Resolvendo o problema

Quando fizemos o teste de envio da requisição para o servidor, nossa aplicação parou. Isto aconteceu por causa de um Exception. Por que é ruim fazermos uma operação de rede na thread principal? No Android, toda aplicação tem uma thread principal, que é responsável por atualizar a tela, colocar os componentes, fazer a animação e tratar os eventos do usuário. Se passamos muito tempo fazendo uma operação na thread principal, o usuário terá o celular travado. Ele tentará fazer scroll ou clicar em um componente e não funcionará. As animações que deveriam acontecer, também não acontecerão...

Para que isto não acontecesse, o Android passou a lançar o NetworkOnMainThreadException. Então, qualquer coisa que fizermos que pudesse vir a travar o sistema operacional, essa aplicação seria derrubada, porque isso seria uma experiência ruim para o usuário.

No nosso caso, quando tentamos enviar uma requisição e o servidor demorou muito para nos dar um resposta, o Android cancelou a aplicação. Para conseguirmos fazer a requisição, teremos que fazê-la em *background*. Manteremos a thread principal rodando normalmente, e a requisição será executada em uma thread secundária. E quando terminasse, seríamos avisados e mostraríamos a resposta na nossa tela.

Faremos isto, criando uma nova Thread no código. De volta ao ListaAlunosActivity.java. Dentro da Thread passaremos um Runnable e vamos instanciar uma classe anônima. Vamos mover o trecho de código referente ao WebClient para dentro da nova Thread. Adicionaremos um start, para dizer que a Thread deverá rodar, depois de criada.

Thread t = new Thread(new Runnable() {  
 @Override  
 public void run() {  
 WebClient client = new WebClient();  
 String resposta = client.post(json);  
 Toast.makeText(this, resposta, Toast.LENGTH\_LONG).show();  
  
 }  
 }).start();  
  
 break;  
  
}  
return super.onOptionItemSelected(item);  
  
}

Chegará o momento, no qual a nossa thread irá receber uma resposta e terá que mostrar algo na tela, ou seja, fazer o Toast, que irá mexer com o contexto da aplicação (this). Quando tentarmos atualizar algo na tela, receberemos outra exception. O sistema operacional irá nos dizer que não podemos manipular a tela, se não estivermos na thread principal. Isto significa que se tentarmos executar a requisição na thread secundária, não conseguimos atualizar a tela. O que fazer?

Iremos encontrar uma maneira de enviar o resultado da thread secundária para a principal. Poderíamos fazer o Toast fora da Thread, mas isto poderia implicar em uma série de novas dificuldades. Em vez disso, existe uma classe no Android, que nos permite encapsular a criação da thread secundária, e que permite fazero envio da resposta para a thread primária.

Então, vamos voltar o nosso código ao que era antes.

case R.id.menu\_enviar\_notas:  
 AlunoDAO dao = new AlunoDAO(this);  
 List<Aluno> alunos = dao.buscaAlunos();  
 dao.close();  
  
 AlunoConverter conversor = new AlunoConverter();  
 String json = conversor.converteParaJSON(alunos);  
  
 AlunoConverter conversor = new AlunoConverter();  
 String json = conversor.converteParaJSON(alunos);  
  
 WebClient client = new WebClient();  
 String resposta = client.post(json);  
 Toast.makeText(this, resposta, Toast.LENGTH\_LONG).show();  
 break;  
 }  
 return super.onOptionsItemSelected(item);  
  
 }

Este trecho será movido para outra classe, que irá representar uma tarefa assíncrona, e irá disparar uma ação para o servidor. Quando ela retornar, iremos voltar à thread principal.

Primeiramente, vamos criar a nova classe no pacote inicial, que chamaremos de EnviaAlunosTask. Em seguida, faremos um extends AsyncTask para indicar que esta classe representa o AsyncTaskjá existente no Android.

Após implementá-la, obrigatoriamente teremos que importar o método doInBackground. Isto é exatamente o que queremos fazer na thread secundária, executar uma tarefa em background.

public class EnviaAlunosTask extends AsyncTask {  
 @Override  
 protected Object doInBackground(Object[] params) {  
 return null;  
 }  
}

Agora, iremos mover para a nova classe o trecho de código que estava na thread principal. Ele irá importar todas as classes necessárias.

public class EnviaAlunosTask extends AsyncTask {  
 @Override  
 protected Object doInBackground(Object[] params) {  
 AlunoDAO dao = new AlunoDAO(this);  
 List<Aluno> alunos = dao.buscaAlunos();  
 dao.close();  
  
 AlunoConverter conversor = new AlunoConverter();  
 String json = conversor.converteParaJSON(alunos);  
  
 AlunoConverter conversor = new AlunoConverter();  
 String json = conversor.converteParaJSON(alunos);  
  
 WebClient client = new WebClient();  
 String resposta = client.post(json);  
 Toast.makeText(this, resposta, Toast.LENGTH\_LONG).show();  
 return null;  
 }  
 }

Para instanciarmos o AlunoDAO, precisamos de um contexto. Mas dentro da AsyncTask, não teremos um acesso fácil ao contexto e iremos trazê-lo de um construtor. Vamos adicionar um context e criar um atributo na classe. Depois, precisaremos de um construtor para popular um contexto.

Faremos isto usando atalho command + N (no Mac) ou ALT + Insert (no Windows) e depois, selecionando a opção Constructor - nós iremos gerar um construtor, pedindo como parâmetro o contexto.

public class EnviaAlunosTask extends AsyncTask {  
 private Context context;  
  
 public EnviaAlunosTask(Context context) {  
 this.context = context;  
  
 }  
\...

Mais abaixo, no Toast, iremos substituir o this pelo context.

public class EnviaAlunosTask extends AsyncTask {  
 private Context context;  
  
 public EnviaAlunosTask(Context context) {  
 this.context = context;  
  
 }  
  
 @Override  
 protected Object doInBackground(Object[] params) {  
 AlunoDAO dao = new AlunoDAO(this);  
 List<Aluno> alunos = dao.buscaAlunos();  
 dao.close();  
  
 AlunoConverter conversor = new AlunoConverter();  
 String json = conversor.converteParaJSON(alunos);  
  
 AlunoConverter conversor = new AlunoConverter();  
 String json = conversor.converteParaJSON(alunos);  
  
 WebClient client = new WebClient();  
 String resposta = client.post(json);  
 Toast.makeText(context, resposta, Toast.LENGTH\_LONG).show();  
 return null;  
 }  
 }

Porém, não podemos mostrar o Toast na thread secundária. Por isso, não iremos executá-lo neste método. Em vez disso, iremos devolver a resposta.

WebClient client = new WebClient();  
String resposta = client.post(json);  
return resposta;

O return irá enviar a resposta para thread principal, usando outro método da AsyncTask. Iremos sobrescrever onPostExecute, que será executado após doInBackground. O método onPostExecute, usa como parâmetro Object, que é referente ao retorno do doInBackground

@Override  
protect void onPostExecute(Object o) {  
 super.onPostExecute(o);  
}

E por que mudamos de método? Porque o onPostExecute será executado na thread principal. Dentro dele, podemos executar o Toast, adicionando o objeto o e especificando que é uma String.

@Override  
protect void onPostExecute(Object o) {  
 Toast.makeText(conext, (String) o, Toast.LENGTH\_LONG).show();  
  
}

Mas para deixarmos o código mais claro, podemos não trabalhar com o Object. Como nós sabemos o tipo de dados que estamos trabalhando, podemos colocá-los como parâmetros do genérico da classe AsyncTask. Por exemplo, poderíamos adicionar Integer e depois, usar como substituto de Object.

public class EnviaAlunosTask extends AsyncTask<Integer> {  
 private Context context;  
  
 public EnviaAlunosTask(Context context) {  
 this.context = context;  
  
 }  
  
 @Override  
 protected Object doInBackground(Integer... params) {  
 AlunoDAO dao = new AlunoDAO(this);  
 List<Aluno> alunos = dao.buscaAlunos();  
 dao.close();  
\...

Outra opção, para deixarmos igual ao que estava, podemos adicionar como parâmetros do genérico da classe AsyncTask, Object, Object, Object.

public class EnviaAlunosTask extends AsyncTask<Object, Object, Object> {  
 private Context context;  
  
 public EnviaAlunosTask(Context context) {  
 this.context = context;  
  
 }  
  
 @Override  
 protected Object doInBackground(Object... params) {  
 AlunoDAO dao = new AlunoDAO(this);  
 List<Aluno> alunos = dao.buscaAlunos();  
 dao.close();  
\...

Se quisermos mudar o valor de resposta, nós precisaremos mudar o último parâmetro da AsyncTask. Podemos alterá-lo para String e depois, fazer o mesmo na assinatura do método.

public class EnviaAlunosTask extends AsyncTask<Object, Object, String> {  
 private Context context;  
  
 public EnviaAlunosTask(Context context) {  
 this.context = context;  
  
 }  
  
 @Override  
 protected String doInBackground(Object... params) {  
 AlunoDAO dao = new AlunoDAO(this);  
 List<Aluno> alunos = dao.buscaAlunos();  
 dao.close();  
\...

O método onPostExecute também terá que se tornar uma String:

@Override  
protect void onPostExecute(String o ) {  
 Toast.makeText(context, (String) o, Toast.LENGTH\_LONG).show();  
  
}

Perceba que ao fazermos a alteração no genérico, conseguimos mudar a assinatura dos métodos e conseguimos trabalhar com tipos de dados mais apropriados.

No método onPostExecute, vamos dar um nome mais apropriado para a String.

@Override  
protect void onPostExecute(String resposta ) {  
 Toast.makeText(context, resposta, Toast.LENGTH\_LONG).show();  
  
}

Depois, voltaremos para o ListaAlunosActivity.java, e no menu de notas, iremos instanciar a Task. Passaremos como parâmetro o this e para executá-la, adicionaremos o método execute().

@Override  
public boolean onOptionsItemSelected(MenuItem item) {  
 switch (item.getItemId()) {  
 case R.id.menu\_enviar\_notas;  
 new EnviaAlunosTask(this).execute();  
 break.  
  
 }  
 return super.onOptionsItemSelected(item);  
}

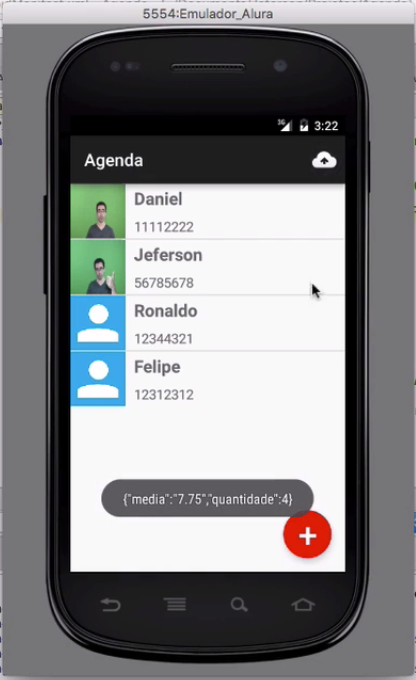
Vamos rodar a nossa aplicação e novamente teremos um problema. A execução do aplicativo foi cancelada. Mas dessa vez, a solução é mais simples. Se pesquisarmos, veremos que encontramos um problema de SecurutyException.



Ele está nos dizendo que a permissão foi negada. Como tentamos fazer uma requisição de um servidor remoto, mas na própria mensagem ele fala sobre a permissão de internet.

Então, vamos para o AndroidManifest.xml. No topo, encontraremos as permissões. Iremos adicionar uses-permission e depois, procuraremos por permission.INTERNET. Esta é uma permissão que o Android considera segura, isto significa que será pedida uma única permissão para o usuário. Depois, a aplicação já estará funcionando e não precisará ativar nas preferências da aplicação.

Faremos um novo teste com o emulador.



Desta vez, enviamos uma requisição e recebemos um json com a média e a quantidade de alunos. Então, está funcionando a requisição e a resposta.

Para fazer melhorias, vamos mostrar para o usuário o progresso do envio da requisição. Para isto, voltaremos para EnviaAlunosTask.java, responsável por fazer a requisição e vamos trabalhar com um objeto novo, ProgressDiolog.

Queremos que a janela mostrando o progresso, apareça na thread principal, antes de executarmos o envio da requisição. Da mesma maneira como temos o onPostExecute, que é um método para **após** a execução, iremos adicionar o onPreExecute, que usaremos para executar antes do doInBackground. Este método também será executado na thread inicial.

@Override  
protect void onPreExecute() {  
 super.onPreExecute();  
}  
  
@Override  
protected String doInBackground(Object... params) {  
 AlunoDAO dao = new AlunoDAO(context);  
 List<Aluno> Alunos = dao.buscaAlunos();  
 dao.close();  
  
 AlunoConverter conversos = new AlunoConverter();  
 String json = conversor.converteParaJSON(alunos);  
  
 WebClient client = new WebClient();  
 String resposta = client.post(json);  
 return resposta;  
}

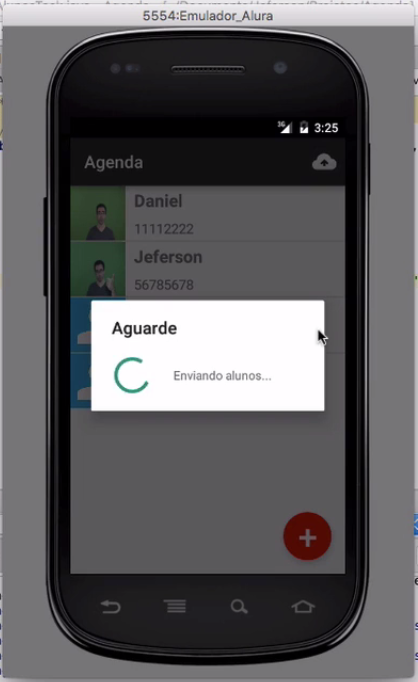
Vamos substituir super.onPreExecute por ProgressDialog. Chamaremos de dialog.

@Override  
protected void onPreExecute() {  
 ProgressDialog dialog = ProgressDialog.show(context, "Aguarde", "Enviando alunos...")  
}

Ele nos pediu vários parâmetros: contexto, títulos e mensagem. Podemos adicionar mais duas opções, que são valores boolean. A primeira será referente se o valor será indeterminado, porque não terá um fim, a segunda será para definir se o valor é cancelable, dando ao usuário a chance de cancelar a ação. Iremos responder true para as duas.

@Override  
protected void onPreExecute() {  
 ProgressDialog dialog = ProgressDialog.show(context, "Aguarde", "Enviando alunos...", true, true);  
}

Vamos testar o nosso aplicativo novamente. Já sabemos que ele está funcionando, agora, nosso objetivo é verificar se a mensagem de *feedback* está aparecendo para o usuário. Após clicarmos para o envio da requisição para o servidor, veremos que aparece uma mensagem de progresso.



Porém, depois de recebermos a resposta do servidor com a média e a quantidade de alunos, a janela de progresso permanece sendo visualizada na tela. Precisamos que ela saia da tela, após a tareda ser executada. Faremos isto, adicionando dialog.dismiss() no método onPOstExecute.

@Override  
protected void onPostExecute(String resposta) {  
 dialog.dismiss();  
 Toast.makeText(context, resposta, Toast.LENGTH\_LONG).show();  
}

Mas ainda não temos acesso ao dialog, porque o criamos no onPreExecute. Teremos que transformá-lo em um atributo da classe. Iremos criar um field mais acima, do ProgressDialog.

public class EnviaAlunosTask extends AsyncTask<Object, Object, String> {  
 private Context context;  
 private ProgressDialog dialog;  
  
 public EnviaAlunosTask(Context context) {  
 this.context = context;  
 }  
  
 @Override  
 protected void onPreExecute() {  
 dialog = ProgressDialog.show(context, "Aguarde", "Enviando alunos...", true, true);  
 }  
\...

Agora, o dismiss() do onPostExecute está funcionando.

Vamos fazer um novo envio de requisição do aplicativo. Depois do recebimento da resposta, a janela de progresso irá desaparecer da tela.

Para concluirmos, falaremos sobre os parâmetros adicionados ao AsyncTask.

public class EnviaAlunosTask extends AsyncTask<Object, Object, String>

O primeiro é o mais importante, porque ele irá alterar os outros que serão recebidos no método doInBackground. Se você não for utilizar o primeiro (e talvez, o segundo) parâmetro, o recomendável é usar Void.

public class EnviaAlunosTask extends AsyncTask<Void, Void, String>

O segundo parâmetro é mais usado quando queremos fazer atualizações no meio da tarefa.

Com o conteúdo que vimos até aqui, conseguimos fazer requisições da forma que nos propomos.

Então, recapitulando:

* Pegamos o código que estava sendo utilizado na thread principal e passamos para uma secundária, usando a classe AsyncTask do Android, que gerenciará todo o processo.
* Para criarmos o AsynTask, criamos a classe EnviaAlunosTask, extendemos. Depois definimos alguns parâmetros do *generic*.
* Depois, precisamos implementar um método obrigatório, o doInBackground, que de fato será feito em um thread secundária. Nela, iremos buscar os alunos, convertê-los para o formato json, fazer a requisição para o servidor e pedir a resposta.
* Nós iremos devolver a resposta, o Android irá tirá-la da thread secundária e irá executá-la na principal. Para fazer isto, ela invocará o método onPostExecute. A resposta do doInBackground entrará como parâmetro no onPostExecute e então, conseguimos fazer as ações desejadas: mostrar o Toast na tela, tirar a janela de dialog. Ou seja, conseguimos atualizar a tela. Como estaremos na thread principal, não teremos problema com o Android.
* Precisamos também adicionar uma requisição de internet, por isso, declaramos no AndroidManifest para o momento da instalação do usuário.

Com isto, encerramos a segunda parte do curso de Android. Parabéns por ter chegado até aqui.

Em seguida, iremos continuar com a terceira parte do curso.