Flutter网络请求

原创 coderwhy coderwhy

> 项目中展示的大部分数据都是来自服务器，我们需要向服务器请求数据，并且对他们进行解析展示。
>
> 向服务器发出请求就需要用到网络请求相关的知识。

一. 网络请求的方式

> 在Flutter中常见的网络请求方式有三种：HttpClient、http库、dio库；

1.1. HttpClient

HttpClient是dart自带的请求类，在io包中，实现了基本的网络请求相关的操作。

网络调用通常遵循如下步骤：

1. 创建 client.
2. 构造 Uri.
3. 发起请求, 等待请求，同时您也可以配置请求headers、 body。
4. 关闭请求, 等待响应.
5. 解码响应的内容.

网络请求实例：

```
void requestNetwork() async {
  // 1.创建HttpClient对象
  final httpClient = HttpClient();

  // 2.构建请求的uri
  final uri = Uri.parse("http://123.207.32.32:8000/api/v1/recommend");

  // 3.构建请求
  final request = await httpClient.getUrl(uri);

  // 4.发送请求，必须
  final response = await request.close();
  if (response.statusCode == HttpStatus.ok) {
    print(await response.transform(utf8.decoder).join());
  } else {
    print(response.statusCode);
  }
}
```

OK，其实HttpClient也可以发送post相关的请求，我们这里就不再演练。

HttpClient虽然可以发送正常的网络请求，但是会暴露过多的细节：

- 比如需要主动关闭request请求，拿到数据后也需要手动的进行字符串解码

在开发中，我们一般很多直接面向HttpClient进行网络请求，而是使用一些库来完成。

1.2. http库

http 是 Dart 官方提供的另一个网络请求类，相比于 HttpClient，易用性提升了不少。

但是，没有默认集成到Dart的SDK中，所以我们需要先在pubspec中依赖它：

```
  http: ^0.12.0+2
```

导入并且使用即可

```
import 'package:http/http.dart' as http;

void httpNetwork() async {
  // 1.创建Client
  final client = http.Client();

  // 2.构建uri
  final url = Uri.parse("http://123.207.32.32:8000/api/v1/recommend");

  // 3.发送请求
  final response = await client.get(url);

  // 4.获取结果
  if (response.statusCode == HttpStatus.ok) {
    print(response.body);
  } else {
    print(response.statusCode);
  }
}
```

1.3. dio三方库

官方提供的HttpClient和http都可以正常的发送网络请求，但是对于现代的应用程序开发来说，我们通常要求的东西会更多：比如拦截器、取消请求、文件上传/下载、超时设置等等；

这个时候，我们可以使用一个在Flutter中非常流行的三方库：dio；

官网有对dio进行解释：

> dio是一个强大的Dart Http请求库，支持Restful API、FormData、拦截器、请求取消、Cookie管理、文件上传/下载、超时、自定义适配器等...

使用dio三方库必然也需要先在pubspec中依赖它：

```
  dio: ^3.0.1
```

代码演练：

```
import 'package:dio/dio.dart';

void dioNetwork() async {
  // 1.创建Dio请求对象
  final dio = Dio();

  // 2.发送网络请求
  final response = await dio.get("http://123.207.32.32:8000/api/v1/recommend");

  // 3.打印请求结果
  if (response.statusCode == HttpStatus.ok) {
    print(response.data);
  } else {
    print("请求失败: ${response.statusCode}");
  }
}
```

1.4. dio库的封装

http_config.dart

```
class HTTPConfig {
  static const baseURL = "https://httpbin.org"
;
  static const timeout = 5000;
}
```

http_request.dart

```dart
import 'package:dio/dio.dart';
import 'package:testflutter001/service/config.dart';

class HttpRequest {
  static final BaseOptions options = BaseOptions(
      baseUrl: HTTPConfig.baseURL, connectTimeout: HTTPConfig.timeout);
  static final Dio dio = Dio(options);

  static Future<T> request<T>(String url,
      {String method = 'get', Map<String, dynamic> params, Interceptor inter}) async {
    // 1.请求的单独配置
    final options = Options(method: method);

    // 2.添加第一个拦截器
    Interceptor dInter = InterceptorsWrapper(
      onRequest: (RequestOptions options) {
        // 1.在进行任何网络请求的时候, 可以添加一个loading显示

        // 2.很多页面的访问必须要求携带Token,那么就可以在这里判断是有Token

        // 3.对参数进行一些处理,比如序列化处理等
        print("拦截了请求");
        return options;
      },
      onResponse: (Response response) {
        print("拦截了响应");
        return response;
      },
      onError: (DioError error) {
        print("拦截了错误");
        return error;
      }
    );
    List<Interceptor> inters = [dInter];
    if (inter != null) {
      inters.add(inter);
    }
    dio.interceptors.addAll(inters);

    // 3.发送网络请求
    try {
      Response response = await dio.request<T>(url, queryParameters: params, options: options);
      return response.data;
    } on DioError catch(e) {
      return Future.error(e);
    }
  }
}
```

代码使用：

```dart
HttpRequest.request("https://httpbin.org/get", params: {"name": "why", 'age': 18}).then((res) {
  print(res);
});

HttpRequest.request("https://httpbin.org/post",
                    method: "post", params: {"name": "why", 'age': 18}).then((res) {
  print(res);
});
```