

第17章(17-4, 17-5)

ネットワーク

プログラミング(2)

2015-05-17 講師: 猪俣 充央

# 自己紹介

- ・ 株式会社 tech vein 代表 猪俣 充央
- ・ 主にiOSアプリ, Androidアプリを作る仕事をしています。

# 今日のタイムスケジュール

- ・ 13:00-14:20 【17-4章】Bluetooth 前半
- ・ 14:20-14:30 10分休憩
- ・ 14:30-15:40 【17-4章】Bluetooth 後半
- ・ 15:40-16:10 30分休憩
- ・ 16:10-17:30 【17-5章】Wi-Fi
- ・ 17:30-17:40 10分休憩
- ・ 17:50-19:00 ネットワークのまとめ

# はじめに

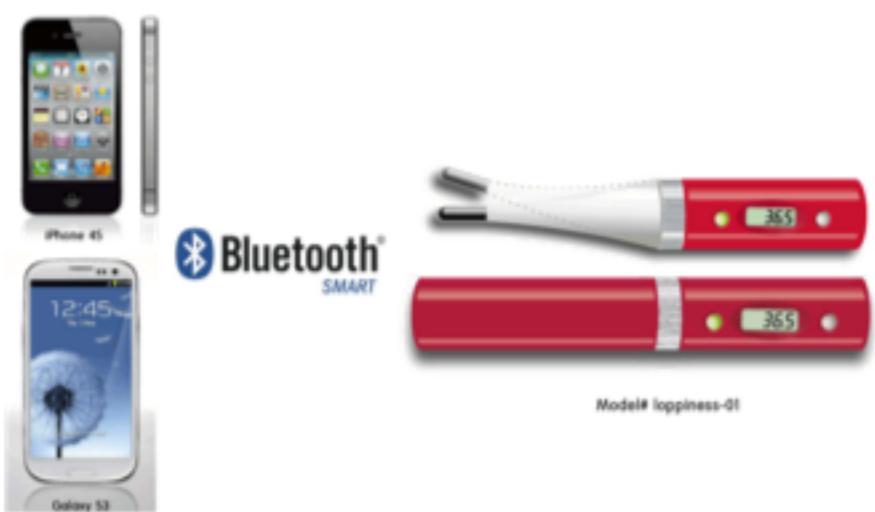
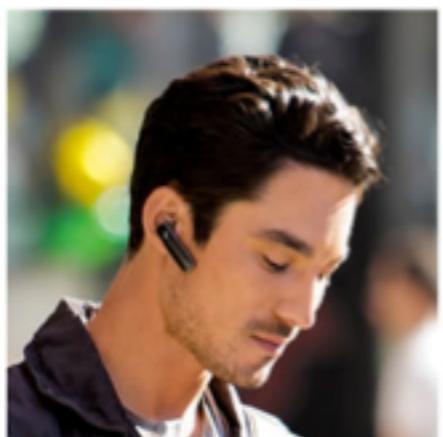
- ・ 今日の実習で必要なファイルを予めダウンロードしておいてください。
- ・ <http://bit.ly/1QRg8NS>
- ・ いち・きゅー・アール・ジー・はち・エヌ・エス

# Bluetoothとは

- 近距離無線通信を行うための規格
- コンピュータ、スマートフォンなどの端末から、無線で様々な周辺機器と通信できる。

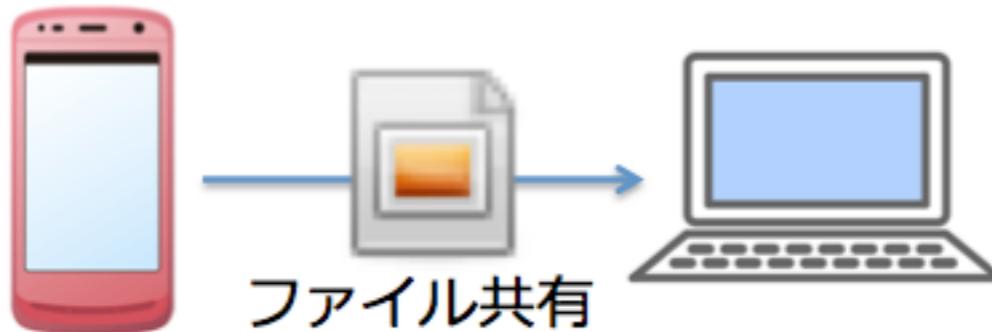


# Bluetooth対応機器





# Bluetooth対応機器



# Bluetoothの仕様

- 2.4GHzの電波帯を利用している
- 通信速度は最大24Mbps(Bluetooth 3.0)
- バージョン2.0～4.0の製品が流通している
- 機器の電波強度(クラス)次第で、最大100mまで届く

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

# Bluetooth のプロファイルとは？

- ・ キーボードやヘッドホンなど、機器の種類ごとに決められている通信プロトコルの使用方法のこと。
- ・ 同じプロファイルを持つ機器同士だけが通信できる。

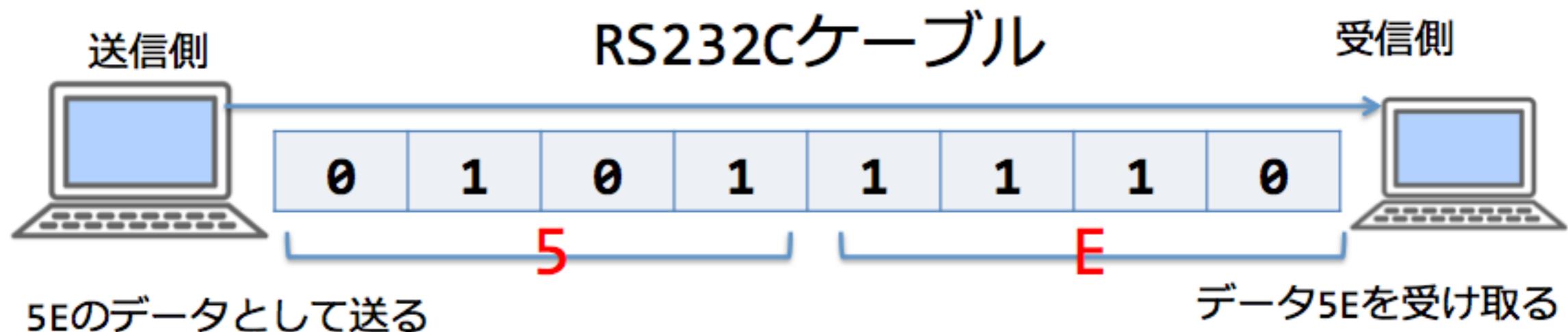
Android の  
API(Java) からは  
★のプロファイル  
が使用できます。  
(将来増えるかも!)

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

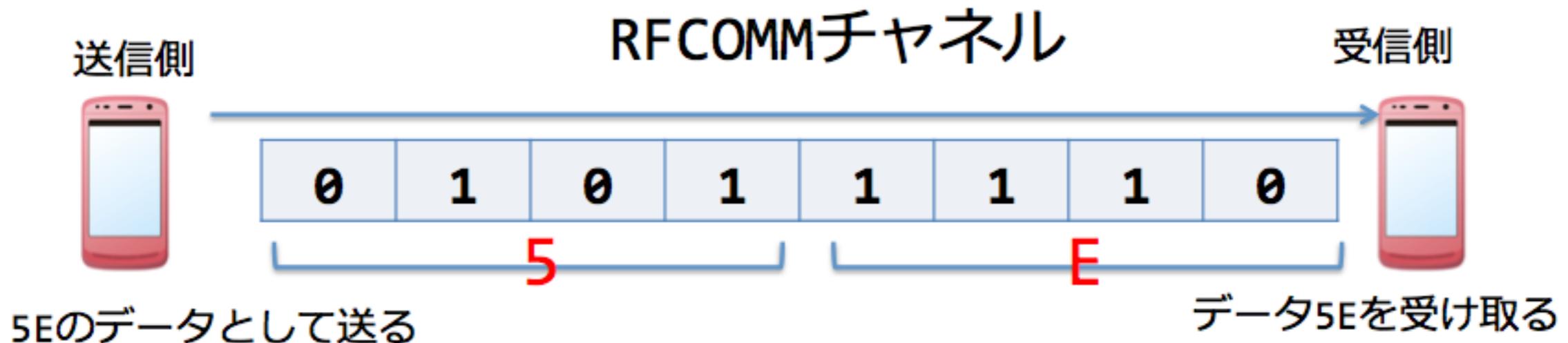
# シリアル通信とは

- データを送受信するために 1 本の伝送路にデータを 1 ビットずつ連續的に送受信する通信方式。
- 受信側でそれを復元して、意味のあるデータ(1 バイト=8 ビットのデータ)として処理する。



# RFCOMMとは

- Bluetoothでシリアル通信をエミュレートするトランスポート層プロトコルのこと。
- Radio Frequency Communication の略。
- RFCOMMチャネルを使うとBluetooth上でシリアル通信ができる。



# Bluetoothのバージョン

- Bluetooth 2.0～3.0
  - クラシックBluetoothと呼ばれる。
  - SPP,HSPなどのプロファイルはクラシック用。
- Bluetooth 4.0～
  - Bluetooth Low Energy(BLE)が追加された。
  - BLEはクラシックBluetoothと互換性がない。
    - BLEとクラシックBLEの共存ができる。
  - BLEはGATTと呼ばれるプロファイルを使う。

# 実習

- Bluetooth機能をアプリから制御してみよう。

# Bluetooth機能を制御するには

- android.bluetoothパッケージのAPI

BluetoothAdapter

端末のBluetooth機能を扱うクラス

BluetoothDevice

Bluetooth対応の外部機器のデバイス情報を扱うクラス

- パーミッションの指定(AndroidManifest.xml)

android.permission.BLUETOOTH

android.permission.BLUETOOTH\_ADMIN

# この実習でやること

1. Bluetooth 機能を有効にする。
2. 他の Bluetooth 機器をスキャンする。
3. すでにペアリングされたBluetooth機器の情報を取得する。
4. 外部機器のデバイスのスキャンに対して応答する。

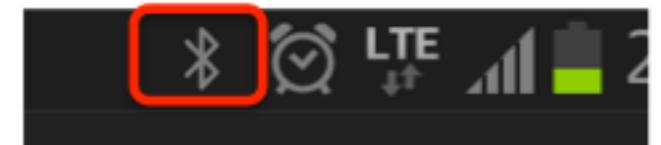
# BluetoothPractice1 プロジェクト

【実装するボタン】



# BluetoothPractice1

## 1. Bluetooth機能を有効にする



# BluetoothPractice1

1 zipファイルを展開して 「BluetoothPractice1」 プロジェクトを import する

2 AndroidManifest.xmlにパーミッションを追加する

```
<uses-permission  
    android:name="android.permission.BLUETOOTH_ADMIN"/>  
<uses-permission  
    android:name="android.permission.BLUETOOTH"/>
```

3 MainActivity.javaのコードを埋める

- initEnableButton() メソッドを完成させる  
(en1-1) BluetoothAdapterのインスタンス取得

```
btAdapter = BluetoothAdapter.getDefaultAdapter();
```

# BluetoothPractice1

- (en1-2)取り出したインスタンスの内容によって処理を分ける。
- Bluetooth未サポートまたは既に有効な場合はその旨をToastで表示する。無効だったら有効にするリクエストを投げる。

```
if (btAdapter == null) {  
    Toast.makeText(getApplicationContext(),  
        "この端末はBluetoothがサポートされていません。",  
        Toast.LENGTH_SHORT).show();  
    return;  
}  
  
if (btAdapter.isEnabled()) {  
    Toast.makeText(getApplicationContext(),  
        "Bluetoothは既に有効になっています。",  
        Toast.LENGTH_SHORT).show();  
} else {  
    // 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  
}
```

# BluetoothPractice1

- (en1-3) onActivityResultコールバックメソッドの実装

```
@Override  
protected void onActivityResult(int requestCode,  
                               int resultCode, Intent data) {  
    super.onActivityResult(requestCode, resultCode, data);  
  
    if (requestCode == REQUEST_BT_ENABLE) {  
        if (resultCode == Activity.RESULT_OK) {  
            Toast.makeText(getApplicationContext(),  
                "Bluetoothを有効にしました。", Toast.LENGTH_SHORT).show();  
        } else {  
            Toast.makeText(getApplicationContext(),  
                "Bluetoothの有効化に失敗しました。", Toast.LENGTH_SHORT).show();  
        }  
        return;  
    }  
}
```

# 確認

- ・ ここまでできたら動作確認。
- ・ ちゃんとBluetoothが有効にできたでしょうか？

復習

## アクティビティの起動

- 別のアクティビティを起動するには、**ContextクラスのstartActivityメソッド**を使う。
  - 引数：起動するアクティビティ情報のIntentオブジェクト

```
Public abstract void startActivityForResult(Intent intent)
```

- 別のアクティビティを起動し、起動したアクティビティから処理の結果を元のアクティビティで受け取るには、**ActivityクラスのstartActivityForResultメソッド**を使う。
  - 引数：起動するアクティビティ情報のIntentオブジェクト
  - 引数：要求コード

```
public void startActivityForResult(Intent intent,  
                                int requestCode)
```

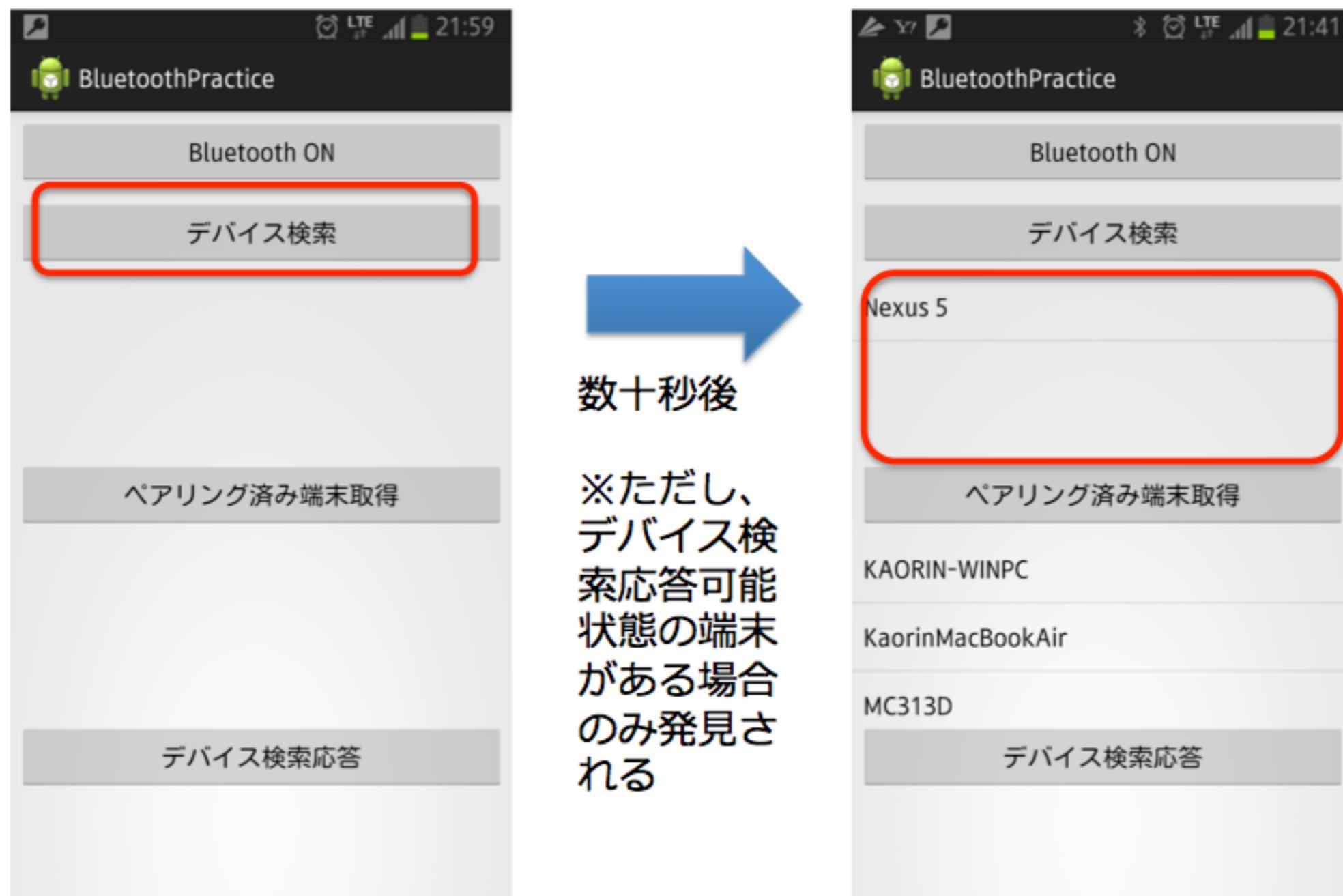
復習

## アクティビティの起動

- startActivityForResultメソッドで起動したアクティビティの処理結果を受け取るには、起動元のアクティビティで**onActivityResult**コールバックメソッドをオーバーライドする
  - 引数：要求コード
  - 引数：起動先のアクティビティの結果コード
  - 引数：起動先のアクティビティのIntentオブジェクト

```
protected void onActivityResult(int requestCode,  
                             int resultCode,  
                             Intent data)
```

## 2. 他のBluetooth機器をスキャンする



数十秒後

※ただし、  
デバイス検  
索応答可能  
状態の端末  
がある場合  
のみ発見さ  
れる

## ① MainActivity.javaのコードを埋める

- initDiscoverButton()メソッドを完成させる

(en2-1)BluetoothAdapterを取得し、もう既に探索中だったら一度キャンセルする

```
if (btAdapter == null) {  
    btAdapter = BluetoothAdapter.getDefaultAdapter();  
}  
if (btAdapter.isDiscovering()) {  
    btAdapter.cancelDiscovery();  
}
```

(en2-2)外部機器探索開始

```
btAdapter.startDiscovery();
```

- ブロードキャストリシーバークラス生成

(en2-3)外部機器探索中に発見した場合に受け取るアクションインテントを受け取るためのブロードキャストリシーバークラス作成

**ACTION\_FOUND** : デバイスが発見された場合

→BluetoothDeviceオブジェクトを取り出してListに格納する

**ACTION\_DISCOVERY\_FINISHED** : デバイス検索が終了した場合

→デバイス探索をキャンセルして終了通知をToastで表示する

```
private BroadcastReceiver mReceiver = new BroadcastReceiver() {  
    public void onReceive(Context context, Intent intent) {  
        // 受け取ったアクションを取り出す  
        String action = intent.getAction();
```

```
        if (BluetoothDevice.ACTION_FOUND.equals(action)) {  
            // BluetoothDeviceオブジェクトを取り出してListに格納する  
            BluetoothDevice device =  
                intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);  
            newDeviceList.add(device);  
        }  
    }
```

## (en2-3) 前のスライドからの続きです

```
else if (BluetoothAdapter.ACTION_DISCOVERY_FINISHED.equals(action)) {  
    // デバイス探索の終了通知をToastで表示する。  
    Toast.makeText(getApplicationContext(),  
        "外部機器の探索を終了しました",  
        Toast.LENGTH_SHORT).show();  
    adapter.clear();  
    if (newDeviceList.size() == 0) {  
        adapter.add("デバイスがありません");  
    }  
    for (BluetoothDevice device : newDeviceList) {  
        String name = device.getName();  
        Log.i(TAG, "デバイス名 = " + name);  
        adapter.add(name);  
    }  
    adapter.notifyDataSetChanged();  
}
```

- onCreateコールバックメソッド内  
(en2-4)外部機器探索状態のアクションインテントを受け取るための IntentFilter生成

```
IntentFilter filter = new IntentFilter();
filter.addAction(BluetoothAdapter.ACTION_DISCOVERY_FINISHED);
filter.addAction(BluetoothDevice.ACTION_FOUND);
```

- (en2-5)IntentFilterとレシーバーをシステムに登録する

```
registerReceiver(mReceiver, filter));
```

- onDestroyメソッド内

- (en2-6)外部機器探索のキャンセルを行い、レシーバーを登録解除する

```
@Override
protected void onDestroy() {
    if (btAdapter != null) {
        btAdapter.cancelDiscovery();
    }
    // レシーバーをシステムから登録解除する
    if (mReceiver != null) {
        unregisterReceiver(mReceiver);
        mReceiver = null;
    }
    super.onDestroy();
}
```

## ブロードキャストレシーバーとは？

- ・ システムで非同期に発生したイベントをインテントとして受け取るアプリケーションコンポーネント。
- ・ システムの起動、バッテリ残量の低下、日付や時刻の設定などが発生すると、発生したイベントを表すアクションが設定されたインテントがブロードキャストされる。
- ・ 独自に作成したアクションインテントをブロードキャストしたり、受け取ったりすることもできる。

## 復習

# システムからブロードキャストされる 代表的なアクションインテント

アクション	発生したイベント
ACTION_PACKAGE_CHANGED	パッケージの変更
ACTION_PACKAGE_FULLY_REMOVED	パッケージの削除
ACTION_BATTERY_LOW	バッテリ残量の低下
ACTION_BATTERY_OKAY	バッテリ残量OK
ACTION_SCREEN_OFF	画面のオフ
ACTION_SCREEN_ON	画面のオン
ACTION_SHUTDOWN	デバイスのシャットダウン
ACTION_BOOT_COMPLETED	システムのブート
ACTION_REBOOT	デバイスの再起動
ACTION_CONFIGURATION_CHANGED	設定の変更
ACTION_LOCALE_CHANGED	ロケールの変更
ACTION_TIME_TICK	現在時刻の変更
ACTION_DATE_CHANGED	日付の設定
ACTION_TIME_CHANGED	時刻の設定
ACTION_TIMEZONE_CHANGED	タイムゾーンの設定

復習

## ブロードキャストレシーバの使用例

例) Android端末起動時にメールをチェックする

- ・ ブロードキャストレシーバでAndroid端末起動完了時のブロードキャストインテントを受信し、メールチェックサービスを実行

例) バッテリー管理アプリを作る

- ・ ブロードキャストレシーバでバッテリー残量低下時のブロードキャストインテントを受信し、バッテリー残量を表す画像を変更して表示する

復習

## ブロードキャストレシーバを実装する方法

- ブロードキャストレシーバのクラスを新規作成する
  - BroadcastReceiverクラスのnewする
  - onReceiveコールバックメソッドの実装
- 作成したブロードキャストレシーバを登録する
  - 2通りの登録方法
    - AndroidManifest.xmlに登録する
      - <receiver><intent-filter>
    - Javaのコードで registerReceiverメソッドで IntentFilterとともに登録する

### 3.既にペアリングされたBluetooth機器の情報を取得する



- ① MainActivity.javaのコードを埋める
- initBondedButton()メソッドを完成させる
- (en3-1) ペアリング済みの端末情報一覧を取得する

```
Set<BluetoothDevice> bondedDevices =  
    btAdapter.getBondedDevices();
```

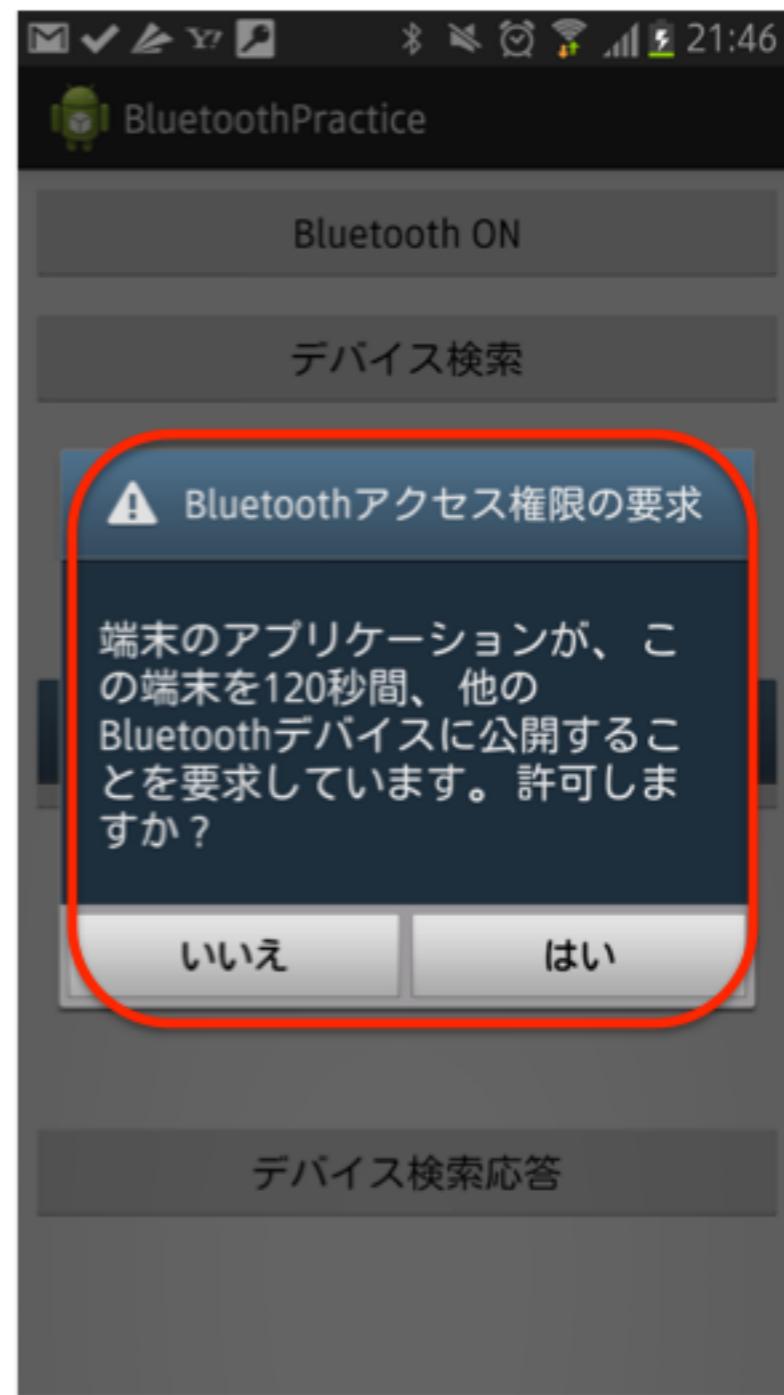
- (en3-2) リスト表示するためにアダプタにセットする

```
adapter2.clear();  
  
if (bondedDevices.size() == 0) {  
    adapter2.add("ペアリング済み端末なし");  
}  
  
for (BluetoothDevice device : bondedDevices) {  
    adapter2.add(device.getName());  
}  
  
adapter2.notifyDataSetChanged();
```

# 動作確認

- ・ ここまでできたら動作確認してみましょう。
- ・ ※ペアリング済み端末がなければ「ペアリング済み  
端末なし」と表示されればOKとします。

## 4. 外部機器のデバイスのスキャンに対して応答する



## ① MainActivity.javaのコードを埋める

- initResButton()メソッドを完成させる

(en4-1)外部機器から自端末を発見可能にするためのアクションインテントを発行

```
Intent intent = new Intent(  
    BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
```

(en4-2) onActivityResultメソッドで結果を受けるようにする

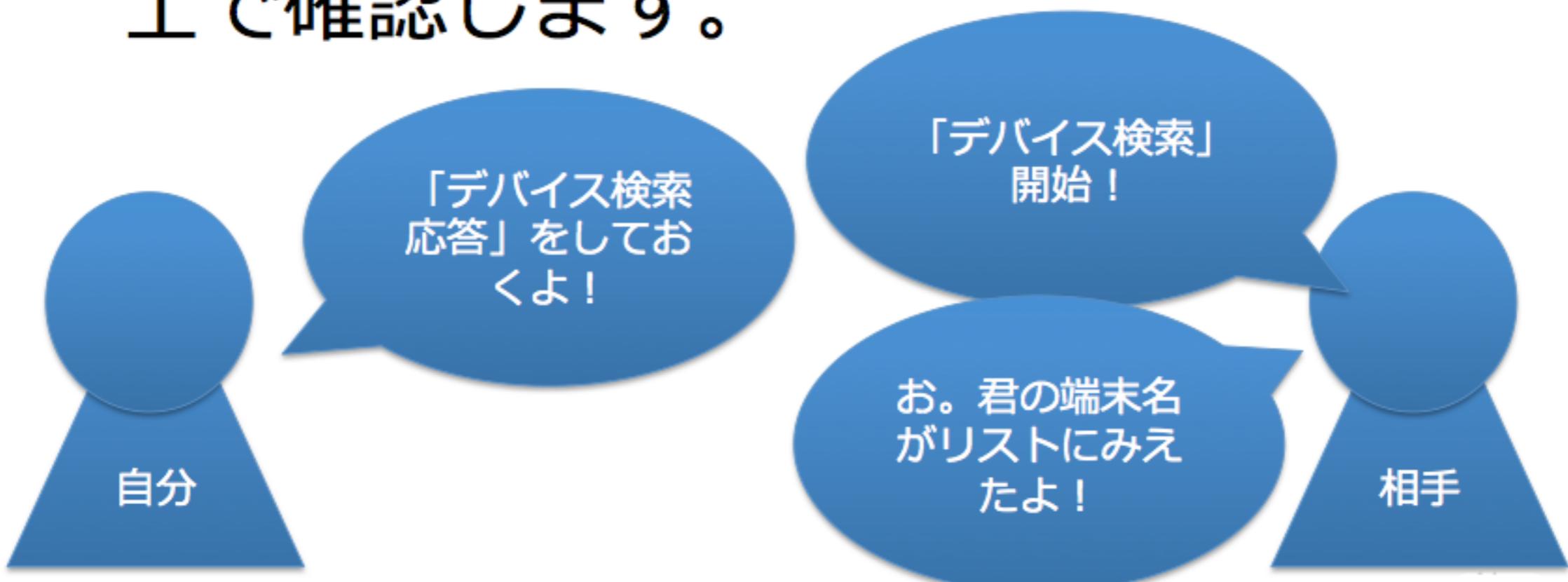
```
startActivityForResult(intent, REQUEST_BT_DISCOVERABLE);
```

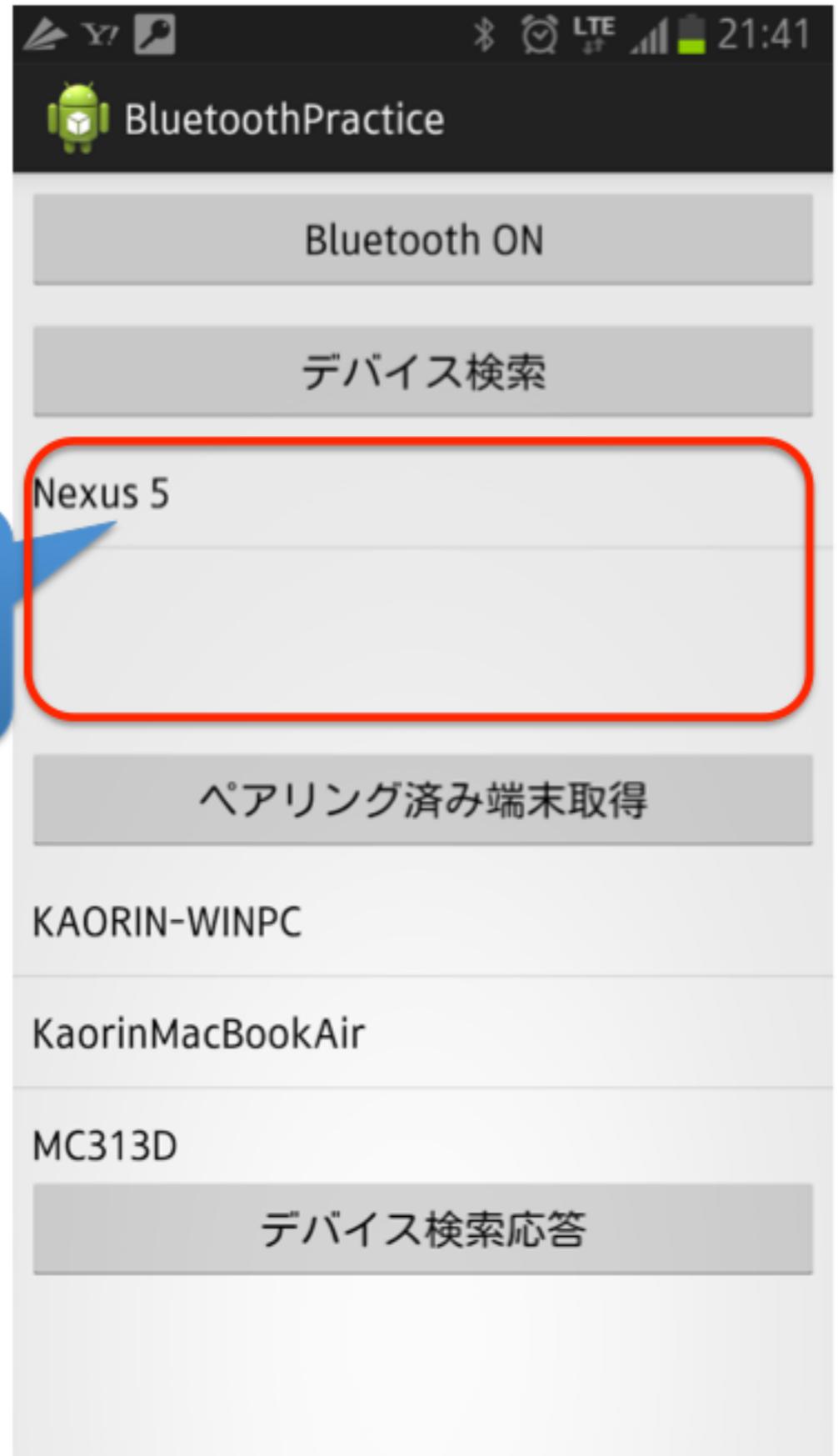
- `onActivityResult()`コールバックメソッド内に追加する  
(en4-2)の応答処理

```
if (requestCode == REQUEST_BT_DISCOVERABLE) {
    if (resultCode == Activity.RESULT_CANCELED) {
        Toast.makeText(getApplicationContext(),
            "端末は探索可能な状態ではありません",
            Toast.LENGTH_SHORT).show();
    } else {
        Toast.makeText(getApplicationContext(),
            resultCode + "秒間外部機器から探索可能状態です",
            Toast.LENGTH_SHORT).show();
    }
}
```

# 動作確認

- ここまでできたら動作確認をしましょう。「デバイス検索」機能の確認には2台必要です。席の近い人同士で確認します。





5. デバイス検索結果のリストから端末名をタップすると端末情報が確認できる

## 【おまけ】デバイス名を変えてみよう

- ・ 標準では機種名(SC-01Fなど)になっていますが、好きな名前に変えることができます。
- ・ 設定方法は機種により異なりますが、みなさんの端末では「設定 > 端末情報 > デバイス名」がBluetoothのペアリング時に外部から見えるデバイス名になります。  
(英語設定なら aboutの中にあります)

## 5. Bluetooth対応機器の情報表示



## ① MainActivity.javaのコードを埋める

- MainActivityクラスの宣言部分

(5-1) ListViewのアイテムクリックイベントを取得できるようにするために、MainActivityクラスにOnItemClickListenerをimplementsする

```
//## 演習5 ## OnItemClickListenerをimplementsして
// ListViewのアイテムクリックイベントを取得できるようにする
public class MainActivity extends Activity implements
    OnItemClickListener {
```

(en5-2) OnItemClickListenerのonItemClickコールバックメソッドをオーバーライドする

```
@Override
public void onItemClick(AdapterView<?> parent, View view,
    int position, long id) {
}
```

- onClickItemコールバックメソッド内の実装  
(en5-3) クリックされた端末名の端末情報  
(端末名とMACアドレス)をトーストで表示する。

```
ListView listView = (ListView) parent;
if (listView.getId() == R.id.device_list) {
    if(position < newDeviceList.size()) {
        BluetoothDevice device = newDeviceList.get(position);
        Toast.makeText(
            getApplicationContext(),
            "端末名 = " + device.getName() +
            " MAC ADDRESS = " + device.getAddress(),
            Toast.LENGTH_SHORT).show();
    }
}
```

休憩(10分)

## 動作確認

ここまでできたら動作確認。  
ちゃんと端末情報が表示された  
でしょうか？

# AndroidのBluetooth接続と データ送信(実習)

# Bluetooth接続とデータの送受信

- android.bluetoothパッケージのAPI

## BluetoothSocket

クライアントとしてリモートデバイスに接続するためのクラス

- RFCOMMで通信する場合は、BluetoothDeviceクラスのcreateRfcommSocketToServiceRecordメソッドでBluetoothSocketオブジェクトを取得できる。
- connectメソッドで接続要求。
- 接続完了後はBluetoothSocketオブジェクトから取得できるInputStream、OutputStreamでデータの送受信を行う。

Bluetooth通信の接続処理は画面のスレッド(ActivityのUIスレッド)とは別のスレッドで行う

# Bluetooth接続とデータの送受信

- android.bluetoothパッケージのAPI

## BluetoothServerSocket

サーバとしてリモートデバイスからの接続要求を受け付けるためのクラス

- RFCOMMで通信する場合は、BluetoothAdapterクラスのlistenUsingRfcommSocketWithServiceRecordメソッドでBluetoothServerSocketオブジェクトを取得できる。
- acceptメソッドで接続要求受付。受付完了後の戻り値としてBluetoothSocketオブジェクトを取得。
- InputStream、OutputStreamを使用したデータの送受信が可能になる。

Bluetooth通信の接続処理は画面のスレッド(ActivityのUIスレッド)とは別のスレッドで行う

# Bluetooth接続に必要な条件

- 接続する機器同士が同じプロファイルを所有していること。
  - 所有有無はBluetooth搭載機器内部にある「SDP対象のサービス一覧データベース」に保有しているか否か
- MACアドレスとUUIDを指定して接続する
  - MACアドレス・・・ネットワーク機器のハードウェアに（原則として）一意に割り当てられる物理アドレス
  - UUID・・・Universally Unique Identifierとは、2つ以上のアイテムが同じ値を持つことがない一意な識別子のこと。

# SDP

- Service Discovery Protocol
- 相手の機器がサポートしているサービス（プロファイル）を検索するためのプロトコル。
- SDPで検索可能なプロファイルのUUIDの規定値はBluetoothSIGで定められている。

<http://goo.gl/LyCtrb>

UUID名	UUID	許可された 使用法	
BASE_UUID	00000000 0000-1000-8000-00805F9B34FB		
サービスクラス名	UUID	仕様	サービスクラス
ServiceDiscoveryServerServiceClassID	0x1000	Bluetooth コア仕様	
BrowseGroupDescriptorServiceClassID	0x1001	Bluetooth コア仕様	
SerialPort	0x1101	Serial Port Profile (SPP) <small>注意: SPP v1.0 の SDP レコード例は、BluetoothProfileDescriptorList 属性を含まないが、いくつかの実装はこの UUID をプロファイル識別子用にも使用できる。</small>	サービスクラス/プロファイル

## [実習2] Bluetooth接続と 文字列データの送受信

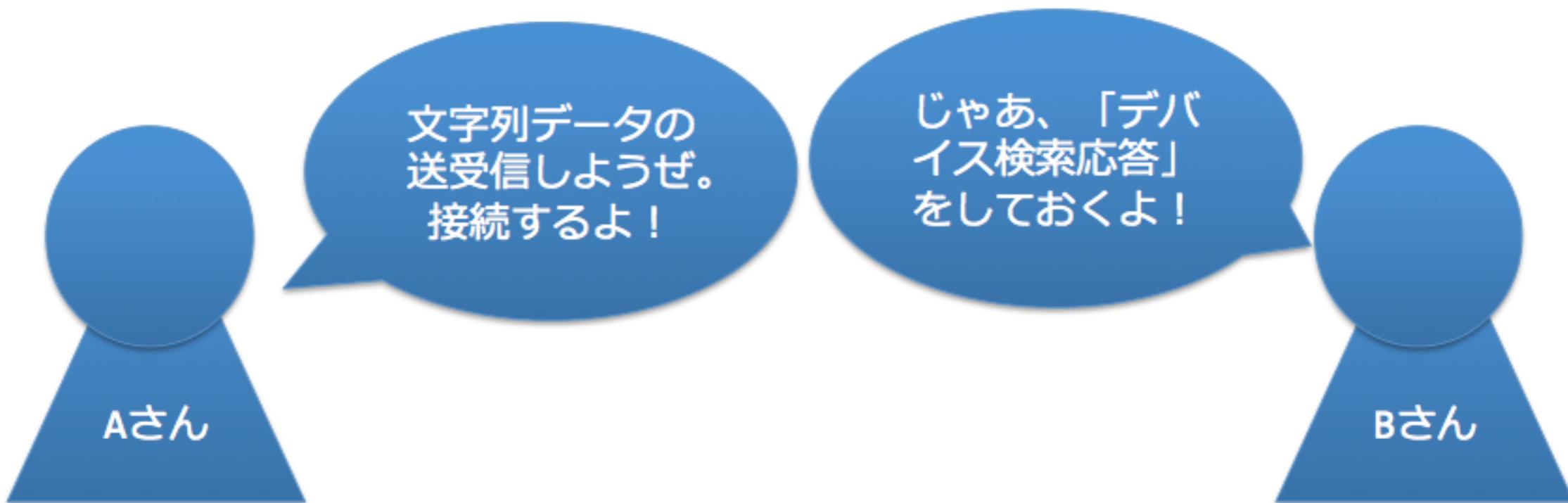
この実習でやること

1. 他のBluetooth機器とRFCOMMチャネルを確立し、接続する
2. 接続された他のBluetooth機器との間で文字列データをやりとりする

※本実習は既に用意された「**BluetoothPractice2**」プロジェクトを importして各自の端末にインストールして動作を確認し、その後、コードの説明をします。

# 動作確認

- 最初に動作確認をしましよう。「文字列データの送受信」の確認には2台必要です。席の近い人同士で確認します。



# BluetoothPractice2

## 起動



# BluetoothPractice2

## 接続

DeviceSelectActivity

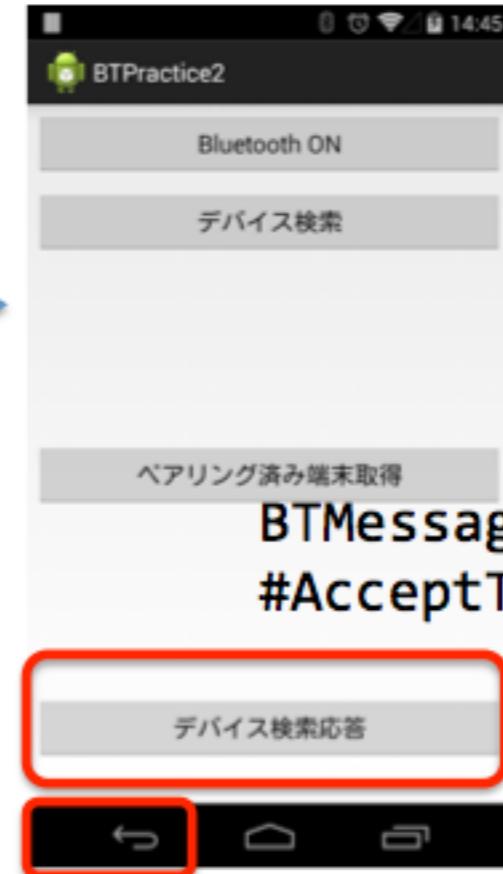


クライアント



Aさん

DeviceSelectActivity



サーバー  
BTMessagingThreads  
#AcceptThread



Bさん

デバイス検索をして相手の端末名をクリック。  
クライアントとして接続する処理が行われる。

# BluetoothPractice2

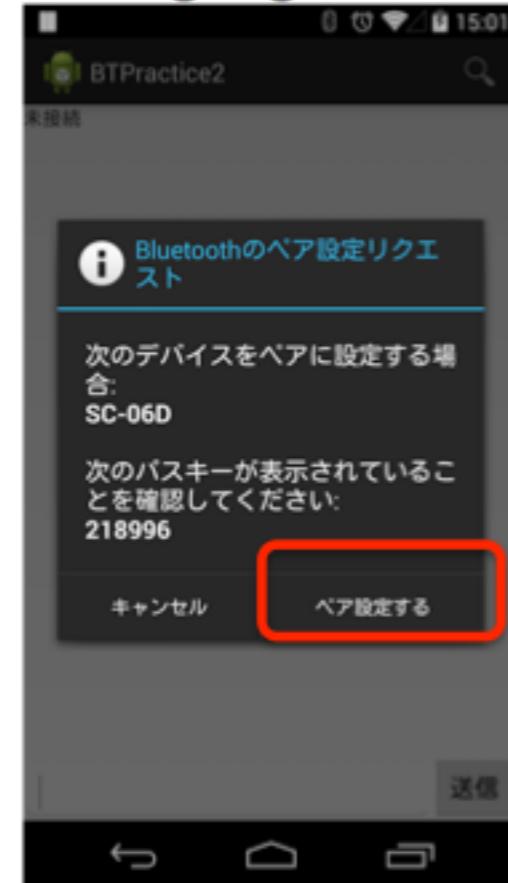
## ペアリング

MessagingActivity



クライアント

MessagingActivity



サーバー

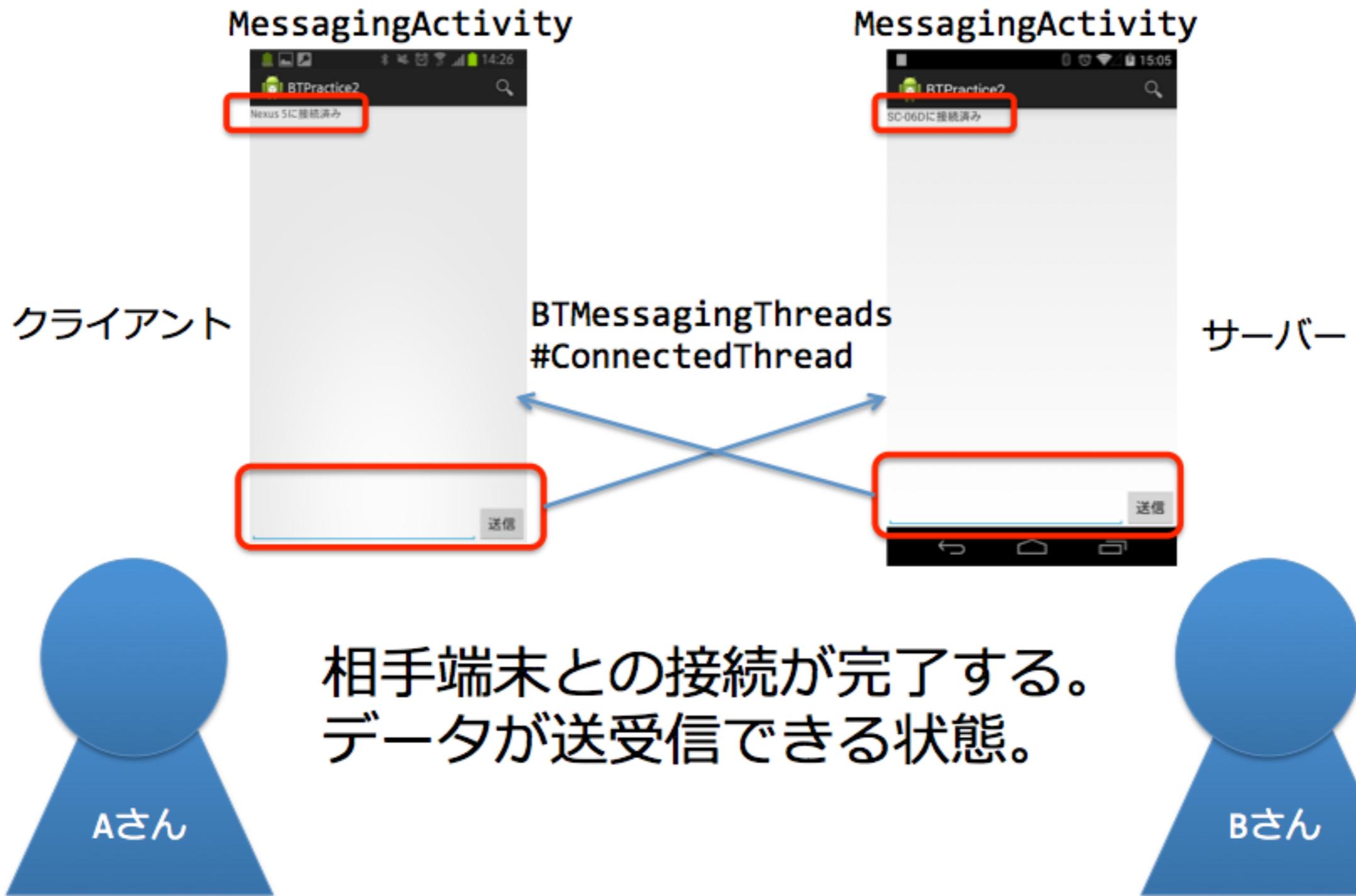
両端末にペアリングのダイアログが表示されるのでペア設定する。

Aさん

Bさん

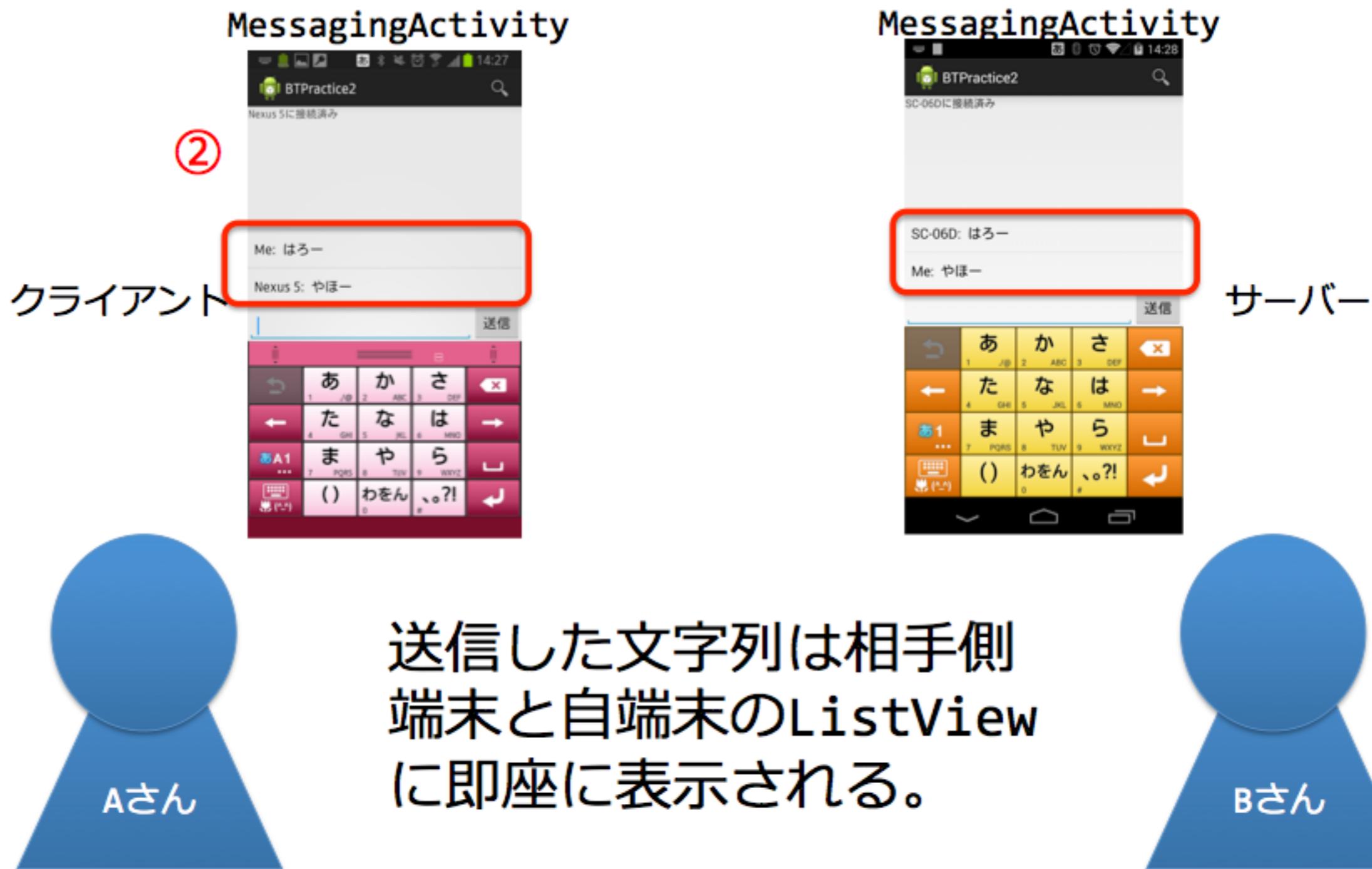
# BluetoothPractice2

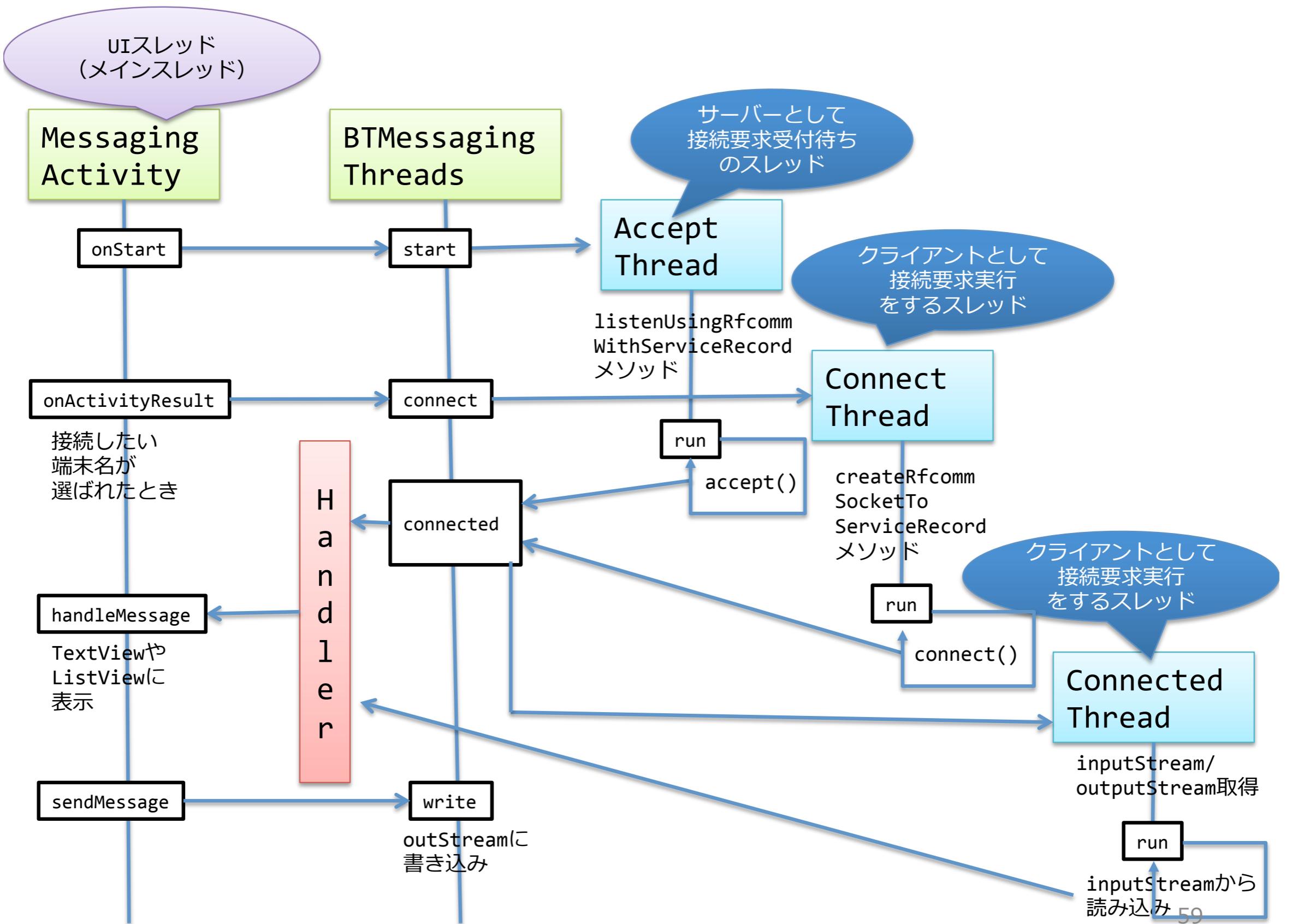
## 接続完了



# BluetoothPractice2

## 文字列データの送受信





復習

# Javaのスレッド

- ・スレッドはそのプログラム中の一連の処理を意味する。
- ・スレッドはメモリ空間を共有する。
- ・Javaでは複数のスレッドを同時実行させる並列処理（マルチスレッド）が可能。

```
java.lang.Thread  
class クラス名 extends Thread {  
    public void run() {  
        // スレッドの処理をかく  
    }  
}
```

# スレッドセーフ

- 複数のスレッドから無秩序にオブジェクトのフィールドへアクセスされた場合でも、正しい状態を維持し続けられるようにクラスをつくることが大切。
- メソッドに **synchronized** をつけることにより他のスレッドからのアクセスをロックすることができる。（オブジェクトの排他制御）
- 複数のオブジェクトに対して排他制御を行う場合は、デッドロックに注意。

# Handler

- Androidの場合はUIスレッドからでないとUI部品を操作できないという制約がある。
- Androidの**Handlerクラス**を使う事により、別スレッドからUI部品操作を用いることができる。
  - Handlerインスタンスの生成元のスレッドへイベントを送るための仕組みなので、UIスレッドで生成する。

復習

# Handler

UIスレッド

```
private Handler mHandler;  
  
mHandler = new Handler(){  
    //メッセージ受信  
    public void handleMessage(Message message) {  
        //受け取ったメッセージをTextViewなどのUI部品に表示など  
    };  
};
```

別のスレッド

```
//ハンドラへのメッセージの生成  
Message msg = mHandler.obtainMessage("メッセージ");  
//UIスレッドのハンドラへのメッセージ送信  
mHandler.sendMessage(msg);
```

## 確認

以上をふまえて、  
**「BluetoothPractice2」**  
プロジェクトの  
コードトレースを  
してください。

Bluetooth  
プロファイルのサポート

# Androidでサポートされている プロファイル

AndroidのAPIでデータの取得やBluetooth機器のコントロール) ができるプロファイル

- **HSP (Headset Profile)**
  - 携帯電話といっしょに使うBluetoothヘッドセットの接続機能。
- **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)をするために携帯電話を使用するための機能。
- **A2DP (Advanced Audio Distribution Profile)**
  - Bluetooth機器間で高品質のオーディオストリームを流すしくみ
- **HDP (Health Device Profile)**
  - Bluetooth搭載の健康管理機器間を接続、データ通信

# どうやるのか？

対応の機器がないと確認ができないので概要のみ説明します。

- ① android.bluetooth.BluetoothProfileインターフェースのServiceListenerを生成
- ② 指定のプロファイルとともにリスナーをBluetoothAdapterに登録
- ③ 指定のプロファイルを持つBluetooth機器への接続時と切断時のイベントメソッドが有効になる
- ④ 接続時、プロファイル固有のオブジェクトを取得できる

# なにができるのか？

ブロードキャストレシーバーでアクションインテントを受け取ることにより機器の状態を管理する

## BluetoothHeadset (HSP/HFP)

ACTION_AUDIO_STATE_CHANGED	A2DPの音声状態変化
ACTION_CONNECTION_STATE_CHANGED	HSPの接続状態の変化
ACTION_VENDOR_SPECIFIC_HEADSET_EVENT	ベンダー固有のイベント通知

## BluetoothA2dp (A2DP)

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

# なにができるのか？

制御用のメソッドを使う

BluetoothHeadset (HSP/HFP)

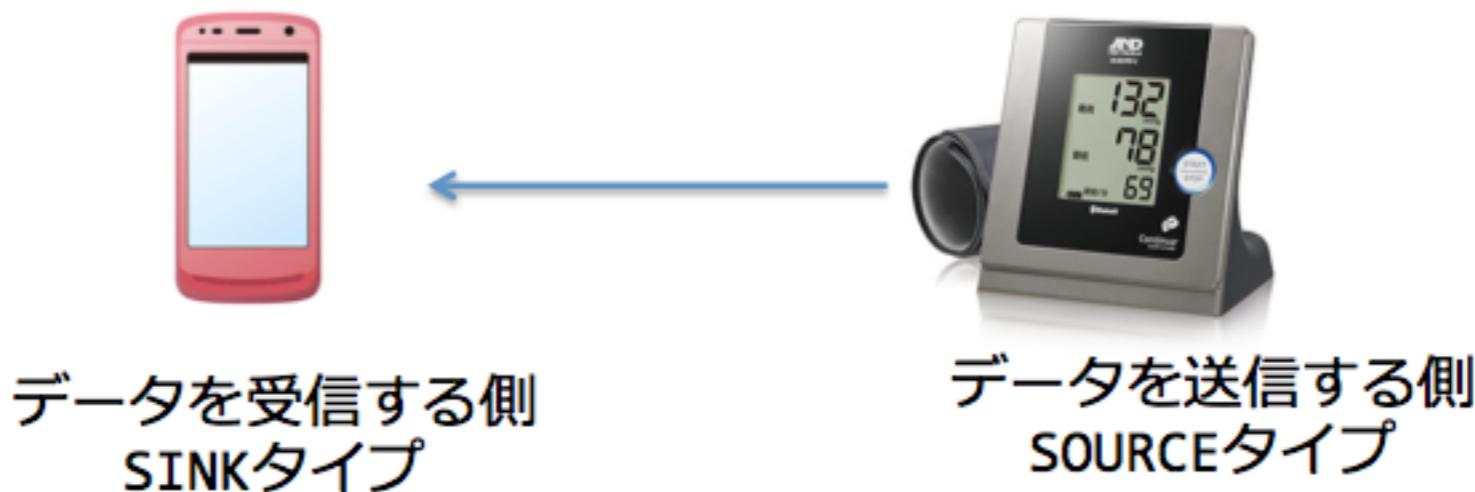
音声認識の開始と終了

```
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)
```

# ヘルス機器(HDP)との接続と データ取得方法



- ① BluetoothHealthオブジェクトの取得（前述のとおり）
- ② android.bluetooth.BluetoothHealthCallbackクラスを生成
- ③ BluetoothHealthオブジェクトにコールバックを登録
- ④ Bluetoothヘルス機器への接続
- ⑤ ヘルス機器への接続状態変化通知の取得
  - onHealthAppConfigurationStatusChangeコールバックメソッド
- ⑥ ヘルス機器からデータを取得
  - onHealthChannelStateChangeコールバックメソッド

# ヘルス機器(HDP)のデータ解析

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

心拍計ならIEEE 11073-10407

歩数計ならIEEE 11073-10441

・・・

# Bluetooth Low Energy (BLE)

# Bluetooth Low Energy

- ・超低消費電力
- ・リチウム乾電池 1つで 1年程度動作可能
- ・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）
- ・プロファイルはGATT（Generic Attribute）

# 互換性

- ・クラシックBluetoothとBLEは全く互換性がない
- ・ハードウェアとして共存させることは可能

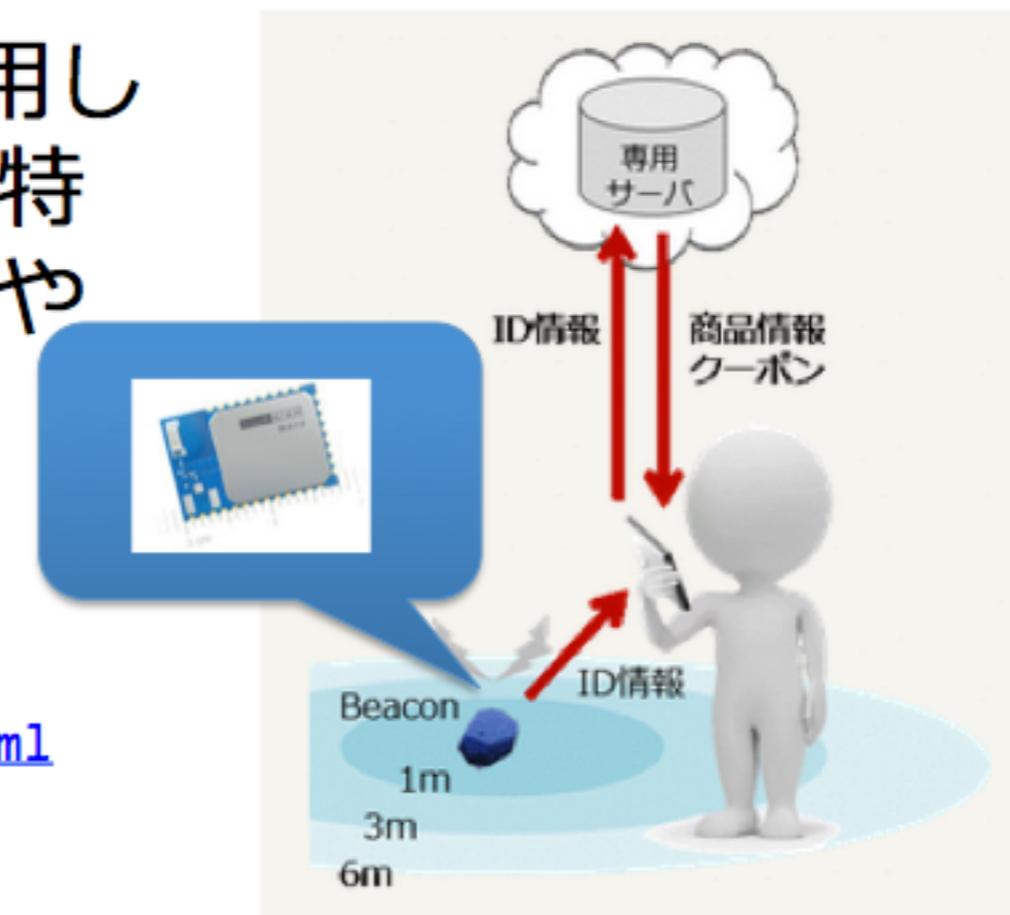
名称	説明
Bluetooth SMART	BLEでのみ通信可能で、クラシックとの通信はできない
Bluetooth SMART READY	BLEとクラシックの両方の通信に対応

# BLEは何につかうの？

- ・フィットネス系のセンサーヤ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

ビーコン・・・BLEを利用して位置情報の取得や端末特定を行い、個別の情報をやりとりする仕組み

(参考)  
おもてナビ  
<http://omotenavi.jp/beacon/index.html>  
東京デバイセズ  
<http://tokyodevices.jp/items/164>



# AndroidでBLE

## Android4.3以上

LeScanCallback	GATTプロファイル対応の外部機器の探索。
BluetoothGattCallback	指定したBLE機器と接続。
BluetoothGatt	指定したUUIDを持つGATTサービスの検索、キャラクタリストイックの読み書き、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。
BluetoothGattService	指定のGATTサービスのキャラクタリストイックが取得できる。
BluetoothGattCharacteristic	GATTプロファイルのキャラクタリストイックのクラス。ディスクリプタを取得など。
BluetoothGattDescriptor	アプリケーションからの値の読み書きの対象。

休憩(30分)

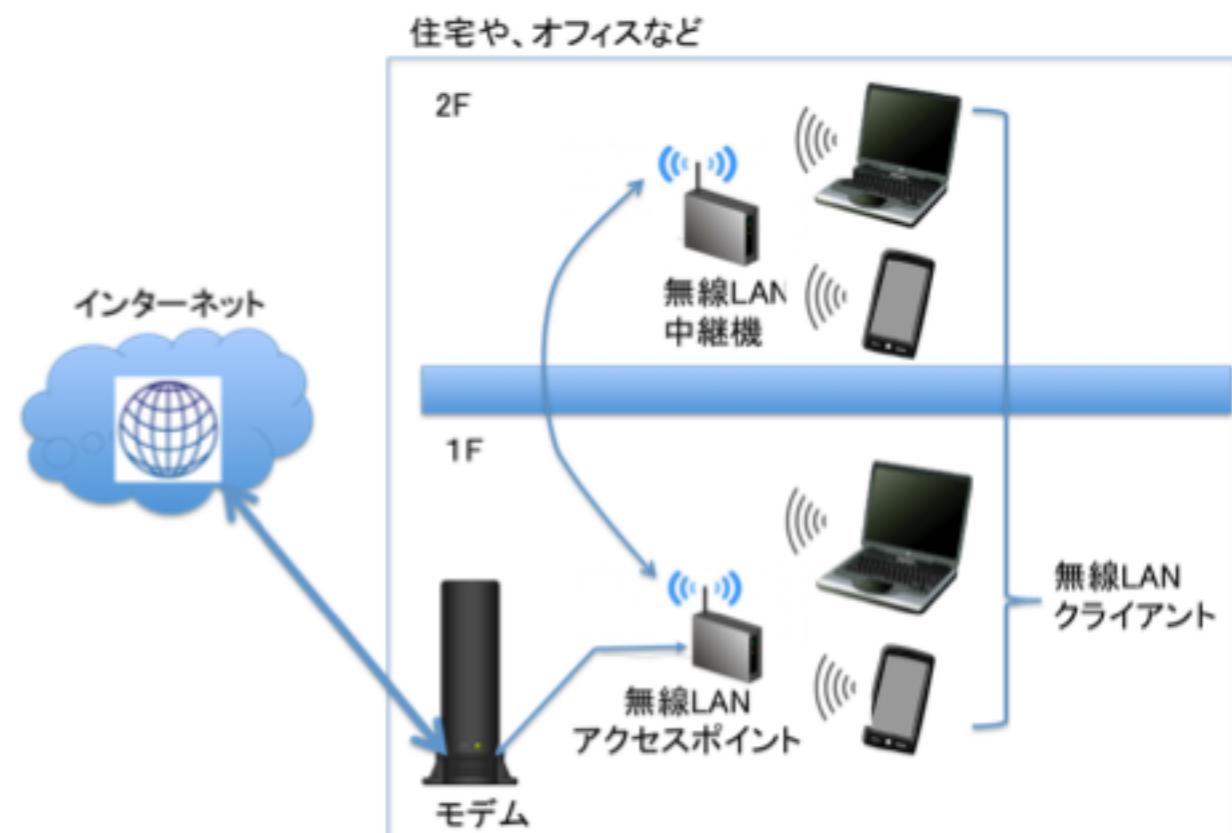
Wi-Fi通信

17-5章

Wi-Fiとは

# 無線LANとは

- 有線ケーブルを使わずに  
アクセスポイントと呼ば  
れる中継機やルータから  
発する電波を用いて数m  
～数十m程度の範囲内で  
PCや電子機器間、もしく  
はインターネットへの接  
続で高速なデータ通信を  
行う通信技術



# 規格

規格	周波数帯	伝送速度
802.11b	2.4~2.5GHz	最大11Mbps
802.11a	5.15~5.35GHz 5.47~5.725GHz	最大54Mbps
802.11g	2.4~2.5GHz	最大54Mbps
802.11n	2.4~2.5GHz 5.15~5.35GHz 5.47~5.725GHz	最大600Mbps

参考

<http://www.allied-telesis.co.jp/products/list/wireless/know1.html>

# アドホックモード

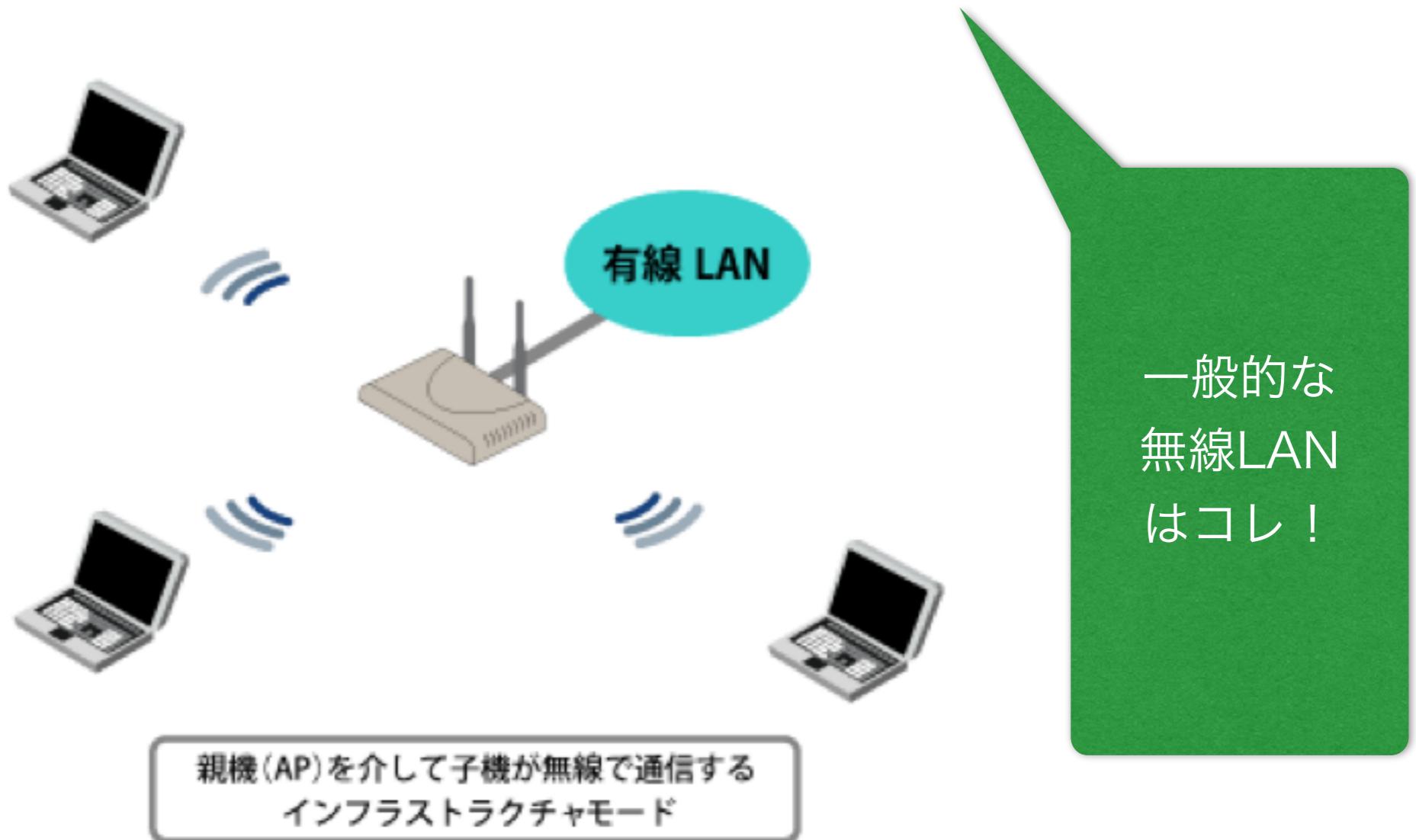


子機同士が無線で通信を行なう  
アドホックモード

参考

<http://www.allied-telesis.co.jp/products/list/wireless/knowl.html>

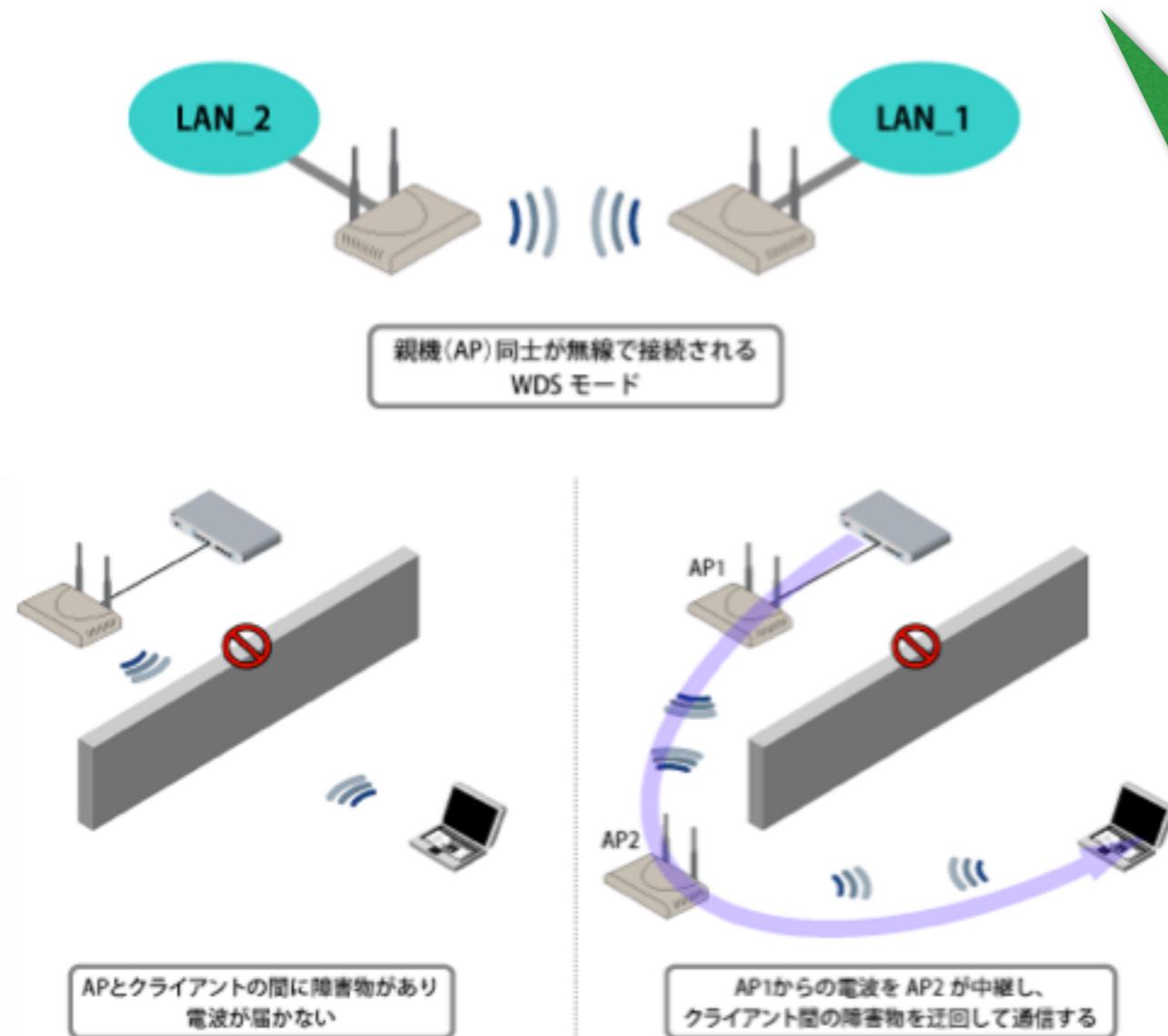
# インフラストラクチャモード



参考

<http://www.allied-telesis.co.jp/products/list/wireless/know1.html>

# WDSモード



メーカーごとに別々の仕様なので同機種同士でないと使えない。

## 参考

<http://www.allied-telesis.co.jp/products/list/wireless/know1.html>

Wi-Fi (Wireless Fidelity) とは

- 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。

# 接続

- 一般的に接続したいアクセスポイントのSSIDを指定し、セキュリティ保護用の認証用パスワードを指定する。

## SSID

Wi-FiのアクセスポイントのID。

最大32文字までの英数字を任意に設定できる

# 認証方式

WEP	固定されたPSK (Pre-Shared Key) を使用した暗号化を行う。鍵は13文字までの数字か、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ない。
WPA	WEPの強化版となる暗号化方式を採用。
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。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。
★ 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。CCMP (Counter Mode with Cipher Block Chaining Message Authentication Code Protocol) と呼ばれる技術を使用。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。
★ WPA2	WPAの後継規格。AESが義務化されるかたちで標準化。WPA同様、TKIPやAESを使用する事が可能な暗号化方式。

AndroidのWi-Fi機能を  
アプリケーションで制御する

Wi-Fi機能用のフレームワークでできること

- 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

# Androidのフレームワーク

- android.net.wifiパッケージ

`WifiManager`

端末のWi-Fiの設定制御を行うクラス

`WifiConfiguration`

接続に必要な設定情報のクラス

`WifiInfo`

Wi-Fi接続情報を扱うクラス

- パーミッションの指定(`AndroidManifest.xml`)

`android.permission.ACCESS_WIFI_STATE`

`android.permission.CHANGE_WIFI_STATE`

`android.permission.CHANGE_WIFI_MULTICAST_STATE`

## フレームワークの使い方

- Wi-Fiの設定制御 (WifiManager)
- WiFiネットワークの検索
- Wi-Fi機器への接続と認証
- WiFiネットワーク設定情報の取得
- Wi-Fi接続情報

# 動作確認

- 最初にダウンロードしたzipファイルの中にある、 WiFiPractice プロジェクトをインポートして実行して下さい。  
( practice/WiFi-Sample/WiFiPractice)  
※接続ボタンはSSIDとパスワードにダミーを設定しているのでそのままでは動きません。

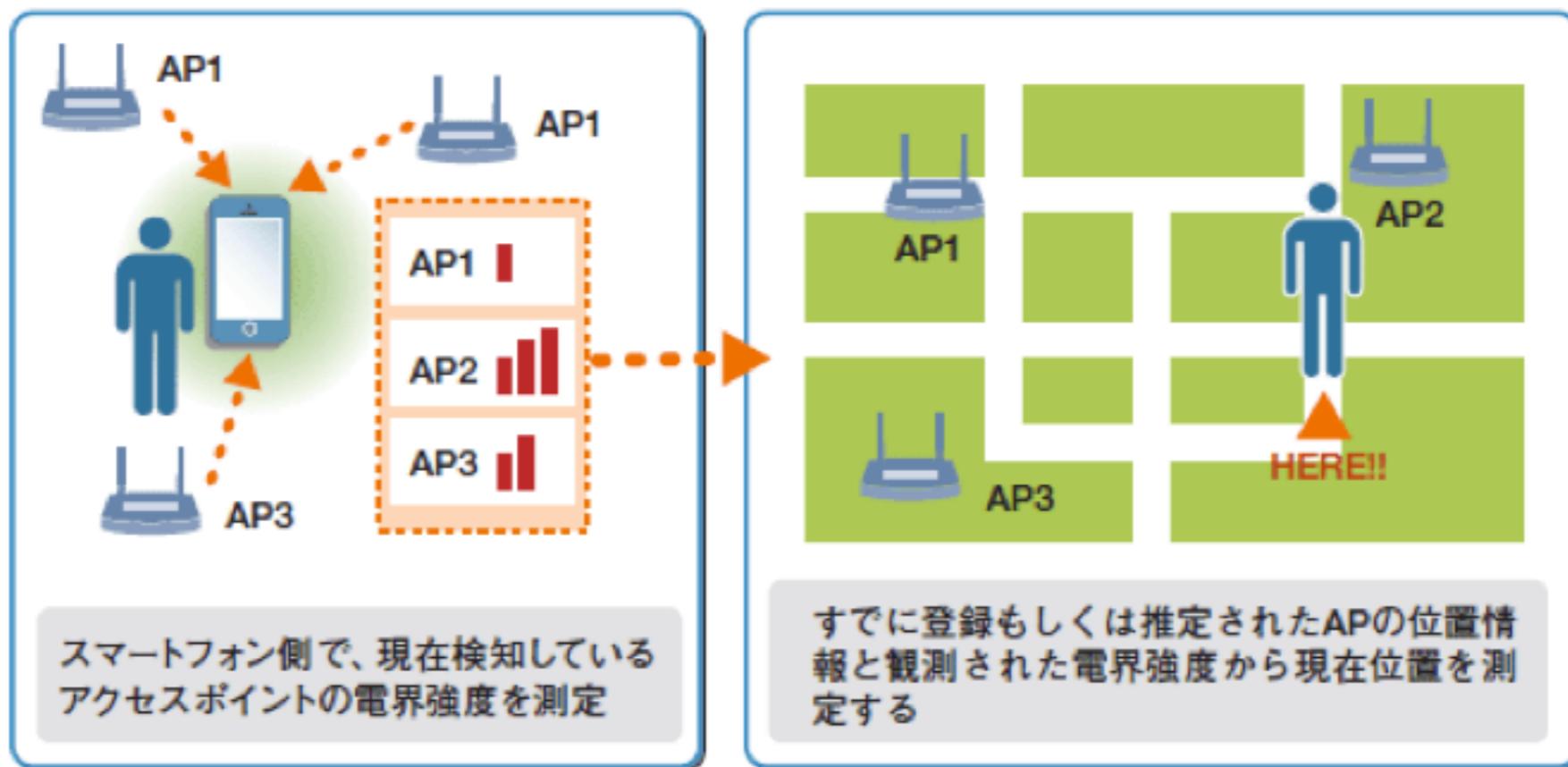
# WiFiPractice



## Wi-Fi接続情報

SSID	無線識別用のID (Service Set Identifier)
IPアドレス	機器のIPアドレス
MACアドレス	機器のMAC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

# アプリケーション例（屋内測位）



参考：

<http://businessnetwork.jp/Detail/tabid/65/artid/3108/Default.aspx>

IPマルチキャスト

## IPマルチキャスト

- 複数の特定のマシンに対してデータを送信する。
- 宛先IPアドレスとしてマルチキャストアドレス(`WiFiManager`で取得可能)を指定。
- Androidでは通常、バッテリ消費の観点からマルチキャストアドレス宛てパケットを通さないようになっている。

# Androidのフレームワーク

- android.net.wifiパッケージ

WifiManager.MulticastLock

IP Multicastのパケットを受け取る

- java.netパッケージ

MulticastSocket

ソケットの生成

- パーミッションの指定(AndroidManifest.xml)

android.permission.CHANGE\_WIFI\_MULTICAST\_STATE

# Wi-Fi Direct

# Wi-Fi Direct

- ・アクセスポイントがなくてもWi-Fi Directに対応する機器同士が1対1（アドホック）で通信することができる仕組み。



# 特徴

- Bluetooth接続に比べて通信速度が高速(最大250Mbps)。最大90mの距離で通信できる。
- 「1対多」の通信も行える。ただし同時に通信できるのは「1対1」になる。
- 無線の接続設定には簡易接続方式のWPSが使われる。
- 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 SDKに  
「WiFi Direct Demo」という  
サンプルプロジェクトがあるので参考にしましょう。

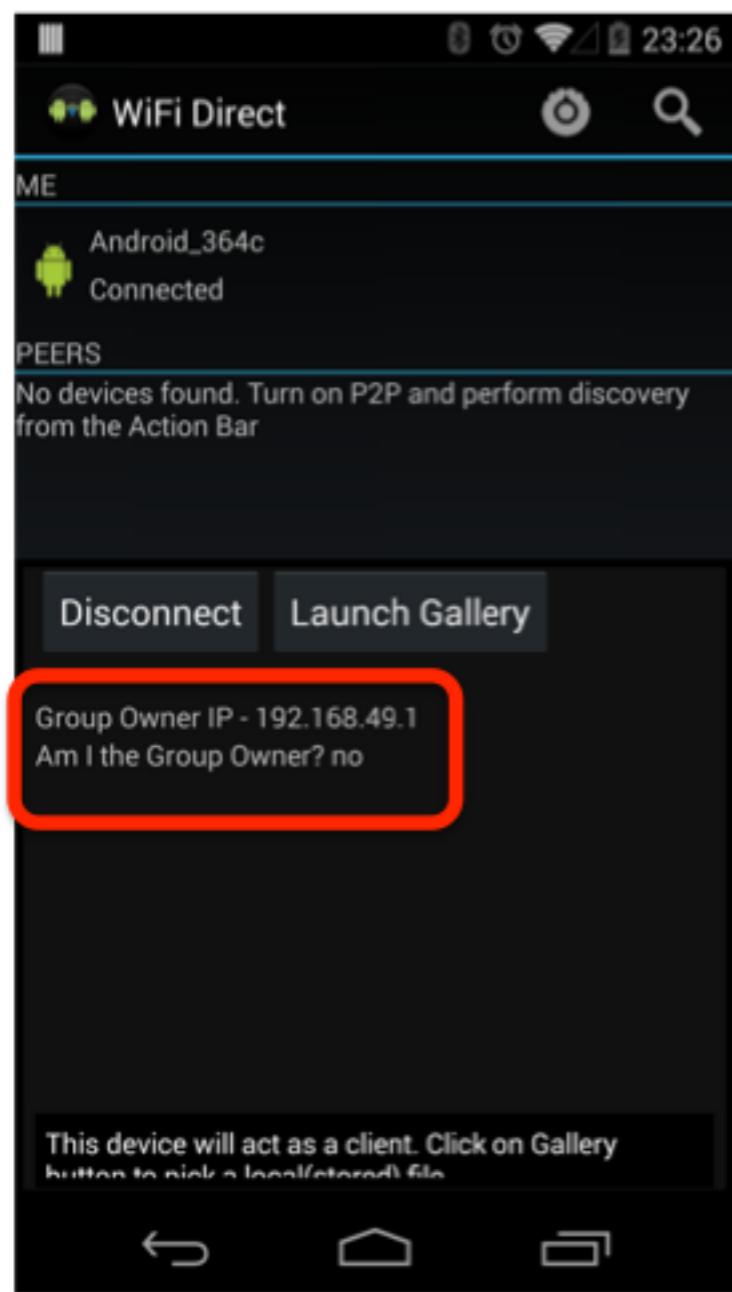
- 最初にダウンロードしたzipファイルの中にも用意してあります。  
( practice/WiFi-Sample/WiFiDirectDemo)

# WiFi Direct 接続

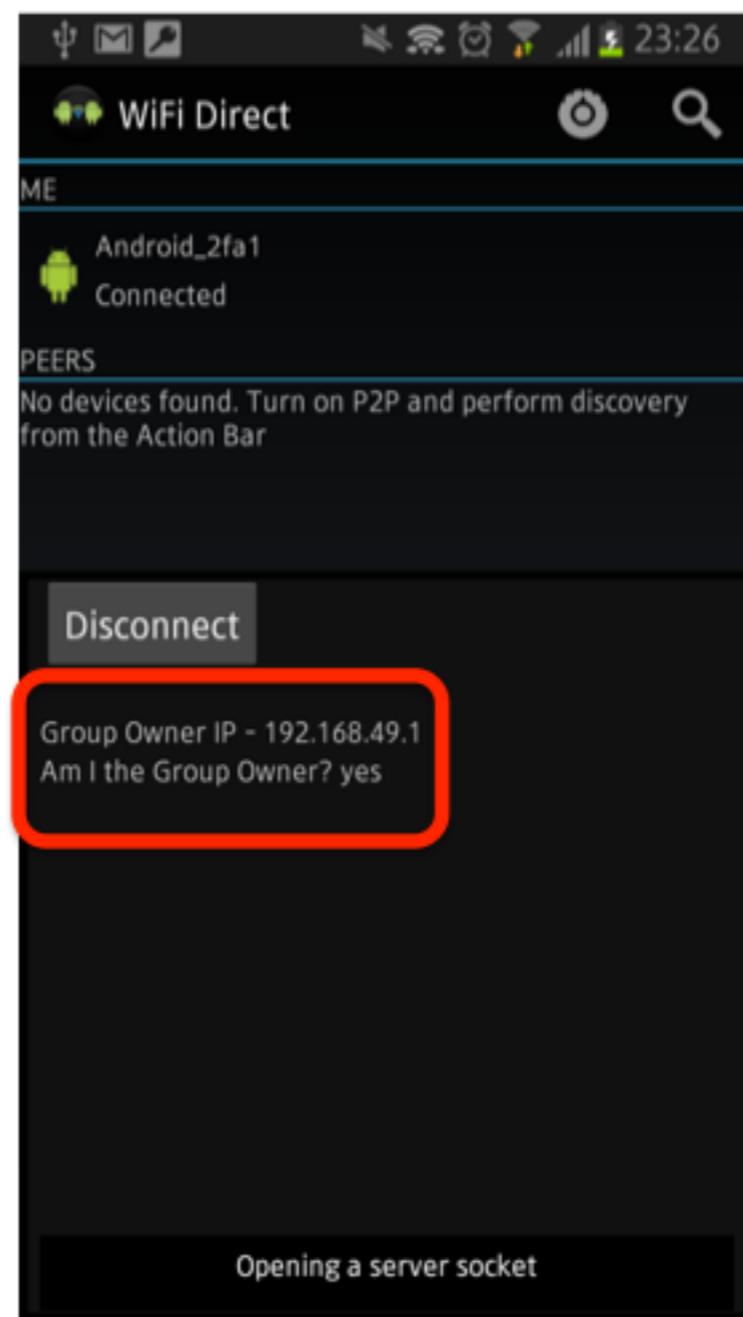


# WiFi Direct Demo

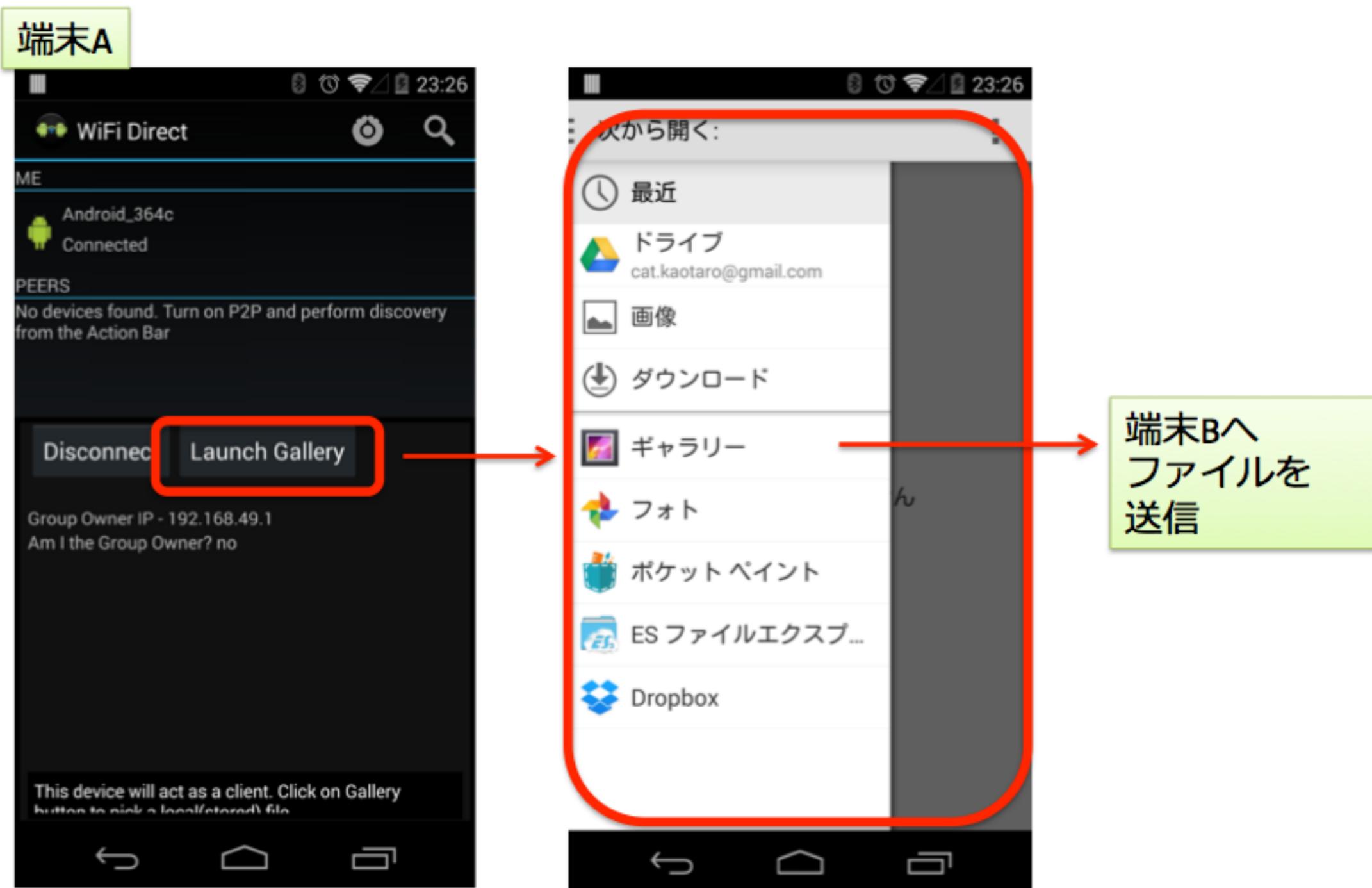
端末A



端末B



# WiFi Direct Demo



# Wi-Fi Direct

## アプリケーションの作成

1. ブロードキャストトレシーバーの作成と登録
2. 端末の発見
3. 端末の接続
4. 接続端末へのデータ送信  
(クライアント・サーバによるソケットを作成)

# AndroidでWiFi Direct

Android4.0以上

WifiP2pManager	WiFi Direct対応デバイス(P2Pデバイス)の情報、接続状態を管理するクラス
Channel	接続チャンネル
PeerListListener	接続可能なデバイス情報(リスト)を取得するためのリスナ
WifiP2pDeviceList	接続なデバイス情報(リスト)
WifiP2pDevice	接続先情報などのデバイス情報が格納されている
ConnectionInfoListener	接続状態の変化を取得するリスナ
WifiP2pInfo	接続状態が格納されている
WifiP2pConfig	接続設定情報

# AndroidでWiFi Direct

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が投げられるのでBroadcastReceiverで受信する必要がある。

<code>WIFI_P2P_STATE_CHANGED_ACTION</code>	WiFi Directの有効/無効状態が通知される
<code>WIFI_P2P_PEERS_CHANGED_ACTION</code>	デバイス情報の変更通知（通信可能なデバイス（Peersと呼ばれる）の発見・リストなど）。
<code>WIFI_P2P_CONNECTION_CHANGED_ACTION</code>	IPアドレスなどコネクション情報。通信状態の変更通知
<code>WIFI_P2P_THIS_DEVICE_CHANGED_ACTION</code>	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)

休憩(10分)

# ネットワークまとめ