04

Integração com outros sistemas: conectividade

Bem-vindo ao curso de Android da Alura

Nesse curso, utilizamos o eclipse com o plugin do ADT, este que não é mais recomendado pelo Google, então recomendamos esse curso apenas se você está envolvido com algum projeto legado. Se você estar começando com o android recomendamos começar com esse curso : [android-studio (www.alura.com.br/course/android-studio-do-zero-a-persistencia)]

No nosso sistema vamos sincronizar dados com um servidor web usando JSON. Neste capítulo, nosso foco é o lado da aplicação Android e não tanto o servidor. Se quiser aprender em detalhes como fazer um servidor, dê uma olhada nesses [cursos da Alura (http://www.alura.com.br/cursos-online-java-web)].

Gerando JSON

O JSON é uma estrutura muito leve e simples de trabalhar, exatamente o que precisamos num desenvolvimento mobile. Com ele podemos representar uma entidade/objeto em um arquivo de formato fácil, que poderá ser lido e absorvido por diversas plataformas sem muito esforço para parseá-lo. Apesar do XML também ser um formato com características semelhantes, o JSON ganhou popularidade na Web pela fácil integração com javascript, e é amplamente utilizado na comunicação de uma aplicação mobile com os websites.

Para criar uma string JSON, precisamos de um `JSONStringer`, que já faz parte do Android. Sendo assim, vamos criar uma classe *helper* para fazer a conversão de uma lista de `Aluno`s para um JSON:

```
public class AlunoConverter {
    public String toJSON(List<Aluno> alunos) {
        try {
            JSONStringer jsonStringer = new JSONStringer();
            jsonStringer.object().key("list").array()
                .object().key("aluno").array();

            for (Aluno aluno : alunos) {
                jsonStringer.object()
                        .key("id").value(aluno.getId())
                        .key("nome").value(aluno.getNome())
                        .key("telefone").value(aluno.getTelefone())
                        .key("endereco").value(aluno.getEndereco())
                        .key("site").value(aluno.getSite())
                        .key("nota").value(aluno.getNota())
                .endObject();
            }

            return jsonStringer.endArray().endObject()
                .endArray().endObject().toString();
        } catch (JSONException e) {
            throw new RuntimeException(e);
        }
    }
}
```

Este método retornará algo como:

```
{
    "list": [
        {
            "aluno": [
                {
                    "id":1,
                    "nome":"Ettore",
                    "telefone":"1155712751",
                    "endereco":"Rua Vergueiro, 3185",
                    "site":"http://www.alura.com.br",
                    "nota":7
                },
                {
                    "id":2,
                    "nome":"Fulano",
                    "telefone":"123123123",
```

```
                    "endereco":"Rua Vergueiro, 3185",
                    "site":"http://www.google.com.br",
                    "nota":5
                }
            ]
        }
    ]
}
```

Para ler um json, basta usar o `JSONObject` com o construtor que recebe uma String:

```
JSONObject objectAluno = new JSONObject(dado) ;

long id = objectAluno.getLong("id");
boolean permissao = objectAluno.getBoolean("permissao");
int idade = objectAluno.getInt("idade");
String nome = objectAluno.getString("nome");
```

Além do `getLong` , `getBoolean` , `getInt` e `getString` , há o `getObject` , que retorna o `JSONObject` contido numa determinada chave. Já o `JSONArray` representa uma coleção de objetos, e sua representação texto é feita usando colchetes.

Enviando dados para o servidor

Desejamos agora enviar para um servidor o `JSON` de nossa lista de contatos e esperar uma resposta, também no formato `JSON` .

A própria plataforma do `Android` possui uma biblioteca da **Apache Foundation** para realização de requests *HTTP*. Se quisermos realizar um *GET* ou *POST* para um servidor basta usarmos a classe `HttpClient` . Como desejamos enviar uma quantidade de informações possivelmente grande (a lista de contatos) é mais indicado que usemos o método `POST` já que o `GET` faz o envio pela URL, possuindo uma limitação quanto ao máximo de caracteres. Para fazer um `POST` a uma URL usaremos a classe `HttpPost` :

```
HttpPost post = new HttpPost("http://www.meuservidor.com.br");
post.setEntity(new StringEntity(json));
```

Para que o servidor seja notificado que estamos enviando a informação no formato `JSON` podemos informar o `Content-Type` no corpo de nossa requisição, além de avisar também que desejamos a resposta no mesmo formato:

```
post.setHeader("Accept", "application/json");
post.setHeader("Content-type", "application/json");
```

Com o **post** corretamente configurado podemos utilizar uma instância de `HttpClient` para fazer o `POST` para o servidor:

```
HttpClient httpClient = new DefaultHttpClient();

//httpPost...

HttpResponse response = httpClient.execute(post);
String jsonDeResposta = EntityUtils.toString(response.getEntity());
```

Para que o código de envio do `JSON` não fique espalhado pelas `Activities` vamos isolar esse comportamento em uma classe chamada **WebClient**:

```
public class WebClient {
    private final String url;

    public WebClient(String url) {
        this.url = url;
    }

    public String post(String json) {
        DefaultHttpClient httpClient = new DefaultHttpClient();

        HttpPost post = new HttpPost(url);
        //... restante do código
    }
}
```