1、概述

Android提供了5种方式来让用户保存持久化应用程序数据。根据自己的需求来做选择，比如数据是否是应用程序私有的，是否能被其他程序访问，需要多少数据存储空间等，分别是：

① 使用SharedPreferences存储数据

② 文件存储数据

③ SQLite数据库存储数据

④ 使用ContentProvider存储数据

⑤ 网络存储数据

Android提供了一种方式来暴露你的数据（甚至是私有数据）给其他应用程序 - ContentProvider。它是一个可选组件，可公开读写你应用程序数据。

2、SharedPreferences存储

SharedPreference类提供了一个总体框架，使您可以保存和检索的任何基本数据类型（ boolean, float, int, long, string）的持久键-值对（基于XML文件存储的“key-value”键值对数据）。

通常用来存储程序的一些配置信息。其存储在“data/data/程序包名/shared_prefs目录下。

xml 处理时Dalvik会通过自带底层的本地XML Parser解析，比如XMLpull方式，这样对于内存资源占用比较好。

2.1 我们可以通过以下两种方法获取SharedPreferences对象（通过Context）：

> ① getSharedPreferences (String name, int mode)

当我们有多个SharedPreferences的时候，根据第一个参数name获得相应的SharedPreferences对象。

> ② getPreferences (int mode)

如果你的Activity中只需要一个SharedPreferences的时候使用。

这里的mode有四个选项：

```
Context.MODE_PRIVATE
```

该SharedPreferences数据只能被本应用程序读、写。

```
Context.MODE_WORLD_READABLE
```

该SharedPreferences数据能被其他应用程序读，但不能写。

```
Context.MODE_WORLD_WRITEABLE
```

该SharedPreferences数据能被其他应用程序读和写。

```
Context.MODE_MULTI_PROCESS
```

sdk2.3后添加的选项，当多个进程同时读写同一个SharedPreferences时它会检查文件是否修改。

2.2 向Shared Preferences中**写入值**

首先要通过 SharedPreferences.Editor获取到Editor对象；

然后通过Editor的putBoolean() 或 putString()等方法存入值；

最后调用Editor的commit()方法提交；

```
//Use 0 or MODE_PRIVATE for the default operation
SharedPreferences settings = getSharedPreferences("fanrunqi", 0);
SharedPreferences.Editor editor = settings.edit();
editor.putBoolean("isAmazing", true);

// 提交本次编辑
editor.commit();
```

同时Edit还有两个常用的方法：

> editor.remove(String key)： 下一次commit的时候会移除key对应的键值对
>
> editor.clear()：移除所有键值对

2.3 从Shared Preferences中**读取值**

读取值使用 SharedPreference对象的getBoolean()或getString()等方法就行了（没Editor 啥子事）。

```
SharedPreferences settings = getSharedPreferences("fanrunqi", 0);
boolean isAmazing= settings.getBoolean("isAmazing",true);
```

2. 4 Shared Preferences的优缺点

可以看出来Preferences是很轻量级的应用，使用起来也很方便，简洁。但存储数据类型比较单一（只有基本数据类型），无法进行条件查询，只能在不复杂的存储需求下使用，比如保存配置信息等。

3、文件数据存储

3.1 使用内部存储

当文件被保存在内部存储中时，默认情况下，文件是应用程序私有的，其他应用不能访问。当用户卸载应用程序时这些文件也跟着被删除。

文件默认存储位置：/data/data/包名/files/文件名。

3.1.1 创建和写入一个内部存储的私有文件：

① 调用Context的openFileOutput()函数，填入文件名和操作模式，它会返回一个FileOutputStream对象。

② 通过FileOutputStream对象的write()函数写入数据。

③ FileOutputStream对象的close ()函数关闭流。

例如：

```
        String FILENAME = "a.txt";
        String string = "fanrunqi";

        try {
            FileOutputStream fos = openFileOutput(FILENAME,
Context.MODE_PRIVATE);
            fos.write(string.getBytes());
            fos.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
```

在 `openFileOutput(String name, int mode)` 方法中

- name参数： 用于指定文件名称，不能包含路径分隔符“/” ，如果文件不存在，Android 会自动创建它。
- mode参数：用于指定操作模式，分为四种：

> Context.MODE_PRIVATE = 0

为默认操作模式，代表该文件是私有数据，只能被应用本身访问，在该模式下，写入的内容会覆盖原文件的内容。

> Context.MODE_APPEND = 32768

该模式会检查文件是否存在，存在就往文件追加内容，否则就创建新文件。

> Context.MODE_WORLD_READABLE = 1

表示当前文件可以被其他应用读取。

> MODE_WORLD_WRITEABLE

表示当前文件可以被其他应用写入。

3.1.2 读取一个内部存储的私有文件：

① 调用openFileInput( )，参数中填入文件名，会返回一个FileInputStream对象。

② 使用流对象的 read()方法读取字节

③ 调用流的close()方法关闭流

例如：

```
    String FILENAME = "a.txt";
        try {
            FileInputStream inStream = openFileInput(FILENAME);
            int len = 0;
            byte[] buf = new byte[1024];
            StringBuilder sb = new StringBuilder();
            while ((len = inStream.read(buf)) != -1) {
                sb.append(new String(buf, 0, len));
            }
            inStream.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
```

其他一些经常用到的方法：

- getFilesDir()：　得到内存储文件的绝对路径
- getDir()：　在内存储空间中**创建**或**打开一个已经存在**的目录
- deleteFile()：　删除保存在内部存储的文件。
- fileList()：　返回当前由应用程序保存的文件的数组（内存储目录下的全部文件）。

3.1.3　保存编译时的静态文件

如果你想在应用编译时保存静态文件，应该把文件保存在项目的 **res/raw/** 目录下，你可以通过openRawResource()方法去打开它（传入参数R.raw.filename），这个方法返回一个 InputStream流对象你可以读取文件但是不能修改原始文件。

```
InputStream is = this.getResources().openRawResource(R.raw.filename);
```

3.1.4　保存内存缓存文件

有时候我们只想缓存一些数据而不是持久化保存，可以使用getCacheDir（）去打开一个文件，文件的存储目录（ /data/data/包名/cache ）是一个应用专门来保存临时缓存文件的内存目录。

当设备的内部存储空间比较低的时候，Android可能会删除这些缓存文件来恢复空间，但是你不应该依赖系统来回收，要自己维护这些缓存文件把它们的大小限制在一个合理的范围内，比如1ＭＢ．当你卸载应用的时候这些缓存文件也会被移除。

3.2 使用外部存储（sdcard）

因为内部存储容量限制，有时候需要存储数据比较大的时候需要用到外部存储，使用外部存储分为以下几个步骤：

3.2.1　添加外部存储访问限权

首先，要在AndroidManifest.xml中加入访问SDCard的权限，如下:

```
  <!-- 在SDCard中创建与删除文件权限 -->
    <uses-permission
android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS"/>

   <!-- 往SDCard写入数据权限 -->
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

3.2.2　检测外部存储的可用性

在使用外部存储时我们需要检测其状态，它可能被连接到计算机、丢失或者只读等。下面代码将说明如何检查状态：

```
//获取外存储的状态
String state = Environment.getExternalStorageState();
if (Environment.MEDIA_MOUNTED.equals(state)) {
    // 可读可写
    mExternalStorageAvailable = mExternalStorageWriteable = true;
} else if (Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {
    // 可读
} else {
    // 可能有很多其他的状态，但是我们只需要知道，不能读也不能写
}
```

3.2.3 访问外部存储器中的文件

1、如果 API 版本大于或等于 8，使用

> getExternalFilesDir (String type)

该方法打开一个外存储目录，此方法需要一个类型，指定你想要的子目录，如类型参数 DIRECTORY_MUSIC和 DIRECTORY_RINGTONES（传null就是你应用程序的文件目录的根目录）。通过指定目录的类型，确保Android的媒体扫描仪将扫描分类系统中的文件（例如，铃声被确定为铃声）。如果用户卸载应用程序，这个目录及其所有内容将被删除。

例如：

```
File file = new File(getExternalFilesDir(null), "fanrunqi.jpg");
```

2、如果API 版本小于 8（7或者更低）

> getExternalStorageDirectory ()

通过该方法打开外存储的根目录，你应该在以下目录下写入你的应用数据，这样当卸载应用程序时该目录及其所有内容也将被删除。

```
/Android/data/<package_name>/files/
```

读写数据：

```
if(Environment.getExternalStorageState().equals(Environment.MEDIA_MOUNTED)){
            File sdCardDir = Environment.getExternalStorageDirectory();//获取
SDCard目录  "/sdcard"

               File saveFile = new File(sdCardDir,"a.txt");

               //写数据
                try {
                    FileOutputStream fos= new FileOutputStream(saveFile);
                    fos.write("fanrunqi".getBytes());
                    fos.close();
                } catch (Exception e) {
                    e.printStackTrace();
                }

                //读数据
                 try {
                    FileInputStream fis= new FileInputStream(saveFile);
                    int len =0;
```

```
                byte[] buf = new byte[1024];
                StringBuffer sb = new StringBuffer();
                while((len=fis.read(buf))!=-1){
                    sb.append(new String(buf, 0, len));
                }
                fis.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
```

我们也可以在 /Android/data/package_name/cache/目录下做外部缓存。

4、 网络存储数据

HttpUrlConnection

HttpUrlConnection是Java.net包中提供的API，我们知道Android SDK是基于Java的，所以当然优先考虑HttpUrlConnection这种最原始最基本的API，其实大多数开源的联网框架基本上也是基于JDK的HttpUrlConnection进行的封装罢了，掌握HttpUrlConnection需要以下几个步骤：

1、将访问的路径转换成URL。

> URL url = new URL(path);

2、通过URL获取连接。

> HttpURLConnection conn = (HttpURLConnection) url.openConnection();

3、设置请求方式。

> conn.setRequestMethod(GET);

4、设置连接超时时间。

> conn.setConnectTimeout(5000);

5、设置请求头的信息。

> conn.setRequestProperty(User-Agent, Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Trident/5.0));

6、针对不同的响应码，做不同的操作（请求码200，表明请求成功，获取返回内容的输入流）

工具类：

```
public class StreamTools {
    /**
     * 将输入流转换成字符串
     *
     * @param is
     *             从网络获取的输入流
     * @return
     */
    public static String streamToString(InputStream is) {
        try {
            ByteArrayOutputStream baos = new ByteArrayOutputStream();
            byte[] buffer = new byte[1024];
```

```
            int len = 0;
            while ((len = is.read(buffer)) != -1) {
                baos.write(buffer, 0, len);
            }
            baos.close();
            is.close();
            byte[] byteArray = baos.toByteArray();
            return new String(byteArray);
        } catch (Exception e) {
            Log.e(tag, e.toString());
            return null;
        }
    }
}
```

HttpUrlConnection发送GET请求

```
public static String loginByGet(String username, String password) {
        String path = http://192.168.0.107:8080/WebTest/LoginServerlet?username=
+ username + &password= + password;
        try {
            URL url = new URL(path);
            HttpURLConnection conn = (HttpURLConnection) url.openConnection();
            conn.setConnectTimeout(5000);
            conn.setRequestMethod(GET);
            int code = conn.getResponseCode();
            if (code == 200) {
                InputStream is = conn.getInputStream(); // 字节流转换成字符串
                return StreamTools.streamToString(is);
            } else {
                return 网络访问失败;
            }
        } catch (Exception e) {
            e.printStackTrace();
            return 网络访问失败;
        }
    }
```

HttpUrlConnection发送POST请求

```
public static String loginByPost(String username, String password) {
        String path = http://192.168.0.107:8080/WebTest/LoginServerlet;
        try {
            URL url = new URL(path);
            HttpURLConnection conn = (HttpURLConnection) url.openConnection();
            conn.setConnectTimeout(5000);
            conn.setRequestMethod(POST);
            conn.setRequestProperty(Content-Type, application/x-www-form-
urlencoded);
            String data = username= + username + &password= + password;
            conn.setRequestProperty(Content-Length, data.length() + );
            // POST方式，其实就是浏览器把数据写给服务器
            conn.setDoOutput(true); // 设置可输出流
            OutputStream os = conn.getOutputStream(); // 获取输出流
            os.write(data.getBytes()); // 将数据写给服务器
            int code = conn.getResponseCode();
```

```
            if (code == 200) {
                InputStream is = conn.getInputStream();
                return StreamTools.streamToString(is);
            } else {
                return 网络访问失败;
            }
        } catch (Exception e) {
            e.printStackTrace();
            return 网络访问失败;
        }
    }
```

HttpClient

HttpClient是开源组织Apache提供的Java请求网络框架，其最早是为了方便Java服务器开发而诞生的，是对JDK中的HttpUrlConnection各API进行了封装和简化，提高了性能并且降低了调用API的繁琐，Android因此也引进了这个联网框架，我们再不需要导入任何jar或者类库就可以直接使用，值得注意的是Android官方已经宣布不建议使用HttpClient了。

HttpClient发送GET请求

1、 创建HttpClient对象

2、创建HttpGet对象，指定请求地址（带参数）

3、使用HttpClient的execute(),方法执行HttpGet请求，得到HttpResponse对象

4、调用HttpResponse的getStatusLine().getStatusCode()方法得到响应码

5、调用的HttpResponse的getEntity().getContent()得到输入流，获取服务端写回的数据

```
public static String loginByHttpClientGet(String username, String password) {
        String path = http://192.168.0.107:8080/WebTest/LoginServerlet?username=
                + username + &password= + password;
        HttpClient client = new DefaultHttpClient(); // 开启网络访问客户端
        HttpGet httpGet = new HttpGet(path); // 包装一个GET请求
        try {
            HttpResponse response = client.execute(httpGet); // 客户端执行请求
            int code = response.getStatusLine().getStatusCode(); // 获取响应码
            if (code == 200) {
                InputStream is = response.getEntity().getContent(); // 获取实体内容
                String result = StreamTools.streamToString(is); // 字节流转字符串
                return result;
            } else {
                return 网络访问失败;
            }
        } catch (Exception e) {
            e.printStackTrace();
            return 网络访问失败;
        }
    }
```

HttpClient发送POST请求

1，创建HttpClient对象

2，创建HttpPost对象，指定请求地址

3，创建List，用来装载参数

4，调用HttpPost对象的setEntity()方法，装入一个UrlEncodedFormEntity对象，携带之前封装好的参数

5，使用HttpClient的execute()方法执行HttpPost请求，得到HttpResponse对象

6， 调用HttpResponse的getStatusLine().getStatusCode()方法得到响应码

7， 调用的HttpResponse的getEntity().getContent()得到输入流，获取服务端写回的数据

```
public static String loginByHttpClientPOST(String username, String password) {
        String path = http://192.168.0.107:8080/WebTest/LoginServerlet;
        try {
            HttpClient client = new DefaultHttpClient(); // 建立一个客户端
            HttpPost httpPost = new HttpPost(path); // 包装POST请求
            // 设置发送的实体参数
            List parameters = new ArrayList();
            parameters.add(new BasicNameValuePair(username, username));
            parameters.add(new BasicNameValuePair(password, password));
            httpPost.setEntity(new UrlEncodedFormEntity(parameters, UTF-8));
            HttpResponse response = client.execute(httpPost); // 执行POST请求
            int code = response.getStatusLine().getStatusCode();
            if (code == 200) {
                InputStream is = response.getEntity().getContent();
                String result = StreamTools.streamToString(is);
                return result;
            } else {
                return 网络访问失败;
            }
        } catch (Exception e) {
            e.printStackTrace();
            return 访问网络失败;
        }
    }
```

Android提供的其他网络访问框架

HttpClient和HttpUrlConnection的两种网络访问方式编写网络代码，需要自己考虑很多，获取数据或许可以，但是如果要将手机本地数据上传至网络，根据不同的web端接口，需要组织不同的数据内容上传，给手机端造成了很大的工作量。