04

Enviando alunos não sincronizados

Transcrição

Agora que conseguimos marcar os alunos como não sincronizados, criaremos um método com uma *query*. Ela devolverá todos os alunos que estão como "não sincronizados".

Acessaremos a classe `AlunoDAO` e em seguida, criaremos o método `listaNaoSincronizados()`. Antes de executarmos a *query*, será necessário o uso de uma instância do banco de dados que podemos pegar por meio do `getReadableDatabase();`.

```
// ...
public List<Aluno> listaNaoSincronizados(){
    SQLiteDatabase db = getReadableDatabase();
}

// ...
```

Com a referência do banco de dados, iremos preparar a instrução *SQL* que buscará todos os alunos com a condição de "não sincronizados".

```
// ...

public List<Aluno> listaNaoSincronizados(){
    SQLiteDatabase db = getReadableDatabase();
    String sql = "SELECT * FROM Alunos WHERE sincronizado = 0";
}

// ...
```

Agora que temos a referência do banco e a *query*, vamos invocar o método `db.rawQuery(sql, null)` usando o objeto `db`. O método nos retorna um objeto do tipo `Cursor`, no qual passaremos como argumento para o `populaAlunos(cursor)`.

```
// ...

public List<Aluno> listaNaoSincronizados(){
    SQLiteDatabase db = getReadableDatabase();
    String sql = "SELECT * FROM Alunos WHERE sincronizado = 0";
    Cursor cursor = db.rawQuery(sql, null);
    return populaAlunos(cursor);
}

// ...
```

Pronto, já temos o método que retorna a lista de "não sincronizados". O próximo passo é enviá-los em uma requisição para o servidor, e o responsável será o `AlunoSincronizador`.

Vamos criar um método que executará a ação de mandar os alunos armazenados internamente. O método será chamado de `sincronizaAlunosInternos()`.

Dentro do método `sincronizaAlunosInternos()`, criaremos uma *call* que vai mandar os alunos para o servidor. Vamos começar pegando os alunos, lembrando que é necessário passar o `context` no construtor do `AlunoDAO()`.

```
// ...

private void sincronizaAlunosInternos(){
    AlunoDAO dao = new AlunoDAO(context);
}

// ...
```

Com o `dao` em mãos, pegaremos a lista dos alunos não sincronizados chamando `dao.listaNaoSincronizados()`.

```
// ...

private void sincronizaAlunosInternos(){
    AlunoDAO dao = new AlunoDAO(context);
    List<Aluno> alunos = dao.listaNaoSincronizados();
}

// ...
```

Nosso próximo passo será criar a *call* chamando o método que atualizará a lista de alunos.

```
// ...

private void sincronizaAlunosInternos(){
    AlunoDAO dao = new AlunoDAO(context);
    List<Aluno> alunos = dao.listaNaoSincronizados();
    new RetrofitInicializador().getAlunoService().atualiza(alunos);
}

// ...
```

Mas não temos o método `atualiza()`, então vamos criá-lo na interface `AlunoService`. Usaremos sempre o auxílio da IDE para criar os métodos.

Colocaremos o retorno do método como `<AlunoSync>`, pois esperamos que o servidor nos devolva todos os alunos que ele atualizar. Como vamos mandar muitos parâmetros, vamos enviá-los pelo `@Body`. Na arquitetura REST, nós usamos o verbo HTTP `PUT` para atualizar dados, e considerando que estamos mandando muitos alunos, o endereço de mapeamento será `@PUT("aluno/lista")`.

```
// ...

    @PUT("aluno/lista")
    Call<AlunoSync> atualiza(@Body List<Aluno> alunos);

}
```

Voltando a classe `AlunoSincronizador`, vamos pegar o retorno da chamada e depois, armazenaremos em uma nova variável que nomearemos como `call`. Agora basta executarmos o `call.enqueue()` passando por argumento a função de *Callback*, como já fizemos diversas vezes.

```
// ...

private void sincronizaAlunosInternos(){
    AlunoDAO dao = new AlunoDAO(context);
    final List<Aluno> alunos = dao.listaNaoSincronizados();
    Call<AlunoSync> call = new RetrofitInicializador().getAlunoService().atualiza(alunos);

     call.enqueue(new Callback<AlunoSync>() {
            @Override
            public void onResponse(Call<AlunoSync> call, Response<AlunoSync> response) {

            }

            @Override
            public void onFailure(Call<AlunoSync> call, Throwable t) {

            }
        });
}

// ...
```

O que deve acontecer no *Callback*? Estamos mandando informações para o servidor, que processará e em seguida, vai nos devolver tudo aquilo que ele conseguiu atualizar. Mas, o que faremos com as informações atualizadas? Precisamos marcá-las como "sincronizadas".

Então dentro do método `onResponse()`, guardaremos o `response.body()` em uma variável chamada `alunoSync`. Com isto, podemos invocar o `dao.sincroniza(alunoSync)`.

```
// ...

call.enqueue(new Callback<AlunoSync>() {
    @Override
    public void onResponse(Call<AlunoSync> call, Response<AlunoSync> response) {
        AlunoSync alunoSync = response.body();
        dao.sincroniza(alunoSync);
    }

    @Override
    public void onFailure(Call<AlunoSync> call, Throwable t) {

    }
});

// ...
```

O código pode quebrar no `dao.sincroniza(alunoSync)`, justamente porque estamos usando uma implementação dentro de uma interface, por isso, a declaração da variável `dao` precisa ser `final`. Lembre-se de fechar a conexão com o

`dao.close()` .

```
// ...

private void sincronizaAlunosInternos(){
    final AlunoDAO dao = new AlunoDAO(context);
    final List<Aluno> alunos = dao.listaNaoSincronizados();
    Call<AlunoSync> call = new RetrofitInicializador().getAlunoService().atualiza(alunos);

     call.enqueue(new Callback<AlunoSync>() {
        @Override
        public void onResponse(Call<AlunoSync> call, Response<AlunoSync> response) {
            AlunoSync alunoSync = response.body();
            dao.sincroniza(alunoSync);
            dao.close();
        }

        @Override
        public void onFailure(Call<AlunoSync> call, Throwable t) {

        }
    });
}
// ...
```

O que exatamente o método `sincroniza()` está fazendo? Ao acessarmos o `sincroniza()` dentro da classe `AlunoDAO` , perceberemos que ele percorre a lista de alunos verificando se existe e se ele está desativado, assim por diante. Mas em nenhum momento ele está indicando que o dado está sincronizado.

Todas as vezes que o `sincroniza()` tentar sincronizar os alunos que vieram do servidor, precisaremos *settar* o valor de "sincronizado". Mas isso não é função do `sincroniza()` , então, criaremos um método na classe `Aluno` .

```
// ...

public void sincroniza(List<Aluno> alunos) {
    for (Aluno aluno :
            alunos) {

        aluno.sincroniza();


        if (existe(aluno)) {
            if(aluno.estaDesativado()){
                deleta(aluno);
            } else {
                altera(aluno);
            }
        } else if (!aluno.estaDesativado()){
            insere(aluno);
        }


    }
}
```

```
// ...
```

Na classe `Aluno`, o método `sincroniza()` ficará da seguinte maneira:

```
// ...

public void sincroniza() {
    this.sincronizado = 1;
}

// ...
```

Conseguimos configurar tudo, porém, não estamos utilizando o método. O que precisamos fazer é, no mesmo momento em que tentarmos buscar os alunos ou fazer um *swipe*, enviaremos os alunos internos.

Na classe `ListaAlunosActivity`, dentro do método `onCreate()`, nós buscamos os alunos quando a tela é criada. Podemos simultaneamente sincronizar os alunos internos.

```
// ...

registerForContextMenu(listaAlunos);
sincronizador.buscaTodos();
sincronizador.sincronizaAlunosInternos();

// ...
```

Mas não podemos deixar a chamada `sincronizador.sincronizaAlunosInternos()` apenas para a criação da tela. Se o usuário nunca fechar a aplicação, o método não será chamado novamente. Nós temos um mecanismo para atualizar as informações sem a necessidade de fechar o aplicativo, o *swipe*. Desta forma, dentro do método `onRefresh()`, nós colocaremos o `sincronizador.sincronizaAlunosInternos()`.

```
// ...

swipe.setOnRefreshListener(new SwipeRefreshLeyout.OnRefreshListener(){
    @Override
    public void onRefresh(){
        sincronizador.buscaTodos();
        sincronizador.sincronizaAlunosInternos();
    }
});

// ...
```

Vamos executar a aplicação para testar. O Android Studio executou normalmente, e se atualizarmos o servidor, vamos notar que o aluno adicionado quando estávamos offline foi exibido.

Tentaremos uma nova simulação. Colocaremos o celular no modo avião e em seguida, vamos adicionar um novo contato. Após adicioná-lo, se analisarmos o *Log* no "Android Monitor", podemos notar que todos os alunos estão marcados como "sincronizados" e apenas o último adicionado está como "não sincronizado".

Sairemos do modo avião para estabelecer a conexão, e ao fazermos um *swipe*, o novo aluno será enviado para o servidor.