06

dio

네트워킹과 비동기

dio 패키지 이용하기

• dio 패키지 등록하기

```
dependencies:
  dio: ^4.0.4
```

- Dio 객체의 get() 함수를 호출하여 서버에 요청하며 결과는 Response 타입의 객체

• GET 방식으로 요청하기

```
try {
  var response = await Dio().get('https://reqres.in/api/users?page=2');
  if (response.statusCode == 200) {
    String result = response.data.toString();
    print("result... $result");
  }
} catch (e) {
  print(e);
}
```

dio 패키지 이용하기

- 서버에 전송할 데이터를 다음처럼 queryParameters 매개변수에 Map 객체로 지정

• ueryParameters 매개변수로 데이터 전달하기

```
var response = await Dio().get('https://reqres.in/api/users', queryParameters: {'page':2});
```

- GET 이외에 POST, PUT, DELETE 방식으로 요청하는 post(), put(), delete() 함수도 제공

• POST 방식으로 요청하기

```
var response = await Dio().post(
    'https://reqres.in/api/users',
    data: {
      "name": "kkang",
      "job": "instructor"
    }
);
```

dio 패키지 이용하기

request() 함수로 요청하기

- 서버에 요청할 때 get(), post(), put(), delete() 함수를 이용해도 되지만, request() 함수를 이용해 어떤 방식으로 요청할지 options 매개변수로 지정

• request() 함수로 요청하기

```
var response = await Dio().request(
    'https://reqres.in/api/users',
    data: {
      "name": "kkang",
      "job": "instructor"
    },
    options: Options(method: 'POST')
);
```

dio 패키지 이용하기

BaseOptions로 Dio 속성 지정하기

- BaseOptions 객체를 지정하여 다양하게 설정
- connectTimeout, receiveTimeout 등 타임 아웃을 설정
- baseUrl로 서버 URL의 공통 부분을 명시

• BaseOptions로 Dio 속성 지정하기

```
var dio = Dio(BaseOptions(
  baseUrl: "https://reqres.in/api/",
  connectTimeout: 5000,
  receiveTimeout: 5000,
  headers: {
    HttpHeaders.contentTypeHeader: 'application/json',
    HttpHeaders.acceptHeader: 'application/json'
  },
));
var response = await dio.get('users?page=2');
```

dio 패키지 이용하기

동시 요청하기

- dio에서는 여러 요청을 List 타입으로 지정하여 동시에 처리

• 동시 요청하기

```
List<Response<dynamic>> response =
  await Future.wait([dio.get('https://reqres.in/api/users?page=1'),
  dio.get('https://reqres.in/api/users?page=2')]);

response.forEach((element) {
  if (element.statusCode == 200) {
    String result = element.data.toString();
    print("result... $result");
  }
});
```

dio 패키지 이용하기

파일 전송하기 — MultifileUpload

- MultipartFile 객체 하나가 전송할 파일 하나를 의미
- fromFile() 생성자로 전송할 파일을 지정해 MultipartFile을 생성

• 파일 전송하기

```
MultipartFile.fromFile('./test.txt',filename: 'upload.txt')
```

- 데이터를 지정해 MultipartFile을 생성하려면 다음처럼 fromBytes() 생성자를 이용

• 파일 데이터를 지정해서 전송하기

```
MultipartFile multipartFile = new MultipartFile.fromBytes(
  imageData,   // 파일 데이터
  filename: 'load_image',
  contentType: MediaType("image", "jpg"),
);
```

dio 패키지 이용하기

파일 전송하기 — MultifileUpload

- MultipartFile 객체를 전송하려면 FormData 객체에 담아야 합니다.
- FormData는 MultipartFile뿐만 아니라 서버에 전송할 여러 가지 데이터를 표현하는 객체

• FormData 객체로 파일 전송하기

```
var formData = FormData.fromMap({
  'name': 'kkang',
  'file': await MultipartFile.fromFile('./test.txt',filename: 'upload.txt')
});
var response = await dio.post('/info', data: formData);
```

dio 패키지 이용하기

요청이나 응답 가로채기 — Interceptor

- 인터셉터를 이용하면 요청이나 응답을 가로챌 수 있습니다.
- Interceptor를 상속받는 클래스를 작성하거나 이미 만들어진 Interceptors Wrapper 클래스를 이용

• 인터셉터 작성하기

```
class MyInterceptor extends Interceptor {
  @override
  void onRequest(RequestOptions options, RequestInterceptorHandler handler) {
    print('request... ${options.method} , ${options.path}');
    print('request data : ${options.data}');
    super.onRequest(options, handler);   // 서버 요청
  }
  @override
  void onResponse(Response response, ResponseInterceptorHandler handler) {
    print('response... ${response.statusCode}, ${response.requestOptions.path}');
    print('response data : ${response.data}');
    super.onResponse(response, handler);   // 결괏값 반환
  }
  @override
  void onError(DioError err, ErrorInterceptorHandler handler) {
    super.onError(err, handler);
    print('error... ${err.response?.statusCode}, ${err.requestOptions.path}');
  }
}
```

dio 패키지 이용하기

요청이나 응답 가로채기 — Interceptor

- Interceptor 클래스의 객체를 서버에 요청하기 전에 dio에 설정

• dio에 인터셉터 추가하기

```
var dio = Dio();
dio.interceptors.add(MyInterceptor());
await dio.post(
    'https://reqres.in/api/users',
    data: {
      "name": "kkang",
      "job": "instructor"
    });
```

dio 패키지 이용하기

요청이나 응답 가로채기 — Interceptor

- InterceptorsWrapper를 이용한다면 개발자 클래스를 만들지 않아도 되며 생성자의 onRequest, onResponse 매개변수에 함수를 등록

• InterceptorsWrapper 이용하기

```
dio.interceptors.add(InterceptorsWrapper(
  onRequest: (options, handler) {
    print('request... ${options.method} , ${options.path}');
    print('request data : ${options.data}');
    handler.next(options);   // 서버 요청
  },
  onResponse: (response, handler) {
    print('response... ${response.statusCode}, ${response.requestOptions.path}');
    print('response data : ${response.data}');
    handler.next(response);   // 결괏값 반환
  }
));
```

dio 패키지 이용하기

요청이나 응답 가로채기 — Interceptor

- onRequest() 함수에서 handler.next() 함수를 호출하지 않고 handler.resolve() 함수로 임의의 데이터를 만들어 서버에서 응답한 것처럼 처리

• 서버 대신 응답하기

```
onRequest: (options, handler) {
  print('request... ${options.method} , ${options.path}');
  print('request data : ${options.data}');
  // handler.next(options);  // 서버 요청
  handler.resolve(Response(requestOptions: options, data: {"hello":"world"}));
},
```

감사합니다

단단히 마음먹고 떠난 사람은
산꼭대기에 도착할 수 있다.
산은 올라가는 사람에게만 정복된다.

윌리엄 셰익스피어
William Shakespeare