やMACアドレスなどのデバイス情報が取得できます。

デバイス検索のアクションインテントの受信処理

```
// BroadcastReceiverのonReceiveメソッド
public void onReceive(Context context, Intent intent) {
  String action = intent.getAction();
  if (BluetoothDevice.ACTION_FOUND.equals(action)) {
    BluetoothDevice device =
      intent .getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
  // BluetoothDeviceオブジェクトからデバイス情報などを取得する処理
      :
  } else if(BluetoothAdapter.ACTION_DISCOVERY_FINISHED.equals(action)) {
    // デバイス探索が完了したので探索キャンセル、レシーバーの削除など
      :
  }
};
```

14-4-5 ペアリング済みデバイスの取得（BluetoothAdapter）

Bluetoothで接続しデータの送受信を行うためには、接続する2台の機器で、あらかじめ接続設定が完了している状態でなければいけません。この接続設定のことを「ペアリング」と呼びます。

Androidではペアリングしていない別のBluetooth機器と初めて接続する際、自動的に双方の機器にペアリング要求ダイアログが表示されます（**図2**）。

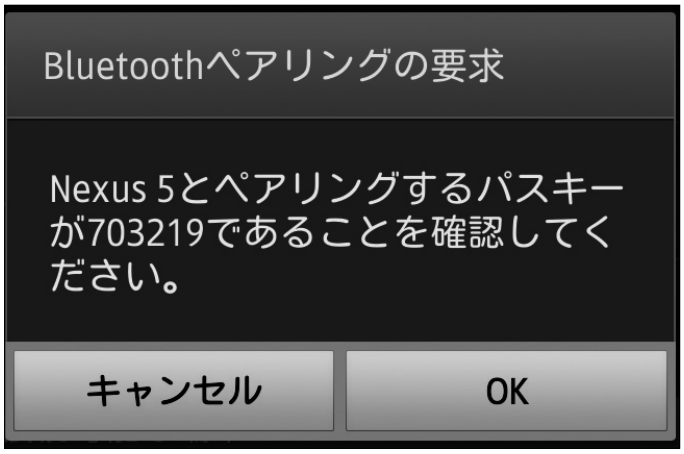


図2: Bluetoothペアリング要求ダイアログ

ダイアログで表示している接続先のPINコードを入力することでペアリングが完了します。この処理はすべてAndroidのシステムで行われるので、アプリケーション開発時に実装する必要はありません。

アプリケーションの中でペアリング済みのデバイス情報一覧を取得するには、BluetoothAdapterクラスのgetBondedDevicesメソッドを使用します。次のサンプルプログラムでは、取得したペアリング済みデバイスの検索結果（Setクラスのオブジェクト）からデバイス名が"SampleDevice"のデバイス情報を取得しています。

ペアリング済みデバイス一覧を取得

```
BluetoothAdapter btAdapter = BluetoothAdapter.getDefaultAdapter();
Set<BluetoothDevice> bondedDevices = btAdapter.getBondedDevices();
BluetoothDevice device;
for (BluetoothDevice bluetoothDevice : bondedDevices) {
  if (bluetoothDevice.getName().equals("SampleDevice")) {
    device = bluetoothDevice;
    break;
  }
}
```

14-4-6 検索されたデバイスに接続（クライアント端末として振る舞う）

検索されたデバイスにクライアントとして、SPPプロファイルで接続をするには、BluetoothSocketクラスのオブジェクトを使用します。次にサンプルプログラムを示します。このサンプルでは、ペアリング済みデバイスの検索結果の中から選択したデバイスと、SPPで接続しています。

deviceにクライアントとしてSPP接続する

```
BluetoothSocket socket =
  device.createRfcommSocketToServiceRecord(
      UUID.fromString("00001101-0000-1000-8000-00805F9B34FB")); //SPPのUUIDを指定
socket.connect();// 接続実行

InputStream in= socket.getInputStream();  // データ受信
// InputStreamのオブジェクトから受信データを取り出す処理など

OutputStream out= socket.getOutputStream();  // データ送信
// OutputStreamのオブジェクトに送信データを設定、write処理など
```

BluetoothSocketクラスのconnectメソッドで接続処理が実行されます。正常に接続されると、BluetoothSocketクラスのオブジェクトからInputStreamとOutputStreamを取得できるので、それらを使ってデータの送受信を行います。

Bluetoothでは、「接続する機器同士が同じサービス（プロファイル）をSDP対象のサービス一覧データベースに保有していること」が接続の条件となります。

TCP/IPのソケット通信では接続先のIPアドレスとポート番号を指定して接続しますが、Bluetooth通信では接続先のMACアドレスとUUIDを指定して接続します。

サンプルプログラム中のcreateRfcommSocketToServiceRecordメソッドは、RFCOMMチャンネルで接続を可能にするためのソケットオブジェクトを作成します。引数には特定のサービス（プロファイル）のUUIDを指定します。今回はSPPで接続するので、UUIDは "00001101-0000-1000-8000-00805F9B34FB" を用います。

SDP（Service Discovery Protocol）とは相手の機器がどのようなサービスをサポートしているのかを検索するのに利用されるプロトコルです。SDPで使用されるUUIDの規定値はこちらのリンクを参照してください。https://www.bluetooth.org/ja-jp/specification/assigned-numbers/service-discovery

14-4-7 自端末を別のBluetooth機器から発見可能にする

別のBluetooth機器からのデバイス検索に対して自端末を発見可能な状態にするには、「ACTION_REQUEST_DISCOVERABLE」を指定したインテントをstartAcitivityForResultメソッドで発行します。するとBluetooth許可リクエストダイアログが表示されるので、これをユーザーが許可することにより一定時間応答できる状態になります（**図3**）。

図3: Bluetooth許可リクエストダイアログ（発見可能）

発見可能な状態でいる時間をインテントに設定することもできます。その場合、インテントの付加情報としてEXTRA_DISCOVERABLE_DURATIONに時間（秒）を設定します。

サンプルプログラムを次に示します。

デバイス検索への応答

```
Intent intent = new Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
intent.putExtra(BluetoothAdapter.EXTRA_DISCOVERABLE_DURATION, 300);
startActivityForResult(intent,
  REQUEST_BT_DISCOVERABLE); // REQUEST_BT_DISCOVERABLEは"1"などの定数
```

図3のようなBluetooth許可リクエストダイアログが表示されます。ユーザーが許可することで、自端末は一定時間（サンプルコードの場合は300秒間）、別のBluetooth機器からのデバイス検索に対して発見可能な状態になります。

ダイアログでユーザーが許可をすると、onActivityResultメソッドにコールバックがきます。引数resultCodeには設定した発見可能な時間の値（秒）が入っています。

自端末がBluetooth機器として発見可能な状態かどうかの変化が起きた場合、BroadcastReceiverで通知を受け取ることができます。それを実現するためにはあらかじめ「ACTION_SCAN_MODE_CHANGED」インテントをBroadcastReceiverとして登録します。状態が変化した場合、BroadcastReceiverで受け取ったインテントには、古い状態「EXTRA_PREVIOUS_SCAN_MODE」と新

しい状態「EXTRA_SCAN_MODE」に、それぞれ次の**表3**で示したいずれかの値が入っています。

値	説明
SCAN_MODE_CONNECTABLE_DISCOVERABLE	発見可能な状態でかつ、接続も可能な状態
SCAN_MODE_CONNECTABLE	発見可能な状態ではないが、接続可能な状態
SCAN_MODE_NONE	発見可能な状態でもなく、接続もできない状態

表3: Bluetooth機器の状態

14-4-8 別のBluetooth機器からの接続要求を受ける(サーバー端末として振る舞う)

サーバーとして別のBluetooth機器からの接続要求を受け付けるには、BluetoothServerSocketクラスを使用します。

次にサンプルプログラムを示します。

リモートデバイスからの接続要求の受付

```
BluetoothAdapter btAdapter = BluetoothAdapter.getDefaultAdapter();
BluetoothServerSocket serverSocket =
  btAdapter.listenUsingRfcommWithServiceRecord("SampleServerConn",
  UUID.fromString("00001101-0000-1000-8000-00805F9B34FB"));
BluetoothSocket socket = serverSocket.accept();
if (socket != null) {    // クライアントからの接続要求を受け付け接続が完了した状態
  // データの送受信処理などを行う
    :
  serverSocket.close()
}
```

はじめにBluetoothAdapterクラスのlistenUsingRfcommWithServiceRecordメソッドを使用して、BluetoothServerSocketクラスのオブジェクトを生成します。接続を受け付ける条件(プロファイルのUUID)を第2引数に指定します。サンプルコードは、RFCOMMチャンネルでSPPプロファイルの接続受け付けを可能とする実装になっています。

前述のサンプルプログラムで生成したBluetoothServerSocketクラスのオブジェクトのacceptメソッドで、接続要求の受け付けを行います。同じUUIDが指定された接続要求がきた場合、acceptメソッドの戻り値としてBluetoothSocketオブジェクトが得られます。BluetoothSocketオブジェクトからInputStreamとOutputStreamを取得できるので、それらを使ってデータの送受信を行います。

クライアントとなる端末から接続要求がこない限り、acceptメソッドの応答(BluetoothSocketオブジェクト)は得られません。acceptメソッドで受け付け待ち状態のときは処理が一定時間中断してしまうので、接続処理や継続的なデータの送受信の処理はUIスレッドとは別のスレッドで処理するようにします。

接続要求の受け付け処理、およびデータの送受信が完了したら、BluetoothSocketオブジェクトはcloseメソッドで破棄します。

14-4-9 プロファイルのサポート

Bluetoothにはさまざまなプロファイルが存在します。これらすべてのプロファイルをAndroid SDKに含まれるAPIから使用（Bluetooth機器からのデータの取得やBluetooth機器のコントロール）できるわけではありません。AndroidではAndroid 3.0（HonneyComb:API Level 11）以降では、「HSP」、「HFP」と「A2DP」対応のBluetooth機器からの情報、Android 4.0（IceCreamSandwich:API Level 14）以降では「HDP」対応のBluetooth機器から情報を取得できるようになりました。

各プロファイルの概要を説明します。

・HSP（Headset Profile）

HSPは、携帯電話といっしょに使うBluetoothヘッドセットの接続機能を提供します。Androidでは、android.bluetooth.BluetoothHeadsetクラスが提供されています。これはAndroidの端末の内部で動作しているBluetooth Headset Serviceにプロセス間通信（IPC）で接続し、実際に制御するためのプロキシとなります。

・HFP（Hands-Free Profile）

自動車の車載器などのようなハンズフリーデバイスで、通信（発呼、着呼）に携帯電話を使用するための機能を提供します。HFPのバージョン1.5で提供される音質は非常に悪く、バージョン1.6で高音質対応されました。Androidでは4.0以降でバージョン1.6に対応しています。このプロファイルのAndroidにおけるAPIでのサポートは、HSPと同様にandroid.bluetooth.BluetoothHeadsetクラスで提供されています。

・A2DP（Advanced Audio Distribution Profile）

A2DPはBluetooth機器間で高品質のオーディオストリームを流す仕組みを提供します。Androidはandroid.bluetooth.BluetoothA2dpクラスが提供されています。これはAndroidの端末の内部で動作しているBluetooth A2DP Serviceにプロセス間通信（IPC）で接続し、実際に制御するためのプロキシとなります。

・HDP（Health Device Profile）

HDPはBluetooth搭載の健康管理機器間を接続、データ通信を提供します。Androidではandroid.bluetooth.BluetoothHealthクラスが提供されています。

(1)BluetoothProfileインターフェースのServiceListener

これらのプロファイルを持つBluetooth機器に接続して、Androidアプリケーションからそれをサポートするためには、android.bluetooth.BluetoothProfileインターフェースを用います。下記に示すように、BluetoothProfileインターフェースが持つServiceListenerを生成し、BluetoothAdapterに登録することで指定のプロファイルを持つBluetooth機器への接続時のイベント、および切断時のイベントメソッドを使うことができます。

ServiceListenerを使用した各プロファイルの、プロキシオブジェクトの取得例を次に示します。

BluetoothProfile.ServiceListenerの実装

```
private BluetoothHeadset mBluetoothHeadset; // HSPのオブジェクト
private BluetoothA2dp mBluetoothA2dp; // A2DPのオブジェクト
private BluetoothHealth mBluetoothHealth; // HDPのオブジェクト

private BluetoothProfile.ServiceListener mProfileListener =
    new BluetoothProfile.ServiceListener() {

  @Override
  public void onServiceConnected(int profile, BluetoothProfile proxy) {
    // プロキシオブジェクトがBluetoothAdapterに指定したサービスに接続されたときに呼ばれる
    // 接続された機器のプロファイル種別とBluetoothProfileのオブジェクトを受け取り
    // プロファイル専用の型のオブジェクトに保持しておく
    if (profile == BluetoothProfile.HEADSET) {
      mBluetoothHeadset = (BluetoothHeadset) proxy;
    } else if (profile == BluetoothProfile.A2DP) {
      mBluetoothA2dp = (BluetoothA2dp) proxy;
    } else if (profile == BluetoothProfile.HEALTH) {
      mBluetoothHealth= (BluetoothHealth) proxy;
    }

    // 接続されているデバイスをBluetoothDeviceオブジェクトのリストで取得する
    List<BluetoothDevice> devices = proxy.getConnectedDevices();

    // 1件ずつBluetoothDeviceオブジェクトを取り出し、
    // 機器名やMACアドレスなどの必要な情報の処理を行うなど…

  }

  @Override
  public void onServiceDisconnected(int profile) {
    // プロキシオブジェクトがBluetoothAdapterに指定したサービスから切断されたときに呼ばれる

    // 必要な後処理など…

    // 最後に切断されるプロファイルのオブジェクトの参照を切り離す処理を行う
    if (profile == BluetoothProfile.HEADSET) {
      mBluetoothHeadset = null;
    } else if (profile == BluetoothProfile.A2DP) {
      mBluetoothA2dp = null;
    } else if (profile == BluetoothProfile.HEALTH) {
```

```
        mBluetoothHealth = null;
      }
    }
  };

    // BluetoothAdapterにServiceListenerで取得するプロファイルをプロキシオブジェクトとして登録する
    BluetoothAdapter mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
    mBluetoothAdapter.getProfileProxy(this,
      mProfileListener, BluetoothProfile.HEADSET); //HSPを指定
    mBluetoothAdapter.getProfileProxy(this,
      mProfileListener, BluetoothProfile.A2DP); //A2DPを指定
    mBluetoothAdapter.getProfileProxy(this,
      mProfileListener, BluetoothProfile.HEALTH); // HDPを指定
```

(2)プロファイル固有のオブジェクト

次に、リスナーで受け取ったandroid.bluetoothパッケージのプロファイル固有クラスのオブジェクト使用方法について解説します。

BluetoothHeadset

BluetoothHeadsetクラスでは、以下のアクションインテントが定義されています。ブロードキャストレシーバでIntentを取得することで、ヘッドセットの接続状態変化、音声状態変化、ベンダー固有のイベント通知をリアルタイムに取得可能です(**表4**)。

値	説明
ACTION_AUDIO_STATE_CHANGED	A2DPの音声状態変化
ACTION_CONNECTION_STATE_CHANGED HSP	HSPの接続状態の変化
ACTION_VENDOR_SPECIFIC_HEADSET_EVENT	ヘッドセットのベンダー固有のイベント通知

表4: BluetoothHeadsetのアクションインテント

BluetoothHeadsetクラスが持つメソッドで、音声認識やベンダ固有の機器操作用のコマンド送信を行うことが可能です。

音声認識の開始と終了

- **boolean startVoiceRecognition(BluetoothDevice device)**
- **boolean stopVoiceRecognition(BluetoothDevice device)**

ベンダ固有の機器操作用のコマンド送信

- **boolean sendVendorSpecificResultCode(BluetoothDevice device, String command, String arg)**

BluetoothA2dp

BluetoothA2dpクラスでは次のアクションインテントが定義されています（**表5**）。ブロードキャストレシーバでIntentを取得することで、接続状態変化、再生状態変化をリアルタイムに取得可能です。

値	説明
ACTION_CONNECTION_STATE_CHANGED	A2DPの接続状態変化
ACTION_PLAYING_STATE_CHANGED	A2DPプロファイルの再生状態変化

表5: BluetoothA2dpのアクションインテント

ブロードキャストレシーバーで状態変化を取得する例を次に示します。

アクションインテントの登録とブロードキャストレシーバーの実装

```
    IntentFilter filter = new IntentFilter();
    filter.addAction(BluetoothA2dp.ACTION_CONNECTION_STATE_CHANGED);
    filter.addAction(BluetoothA2dp.ACTION_PLAYING_STATE_CHANGED);
    registerReceiver(mReceiver, filter);

  private BroadcastReceiver mReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
      String action = intent.getAction();
      //現在の状態
      int status = intent.getIntExtra(BluetoothProfile.EXTRA_STATE, -1);
      //直前の状態
      int prevStatus =
         intent.getIntExtra(BluetoothProfile.EXTRA_PREVIOUS_STATE, -1);
      // 接続しているBluetooth機器の情報
      BluetoothDevice device =
         intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);

      // 接続状態変更のインテントを受信した場合
      if (action.equals(BluetoothA2dp.ACTION_CONNECTION_STATE_CHANGED)) {
        switch (status){
        case BluetoothProfile.STATE_CONNECTED:  //接続済の状態
          :
        case BluetoothProfile.STATE_DISCONNECTED:  //切断している状態
          :
        case BluetoothProfile.STATE_DISCONNECTING:  //切断処理中の状態
          :
  };
```

BluetoothHealth

Bluetoothヘルス機器と通信をするためにはBluetoothHealthクラスを使用します。ヘルス機器の状態をアプリケーション側で受け取るには、HSPやA2DPで扱えているようなアクションインテントは用意されていません。android.bluetooth.Blue

toothHealthCallbackクラス（Android 4.0以降:API Level 14）を生成して状態変化時にコールバックメソッドが呼ばれるようにします。作成したBluetoothHealthCallbackオブジェクトは、BluetoothHealthクラスのregisterSinkAppConfigurationメソッドで登録します。

Bluetoothヘルス機器には2つのデータタイプが定義されています。1つは、SOURCEタイプ（BluetoothHealth.SOURCE_ROLE）でデータを送信する側（ヘルス機器）を指します。もう1つは、SINKタイプ（BluetoothHealth.SINK_ROLE）でデータを受信する側です。接続した機器のデータタイプは、BluetoothHealthAppConfigurationオブジェクトのgetRoleメソッドで取得できます。SINKタイプは複数のSOURCEタイプの機器から情報を受け取ることができます。

Androidアプリケーションを作成する場合、接続したヘルス機器から情報を取得して画面に表示させることが一般的ですので、上述したregisterSinkAppConfigurationメソッドの引数のデータタイプはBluetoothHealth.SINK_ROLEにします。

BluetoothHealthCallbackが持つコールバックメソッドは、onHealthAppConfigurationStatusChange、onHealthChannelStateChangeがあります。

onHealthAppConfigurationStatusChangeメソッドは、ヘルス機器の登録状態が変わったときにコールバックされるメソッドで、第2引数に次のいずれかの値が入ってきます（**表6**）。

値	説明
APP_CONFIG_REGISTRATION_SUCCESS	登録成功
APP_CONFIG_REGISTRATION_FAILURE	登録失敗
APP_CONFIG_UNREGISTRATION_SUCCESS	登録解除成功
APP_CONFIG_UNREGISTRATION_FAILURE	登録解除失敗

表6:onHealthAppConfigurationStatusChangeメソッド

onHealthChannelStateChangeメソッドは、チャンネルの状態が変更されたときにコールバックされるメソッドで第3引数（変更前の状態）と第4引数（変更後の状態）に次のいずれかの値が渡されます（**表7**）。

値	説明
STATE_CHANNEL_CONNECTING	接続処理中
STATE_CHANNEL_CONNECTED	接続完了
STATE_CHANNEL_DISCONNECTING	切断処理中
STATE_CHANNEL_DISCONNECTED	切断完了

表7:onHealthChannelStateChangeメソッド

コールバックを登録したら、Bluetoothヘルス機器への接続を行います。接続にはBluetoothHealthクラスのconnectChannelToSourceメソッドを使用します。

アプリケーションからBluetoothヘルス機器への接続と状態変化通知を受け取るまでのコード例を次に示します。

Bluetoothヘルス機器への接続と状態変化通知の受信のサンプルコード

```
private BluetoothHealthAppConfiguration mHealthConfig;
private int mChannelId;
private BluetoothDevice mBluetoothDevice;

class MyBluetoothHealthCallback extends BluetoothHealthCallback {

  @Override
  public void onHealthAppConfigurationStatusChange(
      BluetoothHealthAppConfiguration config, int status) {
    super.onHealthAppConfigurationStatusChange(config, status);

    // 登録が成功したら通信に必要なアプリケーションコンフィギュレーションを保持
    mHealthConfig = config;
      :
  }

  @Override
  public void onHealthChannelStateChange(
      BluetoothHealthAppConfiguration config,
      BluetoothDevice device, int prevState, int newState,
      ParcelFileDescriptor fd, int channelId) {

    super.onHealthChannelStateChange(config,
      device, prevState, newState, fd, channelId);

    if (newState == BluetoothHealth.STATE_CHANNEL_CONNECTED){
      // チャンネルに関連づいたIDを保持しておく
      // このIDは切断するときに使用する
      mChannelId = channelId;
      // BluetoothDeviceオブジェクトを保持しておく
      // 接続時に使用する
      mBluetoothDevice = device;
        :
        :
    }
  }
};

private void register() {
  MyBluetoothHealthCallback mCallback = new MyBluetoothHealthCallback();
  // ヘルス受信機器として機能するアプリケーションコンフィギュレーションを登録する
  mBluetoothHealth.registerSinkAppConfiguration("HEALTH_DEVICES",
    BluetoothHealth.SINK_ROLE, mCallback);
}

private void unregister() {
  // アプリケーションコンフィギュレーションを登録解除する
  mBluetoothHealth.unregisterAppConfiguration(mHealthConfig);
}
```

```
private void connect() {
  // Bluetoothヘルス機器に接続する
  mBluetoothHealth.connectChannelToSource(mBluetoothDevice, mHealthConfig);
}

private void disconnect() {
  // Bluetoothヘルス機器から切断する
  mBluetoothHealth.disconnectChannel(mBluetoothDevice,
    mHealthConfig, mChannelId);
}
```

実際にヘルス機器から取得したデータは、onHealthChannelStateChangeメソッドの第5引数に入ってくるファイルディスクリプタを使って読み出します。ただし、データの解析には機器ごとに異なる解析処理が必要となります。

心拍計ならIEEE 11073-10407、歩数計ならIEEE 11073-10441というように、IEEE 11073-104xxの規格に沿ったデータ解析処理の実装がヘルス機器種別ごとに必要となります。また接続する実機が必要です。ここで紹介したサンプルコードを実機にそのまま用いても動作の保証はなく、機器固有の設定が必要です。

14-4-10 Bluetooth Low Energy

本節の冒頭で説明したとおり、Bluetoothのバージョン4.0にはBLE（Bluetooth Low Enagy）という仕組みが採用されています。ここではBLEの概要を説明します。

BLEの最大の特徴は、超低消費電力であることです。BLEの通信機器はリチウム電池1つで1年程度動作可能と言われています。データ転送速度は1Mbps程度（ただし、一度に20byte前後の小さいサイズのデータしかやり取りできない）で、周波数帯も同じく2.4GHz帯です。同じ周波数帯の機器からの干渉を防ぐための仕組みも備わっています。

このような特徴を活かして、フィットネス系のセンサーやウェアラブルデバイス、位置情報サービスや店舗サービスに採用されつつあります。

クラシックBluetoothとBLEには、互換性がまったくないのですが、ハードウェアとして共存させることは可能で、Bluetoothチップセットの中には両モードを同時に利用できるものもあります。

Bluetooth4.0搭載の機器には、BLEやクラシックBluetoothの対応の状態を識別するために、下記のような名称が定められています（**表8**）。

名称	説明
Bluetooth SMART	BLEでのみ通信可能で、クラシックとの通信はできない
Bluetooth SMART READY	BLEとクラシックの両方の通信に対応

表8: Bluetooth4.0搭載の機器の種類

BLEではGATT(Generic Attribute)というプロファイルで通信を行います。Androidアプリケーションで BLE機器と通信する場合、次の条件が揃っていなければいけません。

・Android端末がハード的にBluetooth4.0に対応していること

・Android OSのバージョンがBLEに対応していること(Android4.3以降)

ここからは、BLE機器と通信するアプリケーションの開発で使用する、android.bluetoothパッケージAPIを紹介します。

(1)BluetoothManager

Android4.3 (API Level 18)より、BluetoothManagerクラスがサポートされ、下記のようにBluetoothAdapterのインスタンスの取得などができるようになりました。

BluetoothManager

```
BluetoothManager manager =
   (BluetoothManager) getSystemService(Context.BLUETOOTH_SERVICE);
mBluetoothAdapter = manager.getAdapter();
```

(2)LeScanCallback

GATTプロファイル対応のデバイスの探索結果を受け取るための、コールバックメソッドが定義されたクラスです。

(3)BluetoothGattCallback

指定したBLE機器と接続が完了したときに呼ばれるコールバックメソッドが定義されたクラスです。BluetoothGattクラスのインスタンスを受け取ります。

(4)BluetoothGatt

BLE機器とGATTプロファイルで接続するためのクラスです。GATTプロファイルは、1つのプロファイルに付き複数のサービス(使用用途や機器の種類によって異なる)を持っています。また1つのサービスにつき、データ種別ごとに複数のキャラクタリスティックという単位でデータが管理されています。実際にBLE機器とデータ

をやり取りする場合は、このキャラクタリスティックに対してデータをやり取りします。

BluetoothGattクラスでは、指定したUUIDを持つGATTサービスの検索、キャラクタリスティックの読み書き、ディスクリプタの読み書き、BLE機器からのNotificationの受信設定、GATTプロファイルでの通信の接続や切断などが実行できます。

(5)BluetoothGattService

GATTサービスのクラスです。このオブジェクトからUUIDで指定されたキャラクタリスティックが取得できます。

(6)BluetoothGattCharacteristic

GATTプロファイルのキャラクタリスティックのクラスです。

(7)BluetoothGattDescriptor

BluetoothGattCharacteristicクラスのオブジェクトからディスクリプタを取得できます。アプリケーションからの値の読み書きの対象となります。

演習問題

(1)BluetoothAdapterクラスを使って、BluetoothのON/OFF切り替え、外部デバイスの検索、ペアリング済デバイスの検索を行うアプリケーションを作成してみよう。

・ BluetoothのON/OFF切り替えは、ボタンを実装して行う(ボタンの種類は何でもよい)

・ 外部デバイスの検索結果、ペアリング済デバイスの検索結果は、それぞれリスト表示する

(2)BluetoothのSPPで接続して、テキストデータをやりとりすることができるアプリケーションを作成してみよう。

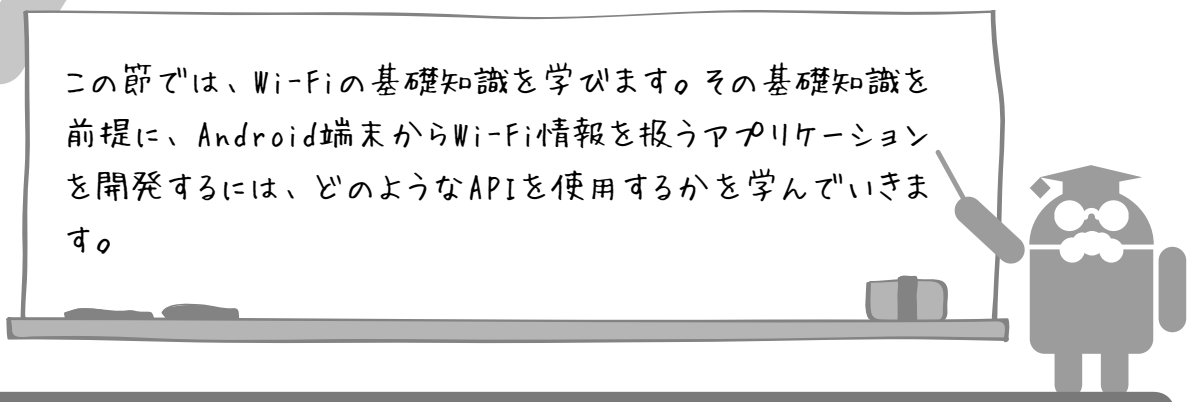
・ Androidの自端末が、サーバー、クライアント端末の両機能を使えるようにする。

・ 画面には送信するテキストをユーザーが入力できるようにEditTextなどを実装する

・ 画面には接続相手から受信したテキストデータを表示できるようにTextViewなどを実装する。

14-5 Wi-Fi通信

著：橋爪香織



この節で学ぶこと

- Wi-Fiの基礎知識（無線LANの規格、セキュリティ）
- AndroidのWi-Fi用APIの使用方法(設定制御、情報取得、アクセスポイントへの接続)
- Wi-Fi Directの概要とAndroidにおけるAPIの使用方法

この節で出てくるキーワード一覧

WifiManager
WifiConfiguration
WifiInfo
WifiP2pManager
Channel
PeerListListener
WifiP2pDeviceList
ConnectionInfoListener
WifiP2pInfo
WifiP2pConfig

14-5-1 Wi-Fiの基礎知識

無線LANの標準規格であるIEEE 802.11シリーズに準拠している無線LANそのもの、および無線LAN機器のことを「Wi-Fi」（Wireless Fidelity）と呼んでいます。無線LANとは、有線ケーブルを使わずに、ルーターが中継機などのアクセスポイントから発する電波を用いて、数m～数十m程度の範囲内でパソコンや電子機器間、もしくはインターネットへ接続して高速なデータ通信を行う通信技術です（**図4**）。

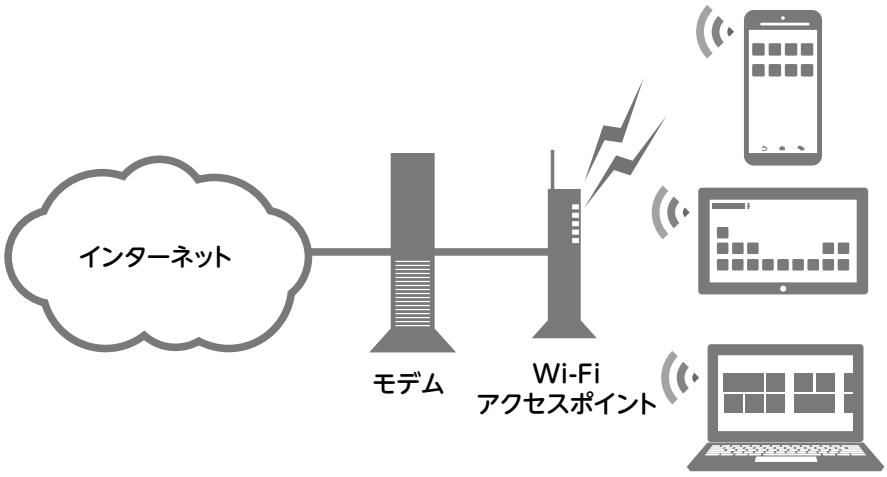


図4: Wi-Fiの概要

Wi-Fiをはじめとした無線LANでの通信では、通信データを傍受される危険性が有線による通信に比べて高いので、セキュリティを確保する必要があります。接続する際の認証方法で用いられる要素に「SSID」、認証方式として「WEP」「WPA」「WPA2」があります。

・SSID（Service Set ID）

Wi-FiのアクセスポイントのIDで、最大32文字までの英数字を任意に設定できます。

・WEP（Wired Equivalent Privacy）

固定されたPSK（Pre-Shared Key）を使用した暗号化を行います。鍵は13文字までの英数字か、26桁までの16進数を使用できますが、セキュリティ強度は非常に低いので現在では推奨されていません。

・WPA（WiFi Protected Access）

WEPの強化版となる暗号化方式です。

・WPA-PSK（TKIP）

暗号化に用いる秘密鍵を一定間隔で更新することができ、セキュリティ強度が高い暗号化技術です。ただし、更新頻度が大きくなるとネットワークに負荷がかかって

しまう問題もあります。

・WPA-PSK(AES)

解読が不可能とされており、非常にセキュリティ強度が高い暗号化技術です。CCMP(Counter Mode with Cipher Block Chaining Message Authentication Code Protocol)と呼ばれる技術が、認証とデータ暗号化に使用されています。現在、最も信頼できる暗号化技術とされており推奨される方式です。

・WPA2

WPAの後継規格です。AESが義務化されるかたちで標準化されました。WPA同様、TKIPやAESを使用する事が可能な暗号化方式です。

14-5-2 アプリケーションでWi-Fi機能を使う

Wi-Fi機能を搭載したAndroidでは無線LANを通してインターネットに接続することが可能です。また、Androidのアプリケーションから Wi-Fi機能を制御したり、情報を取得する事も可能です。Androidのアプリケーションで Wi-Fi機能に関する制御を行うにはandroid.net.wifiパッケージのAPIを使用します。

Wi-Fi機能のAPI をアプリケーションで使用する事によって、次のような機能を実現できます。

android.net.wifiパッケージについてのリファレンスは、こちらを参照して下さい。http://developer.android.com/intl/ja/reference/android/net/wifi/package-summary.html

・自端末のWi-Fi機能の有効/無効設定をする
・Wi-Fiネットワークの検索をする
・Wi-Fi機器との接続を確立する
・Wi-Fiネットワーク設定情報の取得
・IP Multicastを使用する

IP Multicastとは、ネットワーク内で、複数の相手を指定して同じデータを送信することです。

Wi-Fiの状態を取得するには、AndroidManifest.xmlの<uses-permission>に「android.permission.ACCESS_WIFI_STATE」のパーミッションの指定が必要です。Wi-Fiの設定を変更するには「android.permission.CHANGE_WIFI_STATE」のパーミッションが必要です。またIP Multicastを使用するには「android.permission.CHANGE_WIFI_MULTICAST_STATE」のパーミッションが必要です。

14-5-3 Wi-Fiの設定制御（WifiManager）

アプリケーションからWi-Fi機能の有効／無効を設定するサンプルプログラムを次に示します。

Wi-Fi設定制御

```
WifiManager wm = (WifiManager) getSystemService(Context.WIFI_SERVICE);
if (!wm.isWifiEnabled()) {
  wm.setWifiEnabled(true); // Wi-Fiを有効にする
}
```

Wi-Fiが有効かどうか、WifiManagerクラスのisEnableメソッドで確認し、無効ならsetWifiEnabledメソッドで有効にします。Wi-Fi機能の有効／無効状態の取得には、WifiManagerクラスのisEnabledメソッドを使用します。Wi-Fi機能を有効にするにはsetWifiEnabledメソッドを使います。

14-5-4 Wi-Fiネットワークの検索

アプリケーションから実行中のAndroid端末の近隣のWi-Fiネットワーク機器の探索をするサンプルプログラムについて説明します。Wi-Fi機器を探索するには、次のようにWifiManagerクラスのstartScanメソッドを使用します。

Wi-Fiネットワークの探索

```
WifiManager wm = (WifiManager) getSystemService(Context.WIFI_SERVICE);
wm.startScan(); // Wi-Fiネットワーク探索開始
```

発見したWi-Fi機器の情報を受けとるためのインテント「SCAN_RESULTS_AVAILABLE_ACTIOND」を受信できるように、BroadcastReceiverを作成し登録しておきます。次のようになります。

アクションインテントとBroadcastReceiverの登録

```
IntentFilter filter = new IntentFilter();
filter.addAction(WifiManager.SCAN_RESULTS_AVAILABLE_ACTION);
registerReceiver(mReceiver, filter);
```

上記のインテントを受信したとき、WifiManagerのgetScanResultsメソッドを使用する事で、発見したWi-Fi機器の情報（ScanResultオブジェクト）を格納したListが取得できます。

Wi-Fiネットワークの探索のアクションインテントの受信処理

```
BroadcastReceiver  mReceiver = new BroadcastReceiver() {
  public void onReceive(Context context, Intent intent) {
    String action = intent.getAction();
    if (WifiManager.SCAN_RESULTS_AVAILABLE_ACTION.equals(action)) {
      WifiManager wm = (WifiManager) getSystemService(Context.WIFI_SERVICE);
      List<ScanResult> list = wm.getScanResults();

      // 取得したScanResultのリストから 機器データの取得などを行う

    }
  }
};
```

14-5-5 Wi-Fi 機器への接続と認証方法

Wi-Fi機器の検索をして得られた機器のListから、特定のWi-Fi機器で接続するためのサンプルコードを以下に示します。接続に必要な設定（SSIDの値やWEP/WPAなどのキー）は、WifiConfigrationクラスのオブジェクトにセットします。接続する際の認証方式によって、設定内容が異なります。

なお、サンプルコード中の変数ssidはScanResultから取得したSSIDの値です。

認証方式がWEPの場合のWifiConfiguration

```
WifiConfiguration config = new WifiConfiguration();
//SSID
config.SSID = "\"" + ssid + "\"";
//このコンフィギュレーションで管理する認証キー
config.allowedKeyManagement.set(WifiConfiguration.KeyMgmt.NONE);
//IEEE 802.11認証アルゴリズム
config.allowedAuthAlgorithms.set(WifiConfiguration.AuthAlgorithm.SHARED);
//認証されたグループの暗号
config.allowedGroupCiphers.set(WifiConfiguration.GroupCipher.WEP40);
config.allowedGroupCiphers.set(WifiConfiguration.GroupCipher.WEP104);
//WEP用キー
config.wepKeys[0] = "\"password\"";
config.wepTxKeyIndex = 0;
```

WEP認証で接続する際には、WifiConfigrationクラスのallowedKeyManagementに「WifiConfiguration.KeyMgmt.NONE」を設定します。他の設定値に関してもWEP認証に必要な値を設定します。

認証方式がWPA/WPA2-PSKの場合のWifiConfiguration

```
WifiConfiguration config = new WifiConfiguration();
//SSID
config.SSID = "\"" + ssid + "\"";
//このコンフィギュレーションで管理する認証キー
config.allowedKeyManagement.set(WifiConfiguration.KeyMgmt.WPA_PSK);
//IEEE 802.11認証アルゴリズム
config.allowedAuthAlgorithms.set(WifiConfiguration.AuthAlgorithm.OPEN);
//セキュリティプロトコル
config.allowedProtocols.set(WifiConfiguration.Protocol.WPA);
config.allowedProtocols.set(WifiConfiguration.Protocol.RSN);//WPA2
//認証されたグループの暗号
config.allowedGroupCiphers.set(WifiConfiguration.GroupCipher.CCMP);
config.allowedGroupCiphers.set(WifiConfiguration.GroupCipher.TKIP);
//WPA認証用ペア暗号
config.allowedPairwiseCiphers.set(WifiConfiguration.PairwiseCipher.CCMP);
config.allowedPairwiseCiphers.set(WifiConfiguration.PairwiseCipher.TKIP);
//WPA用共通キー
config.preSharedKey = "\"password\"";
```

WPA/WPA2認証で接続する際には、WifiConfigrationクラスのallowedKeyManagementに「WifiConfiguration.KeyMgmt.WPA_PSK」を設定します。他にもWEP認証と異なる値をコンフィギュレーションに設定します。

生成したWifiConfigurationで接続する

```
//設定済のネットワークに新しいネットワーク情報を追加する
if( manager.addNetwork(config) == -1 ){
  // 失敗した場合-1となる
  return false;
};
wifiManager.saveConfiguration(); //設定されたネットワーク情報をこの端末に保存する
// 最新のネットワークへ強制的に接続する
wifiManager.updateNetwork(config);
manager.enableNetwork(config.networkId, true);
```

Wi-Fiの接続はWifiManagerクラスのaddNetworkメソッドを使用します。戻り値としてネットワークIDが戻ってくるので、この値をenableNetworkメソッドの引数に指定して接続を有効にします。saveConfigurationメソッドで接続したWi-Fi機器の設定情報を、システムに保存しておきます。

次に、接続中のWi-Fi機器との接続を切断する場合のサンプルコードを示します。WifiManagerクラスのdisconnectメソッドを使用することで切断できます。

接続中のWi-Fi機器と切断

```
WifiManager wm = (WifiManager) getSystemService(Context.WIFI_SERVICE);
wm.disconnect();
```

14-5-6 Wi-Fi ネットワーク設定情報の取得

接続が可能になったWi-Fi機器の接続設定情報は、WifiManagerクラスのsaveConfigurationメソッドによってシステムに保存されています。保存済みの設定情報を取得するには、WifiManagerクラスのgetConfiguredNetworksメソッドを使用します。戻り値として、WifiConfigurationクラスのオブジェクトが入ったListを取得することができます。

次に示すサンプルコードは、取得した接続設定情報をログに出力しています。

Wi-Fiネットワークの設定情報を取得する

```
WifiManager wm = (WifiManager) getSystemService(WIFI_SERVICE);
List<WifiConfiguration> cfgList = wm.getConfiguredNetworks();
for (int i = 0; i < cfgList.size(); i++) {
    Log.v("WifiConfiguration", "NetworkID = " + cfgList.get(i).networkId);
    Log.v("WifiConfiguration", "SSID = " + config_cfgListlist.get(i).SSID);
    Log.v("…….);
    // 必要な情報の取り出し処理など
    :
    :
}
```

14-5-7 Wi-Fi 接続情報の取得

Wi-Fiの接続状態を取得するためのサンプルコードを次に示します。

接続状態は、WifiManagerクラスのgetConnectionInfoメソッドで取得することができます。取得結果は戻り値のWifiInfoクラスのオブジェクトに含まれています。

Wi-Fiの接続状態を取得する

```
WifiManager wm = (WifiManager) getSystemService(WIFI_SERVICE);
WifiInfo info = wm.getConnectionInfo();
Log.v("WifiInfo", "SSID = " + info.getSSID());
Log.v("WifiInfo", "BSSID = " + info.getBSSID());
Log.v("WifiInfo", "IP Address = " + info.getIpAddress());
Log.v("WifiInfo", "Mac Address = " + info.getMacAddress());
Log.v("WifiInfo", "Network ID = " + info.getNetworkId());
Log.v("WifiInfo", "Link Speed = " + info.getLinkSpeed());
```

このWifiInfoクラスに含まれる接続情報の一覧を次に示します（**表9**）。

値	説明
SSID	無線識別用のID（Service Set Identifier）
IPアドレス	機器のIPアドレス（論理アドレス）
MACアドレス	機器のMACアドレス（物理アドレス）
RSSI	受信信号強度
BSSID	BSS (Basic Service Set)のID（接続しているWi-Fi機器のMACアドレスと同じ）
ネットワークID	接続時にシステムに与えられたID
接続状態	Wi-Fiの接続状態

表9:取得可能なWi-Fi接続情報

IPアドレスはint型で返ってくるので、次のようにオクテット表記にビット演算で直す必要があります。

int型のIPアドレスをオクテット表記に変換

```
int ip_addr_i = w_info.getIpAddress();
String ip_addr = ((ip_addr_i >> 0) & 0xFF) + "."
                              + ((ip_addr_i >> 8) & 0xFF) + "."
                              + ((ip_addr_i >> 16) & 0xFF) + "."
                              + ((ip_addr_i >> 24) & 0xFF);
Log.i("Sample", "IP Address:"+ip_addr);
```

Wi-Fiの接続状態については次の種類があります（**表10**）。

値	説明
WIFI_STATE_DISABLING	切断処理中
WIFI_STATE_DISABLED	切断
WIFI_STATE_ENABLING	接続処理中
WIFI_STATE_ENABLED	接続済み
WIFI_STATE_UNKNOWN	不明な状態

表10: Wi-Fiの接続状態の種類

14-5-8 Wi-Fi Direct とは

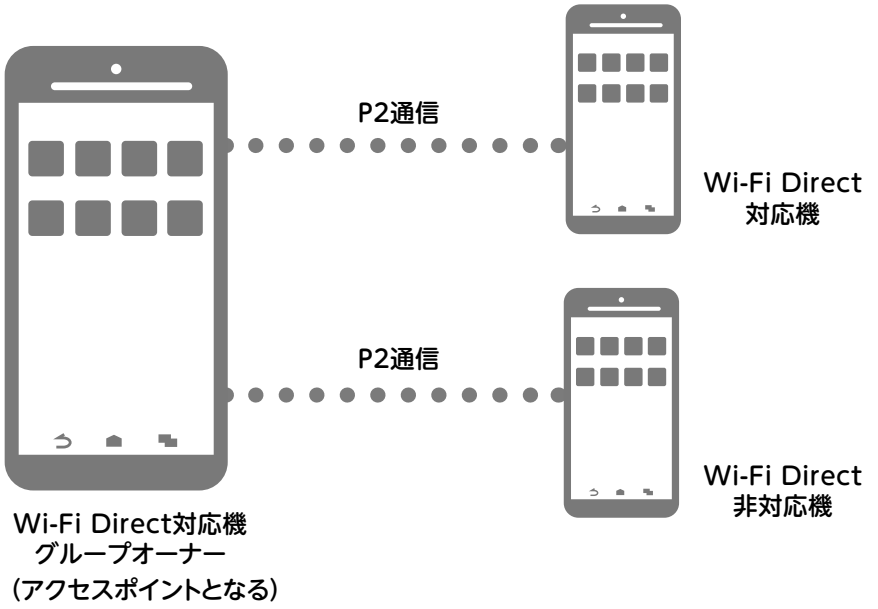


図5: Wi-Fi Directの概要

「Wi-Fi Direct」とは、無線LANを利用した通信方式のひとつで、アクセスポイント(無線LANのルーター機器)がなくても機器同士が1対1で通信できる仕組みです(**図5**)。

Wi-Fi Directに対応している機器は、もともと無線LANのアクセスポイント機能を内蔵しているので、自身がアクセスポイントになることにより通信を実現します。この時、アクセスポイントの役割を果たす機器をグループオーナーと呼び、他の機器からは通常のアクセスポイントと同じように見えます。

- **Wi-Fi Direct対応の機器 <=> Wi-Fi Direct対応の機器**
- **Wi-Fi Direct対応の機器 <=> Wi-Fi Directに対応していない機器**

他の特徴として以下があげられます。

- **「1対多」の通信も行える。但し、同時に通信できるのは「1対1」となる。**
- **無線の接続設定には簡易接続方式「WPS」を使用する。**
- **無線機器のSSID(固有ID)や暗号化キーは自動的に決められる。**

Androidは4.0以降(API Level 14以降)でWi-Fi Directに対応しています。接続方法は、アクションインテントを用いて接続可能な機器を探し、自動的に接続します。本項の前半で説明したWi-Fiネットワーク検索と同じように、Wi-Fi Directで接続できる機器を探すことができます。自ら接続可能な機器を探索する以外に、他の機器からの接続待ちの状態にすることも可能です。他の機器から接続要

求(invite)がきた場合、それに応えて接続します。

Wi-Fi Directは、あくまでも無線による接続機能のみの提供なので、接続後に何かしらの動作をさせたい場合には別途アプリケーションが必要になります。

それでは、次から、Android SDKに付随しているWiFi Direct Demoのソースコードをもとに、関連するAPIの具体的な使用方法について、みていきましょう。

公式解説はhttp://www.androidside.com/docs/resources/samples/WiFiDirectDemo/index.htmlにあります。

14-5-9 WifiP2pManager クラス

AndroidのWi-Fi Direct機能は、WifiP2pManagerクラスによって提供されています。WifiP2pManagerが持つ4つのアクションインテントは、Wi-Fi Directに関する状態に変更があったときにブロードキャストされるので、これらをレシーブ(WiFiDirectBroadcastReceiverを実装、登録)して必要な処理を行っていきます。

これら4つのアクションインテントと、受け取ったときの処理例について、順を追って説明します。

・WIFI_P2P_STATE_CHANGED_ACTION

Wi-Fi Directの有効/無効状態が通知される。機能制限やユーザーへの通知に利用。Wi-Fi Directが無効であれば設定アプリ(Setting)で有効にすることを促す処理(Toastを表示するなど)を加える。

・WIFI_P2P_PEERS_CHANGED_ACTION

デバイス情報の変更通知(Peersと呼ばれる通信可能なデバイスの発見・ロストなど)。WifiP2pManagerクラスのrequestPeersメソッドをつかってデバイス一覧WifiP2pDeviceListを取得。個々のデバイス情報はWifiP2pDeviceオブジェクトに入っている。

・WIFI_P2P_CONNECTION_CHANGED_ACTION

IPアドレスなどコネクション情報。通信状態の変更通知。WifiP2pManagerクラスのrequestConnectionInfoメソッドを使って接続状態WiFiP2pInfoを取得する。

・WIFI_P2P_THIS_DEVICE_CHANGED_ACTION

自分自身のデバイス状態の変更通知(相手デバイスではないことに注意)。WifiP2pDeviceを取り出して、接続状態にあわせた処理を行う。

また、システムからWifiP2pManagerを受け取ったら、WifiP2pManagerの持つinitializeメソッドで初期化を行い、WifiP2pManager.Channelクラスのオブジェクトを受け取ります。

ここまで（WifiP2pManagerのアクションインテントの登録、WifiP2pManagerの取得と初期化（初期化については後述します）、WiFiDirectBroadcastReceiverの登録）のコードサンプルを下記に示します。

WifiP2pManager関連

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    // add necessary intent values to be matched.

    intentFilter.addAction(WifiP2pManager.WIFI_P2P_STATE_CHANGED_ACTION);
    intentFilter.addAction(WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION);
    intentFilter.addAction(WifiP2pManager.WIFI_P2P_CONNECTION_CHANGED_ACTION);
    intentFilter.addAction(WifiP2pManager.WIFI_P2P_THIS_DEVICE_CHANGED_ACTION);

    manager = (WifiP2pManager) getSystemService(Context.WIFI_P2P_SERVICE);
    channel = manager.initialize(this, getMainLooper(), null);
}

/** register the BroadcastReceiver with the intent values to be matched */
@Override
public void onResume() {
    super.onResume();
    receiver = new WiFiDirectBroadcastReceiver(manager, channel, this); ❶
    registerReceiver(receiver, intentFilter);
}

@Override
public void onPause() {
    super.onPause();
    unregisterReceiver(receiver);
}
```

サンプルコード中にでてきた❶WiFiDirectBroadcastReceiverは、BroadcastReceiverを継承したサブクラスです。

それぞれのアクションインテントを受信したときに行う処理の例を、下記のコード内のコメントに記載します。

WiFiDirectBroadcastReceiverの例

```
public class WiFiDirectBroadcastReceiver extends BroadcastReceiver {

    private WifiP2pManager mManager;
    private Channel mChannel;
    private MyWiFiActivity mActivity;

    public WiFiDirectBroadcastReceiver(WifiP2pManager manager, Channel channel,
            MyWifiActivity activity) {
        super();
        this.mManager = manager;
        this.mChannel = channel;
        this.mActivity = activity;
    }

    @Override
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();

        if (WifiP2pManager.WIFI_P2P_STATE_CHANGED_ACTION.equals(action)) {
            // WiFi Directの有効/無効通知
            // Wifi Directが無効であれば設定アプリ(Setting)で有効にするようにToastなどで促す処理などを。
        } else if (WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION.equals(action)) {
            // WiFi Direct通信が出来るデバイス(Peers)の状態変化
            // 画面に表示するデバイス一覧の更新処理など。 ❶
            // 通信相手を切り替えるなどの処理を。
        } else if (WifiP2pManager.WIFI_P2P_CONNECTION_CHANGED_ACTION.equals(action)) {
            // WiFi Direct通信状態の変更通知
            // 相手デバイスとのコネクションが繋がれば接続情報(WifiP2pInfo)を取得して必要な処理を。 ❷
            // 切断された場合は終了処理を。
        } else if (WifiP2pManager.WIFI_P2P_THIS_DEVICE_CHANGED_ACTION.equals(action)) {
            // 自分自身のデバイス状態の変更通知
            // WifiP2pDeviceがインテントで渡されるので、その中に含むデバイス情報を取り出して表示する処理など。
        }
    }
}
```

❶PreeListenerとWifiP2pDeviceListを使う

WifiP2pManager.requestPeersメソッドでPeerListListenerとChannelを登録します。

そうすることでPeerListListenerで実装されたonPeersAvailableメソッドの引数でWifiP2pDeviceListが接続可能なデバイス一覧情報を通知することができます。

❷ConnectionInfoListenerとWifiP2pInfoを使う

接続情報はWifiP2pManager.requestConnectionInfoで登録したConnectionInfoListenerで取得することができます。

ConnectionInfoListenerではonConnectionInfoAvailableメソッドを実装する必要があり、引数に接続情報を含んでいるWifiP2pInfoクラスのオブジェクトが渡されます。

WifiP2pInfoクラス内には、アドレスのIP情報やP2P接続のオーナー（親側）、

WifiP2pInfoクラスのisGroupOwnerメンバなどが含まれています。

ここでWifiP2pManagerクラスが持つメソッドを（**表11**）にまとめます。これまでに出てきていないメソッドもありますが、それは後ほど説明します。

メソッド名	説明
`WifiP2pManager.Channel initialize (Context srcContext, Looper srcLooper, WifiP2pManager.ChannelListener listener)`	初期化を行う。第3引数は切断時のリスナー（WifiP2pManager.ChannelListener）だがnullでもよい。戻り値のChannelは接続チャンネルを識別するために利用する
`connect (WifiP2pManager.Channel c, WifiP2pConfig config, WifiP2pManager.ActionListener listener)`	接続の開始。接続設定情報は第2引数WifiP2pConfigのオブジェクトに含める
`removeGroup (WifiP2pManager.Channel c, WifiP2pManager.ActionListener listener)`	接続の切断。切断したいChannelを指定する
`cancelConnect (WifiP2pManager.Channel c, WifiP2pManager.ActionListener listener)`	接続処理中のキャンセル

表11: WifiP2pManagerクラスのメソッドのまとめ

14-5-10　Wi-Fi Direct での接続処理

(1)connectメソッド

接続処理もWifiP2pManagerクラスを用います。接続チャンネル情報を保持しておくために、必ず、WifiP2pManager.initializeメソッドを実行しましょう。次にWifiP2pManagerクラスのconnectメソッドで対象端末と接続します。connectメソッドの第一引数のchannelは、initializeメソッドで取得した値を渡します。第二引数のconfigはWifiP2pConfigのオブジェクトになります。次のように表します。

接続処理のコード

```
manager.connect(channel, config, new ActionListener() {

    public void onSuccess() {
      // 接続したら呼ばれる
      // 接続情報の詳細はここでは得られないのでBroadcastReceiverで受け取ったアクションインテントから得る
    }

    public void onFailure(int reason) { //エラーコード
    Toast.makeText(WiFiDirectActivity.this, "Connect failed. Retry.",
            Toast.LENGTH_SHORT).show();
    }
```

(1-1)WifiP2pConfigクラス

WifiP2pConfigクラスのオブジェクトには、接続設定情報を含めます。

P2Pで接続する相手先のデバイスのIPアドレスや、悪意のある端末からの接続から端末を保護するための設定などを、WifiP2pConfigクラスのメンバ変数にセットします。

WifiP2pConfigクラスのオブジェクト生成例

```
WifiP2pConfig config = new WifiP2pConfig();
config.deviceAddress = device.deviceAddress; // 接続デバイスのIPアドレス
config.wps.setup = WpsInfo.PBC; //wps(Wi-Fi Protected Setup)
//PBC(Push button configuration:)PINコード入力なしでワンタッチ接続する設定
```

(2)removeGroupメソッド

接続済みのWi-Fi Directのコネクション切断は、removeGroupメソッドを使います。次のようになります。

切断処理のコード

```
manager.removeGroup(channel, new ActionListener() {

    public void onFailure(int reasonCode) { // エラーコード
        // 切断失敗
    }

    public void onSuccess() {
        // 切断成功
    }
```

(3)cancelConnectメソッド

Wi-Fi Directの接続処理中のキャンセル処理はcancelConnectメソッドを使います。このメソッドは対象端末が接続可能(WifiP2pDevice.AVAILABLE)な状態、もしくは要求済み(ネゴシエーション中、WifiP2pDevice.INVITED)の場合に使用します。

キャンセル処理のコード

```
manager.cancelConnect(channel, new ActionListener() {

     public void onSuccess() {
        // キャンセル成功
     }

     public void onFailure(int reasonCode) { // エラーコード
        // キャンセル失敗
     }
```

演習問題

(1)WifiManagerクラスを使って、端末のWi-Fi機能のON/OFF切り替え、ネットワークの検索を行うアプリケーションを作成してみよう。

・Wi-Fi機能のON/OFF切り替えはボタンを実装して行う(ボタンの種類は何でもよい)

・ネットワークの検索結果はリスト表示する

(2)近隣のWi-Fi機器のアクセスポイントへWEP認証で接続してみましょう。

(3)接続したWi-Fi機器の情報を取得し、以下の情報を画面に表示してみましょう。

・SSID/IPアドレス／MACアドレス／RSSI／BSSID／ネットワークID／接続状態

(4)Wi-Fi Directで接続を可能にするアプリケーションを作成してみましょう