03

Dando prioridade para o servidor

Transcrição

Vimos todo o conceito das prioridades do *merge* das informações, agora, veremos como implementar a prioridade do servidor.

Como vimos anteriormente, no código da classe `ListaAlunosActivity`, fizemos uma chamada para `sincronizador.buscaTodos()` buscando todos os alunos. Fizemos também uma chamada simultânea para `sincronizador.sincronizaAlunosInternos()`, sincronizando os alunos internos.

```
// ...

listaAlunos = (ListView) findViewById(R.id.lista_alunos);
swipe = (SwipeRefreshLayout) findViewById(R.id.swipe_lista_aluno);

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

    }
});

// ...
```

Em vez de fazer essas chamadas simultaneamente, iremos ordená-las. Mas como criaremos essa ordem?

Na classe `AlunoSincronizador`, dentro do `onResponse()` do método `buscaAlunosCallback()`, quando este for executado e tiver uma resposta do servidor, a lista de alunos será atualizada, e depois, pediremos para sincronizar as informações com `sincronizaAlunosInternos()`

```
// ...

@NonNull
private Callback<AlunoSync> buscaAlunoCallback() {
    return new Callback<AlunoSync>() {
        @Override
        public void onResponse(Call<AlunoSync> call, Response<AlunoSync> response) {

            AlunoSync alunoSync = response.body();
            String versao = alunoSync.getMomentoDaUltimaModificacao();

            preferences.salvaVersao(versao);

            AlunoDAO dao = new AlunoDAO(context);
            dao.sincroniza(alunoSync.getAlunos());
            dao.close();

            Log.i("versao", preferences.getVersao());
```

```
            bus.post(new AtualizaListaAlunoEvent());
            sincronizaAlunosInternos();
        }

        @Override
        public void onFailure(Call<AlunoSync> call, Throwable t) {
            Log.e("onFailure chamado", t.getMessage());
            bus.post(new AtualizaListaAlunoEvent());
        }
    };
}

// ...
```

Da maneira como é feita atualmente, nós permitimos que a sincronização seja realizada em qualquer lugar e momento. Precisamos tirar o acesso público do método `sincronizaAlunosInternos()` e colocá-lo como privado.

```
// ...

private void sincronizaAlunosInternos(){


    //  implementação do método

}

// ...
```

Desta forma, nenhuma outra classe enxerga o método `sincronizaAlunosInternos()`, sendo responsabilidade da classe `AlunoSincronizador`.

Agora removeremos as chamadas referentes ao `sincronizaAlunosInternos()` de dentro da classe `listaAlunosActivity`.

Vamos executar a aplicação e verificar como ela se comporta. Repetiremos o procedimento de colocar o celular em modo avião, depois, iremos editar no **servidor**, o nome do aluno para **Paulo**, e na **aplicação**, colocaremos o nome do mesmo aluno como **Paulo da Silva**.

Ao retirarmos o modo avião e forçarmos a atualização com o *swipe*, é possível perceber que ele manteve a informação do servidor, alterando o nome do aluno na aplicação como **Paulo**.

Primeiro a aplicação Android pegou as informações que tinha no servidor, atualizou e, depois, a aplicação enviou a lista interna de alunos para o servidor.

Uma coisa importante sobre *merge* das informações é que, em algumas funcionalidades, prestaremos atenção se realmente faz sentido trabalhar em modo online e offline.

Alguns aplicativos não deixam você editar informações do seu perfil em modo offline. Isso porque existem alguns critérios a ser considerados para não ter conflitos de informações. Por isso, sempre avalie se vale a pena o esforço de criar uma funcionalidade que funcione tanto online como offline.