Сила чистой архитектуры

Изменения в слое данных

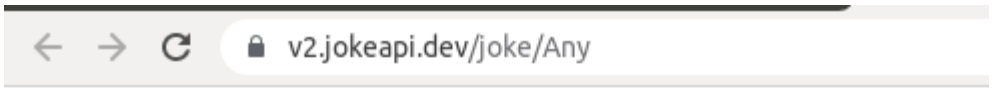
Содержание

1. Меняем апи
2. Логируем вызовы к апи

1. Меняем апи

В предыдущей лекции мы рассмотрели чистую архитектуру и сказали, что она очень сильно помогает в вопросах поддержки, масштабируемости и легко модифицировать код.

Сейчас я покажу как это работает в реальности. Давайте предпложим, что у нас поменялось апи шутки. Например теперь мы получаем данные таким образом




```
{
    "error": false,
    "category": "Pun",
    "type": "twopart",
    "setup": "What kind of car did Whitney Houston drive?",
    "delivery": "A Hyundaiiiiiiiiiii",
    "flags": {
        "nsfw": false,
        "religious": false,
        "political": false,
        "racist": false,
        "sexist": false,
        "explicit": false
    },
    "id": 67,
    "safe": true,
    "lang": "en"
}
```


Здесь у нас разница с текущим апи в урл и в самой структуре и ключах в джейсон. Но я вам покажу как это сделать настолько быстро, что вы удивитесь. Итак, давайте просто добавим новый метод в JokeService

```
interface JokeService {
    @GET( value: "https://official-joke-api.appspot.com/random_joke/")
    fun getJoke(): Call<JokeServerModel>

    @GET( value: "https://v2.jokeapi.dev/joke/Any")
    fun getNewJoke(): Call<NewJokeServerModel>
}
```

Но вы конечно же можете написать новый интерфейс где будет новый метод и просто переключиться на него в апликейшн. Теперь напишем класс серверной модели для этого.

Мне не нужны все поля ответа – я выбираю лишь те, которые мне нужны. Айди, текст и концовка. Но заметьте, что ключ у концовки другой. И поэтому я неизбежно написал целый класс – скопировал существующий и поменял ключ. Это заняло у меня 15 секунд. И так как у меня серверная модель мапится к датамодели то я переиспользую мапер и метод из существующей серверной модели.

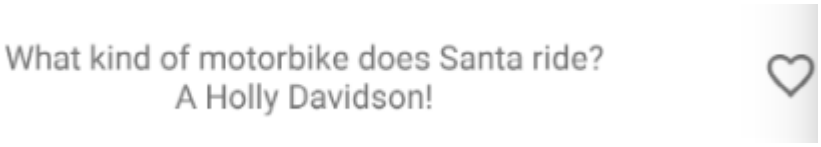
```
class NewJokeServerModel(
    @SerializedName( value: "id")
    private val id: Int,
    @SerializedName( value: "setup")
    private val text: String,
    @SerializedName( value: "delivery")
    private val punchline: String
) : Mapper<JokeDataModel> {
    override fun to() = JokeDataModel(id, text, punchline)
}
```

Ну и в конце концов просто заменяем метод в клаудДатаСторе

```
class BaseCloudDataSource(private val service: JokeService) : CloudDataSource {
    override suspend fun getJoke(): JokeDataModel {
        try {
            return service.getNewJoke().execute().body()!!.to()
        } catch (e: Exception) {
```

Вот и все! Можете запустить проект и ваше приложение сохранит работоспособность но будет уже использовать другой сервер для получения данных. Сколько у вас ушло минут? 5?

А теперь представьте, что вы не написали ни одной модели кроме серверной и даже на юай вы отдавали серверную – вам фиксить очень много классов. И это супер неочевидно. Изменения в серверной части – дата слое – не должны никаким образом влиять на код в других слоях.



Но кто-то скажет, а что если я хочу переключаться между апи и мне не надо трогать существующий клаудДатаСтор. ОК. Давайте тогда напишем следующий код.

Я выделяю интерфейс для работы с любым типом серверной модели которая мапится к датамодели. Далея я перепишу клаудДатаСорс чтобы он работал с сервисом дженерика.

```
**/
class BaseCloudDataSource(private val service: JokeService<*>) : CloudDataSource {
    override suspend fun getJoke(): JokeDataModel {
        try {
            return service.getJoke().execute().body()!!.to()
```

И теперь в апликейшн классе смогу легко заменить один сервис на другой. Смотрите как просто

```
val resourceManager = BaseResourceManager( context: this)
val cloudDataSource = BaseCloudDataSource(retrofit.create(NewJokeService::class.java))
```

```
interface BaseJokeService : JokeService<JokeServerModel> {
    @GET( value: "https://official-joke-api.appspot.com/random_joke/")
    override fun getJoke(): Call<JokeServerModel>
}

interface NewJokeService : JokeService<NewJokeServerModel> {

    @GET( value: "https://v2.jokeapi.dev/joke/Any")
    override fun getJoke(): Call<NewJokeServerModel>
}

interface JokeService<T : Mapper<JokeDataModel>> {

    fun getJoke(): Call<T>
}
```

И давайте еще раз проверим работоспособность нашего проекта

Ай, нет. В ретрофит нельзя наследовать интерфейсы. Значит нужно переписать код. Давайте уберем наследование интерфейсов сервиса для начала

```
interface BaseJokeService {
    @GET( value: "https://official-joke-api.appspot.com/random_joke/")
    fun getJoke(): Call<JokeServerModel>
}

interface NewJokeService  {

    @GET( value: "https://v2.jokeapi.dev/joke/Any")
    fun getJoke(): Call<NewJokeServerModel>
}
```

И теперь нам нужно переписать клаудДатаСтор чтобы он мог работать с любым из 2 сервисов. У них нет ничего общего, кроме того, что они оба отдают Call<T: Mapper<JokeDataModel>> и этим мы и воспользуемся.

Пишем абстрактный класс датасорса, где получаем в протектед абстрактном методе данные в том виде в котором отдает сервис. Остальное уже скопируем из существующего класса датасорса. И теперь осталось написать 2 простые реализации для каждого кейса в 1 линию

```
class NewJokeCloudDataSource(private val service: NewJokeService) :
    BaseCloudDataSource<NewJokeServerModel>() {
    override fun getJokeServerModel() = service.getJoke()
}
```

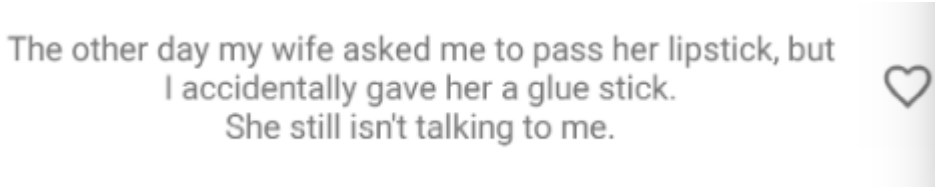
```
abstract class BaseCloudDataSource<T : Mapper<JokeDataModel>> : CloudDataSource {

    protected abstract fun getJokeServerModel(): Call<T>

    override suspend fun getJoke(): JokeDataModel {
        try {
            return getJokeServerModel().execute().body()!!.to()
        } catch (e: Exception) {
            if (e is UnknownHostException) {
                throw NoConnectionException()
            } else {
                throw ServiceUnavailableException()
```

```
class JokeCloudDataSource(private val service: BaseJokeService) :
    BaseCloudDataSource<JokeServerModel>() {
    override fun getJokeServerModel() = service.getJoke()
}
```

И теперь в апликейшн классе просто отдадим нужный клаудДатаСорс и все

```
val cloudDataSource = NewJokeCloudDataSource(retrofit.create(NewJokeService::class.java))
val repository = BaseJokeRepository(cacheDataSource, cloudDataSource, BaseCachedJoke())
val interactor = BaseJokeInteractor(repository, JokeFailureFactory(resourceManager), JokeSuccessMapper())
viewModel = BaseViewModel(interactor, BaseCommunication())
```

Очень жаль конечно что ретрофит не позволяет наследовать интерфейсы, иначе бы было бы намного проще. Но как видите и так неплохо.



Кстати смешно – жена попросила передать ей губную помаду, а я случайно передал ей клей. Она до сих пор не говорит со мной.

И кто-то скажет – а как мы можем быть уверены, что все действительно правильно работает? Хотелось бы логировать запросы-ответы!

3. Логируем вызовы к апи

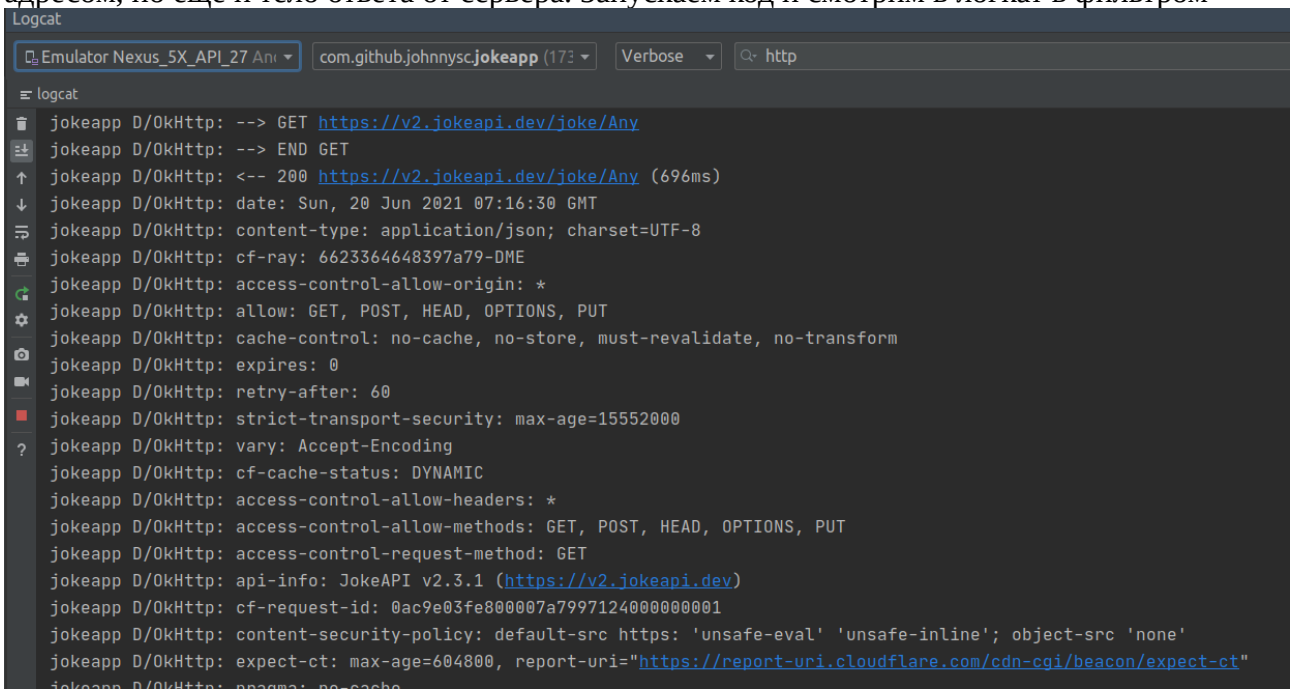
Чтобы залогировать вызовы к апи мы можем просто добавить логирующий инструмент в ретрофит. Сделать это довольно просто. Добавим либу в build.gradle

```
implementation 'com.squareup.okhttp3:logging-interceptor:4.2.1'
```

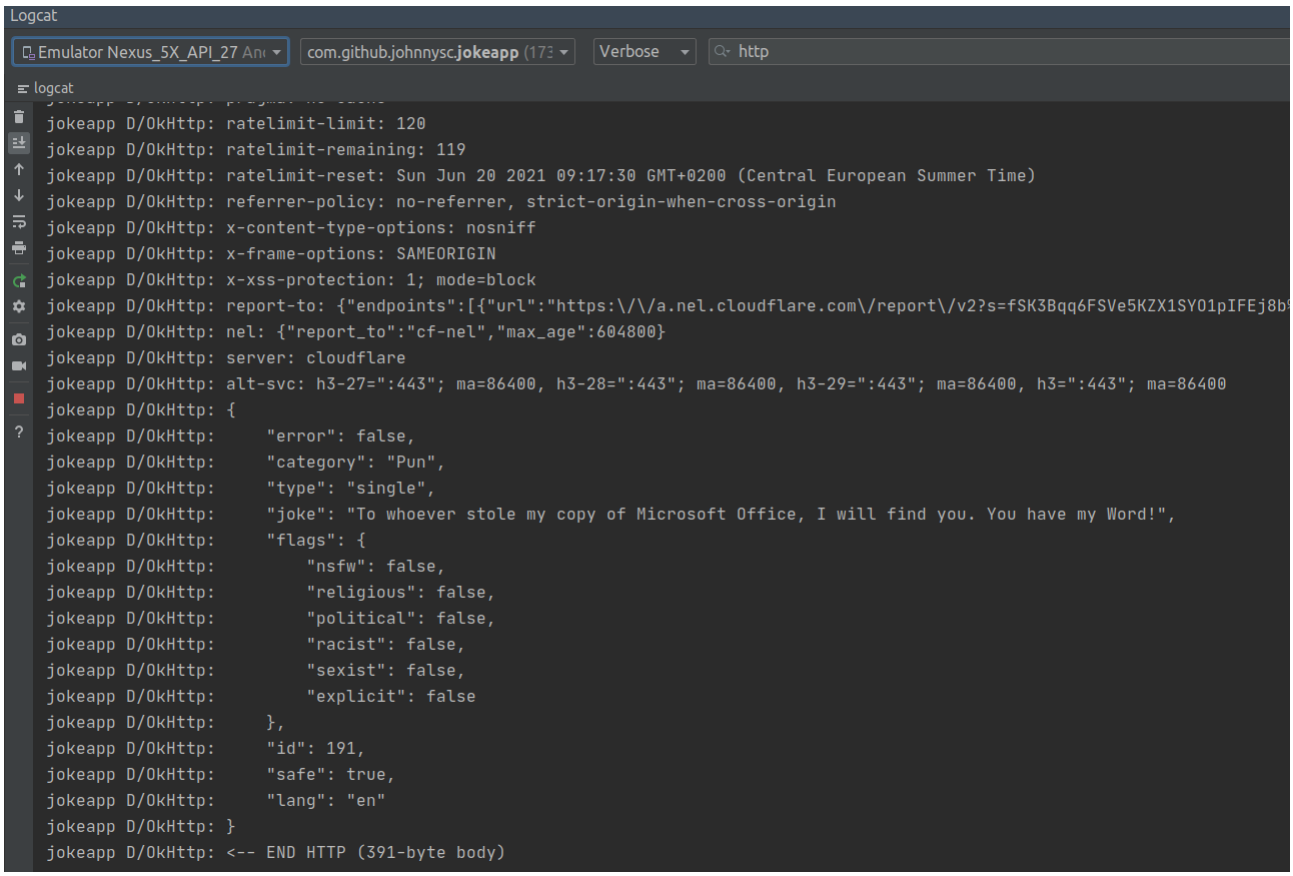
Теперь перенесемся в апликейшн класс и добавим логирование в ретрофит

```
val interceptor = HttpLoggingInterceptor()
interceptor.level = HttpLoggingInterceptor.Level.BODY
val client = OkHttpClient.Builder().addInterceptor(interceptor).build()

val retrofit = Retrofit.Builder()
    .baseUrl( baseUrl: "https://www.google.com")
    .client(client)
    .addConverterFactory(GsonConverterFactory.create())
    .build()
```

Уровень логирования Body означает что мы увидим в логах не только сами запросы к апи с адресом, но еще и тело ответа от сервера. Запускаем код и смотрим в логкат в фильтром



Get запрос, ответ 200 т.е. успешный. И прочая информация. Ниже сам ответ от сервера
Как видите весь джейсон виден в логах и теперь вы можете быть уверены что обращаетесь к нужному апи.
Кстати, вы можете определять с какими параметрами вы хотите получить шутку
документация по апи доступна по адресу https://sv443.net/jokeapi/v2/
Документация по апи первого доступна по такому адресу
https://github.com/15Dkatz/official_joke_api
В последующих лекциях мы применим все возможности этих апи и будем фильтровать шутки на отдельном экране.


Logcat
Emulator Nexus_5X_API_27
com.github.johnnysc.jokeapp
Verbose
http
logcat
jokeapp D/OkHttp: ratelimit-limit: 120
jokeapp D/OkHttp: ratelimit-remaining: 119
jokeapp D/OkHttp: ratelimit-reset: Sun Jun 20 2021 09:17:30 GMT+0200 (Central European Summer Time)
jokeapp D/OkHttp: referrer-policy: no-referrer, strict-origin-when-cross-origin
jokeapp D/OkHttp: x-content-type-options: nosniff
jokeapp D/OkHttp: x-frame-options: SAMEORIGIN
jokeapp D/OkHttp: x-xss-protection: 1; mode=block
jokeapp D/OkHttp: report-to: {"endpoints":[{"url":"https:\/\/a.nel.cloudflare.com\/report\/v2?s=fSK3Bqq6FSVe5KZX1SYO1pIFEj8b
jokeapp D/OkHttp: nel: {"report_to":"cf-nel","max_age":604800}
jokeapp D/OkHttp: server: cloudflare
jokeapp D/OkHttp: alt-svc: h3-27=":443"; ma=86400, h3-28=":443"; ma=86400, h3-29=":443"; ma=86400, h3=":443"; ma=86400
jokeapp D/OkHttp: {
jokeapp D/OkHttp: "error": false,
jokeapp D/OkHttp: "category": "Pun",
jokeapp D/OkHttp: "type": "single",
jokeapp D/OkHttp: "joke": "To whoever stole my copy of Microsoft Office, I will find you. You have my Word!",
jokeapp D/OkHttp: "flags": {
jokeapp D/OkHttp: "nsfw": false,
jokeapp D/OkHttp: "religious": false,
jokeapp D/OkHttp: "political": false,
jokeapp D/OkHttp: "racist": false,
jokeapp D/OkHttp: "sexist": false,
jokeapp D/OkHttp: "explicit": false
jokeapp D/OkHttp: },
jokeapp D/OkHttp: "id": 191,
jokeapp D/OkHttp: "safe": true,
jokeapp D/OkHttp: "lang": "en"
jokeapp D/OkHttp: }
jokeapp D/OkHttp: <-- END HTTP (391-byte body)