Injeção de Dependência com Dagger

iMasters Android Dev Conference 2015

Felipe Lima

@felipecsl

airbnb

Injeção de Dependência

Injeção de Dependência

É um Software Design Pattern

Todo e qualquer sistema possui algum tipo de injeção de dependência

Hollywood principle: "Don't call us, we'll call you”

Frequentemente negligenciado

```java
Retrofit retrofit = new Retrofit.Builder()
    .baseUrl(baseUrl)
    .client(client)
    .callbackExecutor(mainThreadExecutor)
    .addCallAdapterFactory(callAdapterFactory)
    .addConverterFactory(converterFactory)
    .build();
```

```java
HttpUrl baseUrl = HttpUrl.parse("http://10.0.3.2:3000/");

Retrofit retrofit = new Retrofit.Builder()
    .baseUrl(baseUrl)
    .client(client)
    .callbackExecutor(mainThreadExecutor)
    .addCallAdapterFactory(callAdapterFactory)
    .addConverterFactory(converterFactory)
    .build();
```

```
HttpUrl baseUrl = HttpUrl.parse("http://10.0.3.2:3000/");
OkHttpClient client = new OkHttpClient();

Retrofit retrofit = new Retrofit.Builder()
    .baseUrl(baseUrl)
    .client(client)
    .callbackExecutor(mainThreadExecutor)
    .addCallAdapterFactory(callAdapterFactory)
    .addConverterFactory(converterFactory)
    .build();
```

```
HttpUrl baseUrl = HttpUrl.parse(“http://10.0.3.2:3000/");
OkHttpClient client = new OkHttpClient().setCache(cache);
File cacheDir = new File(getCacheDir(), "okhttp");
Cache cache = new Cache(cacheDir, 20L * 1024 * 1024);

Retrofit retrofit = new Retrofit.Builder()
    .baseUrl(baseUrl)
    .client(client)
    .callbackExecutor(mainThreadExecutor)
    .addCallAdapterFactory(callAdapterFactory)
    .addConverterFactory(converterFactory)
    .build();
```

```
HttpUrl baseUrl = HttpUrl.parse(“http://10.0.3.2:3000/");
File cacheDir = new File(getCacheDir(), "okhttp");
Cache cache = new Cache(cacheDir, 20L * 1024 * 1024);
OkHttpClient client = new OkHttpClient().setCache(cache);
ConcurrentUtil.MainThreadExecutor callbackExecutor =
    new ConcurrentUtil.MainThreadExecutor();

Retrofit retrofit = new Retrofit.Builder()
    .baseUrl(baseUrl)
    .client(client)
    .callbackExecutor(callbackExecutor)
    .addCallAdapterFactory(callAdapterFactory)
    .addConverterFactory(converterFactory)
    .build();
```

```
HttpUrl baseUrl = HttpUrl.parse("http://10.0.3.2:3000/");
File cacheDir = new File(getCacheDir(), "okhttp");
Cache cache = new Cache(cacheDir, 20L * 1024 * 1024);
OkHttpClient client = new OkHttpClient().setCache(cache);
ConcurrentUtil.MainThreadExecutor callbackExecutor =
    new ConcurrentUtil.MainThreadExecutor();
RxJavaCallAdapterFactory callAdapterFactory =
    RxJavaCallAdapterFactory.create();

Retrofit retrofit = new Retrofit.Builder()
    .baseUrl(baseUrl)
    .client(client)
    .callbackExecutor(callbackExecutor)
    .addCallAdapterFactory(callAdapterFactory)
    .addConverterFactory(converterFactory)
    .build();
```

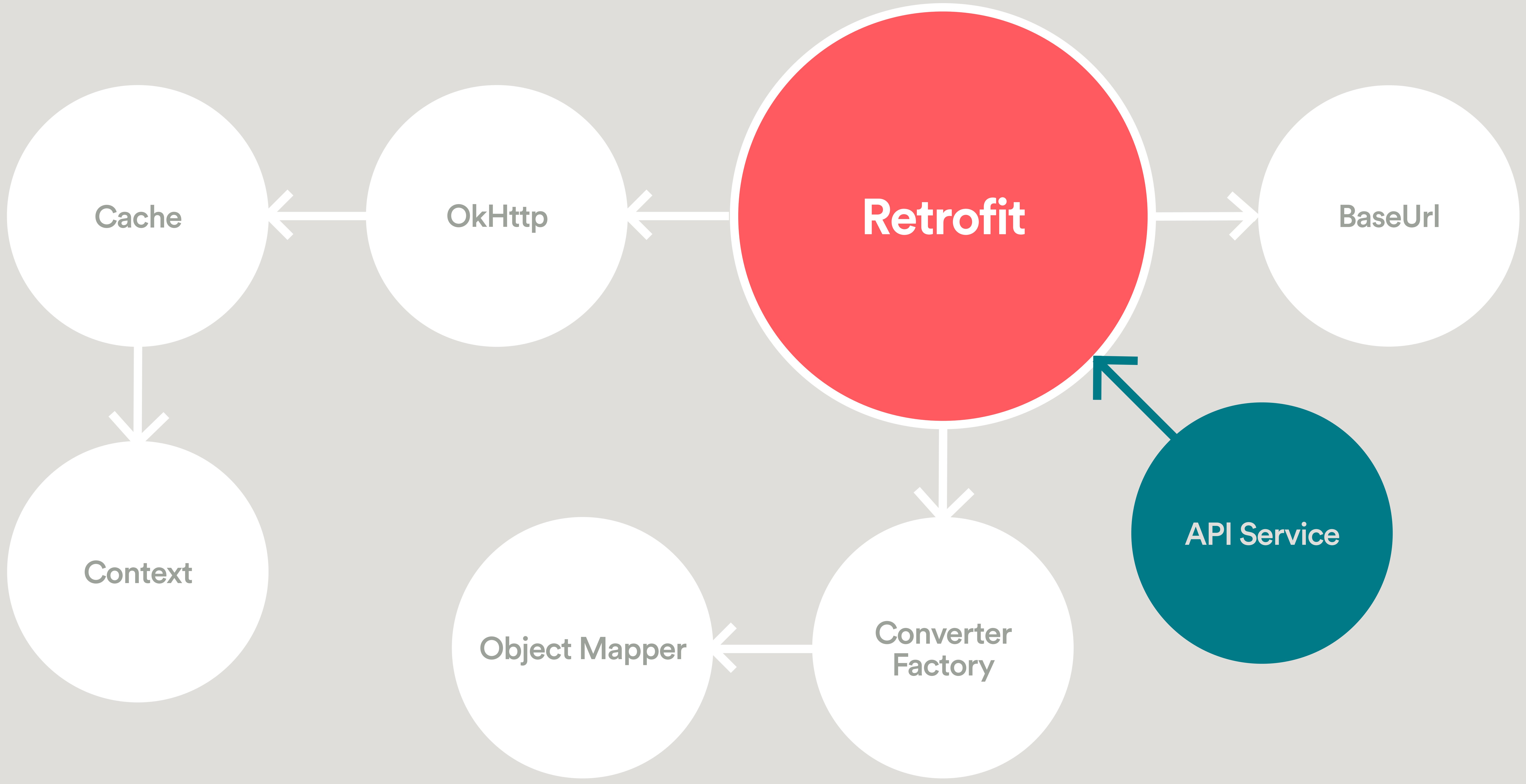
```java
HttpUrl baseUrl = HttpUrl.parse("http://10.0.3.2:3000/");
File cacheDir = new File(getCacheDir(), "okhttp");
Cache cache = new Cache(cacheDir, 20L * 1024 * 1024);
OkHttpClient client = new OkHttpClient().setCache(cache);
ConcurrentUtil.MainThreadExecutor callbackExecutor =
    new ConcurrentUtil.MainThreadExecutor();
RxJavaCallAdapterFactory callAdapterFactory =
    RxJavaCallAdapterFactory.create();
ObjectMapper objectMapper = new ObjectMapper();
Converter.Factory converterFactory =
    JacksonConverterFactory.create(objectMapper);

Retrofit retrofit = new Retrofit.Builder()
    .baseUrl(baseUrl)
    .client(client)
    .callbackExecutor(callbackExecutor)
    .addCallAdapterFactory(callAdapterFactory)
    .addConverterFactory(converterFactory)
    .build();
```

```java
HttpUrl baseUrl = HttpUrl.parse("http://10.0.3.2:3000/");
File cacheDir = new File(getCacheDir(), "okhttp");
Cache cache = new Cache(cacheDir, 20L * 1024 * 1024);
OkHttpClient client = new OkHttpClient().setCache(cache);
ConcurrentUtil.MainThreadExecutor callbackExecutor =
    new ConcurrentUtil.MainThreadExecutor();
RxJavaCallAdapterFactory callAdapterFactory =
RxJavaCallAdapterFactory.create();
ObjectMapper objectMapper = new ObjectMapper();
Converter.Factory converterFactory =
JacksonConverterFactory.create(objectMapper);

Retrofit retrofit = new Retrofit.Builder()
    .baseUrl(baseUrl)
    .client(client)
    .callbackExecutor(callbackExecutor)
    .addCallAdapterFactory(callAdapterFactory)
    .addConverterFactory(converterFactory)
    .build();

ElifutService service = retrofit.create(ElifutService.class);
```

Boilerplate

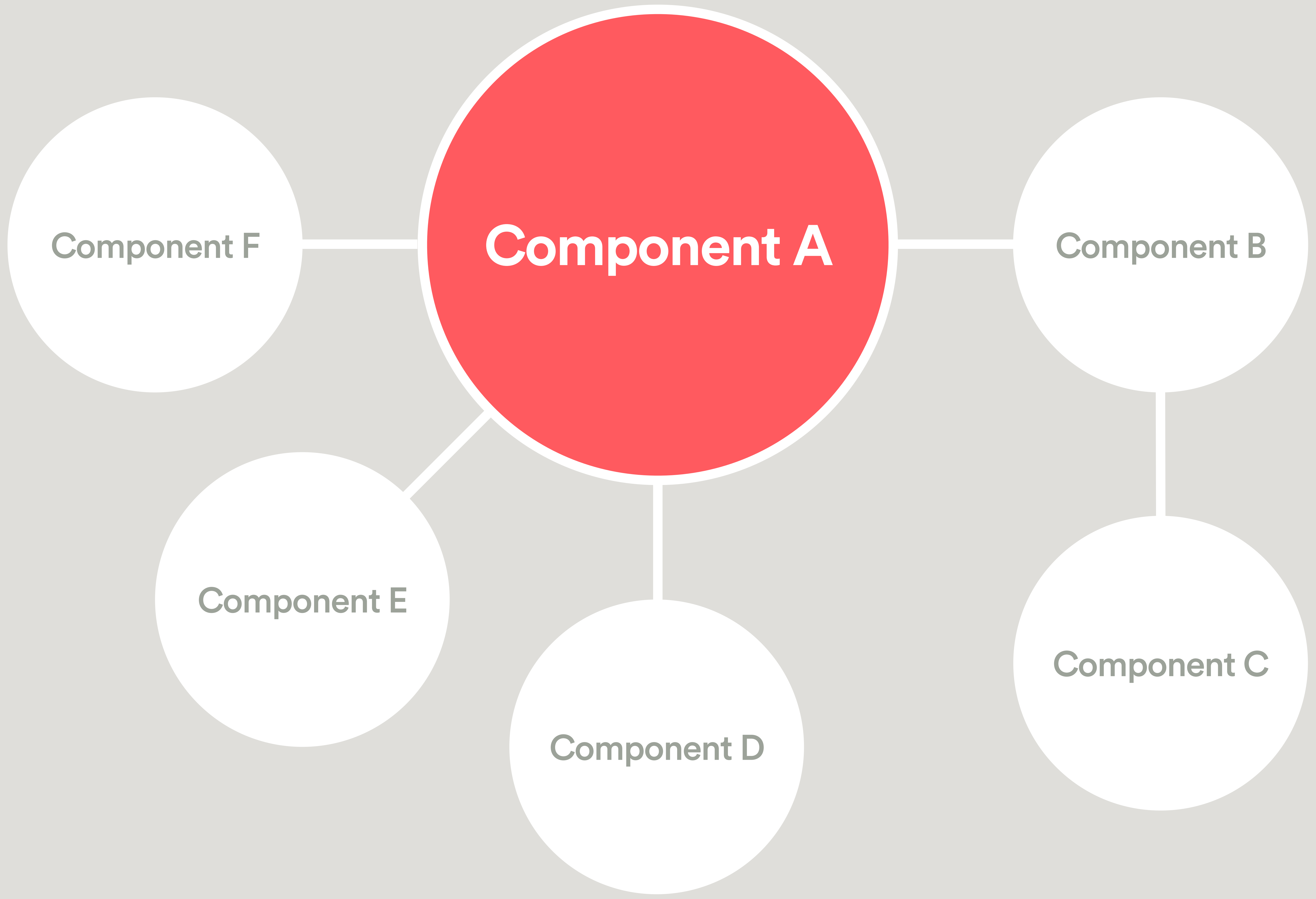
Não é bacana :(

http://cdn.movieweb.com/img.news/NEnBKLdGEGLarg_1_1.jpg


Cache
OkHttp
Retrofit
BaseUrl
Context
API Service
Object Mapper
Converter Factory

Não é este tipo de adaga...

http://runescape.wikia.com/wiki/Steel_dagger



Directed Acyclic Graph

“DAG"-ger

História

História e Linha do Tempo

Spring
Baseado em XML, validado em tempo de execução.

200?

Guice
Baseado em Annotations e 100% Java. Criado pelo Google.

2006

Dagger 1
Evolução do Guice. Mais simples e focado em performance. Criado pela Square.

2013

Dagger 2
Fork do Dagger 1 com foco em performance. Criado pelo Google.

2014

API

Voltando ao nosso exemplo...

```java
HttpUrl baseUrl = HttpUrl.parse("http://10.0.3.2:3000/");
File cacheDir = new File(getCacheDir(), "okhttp");
Cache cache = new Cache(cacheDir, 20L * 1024 * 1024);
OkHttpClient client = new OkHttpClient().setCache(cache);
ConcurrentUtil.MainThreadExecutor callbackExecutor =
    new ConcurrentUtil.MainThreadExecutor();
RxJavaCallAdapterFactory callAdapterFactory =
RxJavaCallAdapterFactory.create();
ObjectMapper objectMapper = new ObjectMapper();
Converter.Factory converterFactory =
JacksonConverterFactory.create(objectMapper);

Retrofit retrofit = new Retrofit.Builder()
    .baseUrl(baseUrl)
    .client(client)
    .callbackExecutor(callbackExecutor)
    .addCallAdapterFactory(callAdapterFactory)
    .addConverterFactory(converterFactory)
    .build();
ElifutService service = retrofit.create(ElifutService.class);
```

```java
public class MainActivity extends Activity {

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
  }
}
```

```java
public class MainActivity extends Activity {
  ElifutService service;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
  }
}
```

```
public class MainActivity extends Activity {
  @Inject ElifutService service;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
  }
}
```

```
public class NetworkModule {

}
```

```java
@Module
public class NetworkModule {

}
```

```java
@Module
public class NetworkModule {
  Retrofit provideRetrofit() {

  }
}
```

```
@Module
public class NetworkModule {
  @Provides
  Retrofit provideRetrofit() {

  }
}
```

```
@Module
public class NetworkModule {
  @Provides
  Retrofit provideRetrofit() {
    return new Retrofit.Builder()
        .baseUrl(HttpUrl.parse("http://10.0.3.2:3000/"))
        .build();
  }
}
```

```java
@Module
public class NetworkModule {
  @Provides
  Retrofit provideRetrofit() {
    return new Retrofit.Builder()
        .baseUrl(HttpUrl.parse("http://10.0.3.2:3000/"))
        .build();
  }

  @Provides
  ElifutService provideService(Retrofit retrofit) {
    return retrofit.create(ElifutService.class);
  }
}
```

```java
public interface ElifutComponent {

}
```

```java
@Component
public interface ElifutComponent {

}
```

```java
@Component(modules = { NetworkModule.class })
public interface ElifutComponent {

}
```

```
@Component(modules = { NetworkModule.class })
public interface ElifutComponent {
  ElifutService service();
}
```

```java
@Singleton
@Component(modules = { NetworkModule.class })
public interface ElifutComponent {
  ElifutService service();
}
```

```java
@Singleton
@Component(modules = { NetworkModule.class })
public interface ElifutComponent {
  ElifutService service();
  void inject(MainActivity mainActivity);
}
```

```java
@Singleton
@Component(modules = { NetworkModule.class })
public interface ElifutComponent {
  ElifutService service();
  void inject(MainActivity mainActivity);

  class Initializer {
    static ElifutComponent init() {
      return DaggerElifutComponent.builder()
          .networkModule(new NetworkModule())
          .build();
    }
  }
}
```

```java
public class MainActivity extends Activity {
  @Inject ElifutService service;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    ElifutComponent.Initializer.init().inject(this);
  }
}
```

Recapitulando

1

`@Module`

Agrupa dependências

2

`@Provides`

Fornece dependências

3

`@Inject`

Solicita dependências

4

`@Component`

“Cola” entre módulos e injeções

Colocando tudo junto…

```java
@Module
public class NetworkModule {
  private final Context context;

  public NetworkModule(Context context) {
    this.context = context;
  }

  @Provides @Singleton ElifutService provideService(Retrofit retrofit) {...}

  @Provides @Singleton Retrofit provideRetrofit(OkHttpClient client, HttpUrl baseUrl,
      Executor callbackExecutor, Converter.Factory converterFactory, CallAdapter.Factory factory) {...}

  @Provides @Singleton Executor provideExecutor() {...}

  @Provides @Singleton CallAdapter.Factory provideCallAdapterFactory() {...}

  @Provides @Singleton HttpUrl provideBaseUrl() { return HttpUrl.parse("http://10.0.3.2:3000/"); }

  @Provides @Singleton Converter.Factory provideConverterFactory(ObjectMapper objectMapper) {...}

  @Provides @Singleton ObjectMapper provideObjectMapper() {...}

  @Provides @Singleton Cache provideCache() {...}

  @Provides @Singleton OkHttpClient provideOkHttpClient(Cache cache) {...}
}
```

```java
public class ElifutApplication extends Application {
  private ElifutComponent component;

  @Override
  public void onCreate() {
    super.onCreate();
    component = ElifutComponent.Initializer.init(this);
  }

  public ElifutComponent component() {
    return component;
  }
}
```

```java
public class MainActivity extends Activity {
  @Inject ElifutService service;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    ElifutApplication application = (ElifutApplication) getApplication();
    application.component().inject(this);
  }
}
```

Tipos de Injeção

HTTP://WWW.BOASNOTICIAS.PT/IMG/1323866894COLESTROL.JPG

Via Construtor

Mecanismo mais simples e intuitivo

Anotação `@Inject` em um construtor da classe

Cada argumento é uma dependência

Dependências podem ser armazenadas em membros `private` e `final`

Classe injetada fica implicitamente disponível para injeções subsequentes

```java
public class SampleClass {
  private final int anInteger;
  private final String aString;

  @Inject
  public SampleClass(int anInteger, String aString) {
    this.anInteger = anInteger;
    this.aString = aString;
  }
}
```

Via Membros da Classe

Anotação `@Inject` em membros da classe

Membros não podem ser privados ou final

Injeção acontece depois de o objeto ter sido instanciado

Método `inject(this)` deve ser chamado explicitamente para ocorrer a injeção

Útil em casos onde não é possível injetar diretamente via construtor (ex.: `Activity` e `Fragment`)

```java
public class MainActivity extends Activity {
  @Inject ElifutService service;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    ElifutApplication application = (ElifutApplication) getApplication();
    application.component().inject(this);
  }
}
```

Via Método

Anotação `@Inject` em métodos da classe

Parâmetros do método são dependências

É o caso de uso menos comum dos três

Código gerado

```java
@Generated("dagger.internal.codegen.ComponentProcessor")
public final class DaggerElifutComponent implements ElifutComponent {
  private Provider<ElifutService> provideServiceProvider;
  private MembersInjector<MainActivity> mainActivityMembersInjector;

  private DaggerElifutComponent(Builder builder) {
    assert builder != null;
    initialize(builder);
  }

  private void initialize(final Builder builder) {
    this.provideServiceProvider = ScopedProvider.create(
      NetworkModule_ProvideServiceFactory.create(
        builder.networkModule, provideRetrofitProvider));
    this.mainActivityMembersInjector =
      MainActivity_MembersInjector.create((MembersInjector)
        MembersInjectors.noOp(), provideServiceProvider);
  }

  @Override
  public void inject(MainActivity mainActivity) {
    mainActivityMembersInjector.injectMembers(mainActivity);
  }
```

```java
@Generated("dagger.internal.codegen.ComponentProcessor")
public final class DaggerElifutComponent implements ElifutComponent {
  private Provider<ElifutService> provideServiceProvider;
  private MembersInjector<MainActivity> mainActivityMembersInjector;

  private DaggerElifutComponent(Builder builder) {
    assert builder != null;
    initialize(builder);
  }

  private void initialize(final Builder builder) {
    this.provideServiceProvider = ScopedProvider.create(
      NetworkModule_ProvideServiceFactory.create(
        builder.networkModule, provideRetrofitProvider));
    this.mainActivityMembersInjector =
      MainActivity_MembersInjector.create((MembersInjector)
        MembersInjectors.noOp(), provideServiceProvider);
  }

  @Override
  public void inject(MainActivity mainActivity) {
    mainActivityMembersInjector.injectMembers(mainActivity);
  }
```

```java
@Generated("dagger.internal.codegen.ComponentProcessor")
public final class NetworkModule_ProvideServiceFactory implements
    Factory<ElifutService> {
  private final NetworkModule module;
  private final Provider<Retrofit> retrofitProvider;

  public NetworkModule_ProvideServiceFactory(NetworkModule module,
      Provider<Retrofit> retrofitProvider) {
    this.module = module;
    this.retrofitProvider = retrofitProvider;
  }

  @Override
  public ElifutService get() {
    return module.provideService(retrofitProvider.get());
  }

  public static Factory<ElifutService> create(NetworkModule module,
      Provider<Retrofit> retrofitProvider) {
    return new NetworkModule_ProvideServiceFactory(module,
      retrofitProvider);
  }
}
```

```java
@Generated("dagger.internal.codegen.ComponentProcessor")
public final class NetworkModule_ProvideServiceFactory implements
    Factory<ElifutService> {
  private final NetworkModule module;
  private final Provider<Retrofit> retrofitProvider;

  public NetworkModule_ProvideServiceFactory(NetworkModule module,
      Provider<Retrofit> retrofitProvider) {
    this.module = module;
    this.retrofitProvider = retrofitProvider;
  }

  @Override
  public ElifutService get() {
    return module.provideService(retrofitProvider.get());
  }

  public static Factory<ElifutService> create(NetworkModule module,
      Provider<Retrofit> retrofitProvider) {
    return new NetworkModule_ProvideServiceFactory(module,
      retrofitProvider);
  }
}
```

airbnb

FEEDBACK? FELIPE.LIMA@GMAIL.COM

Obrigado!