Retrofit2源码分析[插件化适配(线程切换/RxJava)]


BlackSwift (/u/b99b0edd4e77) ＋关注
2016.02.03 19:16* 字数 837 阅读 1185 评论 0 喜欢 8
(/u/b99b0edd4e77)

Retrofit2支持了可插件化，也就是可以让第三方的lib去包装HTTP请求。它通过 `CallAdapter` ，类似于脚本语言中的Map操作。比如说默认的线程切换与外置的RxJava就是通过CallAdapter实现的。

```
//第一步，接口的解析与拼装
Interface --Parse/Proxy--> OkHttpCall
//第二步，请求的适配化
OkHttpCall --Adapter--> RxJava/java8/UICallback
```

通过源码分析，可以发现在Retrofit中，通过接口定义对某个对象 `<T>` 的请求后，第一步是通过动态代理与注解将元数据进行解析并拼装成请求的包装类 `OkHttpCall<T>` ，第二步通过适配器实现 `OkHttpCall<T>` 到其它类型（比如RxJava等第三方库）的适配转换.

本文主要分析第二步

在动态代理等拼装完成请求后，将要调用 `invoke` 进行适配

```
Object invoke(Object... args) {
    //callAdapter来源于CallAdapter.Factory
  return callAdapter.adapt(
      new OkHttpCall<>(callFactory, requestFactory, args, responseConverter));
}
```

也就是

Input	**OkHttpCall(对OkHttp的call的包装)**
↓	↓
invoke	进行适配器转换，实现了 `adapt()` 方法
↓	↓
Output	适配后的对象，比如UI线程回调，RxJava回调等

本文主要分析以下两款主流的转换适配 (https://link.jianshu.com?t=https://github.com/square/retrofit/wiki/Adapters)，对应java对象如下，主要是对 `CallAdapter.Factory` 接口的实现：

1. DefaultCallAdapterFactory: Retrofit自带的线程回调executor，当异步请求入队时，它将自动在主线程以handler的post方式进行回调
2. RxJavaCallAdapterFactory: 用RxJava对请求进行包装，它将根据网络请求 `<T>` 生成一个 `Observable<T>` 进行流式任务执行

默认适配方案 - 跨平台处理

如果在Builder构造时没有指定适配器，Retrofit将会自动 (https://link.jianshu.com?t=https://github.com/square/retrofit/blob/3c195a0cc0c95b82b59bf3f88f6eabed3eeb3621/retrofit/src/main/java/retrofit2/Retrofit.java#L550)使用 `DefaultCallAdapterFactory` ，它可

以通过反射机制自动选择当前平台（对JRE下特殊的包名轮询）的Factory。

```
//反射调用对应的工厂，支持java8,android,ios
Platform.get().defaultCallAdapterFactory(callbackExecutor)
```

通过Platform方法 `get()` 进行寻找 (https://link.jianshu.com?t=https://github.com/square/retrofit/blob/3c195a0cc0c95b82b59bf3f88f6eabed3eeb3621/retrofit/src/main/java/retrofit2/Platform.java#L35)对应平台，当我们的jre是在android下时，将会跳转到Android (https://link.jianshu.com?t=https://github.com/square/retrofit/blob/3c195a0cc0c95b82b59bf3f88f6eabed3eeb3621/retrofit/src/main/java/retrofit2/Platform.java#L91)。 `Android` 用主线程的Handler包装了 `Executor` 方法

```
static class Android extends Platform {
  @Override CallAdapter.Factory defaultCallAdapterFactory(Executor callbackExecutor) {
    //默认retrofit构造时一般是空输入
    if (callbackExecutor == null) {
      callbackExecutor = new MainThreadExecutor();
    }
    return new ExecutorCallAdapterFactory(callbackExecutor);
  }

  static class MainThreadExecutor implements Executor {
    private final Handler handler = new Handler(Looper.getMainLooper());

    @Override public void execute(Runnable r) {
      handler.post(r);
    }
  }
}
```

ExecutorCallAdapterFactory (https://link.jianshu.com?t=https://github.com/square/retrofit/blob/3c195a0cc0c95b82b59bf3f88f6eabed3eeb3621/retrofit/src/main/java/retrofit2/ExecutorCallAdapterFactory.java#L24-L24)是在任何平台下都使用的线程切换回调工具，搜索它的 `adapt()` 方法，我们发现实现了一个ExecutorCallbackCall (https://link.jianshu.com?t=https://github.com/square/retrofit/blob/3c195a0cc0c95b82b59bf3f88f6eabed3eeb3621/retrofit/src/main/java/retrofit2/ExecutorCallAdapterFactory.java#L48)对象，它的enqueue方法如下

```
@Override public void enqueue(final Callback<T> callback) {
    //delegate就是要进行适配的的OkHttpCall
    //它开始入队把任务交给OkHttp内部线程池处理
  delegate.enqueue(new Callback<T>() {
    //这里在OkHttp内部线程池回调
    @Override public void onResponse(final Call<T> call, final Response<T> response) {
        //在UI线程回调，也就是用刚才的handler进行post发送
      callbackExecutor.execute(new Runnable() {
        @Override public void run() {
          if (delegate.isCanceled()) {
            // Emulate OkHttp's behavior of throwing/delivering an IOException on cancelatio
            callback.onFailure(call, new IOException("Canceled"));
          } else {
            callback.onResponse(call, response);
          }
        }
      });
    }

    @Override public void onFailure(final Call<T> call, final Throwable t) {
      callbackExecutor.execute(new Runnable() {
        @Override public void run() {
          callback.onFailure(call, t);
        }
      });
    }
  });
}
```

默认的适配方案就是通过对OkHttpCall进行了一次线程再包装，在OkHttp内部网络线程池处理结束后，用 `MainThreadExecutor` 调用原生MQ发送实际的回调，这就是Retrofit的线程切换原理，android下切换线程也太简单了。

> OkHttpCall是对rawCall的包装，它的入队基于okhttp，具体可以看这里 (https://www.jianshu.com/p/6637369d02e7)

RxJava转换

当使用RxJava时，可以将OkHttpCall转换为Observable对象，以进行更多的流操作。在构造时，需要添加 `CallAdapterFactory`

```
new Retrofit.Builder()
    .addCallAdapterFactory(RxJavaCallAdapterFactory.create())
    ....
    .create();
```

接口需要这样构造

```
public interface GitHub {
  @GET("/repos/{owner}/{repo}/contributors")
  Observable<List<Contributor>> contributors(
      @Path("owner") String owner,
      @Path("repo") String repo);
}
```

我们同样还是分析它的 `adapt()` 方法，在这里 (https://link.jianshu.com?t=https://github.com/square/retrofit/blob/3c195a0cc0c95b82b59bf3f88f6eabed3eeb3621/retrofit-adapters/rxjava/src/main/java/retrofit2/adapter/rxjava/RxJavaCallAdapterFactory.java#L145)，函数中泛型比较多，这里可以把 `<R>` 看作 `<List<Contributor>` 进行分析

```
@Override public <R> Observable<R> adapt(Call<R> call) {
  //创建了RxJava对象,并进行同步网络请求(execute())
  return Observable.create(new CallOnSubscribe<>(call))
      //将 Response对象转换为List<Contributor>
      .flatMap(new Func1<Response<R>, Observable<R>>() {
        @Override public Observable<R> call(Response<R> response) {
          if (response.isSuccess()) {
            //返回List<Contributor>
            return Observable.just(response.body());
          }
          return Observable.error(new HttpException(response));
        }
      });
}
```

CallOnSubscribe对网络请求进行了包装 (https://link.jianshu.com?t=https://github.com/square/retrofit/blob/3c195a0cc0c95b82b59bf3f88f6eabed3eeb3621/retrofit-adapters/rxjava/src/main/java/retrofit2/adapter/rxjava/RxJavaCallAdapterFactory.java#L99)，生成了可以观察的 `Observable<Response>` ，接着进行了flatmap操作，将泛型 `<T>` 转换成了 `Observable<List< Contributor>>` ，接着我们就可以在业务代码中使用Rx流了。

综上，我们可以得出结论：通过对传入的Call进行适配包装，实现请求的RxJava化。

总结

通过适配模式，可以实现：

1. 改变/包装对象的类型，以适应第三方（比如rxjava，jdk8，guava）
2. 实现线程间切换回调（比如android下主线程回调）