



6. Retrofit2访问WebService

- Retrofit简介

1. Retrofit是 Square公司方便APP访问服务器API所开发的库， 基于REST规范。
2. Retrofit 是对 OkHttp 的封装，提供了使用注解更简单的构建各种请求，配置各种参数的方式。本质发起网络请求的还是 OkHttp，但 Retrofit 让这一操作更加的简单优雅。





6. Retrofit2访问WebService

- Retrofit注解(annotation)与请求方法

Retrofit通过给访问接口方法添加相应的注解来表示该方法对应于HTTP的哪种请求

请求方法	请求方法
@GET	表明这是get请求
@POST	表明这是post请求
@PUT	表明这是put请求
@DELETE	表明这是delete请求
@PATCH	表明这是一个patch请求，该请求是对put请求的补充，用于更新局部资源
@HEAD	表明这是一个head请求
@OPTIONS	表明这是一个option请求





6. Retrofit2访问WebService

- Retrofit使用

Retrofit使用分成简单的三个部分

1. 定义访问API接口
2. 创建Retrofit对象
3. 使用创建好的Retrofit创建访问实例





6. Retrofit2访问WebService

- Retrofit使用

定义相应HTTP API访问接口

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



将URL中{user}替换成传入的user参数





6. Retrofit2访问WebService

- Retrofit使用

构建Retrofit对象

```
Retrofit retrofit = new Retrofit.Builder()
```

```
.baseUrl("https://api.github.com/")
```

设置访问服务端的baseUrl

```
.build();
```

创建API访问接口， 并调用接口函数获取相应的数据

```
GitHubService service = retrofit.create(GitHubService.class);
```

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





6. Retrofit2访问WebService

- Retrofit使用

构建Retrofit对象

```
Retrofit retrofit = new Retrofit.Builder()
```

```
.baseUrl("https://api.github.com/")
```

设置访问服务端的baseUrl

```
.build();
```

创建API访问接口， 并调用接口函数获取相应的数据

```
GitHubService service = retrofit.create(GitHubService.class);
```

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





6. Retrofit2访问WebService

- Retrofit使用

在访问接口的定义中会涉及到下列几个参数

1. PATH - 用于URL的参数替换
2. Body - 用于Post, Put请求的数据携带
3. Query - urlQuery的参数携带如 **api/v1/user/list?limit=100&offset=10**

表示limit=100, offset=10





6. Retrofit2访问WebService

- Retrofit-PATH

请求路径中可以包含参数，并在参数中使用 @PATH 注解来动态改变路径，如下例子所示：

```
@PUT( "api/v2/baby/doEdit/{PATH_BABY_ID}")
```

```
Call<BaseResponse> putEditBaby(@Path(PATH_BABY_ID) Long babyId);
```



使用注解 @Path(PATH_BABY_ID) Long babyId 即可改变路径中 {id} 请求时的值





6. Retrofit2访问WebService

- Retrofit-Body

发送 POST、PUT 请求时通常需要携带 body 数据，使用 @Body 注解，如下：

```
@PUT(PUT_EDIT_BABY)
```

```
Call<BaseResponse> putEditBaby(@Body BabyParam babyParam);
```



使用注解 (@Body BabyParam Long babyParam) 携带body数据





6. Retrofit2访问WebService

- Retrofit-Query

在HTTP GET请求常需携带相应的query参数

比如： api/v1/user/list?limit=100&offset=10

第一种方法使用 @Query 注解，如 @Query("userId") Long userId 的形式。这种形式可以传递 null 值，如果某个参数为 null，将不会拼接在 url 后面。



```
Call<UserBabyRelationResp> getBabyRelationList  
(@Query("limit") int limit, @Query("offset") Long offset);
```



6. Retrofit2访问WebService

- Retrofit-Query

在HTTP GET请求常需携带相应的query参数

比如： api/v1/user/list?limit=100&offset=10

第二种使用 @QueryMap 注解，如 @QueryMap
Map<String, String> params 的形式。这种形式传递一个 map 作为
参数，但是 map 中 value 不能为 null，否则会抛出异常。



```
Call<UserBabyRelationResp> getBabyRelationList  
(@QueryMap Map<String, String> map);
```



6. Retrofit2访问WebService

- Retrofit-Call

1. Retrofit2 有了新的类型:Call, 语法与okHttp基本一模一样。即:

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

2. 每一个 call 对象实例只能被用一次，所以说 request 和 response 都是一一对应的。你其实可以通过 Clone 方法来创建一个一模一样的实例，这个开销是很小的。





6. Retrofit2访问WebService

- Retrofit-Call

同步与异步

同步与异步概念在多线程章节中已做相应介绍

同步:提交请求->等待处理(这个期间无法进行其他操作) -> 处理完毕
返回

异步 : 请求通过事件触发->等待处理(期间仍然可以进行其他操作)->
处理完毕回调





6. Retrofit2访问WebService

- Retrofit-Call

Call 同步调用-获取城市Id为101010100的天气情况

```
ApiService apiService = retrofit.create(ApiService.class);
Call<ResponseBody> callObject=apiService.getWeather("101010100");
try {
    System.out.println(callObject.execute().body().string());
    System.out.println("这是同步调用后的打印");
} catch (IOException e) {
    e.printStackTrace();
}
System.out.println("同步调用后")
```



I/System.out: 这是同步调用后的打印
I/System.out: 同步调用后





6. Retrofit2访问WebService

- Retrofit-Call

Call 异步调用-获取城市Id为101010100的天气情况

```
callObject.enqueue(new Callback<ResponseBody>() {  
    @Override  
    public void onResponse(Call<ResponseBody> call, Response<ResponseBody>  
response) {  
    try {  
        System.out.println("这是异步调用的结果");  
    } catch (IOException e) {  
        e.printStackTrace();  
    }  
}  
    @Override  
    public void onFailure(Call<ResponseBody> call, Throwable t) {  
    }  
});  
System.out.println("异步调用后")
```

I/System.out: 异步调用后
I/System.out: 这是异步调用后的打印





7. 搭建Java版WebService

- **WebService**

做Android开发，不可避免会涉及到客户端开发，我们怎么样来实现一个服务端，怎么样来实现一个客户端，并相互传递数据。就算调用别人的服务时，也能知道大概是怎么样实现的。

WebService一般分为.Net版和Java版，今天主要来实现Java版的WebService，.Net版本的还是比较简单的。





7. 搭建Java版WebService

- WebService-Java版
有下列几种方式配置WebService
 - 1. JAX-WS : Jax-WS是Java1.6中才有的,新的WebService模式,基于注解的方式配置WebService,很类似Asp中的WebService,难度已经比Xfire方式的配置降低了很多.
 - 2. REST(JAX-RS) : 是一个Java编程语言的应用程序接口,支持按照表象化状态转变 (REST)架构风格创建Web服务
 - 3. Xfire (已过时)





7. 搭建Java版WebService

- **WebService**

做Android开发，不可避免会涉及到客户端开发，我们怎么样来实现一个服务端，怎么样来实现一个客户端，并相互传递数据。就算调用别人的服务时，也能知道大概是怎么样实现的。

WebService一般分为.Net版和Java版，今天主要来实现Java版的WebService，.Net版本的还是比较简单的。





8. Android WiFi开发

- Wi-Fi

WiFi是一种段程无线传输技术，能够在数百英尺范围内支持互联网接入的无线电信号。随着技术的发展，以及IEEE802.11a和IEEE802.11g等标准的出现，现在IEEE802.11这个标准已被统称作Wi-Fi。从应用层面来说，要使用Wi-Fi，用户首先要有Wi-Fi兼容的用户端装置。





8. Android WiFi开发

- 操作Wi-Fi所需权限

状态名称	描述
CHANGE_NETWORK_STATE	允许应用程序改变网络连接状态
CHANGE_WIFI_STATE	允许应用程序改变WIFI连接状态
ACCESS_NETWORK_STATE	允许应用程序访问网络信息
ACCESS_WIFI_STATE	允许应用程序访问WIFI网络信息





8. Android WiFi开发

- WifiManager

要在应用程序中对Android系统的WiFi设备进行相关操作，需要在项目中的AndroidManifest.xml中选择性地添加如下几句用于声明权限的语句：

```
<uses-permission  
    android:name="android.permission.ACCESS_WIFI_STATE">  
</uses-permission>  
<uses-permission  
    android:name="android.permission.ACCESS_CHECKIN_PROPERTIES">  
</uses-permission>  
<uses-permission  
    android:name="android.permission.WAKE_LOCK"></uses-permission>  
<uses-permission  
    android:name="android.permission.CHANGE_WIFI_STATE">  
</uses-permission>
```





8. Android WiFi开发

- android.net.wifi

ScanResult

用于描述一个已经被检测到的wifi接入点。

WifiConfiguration

该类代表了一个已经配置好的wifi网络，包括了该网络的一些安全设置。例如接入点密码，接入点通讯所采用的安全标准。

WifiConfiguration.

公认的IEEE 802.11标准认证算法。

AuthAlgorithm





8. Android WiFi开发

- Android WiFi相关类介绍

WifiConfiguration.GroupCipher

公认的组密码。

WifiConfiguration.KeyMgmt

公认的密钥管理方案。

WifiConfiguration.PairwiseCipher

公认的用于WPA的成对密码标准。

WifiConfiguration.Protocol

公认的安全协议



WifiConfiguration.Status

网络所可能存在的状态。



8. Android WiFi开发

- Android WiFi相关类介绍

WifiInfo

描述了各个wifi连接的状态，该连接是否处于活动状态或者是否处于识别过程中。

WifiManager

这个类比较重要。它提供了用于管理wifi连接的各种主要API。详见表后说明。

WifiManager.MulticastLock

允许应用程序接收wifi的多播数据包。



ANDROID

WifiManager.WifiLock

允许应用程序永久地保持wifi连接（防止系统自动回收）。



8. Android WiFi开发

- WifiManager

1. Android 操作WiFi的重要类——WifiManager，这个类提供了最主要的应用于管理wifi连接的API。通过调用 Context.getSystemService(Context.WIFI_SERVICE)方法来得到系统提供的WifiManager

```
WifiManager mWifiManager = (WifiManager)  
context.getSystemService(Context.WIFI_SERVICE);
```





8. Android WiFi开发

- WifiManager
 1. 已经配置好的网络连接列表。这个列表可以被用户查看或者更新，而且可以通过它来修改个别接入点的属性；
 2. 如果当前有连接存在的话，可以得到当前正处于活动状态的 wifi连接的控制权，可以通过它建立或者断开连接，并且可以查询该网络连接的动态信息；
 3. 通过对已经扫描到的接入点的足够信息来进行判断，得出一个最好的接入点进行连接。
 4. 定义了很多用于系统广播通知的常量，它们分别代表了WiFi状态的改变。





8. Android WiFi开发

- WifiConfiguration相关子类简介
 1. WifiConfiguration.AuthAlgorithm 用来判断加密方法。
 2. WifiConfiguration.GroupCipher 获取使用GroupCipher 的方法来进行加密。
 3. WifiConfiguration.KeyMgmt 获取使用KeyMgmt 进行。
 4. WifiConfiguration.PairwiseCipher 获取使用WPA 方式的加密。
 5. WifiConfiguration.Protocol 获取使用哪一种协议进行加密。
 6. wifiConfiguration.Status 获取当前网络的状态。





8. Android WiFi开发

- **WifiInfo相关方法简介**

在连接上WiFi后可以通过这个类获得一些已经连通的WiFi 连接的信息

1. getBSSID() 获取BSSID
2. getDetailedStateOf() 获取客户端的连通性
3. getHiddenSSID() 获得SSID 是否被隐藏
4. getIpAddress() 获取IP 地址
5. getLinkSpeed() 获得连接的速度
6. getMacAddress() 获得Mac 地址
7. getRssi() 获得802.11n 网络的信号
8. getSSID() 获得SSID
9. getSupplicantState() 返回具体客户端状态的信息





8. Android WiFi开发

- 实例代码

```
WifiManager wifiManager = (WifiManager)
    context.getSystemService(Context.WIFI_SERVICE);

WifiInfo wifiInfo = wifiManager.getConnectionInfo();
System.out.println("WiFi Status : " + WifiConfiguration.Status.CURRENT);
System.out.println("BSSID : " + wifiInfo.getBSSID());
System.out.println("is hidden SSID : " + wifiInfo.getHiddenSSID());
System.out.println("IP Address : " + wifiInfo.getIpAddress());
System.out.println("Link Speed " + wifiInfo.getLinkSpeed());
System.out.println("MAC Address : " + wifiInfo.getMacAddress());
System.out.println("RSSI : " + wifiInfo.getRssi());
System.out.println("SSID : " + wifiInfo.getSSID());
```





8. Android WiFi开发

- 实例代码 - 打印结果

```
I/System.out: WiFi Status : 0  
I/System.out: BSSID : 01:80:c2:00:00:03  
I/System.out: is hidden SSID : false  
I/System.out: IP Address : 251854858(25.185.48.58)  
I/System.out: Link Speed 0  
I/System.out: MAC Address : 02:00:00:00:00:00  
I/System.out: RSSI : -55  
I/System.out: SSID : "WiredSSID"
```



Questions?



ANDROID