Лабораторная работа №5

ПЕРЕДАЧА ДАННЫХ ПО БЕСПРОВОДНЫМ СЕТЯМ

**Цель работы**: получить навыки работы с беспроводными сетями на мобильных устройствах Android.

**ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Bluetooth – одна из наиболее старых технологий беспроводной связи между устройствами, которая использовалась ещё на КПК. Это сетевой протокол, созданный для связи в условиях низкой пропускной способности и ограниченного радиуса действия. Она пришла на смену протоколу на инфракрасной связи. Сейчас Bluetooth сдаёт свои позиции под напором различных облачных интернет-решений, но тем не менее ещё широко используется пользователями мобильных телефонов и других устройств.

Библиотеки для работы с Bluetooth, которые стали доступны в Android только с версии 2.0 (SDK API level 5), предоставляют разработчику богатые возможности. Работа с Bluetooth состоит из четырех этапов:

* установка настроек bluetooth адаптера,
* поиск доступных для соединения устройств,
* установка соединения,
* передача данных.

**Обзор Android Bluetooth API**

Bluetooth API располагается в пакете *android.bluetooth*. Ниже представлены некоторые из входящих в его состав классов:

* *BluetoothAdapter* – отвечает за работу с установленным в телефоне Bluetooth модулем. Экземпляр этого класса есть в любой программе, использующей bluetooth. В состав этого класса входят методы, позволяющие производить поиск доступных устройств, запрашивать список подключенных устройств, создавать экземпляр класса BluetoothDevice на основании известного MAC адреса и создавать BluetoothServerSocket для ожидания запроса на соединение от других устройств.
* *BluetoothDevice* – класс, ассоциирующийся с удаленным Bluetooth устройством. Экземпляр этого класса используется для соединения через BluetoothSocket или для запроса информации об удаленном устройстве (имя, адресс, класс, состояние).
* *BluetoothSocket* – интерфейс для Bluetooth socket, аналогичный TCP сокетам. Это точка соединения, позволяющая обмениваться данными с удаленным устройством через InputStream и OutputStream.
* *BluetoothServerSocket* – представляет открытый сокет сервера, готовый к обработке входящего запроса. Для того чтобы соединить два Android устройства, одно из них должно открыть сокет с помощью этого класса. Когда удаленное устройство пошлет запрос на соединение, BluetoothServerSocket вернет объект BluetoothSocket.
* *BluetoothClass* – описывает основные параметры Bluetooth модуля. Объект этого класса доступен только в режиме чтения и может быть полезен при определении типа устройства.
* *BluetoothProfile* – интерфейс, представляющий Bluetooth профиль (спецификацию беспроводного интерфейса для соединения устройств через Bluetooth). Примером профиля может служить Hands-Free profile, определяющий порядок работы с беспроводной гарнитурой.

**Установка настроек Bluetooth адаптера из Android**

Для использования в своей программе возможностей Bluetooth модуля, подключим соответствующий пакет API и зададим необходимые разрешения:

import android.bluetooth.\*;

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

<uses-permission android:name="android.permission.BLUETOOTH\_ADMIN" />

Прежде чем начинать работу с модулем Bluetooth, необходимо проверить его наличие на устройстве, после чего убедиться, что модуль, доступен (не отключен), в противном случае можно предложить пользователю включить его.

BluetoothAdapter Bluetooth = BluetoothAdapter.getDefaultAdapter();

if(bluetooth!=null) // на устройстве есть модуль bluetooth

{ // Bluetooth включен?

if (bluetooth.isEnabled()) {

// Bluetooth включен. Дальше работаем с модулем.

}

else {

// Bluetooth выключен. Предложим пользователю включить его:

Intent enableBtIntent = new Intent(BluetoothAdapter.ACTION\_REQUEST\_ENABLE);

startActivityForResult(enableBtIntent, REQUEST\_ENABLE\_BT);

}

}

Если пользователь согласится на включение адаптера, в переменную enableBtIntent будет записано значение RESULT\_OK. В противном случае – RESULT\_CANCELED.

После того, как все проверки выполнены, можно приступать к работе. Чтобы отобразить имя и адрес нашего адаптера, необходимо вызвать методы getName() и getAddress(). Изменить имя устройства можно с помощью метода setName().

\_ String mydeviceaddress= bluetooth.getAddress();

\_ String mydevicename= bluetooth.getName();

\_ status= mydevicename+" : "+ mydeviceaddress;

\_ bluetooth.setName("AndroidNewName");

**Организация поиска доступных bluetooth устройств**

Класс BluetoothAdapter позволяет найти удаленное bluetooth устройство, для этого требуется запустить сканирование или запросить список спаренных устройств. При сканировании осуществляется поиск доступных bluetooth модулей вокруг пользователя. Если в радиусе действия окажется устройство с разрешенным bluetooth, оно отправит в ответ на запрос некоторую информацию о себе: имя, класс, свой уникальный MAC адрес. На основе этой информации можно организовать соединение и передачу данных.

Сразу после установки соединения с удаленным устройством пользователю будет автоматически показан запрос на соединение. В случае положительного ответа полученная информация (имя, класс и MAC адрес) сохраняется и может затем использоваться через bluetooth API. Так при следующем сеансе связи с данным удаленным устройством уже не придется проводить сканирование, поскольку необходимый MAC адрес уже будет занесен в базу телефона и его можно просто выбрать из списка спаренных устройств.

Необходимо различать понятие спаренных и соединенных устройств. Спаренные устройства просто знают о существовании друг друга, имеют ссылку-ключ, которую могут использовать для аутентификации и способны создать шифрованное соединение друг с другом. Соединенные устройства разделяют один радиоканал и могут передавать данные друг другу. Текущая реализация bluetooth API требует, чтобы устройства были спарены перед соединением (спаривание выполняется автоматически, когда начинается шифрованное соединение через Bluetooth API).

**Запрос на соединение со спаренным устройством**

Прежде чем приступать к поиску устройств вокруг имеет смысл показать пользователю список уже известных системе устройств, так как требуемый телефон может оказаться в этом списке. Метод getBondedDevices() возвращает множество (Set) устройств BluetoothDevice, с которыми уже происходило соединение. Можно показать пользователю этот список, например, с помощью ArrayAdapter:

Set<BluetoothDevice> pairedDevices= mBluetoothAdapter.getBondedDevices();

// Если список спаренных устройств не пуст

if(pairedDevices.size()>0) {

// проходимся в цикле по этому списку

for(BluetoothDevice device: pairedDevices) {

// Добавляем имена и адреса в mArrayAdapter, чтобы показать через ListView

mArrayAdapter.add(device.getName()+"\n"+ device.getAddress());

}

}

Для инициализации соединения нужно знать MAC адрес устройства. В приведенном выше примере эти адреса заносятся в Arrayadapter и показываются пользователю. При необходимости можно будет легко достать любой адрес из этого списка.

**Поиск устройств**

Можно начать сканирование радиодиапазона на предмет наличия доступных устройств, вызвав метод startDiscovery(). Сканирование происходит в отдельном асинхронном потоке. Метод возвращает true, если запуск сканирования прошел успешно. Чтобы получить информацию о найденных устройствах, приложение должно зарегистрировать BroadcastReceiver для интента ACTION\_FOUND. Этот интент вызывается для каждого найденного устройства. Интент содержит дополнительные поля EXTRA\_DEVICE и EXTRA\_CLASS, которые содержат объекты BluetoothDevice и BluetoothClass соответственно.

// Создаем BroadcastReceiver для ACTION\_FOUND

private final BroadcastReceiver mReceiver=new BroadcastReceiver(){

public void onReceive(Context context, Intent intent){

String action= intent.getAction();

// Когда найдено новое устройство

if(BluetoothDevice.ACTION\_FOUND.equals(action)){

// Получаем объект BluetoothDevice из интента

BluetoothDevice device= intent.getParcelableExtra(BluetoothDevice.EXTRA\_DEVICE);

//Добавляем имя и адрес в array adapter, чтобы показвать в ListView

mArrayAdapter.add(device.getName()+"\n"+ device.getAddress());

}

}

};

// Регистрируем BroadcastReceiver

IntentFilter filter=new IntentFilter(BluetoothDevice.ACTION\_FOUND);

registerReceiver(mReceiver, filter);

Как только найдено подходящее устройство, следует остановить процесс сканирования. Это можно сделать с помощью метода cancelDiscovery().

**Включение Bluetooth из приложения**

Позволить пользователям сделать телефон видимым для других устройств можно с помощью метода startActivityForResult(Intent, int), вызвав интент ACTION\_REQUEST\_DISCOVERABLE. В результате пользователю будет показано системное окно с запросом на перевод телефона в режим bluetooth видимости. По умолчанию этот режим включается на 120 секунд. Это время можно изменить с передав интенту дополнительный параметр EXTRA\_DISCOVERABLE\_DURATION. Максимально доступное время составляет 3600 секунд. Значение 0 переводит bluetooth модуль вашего телефона в режим постоянной видимости.

Intent discoverableIntent = new Intent(BluetoothAdapter.ACTION\_REQUEST\_DISCOVERABLE);

discoverableIntent.putExtra(BluetoothAdapter.EXTRA\_DISCOVERABLE\_DURATION, 300);

startActivity(discoverableIntent);

Если пользователь подтвердит переход в появившемся диалоговом окне, телефон будет переведен в режим видимости, и будет вызван callback-метод onActivityResult() . В качестве результата методу будет передано число секунд, которое устройство будет видимым. Если пользователь откажется от предложения или произойдет ошибка, то интент вернет код RESULT\_CANCELED. Перевод устройства в режим видимости автоматически включает bluetooth адаптер.

**Соединение устройств**

Чтобы соединить два устройства, необходимо написать серверную и клиентскую часть кода. Одно из устройств должно открыть серверный сокет, а второе – инициализировать соединение, используя MAC адрес сервера. Сервер и клиент считаются соединенными, когда они оба имеют активный BluetoothSocket на одном и том же RFCOMM (Radio Frequency Communication) канале. После этого они могут поучать и отправлять потоки данных. Сервер и клиент по-разному получают требуемый BluetoothSocket. Сервер получает его, когда входящее соединение принято. Клиент – когда открывает RFCOMM для сервера.

**Сервер**

При соединении устройств одно из них должно вести себя как сервер, то есть удерживать открытый BluetoothServerSocket. Цель сервера – ждать запроса на входящее соединение, и когда оно подтверждено, создать BluetoothSocket. После этого BluetoothServerSocket можно закрыть. Рассмотрим поэтапно процедуру соединения с точки зрения сервера:

1. Получить BluetoothServerSocket вызвав метод listenUsingRfcommWithServiceRecord (String, UUID). Первый параметр метода представляет собой идентификационное имя вашего сервиса. Система автоматически добавит его в базу Service Discovery Protocol (SDP). Обычно в качестве этого параметра просто указывают название приложения. Второй параметр также идентифицирует сервис. Этот параметр используется клиентом при подтверждении соединения.
2. Начинаем прослушивать запрос на соединения через метод accept(). Это блокирующий метод, который возвращает результат или когда соединение подтверждено, или когда произошло исключение. Соединение считается подтвержденным, когда удаленное устройство пошлет запрос на соединение с UUID, указанным при регистрации серверного сокета. В случае успеха, accept() возвращает настроенный на соединение BluetoothSocket.
3. Чтобы принять дополнительное соединение, вызовите метод close(). Это приведет к освобождению сокета и всех его ресурсов, но не закроет соединенный BluetoothSocket. В отличие от TCP/IP, RFCOMM позволяет работать только с одним клиентом на канале, поэтому в большинстве случаев имеет смысл вызывать метод close() срезу же после установки принятия сокета.

Поскольку метод accept() является блокирующим, его не стоит вызывать из потока главной деятельности, поскольку это приведет к подвисанию интерфейса. Обычно вся работа с BluetoothServerSocket и BluetoothSocket выполняется в отдельном потоке. Чтобы прекратить выполнение метода accept(), нужно вызвать метод close() для BluetoothServerSocket (или BluetoothSocket) из любого другого потока приложения.

**Клиент**

Для инициализации соединения с удаленным устройствам (устройством, которое держит открытым серверный сокет) необходимо получить объект BluetoothDevice, содержащий информацию о нем. Этот объект используется для получения BluetoothSocket и инициализации соединения. Опишем процедуру соединения:

1. Получим BluetoothSocket вызвав метод BluetoothDevice.createRfcommSocketToServiceRecord(UUID). Значение параметра UUID должно совпадать с значением, указанным при вызове listenUsingRfcommWithServiceRecord сервера.
2. Инициализируем соединение, вызвав метод connect(). После вызова этого метода система будет выполнять SDP поиск на удаленном устройстве, чтобы сопоставить UUID. В случае успеха, при условии подтверждения запроса со стороны сервера, будет открыт RFCOMM канал. Если по каким-то причинам соединение сорвется или выйдет timeout (около 12 секунд), будет сгенерировано исключение.

Как и в случае с accept, метод connect() следует выполнять в отдельном потоке, в противном случае может произойти подвисание интерфейса.

**Передача данных**

После успешного соединения, каждое из соединенных устройств имеет объект BluetoothSocket с помощью которого легко реализовать передачу/прием данных:

1. С помощью методов getInputStream() и getOutputStream() можно получить объекты InputStream и OutputStream, управляющие передачей через сокет.
2. После чего читать и писать данные в поток с помощью методов read(byte[]) и write(byte[]).

Необходимо использовать отдельный поток для чтения и записи данных. Это важно, поскольку методы read(byte[]) и write(byte[]) являются блокирующими и их вызов в основном потоке может парализовать программу. Главный цикл в этом отдельном потоке должен считывать данные из InputStream. Для записи в OutputStream имеет смысл создать отдельный public метод. Ниже приведен пример потока, реализующий описанный выше механизм работы.

privateclass ConnectedThreadextends Thread{

private final BluetoothSocket mmSocket;

private final InputStream mmInStream;

private final OutputStream mmOutStream;

public ConnectedThread(BluetoothSocket socket){

\_ mmSocket= socket;

\_ InputStream tmpIn=null;

\_ OutputStream tmpOut=null;

\_ // Получить входящий и исходящий потоки данных

\_ try{

\_ tmpIn= socket.getInputStream();

\_ tmpOut= socket.getOutputStream();

\_ } catch(IOException e){}

\_ mmInStream= tmpIn;

\_ mmOutStream= tmpOut;

}

public void run(){

byte[] buffer=new byte[1024];// буферный массив

int bytes;// bytes returned from read()

// Прослушиваем InputStream пока не произойдет исключение

while(true){

\_ try{

\_ // читаем из InputStream

\_ bytes= mmInStream.read(buffer);

\_ // посылаем прочитанные байты главной деятельности

\_ mHandler.obtainMessage(MESSAGE\_READ, bytes,-1, buffer).sendToTarget();

\_ } catch(IOException e){ break; }

}

}

/\* Вызываем этот метод из главной деятельности, чтобы отправить данные

удаленному устройству \*/

public void write(byte[] bytes){

try{

\_ mmOutStream.write(bytes);

} catch(IOException e){}

}

/\* Вызываем этот метод из главной деятельности, чтобы разорвать соединение \*/

public void cancel(){

try{

\_ mmSocket.close();

} catch(IOException e){}

}

В конструкторе создаются объекты для работы с потоками данных. После того, как прочитан очередной блок данных из входящего потока, он посылается в главную деятельность посредствам вызова метода Handler родительского класса. Для отправки данных из главной деятельности просто вызывается метод write(). Внутри этого публичного метода происходит вызов write(byte[]). Метод close() также можно вызвать из главной деятельности. Он разрывает соединение.

**Вопросы для самопроверки**

1. Для чего используется технология Bluetooth? Каковы этапы работы с этой технологией?
2. Какие классы пакета android.bluetooth вам известны? В чем их назначение?
3. Какие разрешения могут понадобиться при работе с Bluetooth? Какие проверки следует провести перед началом работы?
4. Как можно получить список уже известных системе устройств?
5. Какие методы могут быть использованы при поиске доступных устройств? Как происходит процесс сканирования?
6. Как можно сделать телефон видимым для других телефонов?
7. Какие шаги необходимо сделать с точки зрения сервера для соединения с другим устройством? А с точки зрения клиента?
8. Каким образом происходит передача данные? Какие методы в этом задействованы?

**Дополнительные материалы**

1. Developers.Android.API Guides – Bluetooth – [Электронный ресурс] – <https://developer.android.com/guide/topics/connectivity/bluetooth.html> (дата обращения 28.02.2017).
2. Android Bluetooth Example – [Электронный ресурс] – <https://examples.javacodegeeks.com/android/core/bluetooth/bluetoothadapter/android-bluetooth-example/> (дата обращения 28.02.2017).
3. Android – Bluetooth – [Электронный ресурс] – <https://www.tutorialspoint.com/android/android_bluetooth.htm> (дата обращения 28.02.2017).
4. Описание работы приложения bluetooth для Андроид – [Электронный ресурс] – <http://interosite.ru/articles/bluetooth-for-android> (дата обращения 28.02.2017)

**Задание**

Создать приложение, которое получает и принимает файл, используя Bluetooth.

**Варианты заданий**

1. Позволить пользователю изменять имя устройства.
2. Отображать пользователю имя, класс и уникальный MAC адрес найденных в радиусе действия устройств.
3. Позволить пользователю сделать устройство видимым постоянно.
4. Позволить пользователю задать время видимости устройства для других телефонов.
5. При отсутствии других устройств в радиусе действия прекращать сканирование и сообщать об этом пользователю через заданный период времени.
6. При отсутствии других устройств в радиусе действия прекращать сканирование и отключать Bluetooth через заданный период времени.
7. Предлагать пользователю включить модуль Bluetooth в начале работы, если он отключен.
8. Отображать состояние адаптера Bluetooth в приложении (bluetooth.getState()).
9. Отображать список спаренных (уже известных пользователю) устройств.
10. Сохранять и отображать информацию о последнем переданном файле (например, имя, путь, размер, имя устройства получателя).