

**Yandex**



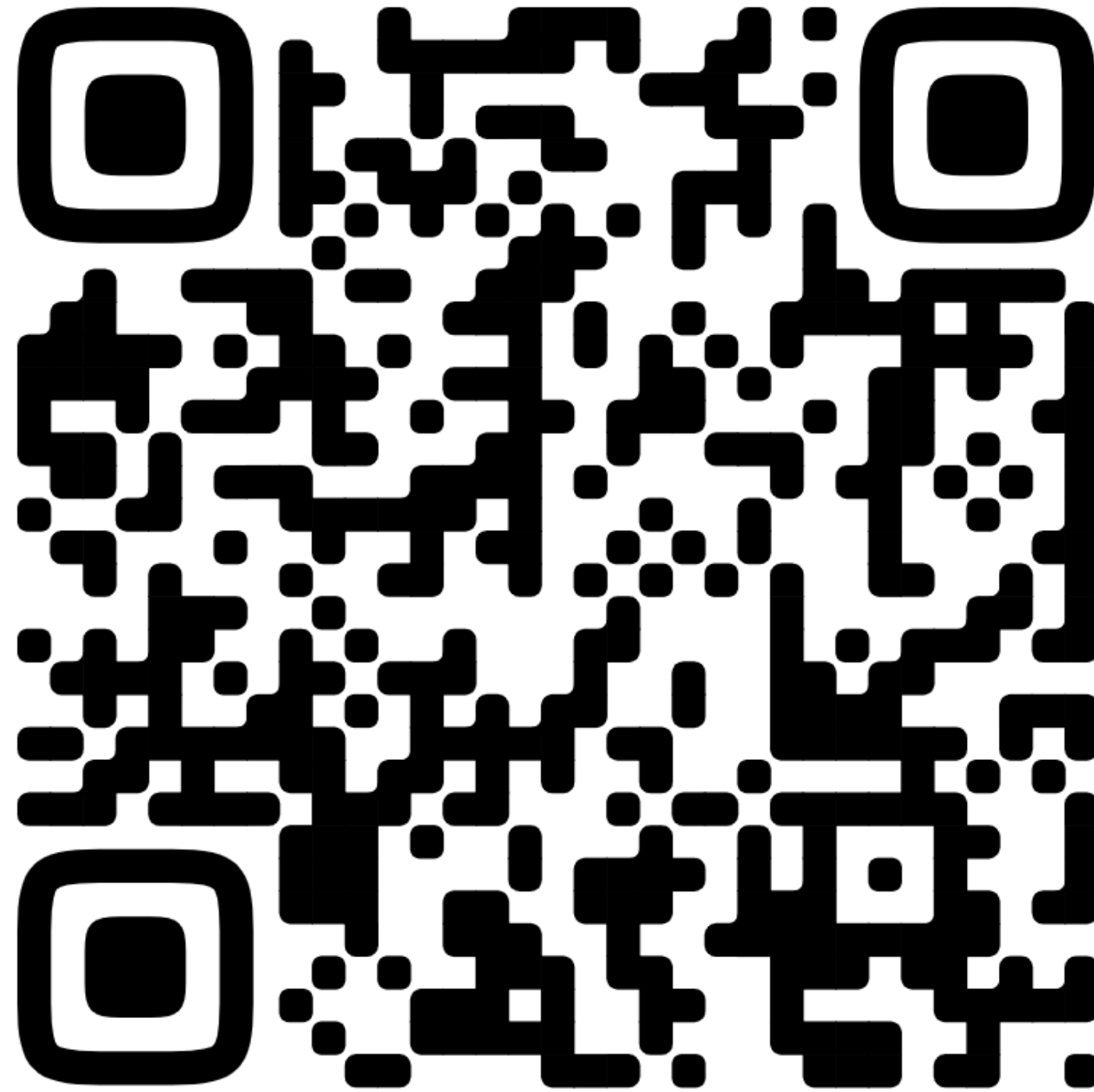
# Basics of async and networking

Sergey Koltsov, Yandex.Pro Team Lead

# Table of Content

1. Sockets and asynchronous programming once again
2. HTTP
3. Serialization
4. Dio

# Networking examples



**bit.ly/networking\_examples**

# **Sockets and asynchronous programming**

(once again)

# Asynchronous programming

- | Future API

- | Async and await

- | Stream

# Future API

```
// HttpRequest.getString(url) - returns Future<String>
void getData(String url) {
    HttpRequest.getString(url).then((String result) {
        print(result); // handle the result
    }).catchError((e) {
        // handle an error
    });
}
```

# async и await

```
// HttpRequest.getString(url) - returns Future<String>
void getData(String url) async { // await keyword inside so make function async
  try {
    // execution will freeze inside getData until result will be returned
    String result = await HttpRequest.getString(url)
    print(result); // handle the result
  } catch (e) {
    // handle an error
  }
}
```



# Live example

```
import 'dart:async';

void main() {
  var counter = 0;
  final timer = Timer.periodic(Duration(seconds: 1), (_) => print(counter++));
  Future.delayed(Duration(seconds: 10)).then((_) {
    print('Finished');
    timer.cancel();
  });
}

void main() async {
  var counter = 0;
  final timer = Timer.periodic(Duration(seconds: 1), (_) => print(counter++));
  await Future.delayed(Duration(seconds: 10));
  print('Finished');
  timer.cancel();
}
```

# Streams

- | Sequence of async events
- | Can subscribe and unsubscribe from a stream
- | Can transform and process a stream

# Stream types

- | Single subscription — subscribe only once. For example, server request.
- | Broadcast — many subscribers. For example, UI events or location.

# Example

```
final subscription = Stream<int>
    .periodic(const Duration(milliseconds: 100), (int event) ⇒ event)
    .listen(
        (event) {
            print(event);    // handle elements
        },
        onError: (e) {
            print(e);        // handle an error
        },
    );
```

# Sockets

```
abstract class Socket implements Stream<Uint8List>, IOSink {  
    // ...  
}  
  
// ----  
  
// open connection  
final socket = await Socket.connect('localhost', 4567);  
  
// receive data  
socket.listen(...) // Function(Uint8List)? onData  
  
// send data  
socket.write(...) // Object? object
```

**HTTP**

pub.dev

http | Dart Package

Sign inHelp

# http 0.13.4

Published 2 months ago • [dart.dev](#) Null safety

[DART](#) [NATIVE](#) [JS](#) [FLUTTER](#) [ANDROID](#) [IOS](#) [LINUX](#) [MACOS](#) [WEB](#) [WINDOWS](#)

3.73K

[Readme](#) [Changelog](#) [Example](#) [Installing](#) [Versions](#) [Scores](#)

A composable, Future-based library for making HTTP requests.

pub

v0.13.4

Dart CI

passing

This package contains a set of high-level functions and classes that make it easy to consume HTTP resources. It's multi-platform, and supports mobile, desktop, and the browser.

## Using

The easiest way to use this library is via the top-level functions. They allow you to make individual HTTP requests with minimal hassle:

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

3734

LIKES

130

PUB POINTS

100%

POPULARITY

Publisher

[dart.dev](#)

Metadata

A composable, multi-platform, Future-based API for HTTP requests.

[Repository \(GitHub\)](#)

[View/report issues](#)

Documentation

[API reference](#)

http

# HTTP

1. Add dependency name into **pubspec.yaml**

```
dependencies:  
  http: 0.13.4
```

2. flutter pub get

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

4. Use it!

```
http.get(Uri.parse('https://api.ipify.org'));
```



# HTTP

```
Future<Response> get(Uri url,  
                    {Map<String, String>? headers})
```

| get

| patch

| post

| put

| head

| delete

# HTTP

```
final client = http.Client();
```

```
client.send(request);
```

# HTTP

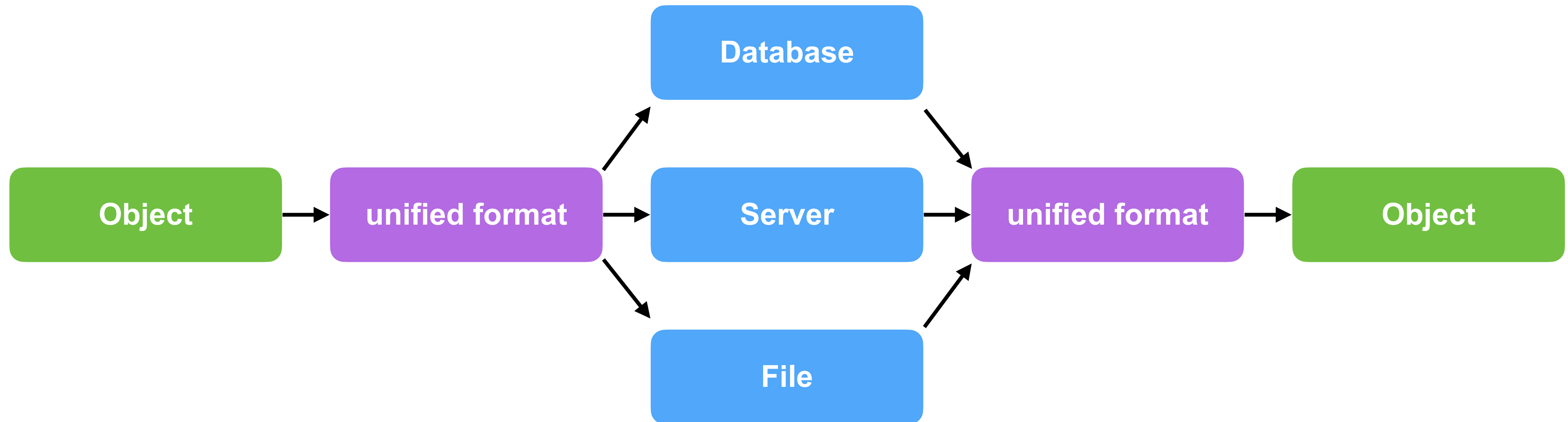
MultipartRequest **extends** BaseRequest

```
final multipart = http.MultipartRequest('POST', uri);  
multipart.files.add(MultipartFile(...));  
  
client.send(multipart);
```

# Serialization

# Serialization

# Deserialization



# Serialization

1. `import 'dart:convert';`

2. Serialization

`jsonEncode(Map<String, dynamic>())` // returns String

3. Deserialization

`jsonDecode('{ "key": "value" })` // returns dynamic

pub.dev

json\_serializable | Dart Package

Sign inHelp

Flutter Favorite

json\_serializable 6.1.1

Published 3 days ago • google.dev Null safety

DARTNATIVE

1.52K

ReadmeChangelogExampleInstallingVersionsScores

pub v6.1.1

Provides [Dart Build System](#) builders for handling JSON.

The builders generate code when they find members annotated with classes defined in [package:json\\_annotation](#).

- To generate to/from JSON code for a class, annotate it with `JsonSerializable`. You can provide arguments to `JsonSerializable` to configure the generated code. You can also customize individual fields by annotating them with `JsonKey` and providing custom arguments. See the table below for details on the [annotation values](#).
- To generate a Dart field with the contents of a file containing JSON, use the `JsonLiteral` annotation.

Setup

To configure your project for the latest released version of, `json_serializable` see the [example](#).

1524LIKES120PUB POINTS99%POPULARITY

Publisher

google.dev

Metadata

Automatically generate code for converting to and from JSON by annotating Dart classes.

[Repository \(GitHub\)](#)

[View/report issues](#)

Documentation

[API reference](#)

License

json\_serializable

# Serialization

1. Add dependency name into **pubspec.yaml**

```
dependencies:  
  json_annotation: 4.4.0  
  
dev_dependencies:  
  json_serializable: 6.1.4  
  build_runner: 2.1.7
```

2. flutter pub get



# Serialization

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

part 'user.g.dart';

@JsonSerializable()
class User {
  final String name;
  final String email;

  User(this.name, this.email);

  factory User.fromJson(Map<String, dynamic> json) => _$UserFromJson(json);
  Map<String, dynamic> toJson() => _$UserToJson(this);
}
```

# Serialization

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

part 'user.g.dart';

@JsonSerializable()
class User {
  final String name;
  final String email;

  User(this.name, this.email);

  factory User.fromJson(Map<String, dynamic> json) => _$UserFromJson(json);
  Map<String, dynamic> toJson() => _$UserToJson(this);
}
```

# Serialization

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

part 'user.g.dart';

@JsonSerializable()
class User {
  final String name;
  final String email;

  User(this.name, this.email);

  factory User.fromJson(Map<String, dynamic> json) => _$UserFromJson(json);
  Map<String, dynamic> toJson() => _$UserToJson(this);
}
```

# Serialization

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

part 'user.g.dart';

@JsonSerializable()
class User {
  final String name;
  final String email;

  User(this.name, this.email);

  factory User.fromJson(Map<String, dynamic> json) => _$UserFromJson(json);
  Map<String, dynamic> toJson() => _$UserToJson(this);
}
```

# Serialization

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

part 'user.g.dart';

@JsonSerializable()
class User {
  final String name;
  final String email;

  User(this.name, this.email);

  factory User.fromJson(Map<String, dynamic> json) => _$UserFromJson(json);
  Map<String, dynamic> toJson() => _$UserToJson(this);
}
```

# Serialization

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

part 'user.g.dart';

@JsonSerializable()
class User {
  final String name;
  final String email;

  User(this.name, this.email);

  factory User.fromJson(Map<String, dynamic> json) => _$UserFromJson(json);
  Map<String, dynamic> toJson() => _$UserToJson(this);
}
```

# Serialization

Run code generation:

```
flutter pub run build_runner build
```

# Serialization

```
const JsonKey({  
  this.defaultValue,  
  this.disallowNullValue,  
  this.fromJson,  
  this.ignore,  
  this.includeIfNull,  
  this.name,  
  this.required,  
  this.toJson,  
  this.unknownEnumValue,  
});
```

```
// ...  
class User {  
  @JsonSerializable(name: 'name')  
  final String name;  
  @JsonSerializable(name: 'email')  
  final String email;  
  
  User(this.name, this.email);  
  
  // ...  
}
```



# Serialization

Other solutions for serialization

| freezed

| built\_value

**Dio**

pub.dev

dio | Dart Package

Search Sign in Help

dio 4.0.4

Published 13 days ago • flutterchina.club • Latest: 4.0.4 / Prerelease: 4.0.5-beta1

DART NATIVE JS FLUTTER ANDROID IOS LINUX MACOS WEB WINDOWS

3.08K

Readme Changelog Example Installing Versions Scores

Language: English | 中文简体

dio

pub v4.0.4 platform flutter|flutter web|dart vm

A powerful Http client for Dart, which supports Interceptors, Global configuration, FormData, Request Cancellation, File downloading, Timeout etc.

Get started

Add dependency

3083 130 100%

LIKES PUB POINTS POPULARITY

Publisher

flutterchina.club

Metadata

A powerful Http client for Dart, which supports Interceptors, FormData, Request Cancellation, File Downloading, Timeout etc.

Repository (GitHub)

View/report issues

dio

# Dio

1. Add dependency name into **pubspec.yaml**

```
dependencies:  
  dio: 4.0.4
```

2. flutter pub get

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

# Dio

4. Create client

```
final Dio _dio = Dio();
```

5. Use it!

```
_dio.get('/path');
```

# Dio: client configuration

```
9      final options = BaseOptions();
```

```
{String method, int connectTimeout, int receiveTimeout,  
int sendTimeout, String baseUrl: '', Map<String,  
dynamic> queryParameters, Map<String, dynamic> extra,  
Map<String, dynamic> headers,  
ResponseType responseType: ResponseType.json,  
String contentType, ValidateStatus validateStatus,  
bool receiveDataWhenStatusError, bool followRedirects,  
int maxRedirects, RequestEncoder requestEncoder,  
ResponseDecoder responseDecoder, ListFormat listFormat,  
setRequestContentTypeWhenNoPayload: false}
```

```
final Dio _dio = Dio(options);
```

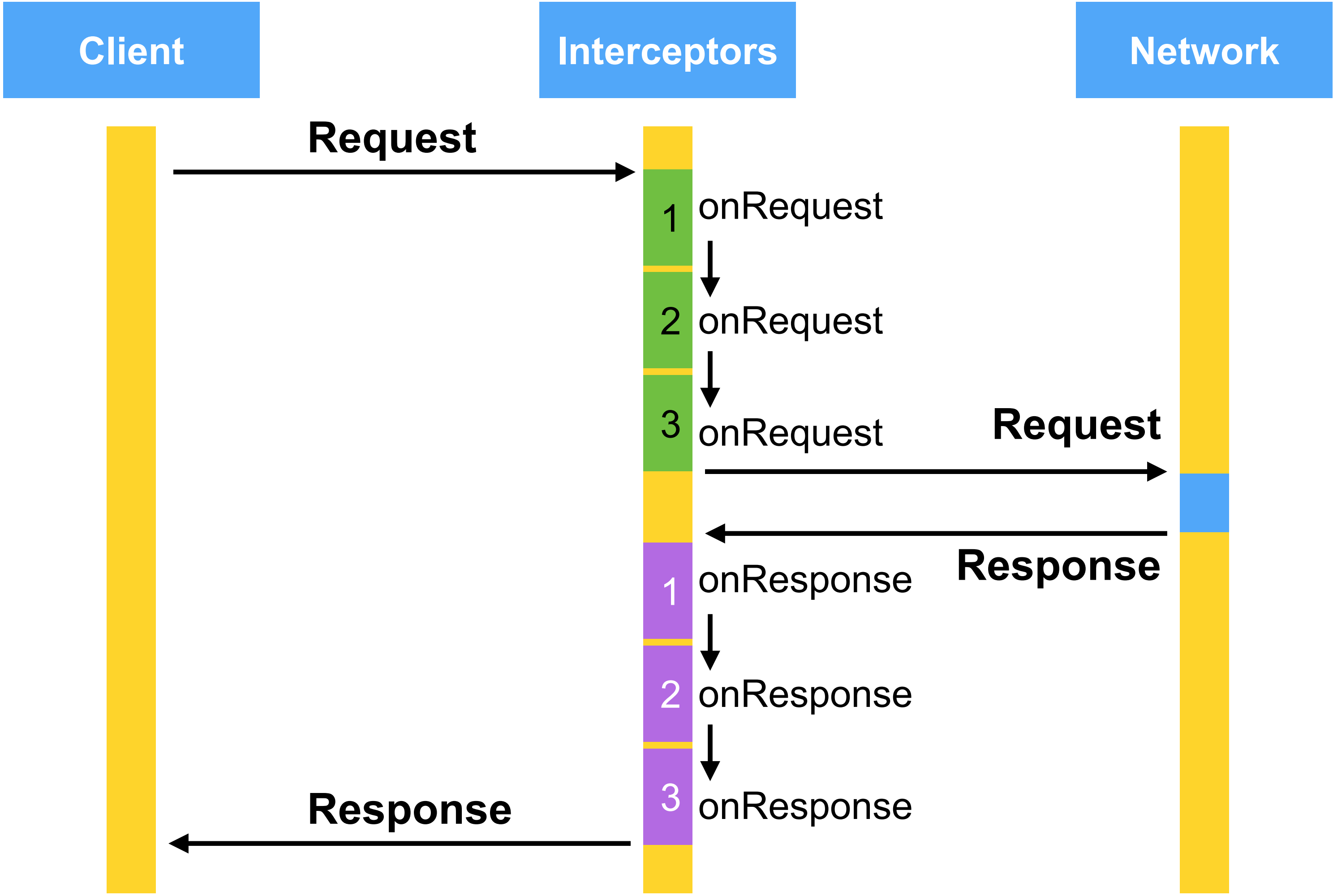
# Decerialization

```
_dio.get<Map<String,dynamic>>( '/path' );
```

# Serialization

```
_dio.post( '/path', data: {  
    'field': 'string'  
} );
```

# Dio: interceptors





# Dio: interceptors

```
class AuthInterceptor extends Interceptor {  
  
    @override  
    void onRequest(RequestOptions options, RequestInterceptorHandler handler) {  
        final newOptions = options.copyWith(  
            headers: {  
                ...options.headers,  
                'Authorization': token  
            }  
        );  
        handler.next(newOptions);  
    }  
}
```

# Dio: errors

```
try {  
    // make request  
} on DioError catch (e) {  
    // handle error  
}
```



# Thank you for attention

Sergey Koltsov, Yandex.Pro Team Lead

[ringov@yandex-team.ru](mailto:ringov@yandex-team.ru)