SÃOJUDAS UNIVERSIDADE	FTCE Faculdade de Tecnologia e Ciências Exatas

USJT - 2018 - Desenvolvimento de Sistemas Móveis

Professor: Bonato e Rodrigo

Aula: 05
Assunto: Multithreading - Acesso a Serviços REST - Arquivos JSON

Conceitos básicos

Introdução

Permissões de Segurança

Há uma série de permissões de segurança para um app Android, como acesso à câmera, à localização, aos contatos, etc. Uma delas é necessária para que o app acesse a internet, a **android.permission.INTERNET**. A outra é necessária para saber se o telefone tem conexão (WIFI, 3G, etc.), a **android.permission.ACCESS_NETWORK_STATE**. Não se esqueça, portanto, de acrescentá-las ao Manifest.xml de seu app, conforme abaixo. Caso contrário o app não vai funcionar:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="br.usjt.cervejap3" >

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

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="Cerveja P3"
```

REST

Para acessar um serviço REST (como os vistos na aula 10) você precisa usar uma API de HTTP. Você pode usar as classes da package java.net ou uma API de terceiros chamada OkHttp, mais simples e eficiente e largamente usada no mercado.

Você pode baixar o *okio.jar* e o *okhttp.jar* manualmente do site e instalá-lo em seu projeto (na visão Project, na pasta External Libraries) ou colocar a seguinte dependência na tag *dependencies* do arquivo **build.gradle (Module: app)** que fica em Gradle Scrips do seu projeto.

```
compile 'com.squareup.okhttp:okhttp:2.5.0'
```

GET: abaixo o exemplo de como fazer o download de uma URL e imprimir seu conteúdo.

```
OkHttpClient client = new OkHttpClient();

String run(String url) throws IOException {
  Request request = new Request.Builder()
      .url(url)
      .build();

  Response response = client.newCall(request).execute();
  return response.body().string();
}
```

POST: abaixo como postar dados em um servidor.

```
public static final MediaType JSON
    = MediaType.parse("application/json; charset=utf-8");

OkHttpClient client = new OkHttpClient();

String post(String url, String json) throws IOException {
  RequestBody body = RequestBody.create(JSON, json);
  Request request = new Request.Builder()
      .url(url)
      .post(body)
      .build();
  Response response = client.newCall(request).execute();
  return response.body().string();
}
```

Obs: para acessar um serviço rodando em um Tomcat instalado na sua própria máquina, use como servidor o IP/Porta 10.0.2.2:8080. Para o Android o localhost é o próprio Emulador (ou telefone).

JSON

Para fazer o parse de arquivos JSON utilize a API org.json, presente na API do Android, conforme visto na aula 10.

TESTE DE CONEXÃO

Para acessar um serviço REST é necessário ter uma conexão internet do seu celular ativa, WIFI ou 3G. Para verificar isso há uma série de métodos na classe ConnectivityManager do pacote android.net. Para simplesmente saber se há uma conexão ativa ou não, use um método parecido com o método abaixo:

```
public boolean isConnected(Context context) {
    ConnectivityManager connectivityManager =
            (ConnectivityManager) context
```

```
                    .getSystemService(Context.CONNECTIVITY_SERVICE);
    return connectivityManager.getActiveNetworkInfo() != null
            && connectivityManager.getActiveNetworkInfo().isConnected();
}
```

THREADS

Quando um app é iniciado é criada uma única thread chamada **main**. Ela é importante porque é responsável por enviar todos dos eventos para os widgets da interface de usuário, inclusive os que desenham os widgets.

Além disso, tudo o que é executado em um app não é executado automaticamente em uma thread separada, mas sim na main thread. Deste modo, atividades demoradas, como uma chamada de uma URL via HTTP para acesso a um serviço REST, irá parar a main thread enquanto a requisição não é atendida e, como consequência, congelar totalmente a interface da aplicação. E, pior, se este congelamento durar mais do que 5 segundos, o Android apresenta para o usuário a mensagem "A aplicação não está respondendo" (ANR, ou *application not responding*).

Para evitar isto as tarefas mais demoradas devem ser disparadas em uma outra thread, conhecida por **worker thread**. Entretanto, você não deve manipular widgets de interface da worker thread, pois ela não é thread safe (isto é, não tem controle de concorrência, como em banco de dados).

Há duas regras para threads em Android:

1. Não bloqueie a UI thread (ou seja, a main thread)
2. Não acesse o UI Android Toolkit (componentes de interface de usuário) de fora da UI thread (main thread).

Worker Thread

Foram criadas para não bloquear a main thread. Veja, por exemplo, como baixar uma imagem da internet para mostrá-la no seu app:

```
public void onClick(View v) {
    new Thread(new Runnable() {
        public void run() {
            Bitmap b = loadImageFromNetwork("http://example.com/image.png");
            mImageView.setImageBitmap(b);
        }
    }).start();
}
```

Em princípio, ela parece OK. Mas tem um problema. Quando o método setImageBitmap é chamado ele viola a regra 2 das threads em Android. Para evitar isso, o código correto deve ser:

```
public void onClick(View v) {
    new Thread(new Runnable() {
        public void run() {
            final Bitmap bitmap =
                    loadImageFromNetwork("http://example.com/image.png");
            mImageView.post(new Runnable() {
                public void run() {
                    mImageView.setImageBitmap(bitmap);
                }
            });
        }
    }).start();
}
```

Na verdade, há três modos de fazer isso:

- Activity.runOnUiThread(Runnable)
- View.post(Runnable)
- View.postDelayed(Runnable, long)

No app exemplo (cerveja), usamos o Activity.runOnUiThread(Runnable).

Async Task

Uma outra maneira de realizar trabalhos assíncronos (a programação multithread é assíncrona) é com o uso de Async Tasks. Desta forma você consegue chamar a tarefa da main thread e, internamente, a Async Task cria uma worker thread para realizar a tarefa desejada. Veja abaixo um exemplo de como você poderia executar o mesmo download de imagem mostrado acima com o uso de Async Tasks:

```
public void onClick(View v) {
    new DownloadImageTask().execute("http://example.com/image.png");
}

private class DownloadImageTask extends AsyncTask<String, Void, Bitmap> {
    /** The system calls this to perform work in a worker thread and
      * delivers it the parameters given to AsyncTask.execute() */
    protected Bitmap doInBackground(String... urls) {
        return loadImageFromNetwork(urls[0]);
    }

    /** The system calls this to perform work in the UI thread and delivers
      * the result from doInBackground() */
    protected void onPostExecute(Bitmap result) {
        mImageView.setImageBitmap(result);
    }
}
```

Para usar, faça uma subclasse de AsyncTask e implemente o método de callback doInBackground. Este método executa as tarefas em uma worker thread. Para sincronizar de volta com a main thread atualizando alguma coisa na interface de

usuário, implemente o método onPostExecute. Para rodar, chame o método execute.

Exercício Prático Aula: Continuação do App de Service Desk

Use o sistema de Service Desk em Spring que você recebeu junto com esta aula (feito na aula de ARQDES) para chamar o serviço de consulta de chamados. Veja a classe ManterChamadosRestController para ver como chamar o serviço que lista todos os chamados e o serviço que lista os chamados por fila.

Exercício Prático: Continuação do Projeto - Entrega 4

Faça toda lógica de acesso aos serviços do restcountries, tanto para o listview como para os detalhes.

Bibliografia

OkHTTP. disponível em: <http://square.github.io/okhttp/>. Acessada em: 13/09/2015.

Android Layouts; disponível em http://developer.android.com/guide/topics/ui/declaring-layout.html. Acessado em 07/09/2015.