Android移动开发

Android Mobile Application Development

第11讲 网络通信

吴以凡

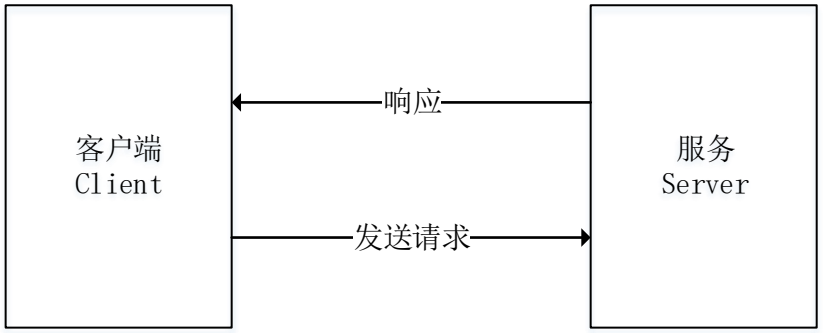
计算机学院 一教505

yfwu@hdu.edu.cn

HTTP通信

HTTP协议简介

- HTTP协议是Hyper Text Transfer Protocol（超文本传输协议）的缩写，是用于从万维网（WWW，World Wide Web）服务器传输超文本到本地浏览器的传送协议
- HTTP是基于TCP/IP通信协议来传递数据（HTML文件,图片文件,查询结果等）的
- HTTP是一个属于应用层的面向对象的协议
- HTTP协议工作于客户端-服务端架构为上。浏览器作为HTTP客户端通过URL向HTTP服务端即WEB服务器发送所有请求。Web服务器根据接收到的请求后，向客户端发送响应信息。



HTTP协议简介

- HTTP协议的主要特点
  - 简单快速：客户向服务器请求服务时，只需传送请求方法和路径。请求方法常用的有GET、HEAD、POST、PUT、DELETE
  - 灵活：HTTP允许传输任意类型的数据对象
  - 无连接：无连接的含义是限制每次连接只处理一个请求。服务器处理完客户的请求，并收到客户的应答后，即断开连接
  - 无状态：HTTP协议是无状态协议。无状态是指协议对于事务处理没有记忆能力。缺少状态意味着如果后续处理需要前面的信息，则它必须重传，这样可能导致每次连接传送的数据量增大
  - 支持B/S及C/S模式

HTTP协议简介

- HTTP请求的方法
  - GET：请求获得资源
  - POST：添加新资源
  - PUT：修改资源
  - PATCH：修改资源
  - HEAD：获取资源响应消息报头
  - DELETE：删除资源
  - TRACE：用于测试或诊断
  - OPTIONS：查询资源是否可用
  - CONNECT：保留将来使用

Java中HTTP相关接口

- Java.net.*提供与互联网有关的类，包括流和数据包套接字、Internet协议和一些Http处理

类/接口	说明
ServerSocket	实现服务器套接字
Socket	实现客户端套接字
DatagramSocket	表示用来发送和接收数据报包的套接字
InterAddress	表示互联网协议（IP）地址
HttpURLConnection	用于管理Http链接的资源连接管理器
URL	代表一个统一资源定位符，它是指向互联网“资源”的指针

Java中HTTP相关接口

- java.net包的HTTP的方法应用

```
try {
   //定义地址
   URL url=new URL("http://localhost:8080/Test/index.jsp");
   //打开链接地址
   HttpURLConnection http = (HttpURLConnection)url.openConnection();
   //得到连接状态
   int status = http.getResponseCode();
   if (status == HttpURLConnection.HTTP_OK) {
      // 取得数据
      InputStream in = http.getInputStream();
      //处理数据
      ...
   }
} catch (Exception e){
}
```

Android网络接口

- Android的网络接口即android.net.*，它实际上是通过对Apache HttpClient的封装来实现一个HTTP编程接口，比java.net.* API功能更强大
- android.net.*除了具备核心的java.net.*外，还包含额外的网络访问Socket。该包包括URI类，在Android应用程序开发中使用较多。同时Android网络接口还提供了HTTP请求队列管理、HTTP连接池管理、网络状态监视等接口。

Android网络通信简史

- 早期的Android 应用中，主要使用HttpURLConnection和Apache HttpClient进行网络请求
- Android团队放弃维护Apache HttpClient，在2.3-5.x版本建议使用HttpURLConnection
- 在6.0以后删除了HttpClient相关的API

HttpURLConnection

- 在Android开发中，应用程序经常需要与服务器进行数据交互，包括访问本地服务器以及远程服务器，这些都可以称为访问网络，此时就可以使用HttpURLConnection对象，它是一个标准的Java类。
- HttpURLConnection继承自URLConnection类，两者都是抽象类，其对象主要通过URL的openConnection方法获得。
- openConnection方法只创建URLConnection或者HttpURLConnection实例，但并不进行真正的连接操作，并且每次openConnection都将创建一个新的实例。因此在连接之前可以对其一些属性进行设置。

HttpURLConnection

- HttpURLConnection常用方法
  - 设置请求方式

```
http.setRequestMethod("GET");
```

  - 设置超时时间

```
http.setConnectionTimeout(4000);
```

    - 在连接时需要设置超时时间，如果不设置超时时间，在网络异常的情况下，会导致取不到数据而一直等待，以至于程序不往下执行
- AndroidManifest.xml中需要添加权限

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

实例：HttpURLConnection

- （1）布局设计

```
<EditText
   android:id="@+id/address"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:text="http://img.mp.sohu.com/....5799_th.png"/>
<Button
   android:id="@+id/get_show"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:text="获取并显示图片"/>
<ImageView
   android:id="@+id/images"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content" />
```

实例：HttpURLConnection

- （2）MainActivity

```
//定义获取到图片和失败的状态码
protected static final int SUCCESS=1;
protected static final int ERROR=2;
//创建消息处理器
private Handler handler = new Handler() {
  public void handleMessage(android.os.Message msg){
     if (msg.what == SUCCESS){
        Bitmap bitmap = (Bitmap)msg.obj;
        iv.setImageBitmap(bitmap);
     } else if (msg.what == ERROR){
        Toast.makeText(MainActivity.this,“显示图片错误”,Toast.LENGTH_SHORT).show();
     }
   }
};
```

实例：HttpURLConnection

- （2）MainActivity

```
public void onClick(View view) {
 //获取输入的网络图片地址
 final String path=path_edit.getText().toString().trim();
  new Thread() {
   private HttpURLConnection conn;
    private Bitmap bitmap;
   public void run() {
     //连接服务器GET请求
     try {
   URL url = new URL(path);
        //根据url发送http的请求
        conn=(HttpURLConnection)url.openConnection();
        //设置请求的方式
        conn.setRequestMethod("GET");
        //设置超时时间
        conn.setConnectTimeout(5000);
        //得到服务器返回的响应码
        int status = conn.getResponseCode();
        if(status == 200){
          //请求网络成功，获取输入流
          InputStream in = conn.getInputStream();
          //将流转换为Bitmap对象
          bitmap = BitmapFactory.decodeStream(in);
          //告诉消息处理器显示图片
          Message msg = new Message();
          msg.what = SUCCESS;
          msg.obj = bitmap;
          handler.sendMessage(msg);
        }else {
          //请求网络失败，提示用户
          Message msg = new Message();
          msg.what = ERROR;
          handler.sendMessage(msg);
        }
      } catch (Exception e) {
        e.printStackTrace();
        Message msg = new Message();
        msg.what = ERROR;
        handler.sendMessage(msg);
      }
     }
    }.start();
}
```

Socket通信

Socket通信

- Android应用程序与服务器通信的方式主要有两种，一种是Http通信，另一种是Socket通信。
  - Http连接使用的是“请求-响应方式”，即在请求时才建立连接
  - Socket通信则是在双方建立连接后直接进行数据传输。它在连接时可实现信息的主动推送，而不用每次等客户端先向服务器发送请求
- Socket即“套接字”，用于描述IP地址和端口，是一个通信链的句柄，支持TCP/IP协议的网络通信基本单元
  - 是网络通信过程中端点的抽象表示，包含进行网络通信的五种必须信息：连接使用的协议、本地主机的IP地址、本地进程的协议端口、远程主机的IP地址、远程进程的协议端口

创建Socket

- 连接Socket连接至少需要两个套接字，一个运行于客户端，一个运行于服务端。Socket常用的构造方法如下所示：
  - Socket(InetAddress address, int port)
  - Socket(InetAddress address,int port, boolean stream)
  - Socket(String host,int port, Boolean stream)
  - ServerSocket(int port)
  - ServerSocket(int port, int backlog)
  - ServerSocket(int port, int backlog, InetAddress bindAddr)
- 创建Socket的代码：

```
Socket socket=new Socket("192.168.56.1","35434");
ServerSocket server=new ServerSocket("35434");
```

  - 需要注意的是，在选择端口时每一个端口对应一个服务。0～1023的端口号为系统所保留，所以在选择端口号时最好选择一个大于1023的数，如上面的35434，以防止发生冲突。

输入/输出流

- Socket提供了getInputStream()和getOutPutStream()来得到对应的输入或输出流进行读写操作
  - getInputStream()方法返回InputStream类对象
  - getOutPutStream()方法返回OutputStream类对象
  - 为便于读写数据，可在输入、输出流对象上建立过滤流
  - 对于文本方式流对象，可以采用InputStreamReader、OutputStreamWriter和PrintWriter处理

```
PrintStream pStream = new PrintStream(new
                                BufferedOutputStream(Socket.getOutputStream()));
DataInputStream diStream = new DataInputStream(socket.getInputStream());
PrintWriter pWriter = new PrintWriter(socket.getOutStream(), true);
BufferedReader bReader = new BufferedReader(new
                                InputStreamReader(Socket.getInputStream()));
```

关闭Socket

- 在Socket使用完毕后需要将其关闭，以释放资源。需要注意的是，在关闭Socket之前，需要将与Socket相关的输入输出流先关闭

```
ps.close();  //输出流先关闭
is.close();  //输入流其次关闭
socket.close();  //socket最后关闭
```

实例：Socket向服务器发送内容

- （1）服务端

```
public class testSocket implements Runnable {
  public static final String Server_ip="127.0.0.1";
  public static final int Server_port=2000;
  @Override
  public void run() {
     System.out.println("S:Connectioning...");
   try {
         ServerSocket serverSocket=new ServerSocket(Server_port);
          while (true){
            Socket client=serverSocket.accept();
       System.out.println("S:Receing...");
       try {
              BufferedReader breader=new BufferedReader(new InputStreamReader(client.getInputStream()));
        String str=breader.readLine();
        System.out.println("S:Received：+str);
             } catch (Exception e) {
        System.out.print("S:Error");
        e.printStackTrace();
             } finally {
        client.close();
             }
          }
    } catch (IOException e) {
         e.printStackTrace();
    }
}
```

```
public static void main(String[] args){
      Thread thread=new Thread(new testSocket());
      thread.start();
   }
}
```

代码中设置了服务器端口为2000，然后通过accept()方法使服务器开始监听客户端的连接，然后通过BufferReader对象来接收输入流。最后关闭Socket和流。

实例：Socket向服务器发送内容

- （2）客户端（Android）

```
public class MainActivity extends Activity {
  private EditText mes;
  private Button btn;
  private String ip = "172.16.39.192";
  private int port = 2000;
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    //查看系统的API级别
    int SDK_INT = android.os.Build.VERSION.SDK_INT;
    if (SDK_INT > 8) {
      StrictMode.ThreadPolicy policy = new
          StrictMode.ThreadPolicy.Builder()
          .permitAll().build();
      StrictMode.setThreadPolicy(policy);
    }
    mes=(EditText)findViewById(R.id.mes);
     btn=(Button)findViewById(R.id.send);
```

```
btn.setOnClickListener(new View.OnClickListener() {
      @Override
    public void onClick(View view) {
        try {
          String string = mes.getText().toString();
           if (!TextUtils.isEmpty(string)) {
             SendMes(ip, port, string);
            } else {
        Toast.makeText(MainActivity.this,"请先输入内容
",Toast.LENGTH_SHORT).show();
        mes.requestFocus();
            }
         } catch (Exception e) {
       e.printStackTrace();
        }
     }
  });
```

实例：Socket向服务器发送内容

- （2）客户端（Android）

```
private void SendMes(String ip, int port, String mes) throws UnknownHostException, IOException {
  try {
    Socket socket = null;
    socket = new Socket(ip,port);
    BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));
    writer.write(mes);
    writer.flush();
    writer.close();
    socket.close();
  } catch (UnknownHostException e ) {
    e.printStackTrace();
  } catch (IOException e){
    e.printStackTrace();
  }
}
```

客户端

```
S:Connectioning...
S:Receing...
S:Received:3534
S:Done
```

服务器端

WebView

WebView

- WebView类是WebKit模块Java层的视图类，所有需要使用Web浏览功能的Android应用程序都可以创建该视图对象显示和处理请求的网络资源
- 可以将WebView当成一个完整的浏览器使用
- WebView提供的API可轻松实现加载网址，本地web支持的文件，浏览缓存历史，清空缓存等等

WebView API

- WebView提供的主要API有：
  - loadUrl()：加载Url信息
  - goBack()：向后浏览历史页面。
  - goForword()：向前浏览历史页面。
  - clearCache ()：清除缓存内容。
  - loadData()：添加一个给定的数据到WebView。
  - loadDataWithBaseURL()：添加一个给定的数据到WebView,如果没有则为baseURL指定数据。
  - addJavascriptInterface()：添加一个JavaScript访问对象。

WebVew实例

- 简单手机浏览器
  - 输入网址后先使用isNetworkUrl判断输入URL是否有效
  - 如果有效，通过loadUrl跳转到页面
  - menu提供向前向后浏览历史页面
  - AndroidManifest.xml中添加允许访问网络的权限

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