11-Networking

Android & Networking

Le applicazioni scritte con componenti di rete sono molto più dinamiche e ricche di contenuti rispetto a quelle che non lo sono, infatti le capacità di rete possono essere utilizzate per una varietà di motivi:

- Per fornire contenuti freschi e aggiornati
- Per abilitare funzionalità di social networking
- Per alleggerire il carico di elaborazione tramite server ad alta potenza
- Per consentire l'archiviazione di dati oltre le possibilità dell'utente sul dispositivo

La connettività di rete sulla piattaforma Android è standardizzata, utilizzando una combinazione di soluzioni potenti e facili da usare, come `java.net`.

> Nota
> Android supporta API di rete tradizionali come Socket e SocketServer e HTTP (Hypertext Transfer Protocol)

Accessing the Web (HTTP)

Il modo più comune per trasferire dati da e verso la rete è utilizzare HTTP, utilizzandolo per incapsulare quasi qualsiasi tipo di dato e per proteggerlo con Secure Sockets Layer (SSL), il che può essere importante quando si trasmettono dati che rientrano nei requisiti di privacy.

> Note
> Le porte più comuni utilizzate da HTTP sono tipicamente aperte dalle reti telefoniche

La lettura dei dati dal web può essere davvero semplice, se si ha solo bisogno di leggere alcuni dati da un sito web e si ha a disposizione l'indirizzo web di quei dati, si può utilizzare la classe `URL` per leggere una quantità fissa di un file memorizzato su un server web.

Bisogna ricordare che poiché si sta lavorando con risorse di rete, gli errori possono essere più comuni e dovrebbero essere gestiti correttamente informando l'utente sullo stato del compito e mantenendo l'applicazione attiva e reattiva.

I problemi tipici includono:

- Il telefono potrebbe non avere copertura di rete
- Il server remoto potrebbe essere giù per un breve o lungo periodo

- L'URL potrebbe essere non valido

HTTP and Java URL

```
// Crea l'oggetto URL con l'url HTTP obbiettivo
Url urlObj = new URL("http://....");

// Apre un'oggetto inputStream per leggere i contenuti dall'url
// utilizzando un buffer
InputStream is = urlObj.openStream();
byte[] buffer = new byte[250];
int readSize = is.read(buffer);

Log.d(TAG,"readSize = " + readSize);
Log.d(TAG,"read content = " + new String(buffer));

// Chuide lo l'inputStream alla fine delle operazioni
is.close();
```

HttpURLConnection

L'approccio basato su URL potrebbe funzionare in alcune situazioni semplici (recuperare dati leggeri), ma in scenari complessi bisognerebbe utilizzare soluzioni avanzate per ottenere informazioni dettagliate sui dati disponibili prima di scaricarli e quindi utilizzare funzionalità API aggiuntive per costruire la richiesta. L'oggetto `HttpURLConnection` consente di ottenere alcune informazioni sulla risorsa referenziata dall'oggetto URL, inclusi lo stato HTTP e le informazioni dell'intestazione.

Alcune delle informazioni disponibili sono:

- Lunghezza del contenuto
- Tipo di contenuto
- Informazioni sulla data e l'ora (per verificare se i dati sono stati modificati dall'ultima volta che hai acceduto all'URL)

Utilizzando questa API si può determinare se un contenuto remoto è appropriato e quindi si può recuperarlo utilizzando un oggetto InputStream standard ottenuto tramite il metodo `http.getInputStream()`.

```
Url urlObj = new URL("http://....");

// Crea l'oggetto URL e apre la connessione HTTP
HttpURLConnection connection = (HttpURLConnection)url.openConnection();

// Legge informazioni utili riguardo i dati disponibili
Log.d(TAG,"lenght = " + connection.getContentLenght());
Log.d(TAG,"respCode = " + connection.getResponseCode());
Log.d(TAG,"contentType = " + connection.getContentType());
Log.d(TAG,"content = " + connection.getContent());


// Utilizza un InputSteam e un BufferReader per ottenere i dati dal server remoto
connection.connect();
// read the output from the server
reader = new BufferedReader(new InputStreamReader(connection.getInputStream()));

// Un oggetto StringBuilder è utilizzato per memorizzare i dati ottenuti
StringBuilder stringBuilder = new StringBuilder();
String line = null;

while ((line = reader.readLine()) != null) {
        stringBuilder.append(line + "\n");
}

return stringBuilder.toString();
```

Unirest

Unirest è un insieme di librerie HTTP disponibili in diversi linguaggi, sviluppate e mantenute da Mashape, che gestisce anche il Gateway API open-source Kong.

```
Unirest.post("http://httpbin.org/post")
               .queryString("name", "Mark")
               .field("last", "Polo")
               .asJson()
```

RetroFit

Retrofit trasforma un API HTTP in un'interfaccia Java.

```
public interface GitHubService {
        @GET("users/{user}/repos")
        Call<List<Repo>> listRepos(@Path("user") String user);
}
```

La classe `Retrofit` genera un implementazione dell'interfaccia.

```
Retrofit retrofit = new Retrofit.Builder()
        .baseUrl("https://api.github.com/")
        .build();
GitHubService service = retrofit.create(GitHubService.class);
```

Ogni `Call` proveniente dal service creato può fare richieste HTTP, sincrone o asincrone, ad un server web remoto.

```
Call<List<Repo>> repos = service.listRepos("octocat");
```

Android & Network Status

La Android SDK fornisce strumenti per raccogliere informazioni sullo stato corrente della rete, questo è utile per determinare se è disponibile una connessione di rete prima di tentare di utilizzare una risorsa di rete.

La classe `ConnectivityManager` fornisce diversi metodi per fare ciò, ad esempio è possibile determinare se la rete mobile o WiFi sono disponibili e collegate.

Un'istanza dell'oggetto `ConnectivityManager` viene recuperata utilizzando il metodo Context `getSystemService()`, quindi utilizzando quest'oggetto, è possibile recuperare oggetti `NetworkInfo` per `TYPE_WIFI` e `TYPE_MOBILE`.

Per consentire all'applicazione di leggere lo stato della rete, è necessario ottenere un'autorizzazione esplicita nel file `AndroidManifest.xml`.

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

Connectivity Manager

La classe che risponde alle interrogazioni sullo stato della connettività di rete, notifica anche le applicazioni quando cambia la connettività di rete.

> Nota
> Si può ottenere un'istanza di questa classe chiamando `Context.getSystemService(Context.CONNECTIVITY_SERVICE)`.

Le principali responsabilità di questa classe sono le seguenti:

- Monitorare le connessioni di rete (Wi-Fi, GPRS, UMTS, ecc.)
- Inviare broadcast intents quando cambia la connettività di rete
- Tentare di "passare" a un'altra rete quando la connettività a una rete viene persa
- Fornire un'API che consente alle applicazioni di interrogare lo stato a grana grossa o a grana fine delle reti disponibili

```
// Prende i servizi di sistema
ConnectivityManager cm = (ConnectivityManager)getSystemService(Context.CONNECTIVITY_SERVICE);

// Recupera gli oggetti NetworkInfo per la rete WiFi e la rete mobile.
NetworkInfo mobNi = cm.getNetworkInfo(ConnectivityManager.TYPE_MOBILE);
mobNi.isConnected();
mobNi.isAvailable();
NetworkInfo wifiNi = cm.getNetworkInfo(ConnectivityManager.TYPE_WIFI);
wifiNi.isConnected();
wifiNi.isAvailable();
```

NetworkInfo

La classe `NetworkInfo` descrive lo stato di un'interfaccia di rete di un tipo specifico (attualmente Mobile o Wifi), i principali metodi sono:

- `getState()`, che restituisce lo stato corrente a grana grossa della rete
- `getType()`, che restituisce il tipo di rete (attualmente mobile o Wi-Fi) a cui si riferiscono le informazioni in questo oggetto
- `isAvailable()`, che indica se la connettività di rete è possibile
- `isConnected()`, che indica se esiste una connettività di rete ed è possibile stabilire connessioni e trasferire dati

- `isConnectedOrConnecting()`, che indica se esiste una connettività di rete o se è in fase di stabilimento
- `isFailover()`, che indica se il tentativo corrente di connettersi alla rete è il risultato dal tentativo di failover, del ConnectivityManager, a questa rete a seguito di una disconnessione da un'altra rete (si verifica quando il dispositivo Android cerca di passare automaticamente da una rete a un'altra)
- `isRoaming()`, che indica se il dispositivo è attualmente in roaming su questa rete

Connectivity Manager & Broadcast Receiver

La classe `ConnectivityManager` permette di mandate intents di broadcast quando la connettività network cambia

```
registerReceiver(mConnReceiver,new IntentFilter(ConnectivityManager.CONNECTIVITY_ACTION));
```

Gli Intent Extra disponibili sono:

- `EXTRA_EXTRA_INFO`, la chiave di ricerca per una stringa che fornisce informazioni extra opzionali fornite sullo stato della rete
- `EXTRA_IS_FAILOVER`, la chiave di ricerca per un valore booleano che indica se l'evento di connessione riguarda una rete a cui il connectivity manager stava passando dopo una disconnessione su un'altra rete
- `EXTRA_NETWORK_INFO`, questa costante è deprecata, poiché NetworkInfo può variare in base all'UID, le applicazioni dovrebbero sempre ottenere le informazioni di rete tramite `getActiveNetworkInfo()` o `getAllNetworkInfo()`
- `EXTRA_NO_CONNECTIVITY`, la chiave di ricerca per un valore booleano che indica se c'è una completa mancanza di connettività, ovvero nessuna rete è disponibile
- `EXTRA_OTHER_NETWORK_INFO`, la chiave di ricerca per un oggetto NetworkInfo
- `EXTRA_REASON`, la chiave di ricerca per una stringa che indica il motivo per cui un tentativo di connessione a una rete è fallito

```
private BroadcastReceiver mConnReceiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {

                Log.d(TAG,"Received Connectivity Intent: " + intent);

                boolean noConnectivity =
intent.getBooleanExtra(ConnectivityManager.EXTRA_NO_CONNECTIVITY, false);
                String reason = intent.getStringExtra(ConnectivityManager.EXTRA_REASON);
                boolean isFailover =
intent.getBooleanExtra(ConnectivityManager.EXTRA_IS_FAILOVER, false);

                NetworkInfo currentNetworkInfo = (NetworkInfo)
intent.getParcelableExtra(ConnectivityManager.EXTRA_NETWORK_INFO);
                NetworkInfo otherNetworkInfo = (NetworkInfo)
intent.getParcelableExtra(ConnectivityManager.EXTRA_OTHER_NETWORK_INFO);
                Log.d(TAG,"No Connectivity: " + noConnectivity + " reason:" + reason + "
isFailover: " + isFailover);

                if(currentNetworkInfo != null)
                        Log.d(TAG,"Current Network Info: " + currentNetworkInfo.getTypeName()
                        + " ("+currentNetworkInfo.getType()+")
["+currentNetworkInfo.getState()+"]");
                if(otherNetworkInfo != null)
                        Log.d(TAG,"Other Network Info: " + otherNetworkInfo.getTypeName()
                        + " ("+currentNetworkInfo.getType()+")
["+currentNetworkInfo.getState()+"]");
        }
};
```

Working with WiFi

Il sensore WiFi può leggere lo stato della rete e determinare i punti di accesso wireless nelle vicinanze, la SDK fornisce un insieme di API per recuperare informazioni sulle reti WiFi disponibili al dispositivo e i dettagli della connessione alla rete WiFi.
Queste informazioni possono essere utilizzate per monitorare la potenza del segnale, trovare punti di accesso di interesse o eseguire azioni quando si è connessi a punti di accesso specifici.
La classe utilizzata per lavorare con il WiFi è `WifiManager` e può essere recuperata come un System Service tramite il metodo `Context.getSystemService()`.

Sono necessarie due autorizzazioni Android diverse e aggiuntive:

```
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE"/>
<!-- Più i permessi di posizione -->
```

WifiManager

La classe `WifiManager` fornisce l'API principale per gestire tutti gli aspetti della connettività Wi-Fi, per ottenere un'istanza di questa classe, chiamando `Context.getSystemService(Context.WIFI_SERVICE)`.

Consente di ottenere:

- La lista delle reti configurate può essere visualizzata e aggiornata, e gli attributi delle singole voci possono essere modificati
- La rete Wi-Fi, se presente, attualmente attiva, è possibile stabilire o interrompere la connettività ed interrogare informazioni dinamiche sullo stato della rete
- I risultati delle scansioni dei punti di accesso, contenenti informazioni sufficienti per prendere decisioni su quale punto di accesso connettersi
- Essa definisce i nomi di diverse azioni di intent che vengono trasmesse in broadcast in caso di qualsiasi tipo di cambiamento nello stato del Wi-Fi

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

// controlla se l'itnerfaccia wifi è abilitata
if(wm.isWifiEnabled()){
        Log.d(TAG,"Stopping WiFi");
        wm.disconnect();
        wm.setWifiEnabled(false);
}else{
        Log.d(TAG,"Enabling WiFi");
        wm.setWifiEnabled(true);
}
```

WifiManager and AccessPoint Scan

`WiFiManager` consente di eseguire una scansione manuale dei punti di accesso (Access Point o AP) disponibili vicino al dispositivo mobile utilizzando il metodo `wm.startScan()`.

Questo metodo restituisce immediatamente, e la disponibilità dei risultati viene resa nota successivamente mediante un evento asincrono inviato al completamento della scansione (disponibile tramite BroadcastReceiver).

Per ottenere l'elenco dei punti di accesso rilevati, è possibile utilizzare il metodo `wm.getScanResults()`, che restituisce un elenco di oggetti `ScanResult`.

> Nota
> La classe ScanResult descrive le informazioni su un punto di accesso rilevato, oltre agli attributi descritti qui, il suplicant tiene traccia degli attributi di qualità (quality), rumore (noise) e bitrate massimo (max bitrate)

Le informazioni disponibili sono:

- BSSID, l'indirizzo del punto di accesso
- SSID, il nome della rete
- capabilities, che descrive gli schemi di autenticazione, gestione delle chiavi e crittografia supportati dal punto di access.
- frequency, la frequenza in MHz del canale su cui il client sta comunicando con il punto di accesso
- level, il livello del segnale rilevato in dBm

WifiManager and Configured Networks

Se il `WifiManager` permette anche di recuperare la lista di tutti i network configurati dall'utente.

Mediante l'insieme di cifrature di gruppo supportate da una configurazione WiFii, è possibile ottenere un elenco di `WifiConfiguration` contenente le informazioni seguenti:

- networkId, il numero di ID che il supplicant utilizza per identificare questa voce di configurazione di rete
- SSID, della rete (nota, contiene virgolette aggiuntive - "" - che non sono presenti nel campo SSID di `ScanResult`)
- BSSID, quando impostata questa voce di configurazione di rete dovrebbe essere utilizzata solo quando si associa all'AP con il BSSID specificato
- priority, la priorità determina la preferenze date ad una rete da parte del `wpa_supplicant` quando sceglie un punto di accesso con cui associarsi
- allowedProtocols, l'insieme di protocolli di sicurezza supportati da questa configurazione
- allowedKeyManagement, l'insieme di protocolli di gestione delle chiavi supportati da questa configurazione
- allowedAuthAlgorithms, l'insieme di protocolli di autenticazione supportati da questa configurazione
- allowedPairwiseCiphers, l'insieme di cifre pairwise per WPA supportate da questa configurazione
- allowedGroupCiphers, l'insieme di cifre di gruppo supportate da questa configurazione

Wifi Network Configuration and Connection

WifiManager fornisce anche i metodi per creare e salvare una nuova WifiConfiguration sul dispositivo dell'utente.

```
WifiManager wifi = (WifiManager) getSystemService(Context.WIFI_SERVICE);

WifiConfiguration wc = new WifiConfiguration();
```

```
wc.SSID = "\"SSIDName\"";
wc.preSharedKey = "\"password\"";
wc.hiddenSSID = true;
wc.status = WifiConfiguration.Status.ENABLED;
wc.allowedGroupCiphers.set(WifiConfiguration.GroupCipher.TKIP);
wc.allowedGroupCiphers.set(WifiConfiguration.GroupCipher.CCMP);
wc.allowedKeyManagement.set(WifiConfiguration.KeyMgmt.WPA_PSK);
wc.allowedPairwiseCiphers.set(WifiConfiguration.PairwiseCipher.TKIP);
wc.allowedPairwiseCiphers.set(WifiConfiguration.PairwiseCipher.CCMP);
wc.allowedProtocols.set(WifiConfiguration.Protocol.RSN);

int res = wifi.addNetwork(wc);

Log.d("WifiPreference", "add Network returned " + res );

// Permeete di abilitare un network configurato
boolean b = wifi.enableNetwork(res, true);

Log.d("WifiPreference", "enableNetwork returned " + b );
```

WifiManager & Broadcast Receiver

La classe `WifiManager` consente di impostare un `BroadcastReceiver` dedicato per ottenere lo stato dell'interfaccia WiFi e i risultati delle scansioni dei punti di accesso, contenenti informazioni sufficienti per prendere decisioni su quale punto di accesso connettersi.

Le azioni di intent disponibili sono:

- `NETWORK_IDS_CHANGED_ACTION`, gli ID di rete dei network configurati potrebbero essere cambiati
- `NETWORK_STATE_CHANGED_ACTION`, azione di intento broadcast che indica che lo stato della connettività Wi-Fi è cambiato
- `RSSI_CHANGED_ACTION`, l'intensità del segnale è cambiata
- `SCAN_RESULTS_AVAILABLE_ACTION`, una scansione del punto di accesso è stata completata e i risultati sono disponibili dal supplicant
- `SUPPLICANT_CONNECTION_CHANGE_ACTION`, azione di intent broadcast che indica se è stata stabilita una connessione al supplicant, e che quindi è possibile eseguire operazioni Wi-Fi, o se la connessione al supplicant è stata persa
- `SUPPLICANT_STATE_CHANGED_ACTION`, azione di intent broadcast che indica che lo stato di stabilimento di una connessione a un punto di accesso è cambiato, un extra fornisce il nuovo stato del supplicant (`newSupplicantState`)
- `WIFI_STATE_CHANGED_ACTION`, azione di intent broadcast che indica se il Wi-Fi è stato abilitato/disabilitato, se è in fase di abilitazione/disabilitazione o sconosciuto

Ad esempio è possibile creare un IntentFilter inline per ricevere le notifiche seguenti:

```
IntentFilter intentFilter = new IntentFilter();
intentFilter(WifiManager.NETWORK_STATE_CHANGED_ACTION);
intentFilter(WifiManager.SUPPLICANT_STATE_CHANGED_ACTION);
intentFilter(WifiManager.NETWORK_IDS_CHANGED_ACTION);
intentFilter(WifiManager.SCAN_RESULTS_AVAILABLE_ACTION);
intentFilter(WifiManager.RSSI_CHANGED_ACTION);

wifiNetworkEventsReceiver = new WiFiNetworkChangesReceiver();
registerReceiver(wifiNetworkEventsReceiver, intentFilter);
```

Gli intent extra disponibili sono:

- `EXTRA_BSSID`
- `EXTRA_NETWORK_INFO`
- `EXTRA_NEW_RSSI`
- `EXTRA_NEW_STATE`
- `EXTRA_PREVIOUS_WIFI_STATE`
- `EXTRA_SUPPLICANT_CONNECTED`
- `EXTRA_SUPPLICANT_ERROR`
- `EXTRA_WIFI_INFO`
- `EXTRA_WIFI_STATE`

```
public class WiFiNetworkChangesReceiver extends BroadcastReceiver{
        @Override
        public void onReceive(Context context, Intent intent) {
                if(intent.getAction().equals(WifiManager.NETWORK_STATE_CHANGED_ACTION)){
                        NetworkInfo ni =
(NetworkInfo)intent.getParcelableExtra(WifiManager.EXTRA_NETWORK_INFO);
                        // ...
                }
                if(intent.getAction().equals(WifiManager.SUPPLICANT_STATE_CHANGED_ACTION)){
                        SupplicantState ss =
(SupplicantState)intent.getParcelableExtra(WifiManager.EXTRA_NEW_STATE);
                        // ...
                }
```

```
                if(intent.getAction().equals(WifiManager.NETWORK_IDS_CHANGED_ACTION)){
                        Log.d(TAG, "WiFiNetworkChangesReceiver: action " +
intent.getAction());
                }
                if( intent.getAction().equals(WifiManager.SCAN_RESULTS_AVAILABLE_ACTION)){
                        Log.d(TAG, "WiFiNetworkChangesReceiver: action " +
intent.getAction());
                        // ...
                }
                if(intent.getAction().equals(WifiManager.RSSI_CHANGED_ACTION)){
                        int newRssi = intent.getIntExtra(WifiManager.EXTRA_NEW_RSSI, 0);
                        // ...
                }
        }
}
```

Android & Telephony Utilities

La SDK fornisce una serie di utilità utili per consentire alle applicazioni di integrare le funzionalità telefoniche disponibili sul dispositivo. Queste informazioni sono davvero importanti per aggiungere funzionalità all'applicazione, ad esempio, l'applicazione non dovrebbe interrompere una conversazione telefonica o fermare la riproduzione di media quando arriva una chiamata in entrata.

Per evitare questo tipo di comportamento, l'applicazione deve conoscere qualcosa riguardo a ciò che l'utente sta facendo per reagire correttamente, un altro esempio è quello di adattare le funzionalità dell'applicazione in base al tipo di operatore telefonico dell'utente, ad esempio estrarre informazioni dalla SIM come IMSI, nome dell'operatore e posizioni delle celle.

Telephony Manager

L'oggetto `TelephonyManager` è l'interfaccia di piattaforma per accedere alle informazioni sullo stato del telefono e fornisce accesso alle informazioni sui servizi di telefonia sul dispositivo.
Le applicazioni possono utilizzare i metodi di questa classe per determinare i servizi e gli stati di telefonia, nonché per accedere a alcuni tipi di informazioni degli iscritti.

> Nota
> Le applicazioni possono anche registrare un listener per ricevere notifiche dei cambiamenti dello stato di telefonia

Non bisogna istanziare una classe `TelephonyManager` direttamente, invece bisogna ottienere un riferimento a un'istanza tramite `Context.getSystemService(Context.TELEPHONY_SERVICE)`.

> Nota
> L'accesso ad alcune informazioni di telefonia è protetto da autorizzazioni, l'applicazione non può accedere alle informazioni protette a meno che non abbia le autorizzazioni appropriate dichiarate nel suo file manifesto.
> Dove sono richieste le autorizzazioni, sono indicate nei metodi attraverso i quali si accede alle informazioni protette

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

TelephonyManager & BroadcastReceiver

È possibile registrarsi per ricevere aggiornamenti utilizzando l'azione dell'intent `ACTION_PHONE_STATE_CHANGED`, gli extra disponibili sono:

- `EXTRA_STATE`
- `EXTRA_INCOMING_NUMBER`
- `EXTRA_STATE_IDLE`
- `EXTRA_STATE_OFFHOOK`
- `EXTRA_STATE_RINGING`

```
public class MyPhoneReceiver extends BroadcastReceiver {
        @Override
        public void onReceive(Context context, Intent intent) {
                Bundle extras = intent.getExtras();
                if (extras != null) {
                        String state = extras.getString(TelephonyManager.EXTRA_STATE);
                        Log.w("DEBUG", state);
                        if (state.equals(TelephonyManager.EXTRA_STATE_RINGING)) {
                                String phoneNumber =
extras.getString(TelephonyManager.EXTRA_INCOMING_NUMBER);
```

```
                                Log.w("DEBUG", phoneNumber);
                        }
                }
        }
}
```

TelephonyManager

```
TelephonyManager tm = (TelephonyManager) getSystemService(Context.TELEPHONY_SERVICE);
if(tm != null){
        int simState = tm.getSimState();
        //Checks the status of the SIM CARD
        if(simState != TelephonyManager.SIM_STATE_ABSENT){
                //Returns the serial number of the SIM, if applicable. Return null if it is
unavailable.
                //Requires Permission: READ_PHONE_STATE
                String simSerialNumber = tm.getSimSerialNumber();
                //Returns the MCC+MNC (mobile country code + mobile network code) of the
provider of the SIM.
                String simOperatorCode = tm.getSimOperator();
                //Returns the Service Provider Name (SPN).
                String simOperatorName = tm.getSimOperatorName();
                //Returns the alphabetic name of current registered operator.
                String operatorName = tm.getNetworkOperatorName();
                //Returns the numeric name (MCC+MNC) of current registered operator.
                String operatorCode = tm.getNetworkOperator();
                //Returns the ISO country code equivalent of the current registered
operator's MCC (Mobile Country Code).
                String operatorISO = tm.getNetworkCountryIso();
                //Returns the unique subscriber ID, for example, the IMSI for a GSM phone.
                String subscriberId = tm.getSubscriberId();
        }
}
```

Bluetooth

La piattaforma Android include il supporto per lo stack di rete Bluetooth, che consente a un dispositivo di scambiare dati in modalità wireless con altri dispositivi Bluetooth.
Il framework delle applicazioni fornisce accesso alle funzionalità Bluetooth tramite le API Bluetooth di Android che consentono alle applicazioni di connettersi in modalità wireless ad altri dispositivi Bluetooth, abilitando le funzionalità wireless point-to-point e multipoint.

Utilizzando le API Bluetooth, un'applicazione Android può svolgere le seguenti funzioni:

- Scansionare altri dispositivi Bluetooth
- Interrogare l'adattatore Bluetooth locale per dispositivi Bluetooth accoppiati
- Stabilire canali RFCOMM
- Connettersi ad altri dispositivi tramite il discovery dei servizi
- Trasferire dati da e verso altri dispositivi
- Gestire connessioni multiple

Il package `android.bluetooth` contiene tutte le API bluetooth disponibili, le classi principali sono:

- `BluetoothAdapter`, rappresenta l'adattatore Bluetooth locale (Bluetooth radio), il `BluetoothAdapter` è il punto di ingresso per tutte le interazioni Bluetooth. Utilizzandolo, si può scoprire altri dispositivi Bluetooth, interrogare un elenco di dispositivi accoppiati (paired), istanziare un `BluetoothDevice` utilizzando un indirizzo MAC noto e creare un `BluetoothServerSocket` per ascoltare le comunicazioni da altri dispositivi
- `BluetoothDevice`, rappresenta un dispositivo Bluetooth remoto, si utilizza per richiedere una connessione con un dispositivo remoto tramite un `BluetoothSocket` o per ottenere informazioni sul dispositivo (il nome, l'indirizzo, la classe e lo stato di accoppiamento)
- `BluetoothSocket`, rappresenta l'interfaccia per un socket Bluetooth (simile a un socket TCP), questo è il punto di connessione che consente ad un'applicazione di scambiare dati con un altro dispositivo Bluetooth tramite `InputStream`/`OutputStream`
- `BluetoothServerSocket`, rappresenta un socket server aperto che ascolta le richieste in ingresso (simile a un ServerSocket TCP), per connettere due dispositivi Android, un dispositivo deve aprire un server socket con questa classe. Quando un dispositivo Bluetooth remoto invia una richiesta di connessione a questo dispositivo, il `BluetoothServerSocket` restituirà un `BluetoothSocket` connesso una volta accettata la connessione
- `BluetoothClass`, descrive le caratteristiche generali e le capacità di un dispositivo Bluetooth, si tratta di un insieme di proprietà in sola lettura che definisce le classi principali e minori del dispositivo e i suoi servizi. Tuttavia, questo non descrive in modo affidabile tutti i profili e i servizi Bluetooth supportati dal dispositivo, ma è utile come suggerimento per il tipo di dispositivo
- `BluetoothProfile`, un'interfaccia che rappresenta un profilo Bluetooth, che è una specifica di interfaccia wireless per la comunicazione basata su Bluetooth tra dispositivi. Un esempio è il profilo Hands-Free, per ulteriori discussioni sui profili, consulta "Lavorare con i profili" (Working with Profiles)

Bluetooth Permission

Per utilizzare le funzionalità Bluetooth nella tua applicazione, è necessario dichiarare almeno una delle due autorizzazioni Bluetooth:

- `BLUETOOTH`
- `BLUETOOTH_ADMIN`.

Bisogna richiedere l'autorizzazione `BLUETOOTH` per eseguire qualsiasi comunicazione Bluetooth come:

- Richiedere una connessione
- Accettare una connessione
- Trasferire dati.

Devi richiedere l'autorizzazione `BLUETOOTH_ADMIN` per:

- Avviare la scoperta dei dispositivi
- Manipolare le impostazioni Bluetooth

La maggior parte delle applicazioni necessita del secondo tipo di autorizzazione unicamente per la capacità di scoprire i dispositivi Bluetooth locali.
Le altre capacità concesse da questa autorizzazione non dovrebbero essere utilizzate, a meno che l'applicazione non sia un "power manager" che modificherà le impostazioni Bluetooth su richiesta dell'utente.

> Nota
> Se si utilizza l'autorizzazione `BLUETOOTH_ADMIN` e bisogna avere anche l'autorizzazione `BLUETOOTH`

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

Setting Up Bluetooth

Il `BluetoothAdapter` è richiesto per ogni e per tutte le attività bluetooth.
Per ottenere il `BluetoothAdapter`, chiama il metodo statico `getDefaultAdapter()` che restituisce un BluetoothAdapter che rappresenta l'adattatore Bluetooth del dispositivo (Bluetooth radio).

C'è un solo `BluetoothAdapter` per l'intero sistema e la tua applicazione può interagire con esso utilizzando questo oggetto, se `getDefaultAdapter()` restituisce `null`, significa che il dispositivo non supporta il Bluetooth e qui ci si ferma

```
BluetoothAdapter mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
if (mBluetoothAdapter == null) {
        // Device does not support Bluetooth
}
if (!mBluetoothAdapter.isEnabled()) {
        Intent enableBtIntent = new
        Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
        startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT);
}
```

Finding Devices/Querying paired devices

Prima di avviare la scoperta dei dispositivi, vale la pena interrogare l'insieme di dispositivi accoppiati per verificare se il dispositivo desiderato è già noto.
Per fare ciò bisogna chiamare `getBondedDevices()`, questo metodo restituirà un insieme (`Set`) di `BluetoothDevice` che rappresenta i dispositivi accoppiati.

```
Set<BluetoothDevice> pairedDevices = mBluetoothAdapter.getBondedDevices();
// If there are paired devices
if (pairedDevices.size() > 0) {
        // Loop through paired devices
        for (BluetoothDevice device : pairedDevices) {
                // Add the name and address to an array adapter to show in a ListView
                mArrayAdapter.add(device.getName() + "\n" + device.getAddress());
        }
}
```

Discovering Devices

Per avviare la ricerca dei dispositivi bisogna chiamare `startDiscovery()`.

> Nota
> Il processo è asincrono e il metodo restituirà immediatamente un booleano che indica se la ricerca è stata avviata con successo.

Il processo di ricerca di solito prevede una scansione di indagine di circa 12 secondi, seguita da una scansione di pagina di ciascun dispositivo trovato per recuperare il suo nome Bluetooth.

L'applicazione deve registrare un `BroadcastReceiver` per l'intent `ACTION_FOUND` al fine di ricevere informazioni su ciascun dispositivo scoperto, per ognuno il sistema invierà l'intent `ACTION_FOUND`. Questo intent contiene i campi extra `EXTRA_DEVICE` ed `EXTRA_CLASS`, che contengono rispettivamente un `BluetoothDevice` e una `BluetoothClass`.

Ad esempio, ecco come si può registrarsi per gestire la ricezione della trasmissione quando i dispositivi vengono scoperti:

```
// Create a BroadcastReceiver for ACTION_FOUND
private final BroadcastReceiver mReceiver = new BroadcastReceiver() {
        public void onReceive(Context context, Intent intent) {
                String action = intent.getAction();
                // When discovery finds a device
                if (BluetoothDevice.ACTION_FOUND.equals(action)) {
                        // Get the BluetoothDevice object from the Intent
                        BluetoothDevice device =
intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
                        // Add the name and address to an array adapter to show in a ListView
                        mArrayAdapter.add(device.getName() + "\n" + device.getAddress());
                }
        }
};
// Register the BroadcastReceiver
IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION_FOUND);
registerReceiver(mReceiver, filter); // Don't forget to unregister during onDestroy
```

Bluetooth Connecting Devices

Quando si desidera connettere due dispositivi, uno dei due deve agire come server tenendo aperto un `BluetoothServerSocket`, con lo scopo di ascoltare le richieste di connessione in ingresso e, quando una di esse viene accettata, fornire un `BluetoothSocket` connesso.
Una volta ottenuto il `BluetoothSocket` dal `BluetoothServerSocket`, il `BluetoothServerSocket` può (e dovrebbe) essere scartato, a meno che si desidera accettare ulteriori connessioni.

Per avviare una connessione con un dispositivo remoto (un dispositivo che tiene aperto un server socket):

1. Bisogna ottenere un oggetto `BluetoothDevice` che rappresenta il dispositivo remoto
2. Bisogna utilizzare il `BluetoothDevice` per ottenere un `BluetoothSocket` e avviare la connessione.

Bluetooth Server

```
private class AcceptThread extends Thread {
        private final BluetoothServerSocket mmServerSocket;
        public AcceptThread() {
                BluetoothServerSocket tmp = null;
                try {
                        // MY_UUID is the app's UUID string, also used by the client code
                        tmp = mBluetoothAdapter.listenUsingRfcommWithServiceRecord(NAME,
MY_UUID);
                } catch (IOException e) { }
                mmServerSocket = tmp;
        }
        public void run() {
                BluetoothSocket socket = null;
                while (true) {
                        try {
                                socket = mmServerSocket.accept();
                        } catch (IOException e) {
                                break;
                        }
                        if (socket != null) {
                                manageConnectedSocket(socket);
                                mmServerSocket.close();
                                break;
                        }
                }
        }
        public void cancel() {
                try {
                        mmServerSocket.close();
                } catch (IOException e) { }
        }
}
```

Bluetooth Client

```
private class ConnectThread extends Thread {
        private final BluetoothSocket mmSocket;
        private final BluetoothDevice mmDevice;
        public ConnectThread(BluetoothDevice device) {
                BluetoothSocket tmp = null;
                mmDevice = device;
                try{
                        tmp = device.createRfcommSocketToServiceRecord(MY_UUID);
                }catch(IOException e) { }

                mmSocket = tmp;
        }
        public void run() {
                // Cancel discovery because it will slow down the connection
                mBluetoothAdapter.cancelDiscovery();
                try {
                        // Connect the device through the socket. This will block
                        // until it succeeds or throws an exception
                        mmSocket.connect();
                } catch (IOException connectException) {
                        // Unable to connect; close the socket and get out
                        try {
                                mmSocket.close();
                        } catch (IOException closeException) { }
                        return;
                }
                // Do work to manage the connection (in a separate thread)
                manageConnectedSocket(mmSocket);
        }
        public void cancel() {
                try {
                        mmSocket.close();
                } catch (IOException e) { }
        }
}
```