Dependency Injection
Dagger 2

DI & Dagger 2

Модно

Используется повсеместно

Вызывает сложности

Dependency Injection

```java
public class PaymentApi {

    private String url = "tinkoff.payment.ru";
    private OkHttpClient client = new OkHttpClient().newBuilder()
            .connectTimeout(30, TimeUnit.SECONDS)
            .build();

    public void pay(BigDecimal amount) {
        Request request = // create request
        client.newCall(request).execute();
    }
}
```

```java
public class PaymentApi {

    //...

    public void anotherUrlPay(BigDecimal amount) {
        OkHttpClient client = new OkHttpClient().newBuilder()
                .connectTimeout(40, TimeUnit.SECONDS)
                .build();
        String url = "tinkoff.payment-another.ru";
        Request request = // create request
        client.newCall(request).execute();
    }
}
```

```
public class PaymentApi {

    //...

    private String anotherUrl = "tinkoff.payment-another.ru";
    private OkHttpClient anotherClient = new OkHttpClient().newBuilder()
            .connectTimeout(40, TimeUnit.SECONDS)
            .build();

    public void anotherUrlPay(BigDecimal amount) {
        Request request = // create request with anotherUrl
        anotherClient.newCall(request).execute();
    }
}
```

```java
public class PaymentAnotherApi {

    private String url = "tinkoff.payment-another.ru";
    private OkHttpClient client = new OkHttpClient().newBuilder()
            .connectTimeout(40, TimeUnit.SECONDS)
            .build();

    public void pay(BigDecimal amount) {
        Request request = // create request
        client.newCall(request).execute();
    }
}
```

```java
public class PaymentApi {

    private String url = "tinkoff.payment.ru";
    private OkHttpClient client = new OkHttpClient().newBuilder()
            .connectTimeout(30, TimeUnit.SECONDS)
            .build();

    public void pay(BigDecimal amount) {
        Request request = // create request
        client.newCall(request).execute();
    }
}
```

```java
public class PaymentApi {

    private final String url;
    private final OkHttpClient client;

    public PaymentApi(String url, OkHttpClient client) {
        this.url = url;
        this.client = client;
    }

    public void pay(BigDecimal amount) {
        Request request = // create request
        client.newCall(request).execute();
    }
}
```

```
public class PaymentApi {

    private final String url;
    private final OkHttpClient client;

    public PaymentApi(String url, OkHttpClient client) {
        this.url = url;
        this.client = client;
    }

    public void pay(BigDecimal amount) {
        Request request = // create request
        client.newCall(request).execute();
    }
}
```

```
public class PaymentApi {

    private final String url;
    private final OkHttpClient client;

    public PaymentApi(String url, OkHttpClient client) {
        this.url = url;
        this.client = client;
    }

    public void pay(BigDecimal amount) {
        Request request = // create request
        client.newCall(request).execute();
    }
}
```

Что это дало?

Менее связный код

Упрощенное тестирование

```
String url = "tinkoff.payment-another.ru";
OkHttpClient client = new OkHttpClient().newBuilder()
        .connectTimeout(40, TimeUnit.SECONDS)
        .build();

new PaymentApi(url, client).pay(BigDecimal.TEN);
```

```
String url = "tinkoff.payment-another.ru";
OkHttpClient client = new OkHttpClient().newBuilder()
        .connectTimeout(40, TimeUnit.SECONDS)
        .build();

new PaymentApi(url, client).pay(BigDecimal.TEN);
```

Dagger 2 = Directed Acyclic Graph + ger + 2

Инструмент для реализации DI

Хорошо подходит для Android

Работает на кодогенерации

Поддержка от Google

Пять **основных** аннотаций


```
@Inject

@Module

@Provides

@Component

@Scope
```

Декларирует требование зависимости

Три вида Inject

В конструктор

В поле

В метод

Inject в конструктор

```
public class PaymentApi {

    private final String url;
    private final OkHttpClient client;

    @Inject
    public PaymentApi(String url, OkHttpClient client) {
        this.url = url;
        this.client = client;
    }

}
```

Inject в конструктор

```
public class PaymentApi {

    private final String url;
    private final OkHttpClient client;

    @Inject
    public PaymentApi(String url, OkHttpClient client) {
        this.url = url;
        this.client = client;
    }

}
```

Inject в поле

```
public class PaymentApi {

    @Inject
    String url;
    @Inject
    OkHttpClient client;

}
```

Inject в поле

```
public class PaymentApi {

    @Inject
    private final String url;
    @Inject
    private final OkHttpClient client;

}
```

Inject в поле

```
public class PaymentApi {

    @Inject
    String url;
    @Inject
    OkHttpClient client;

    @Inject
    public PaymentApi() {
        // empty
    }

}
```

Inject в поле

```
public class PaymentApi {

    @Inject
    String url;
    @Inject
    OkHttpClient client;

}


void inject(PaymentApi paymentApi);
```

Inject в метод

```
public class PaymentApi {

    private String url;
    private OkHttpClient client;

    @Inject
    void inject(String url, OkHttpClient client) {
        this.url = url;
        this.client = client;
    }
}
```

Inject в метод

```
private EventsCreator eventsCreator;

@Inject
void inject(EventsCreator eventsCreator) {
    eventsCreator.registerListener(this);
    this.eventsCreator = eventsCreator;
}
```

Inject в метод

```
private EventsCreator eventsCreator;

@Inject
void inject(EventsCreator eventsCreator) {
    eventsCreator.registerListener(this);
    this.eventsCreator = eventsCreator;
}
```

@Module + @Provides

Определяют сами зависимости

```java
@Module
public class PaymentModule {

    @Provides
    OkHttpClient provideOkHttpClient() {
        return new OkHttpClient().newBuilder()
                .connectTimeout(30, TimeUnit.SECONDS)
                .build();
    }

    @Provides
    String providePaymentUrl() {
        return "tinkoff.payment.ru";
    }
}
```

```
@Module
public class PaymentModule {

    @Provides
    OkHttpClient provideOkHttpClient() {
        return new OkHttpClient().newBuilder()
                .connectTimeout(30, TimeUnit.SECONDS)
                .build();
    }

    @Provides
    String providePaymentUrl() {
        return "tinkoff.payment.ru";
    }
}
```

```
@Module
public class PaymentModule {

    @Provides
    OkHttpClient provideOkHttpClient() {
        return new OkHttpClient().newBuilder()
                .connectTimeout(30, TimeUnit.SECONDS)
                .build();
    }

    @Provides
    String providePaymentUrl() {
        return "tinkoff.payment.ru";
    }
}
```

```java
@Module
public class PaymentModule {

    @Provides
    OkHttpClient provideOkHttpClient() {
        return new OkHttpClient().newBuilder()
                .connectTimeout(30, TimeUnit.SECONDS)
                .build();
    }

    @Provides
    String providePaymentUrl() {
        return "tinkoff.payment.ru";
    }

    @Provides
    PaymentApi providePaymentApi(String url, OkHttpClient client) {
        return new PaymentApi(url, client);
    }
}
```

```
@Module
public class PaymentModule {

    @Provides
    OkHttpClient provideOkHttpClient() {
        return new OkHttpClient().newBuilder()
                .connectTimeout(30, TimeUnit.SECONDS)
                .build();
    }

    @Provides
    String providePaymentUrl() {
        return "tinkoff.payment.ru";
    }

    @Provides
    PaymentApi providePaymentApi(String url, OkHttpClient client) {
        return new PaymentApi(url, client);
    }
}
```

```java
@Provides
PaymentApi providePaymentApi(String url, OkHttpClient client) {
    return new PaymentApi(url, client);
}
```

```java
@Inject
public PaymentApi(String url, OkHttpClient client) {
    this.url = url;
    this.client = client;
}
```

```java
@Provides
PaymentApi providePaymentApi(String url, OkHttpClient client) {
    return new PaymentApi(url, client);
}
```

```java
@Inject
public PaymentApi(String url, OkHttpClient client) {
    this.url = url;
    this.client = client;
}
```

@Component

@Component

Осуществляет «инъекции» зависимостей

```
@Component(modules = PaymentModule.class)
public interface PaymentComponent {


}
```

```
@Component(modules = PaymentModule.class)
public interface PaymentComponent {



}
```

```
@Component(modules = PaymentModule.class)
public interface PaymentComponent {



}
```

```java
@Component(modules = PaymentModule.class)
public interface PaymentComponent {

    PaymentApi paymentApi();

}
```

```
PaymentComponent paymentComponent = DaggerPaymentComponent
        .builder()
        .paymentModule(new PaymentModule())
        .build();
PaymentApi paymentApi = paymentComponent.paymentApi();
```

```
PaymentComponent paymentComponent = DaggerPaymentComponent
        .builder()
        .paymentModule(new PaymentModule())
        .build();
PaymentApi paymentApi = paymentComponent.paymentApi();
```

```
@Component(modules = PaymentModule.class)
public interface PaymentComponent {

    PaymentApi paymentApi();
    void inject(PaymentActivity activity);

}
```

```java
public class PaymentActivity extends AppCompatActivity {

    @Inject
    PaymentApi paymentApi;

    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        PaymentComponent paymentComponent = DaggerPaymentComponent
                .builder()
                .paymentModule(new PaymentModule())
                .build();
        paymentComponent.inject(this);
    }
}
```

```
public class PaymentActivity extends AppCompatActivity {

    @Inject
    PaymentApi paymentApi;

    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        PaymentComponent paymentComponent = DaggerPaymentComponent
                .builder()
                .paymentModule(new PaymentModule())
                .build();
        paymentComponent.inject(this);
    }
}
```

```
public class PaymentActivity extends AppCompatActivity {

    @Inject
    PaymentApi paymentApi;

    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        PaymentComponent paymentComponent =
                DaggerPaymentComponent.create()
        paymentComponent.inject(this);
    }
}
```

@Scope

Определяет «время жизни» объекта

```
@Scope
public @interface ActivityScope {
}
```

```
@Scope
public @interface ActivityScope {
}
```

```java
@ActivityScope
@Component(modules = PaymentModule.class)
public interface PaymentComponent {

    void inject(PaymentActivity activity);

}
```

```java
@Module
public class PaymentModule {

    @Provides
    @ActivityScope
    OkHttpClient provideOkHttpClient() {
        return new OkHttpClient().newBuilder()
                .connectTimeout(30, TimeUnit.SECONDS)
                .build();
    }

}
```

```java
@Module
public class PaymentModule {

    @Provides
    @ActivityScope
    OkHttpClient provideOkHttpClient() {
        return new OkHttpClient().newBuilder()
                .connectTimeout(30, TimeUnit.SECONDS)
                .build();
    }

    @Provides
    String providePaymentUrl() {
        return "tinkoff.payment.ru";
    }

}
```

```
@Module
public class PaymentModule {

    @Provides
    @ActivityScope
    OkHttpClient provideOkHttpClient() {
        return new OkHttpClient().newBuilder()
                .connectTimeout(30, TimeUnit.SECONDS)
                .build();
    }

    @Provides
    String providePaymentUrl() {
        return "tinkoff.payment.ru";
    }

}
```

```
public class PaymentActivity extends AppCompatActivity {

    @Inject
    String url;
    @Inject
    OkHttpClient okHttpClient;

    //...
}
```

```java
public class PaymentActivity extends AppCompatActivity {

    @Inject
    String url;
    @Inject
    OkHttpClient okHttpClient;

    //...
}
```

```java
@Inject
public PaymentApi(String url, OkHttpClient client) {
    this.url = url;
    this.client = client;
}
```

```
public class PaymentActivity extends AppCompatActivity {

    @Inject
    String url;
    @Inject
    OkHttpClient okHttpClient;

    //...
}
```

```
@Inject
public PaymentApi(String url, OkHttpClient client) {
    this.url = url;
    this.client = client;
}
```

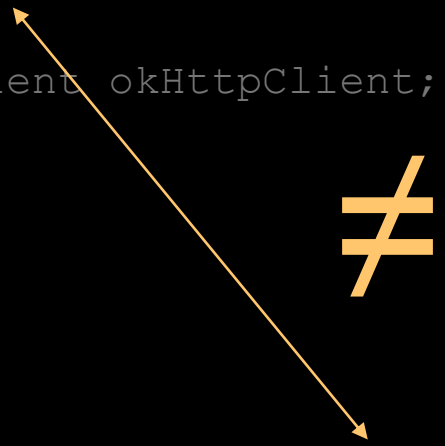
```
public class PaymentActivity extends AppCompatActivity {

    @Inject
    String url;
    @Inject
    OkHttpClient okHttpClient;

    //...
}
```

```
@Inject
 public PaymentApi(String url, OkHttpClient client) {
     this.url = url;
     this.client = client;
 }
```

```
public class PaymentActivity extends AppCompatActivity {

    @Inject
    String url;
    @Inject
    OkHttpClient okHttpClient;

    //...
}
```

```
@Inject
public PaymentApi(String url, OkHttpClient client) {
    this.url = url;
    this.client = client;
}
```

```
public class PaymentActivity extends AppCompatActivity {

    @Inject
    String url;
    @Inject
    OkHttpClient okHttpClient;

    //...
}
```

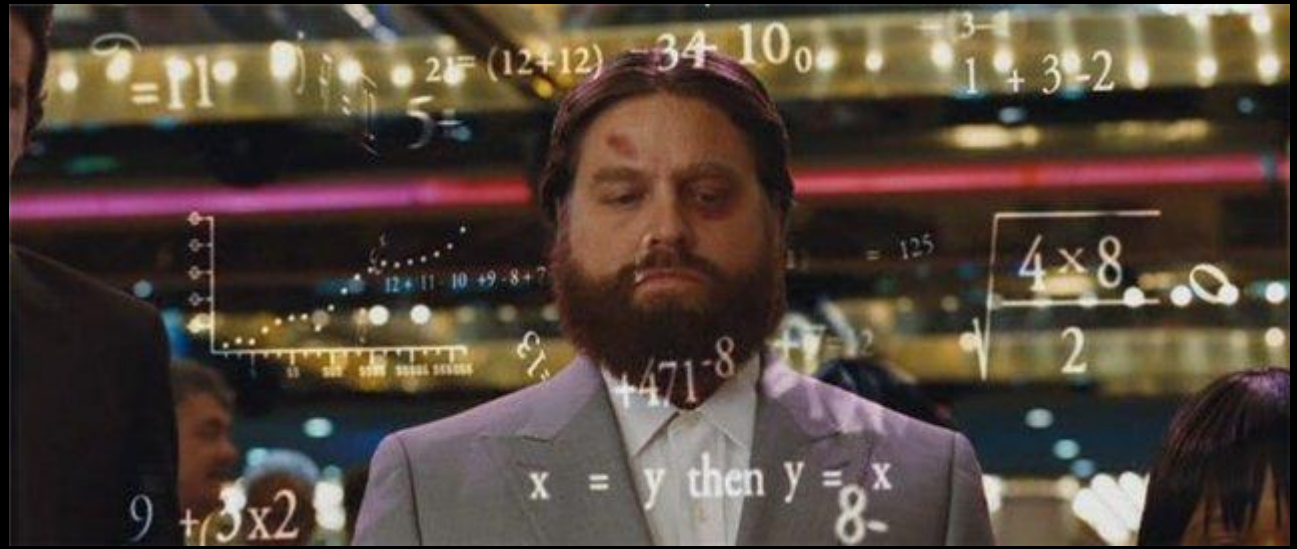


≠

```
@Inject
 public PaymentApi(String url, OkHttpClient client) {
     this.url = url;
     this.client = client;
 }
```

```java
@ActivityScope
public class PaymentApi {

    private final String url;
    private final OkHttpClient client;

    @Inject
    public PaymentApi(String url, OkHttpClient client) {
        this.url = url;
        this.client = client;
    }

}
```

Что в итоге получилось?



```java
@Module
public class PaymentModule {

    @Provides
    @ActivityScope
    OkHttpClient provideOkHttpClient() {
        return new OkHttpClient().newBuilder()
            .connectTimeout(30, TimeUnit.SECONDS)
            .build();
    }

    @Provides
    String providePaymentUrl() {
        return "tinkoff.payment.ru";
    }
}
```

```java
@ActivityScope
public class PaymentApi {

    private final String url;
    private final OkHttpClient client;

    @Inject
    public PaymentApi(
          String url,
          OkHttpClient client
    ) {
        this.url = url;
        this.client = client;
    }
}
```

```java
@ActivityScope
@Component(modules = PaymentModule.class)
public interface PaymentComponent {

    void inject(PaymentActivity activity);

}


public class PaymentActivity extends AppCompatActivity {

    @Inject
    PaymentApi paymentApi;

    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        PaymentComponent paymentComponent = DaggerPaymentComponent.create();
        paymentComponent.inject(this);
    }

}
```

Преимущества Dagger 2

- Compile-time проверки
- No-reflection
- Нет задержек при инициализации приложения
- Поддержка от Google

Недостатки Dagger 2

- Compile-time проверки
- Boiler-plate
- Документация

Другие решения

Guice

Dagger 1

Toothpick

Koin

Kodein

Продвинутые технологии Dagger 2

- Lazy, Provider
- @Qualifier
- @Subcomponent vs @Component-dependencies
- (Sub-)Component Builders + @BindsInstance
- @Binds + static @Provides

Lazy

```
public class PaymentApi {

    private final Lazy<OkHttpClient> lazyClient;

    @Inject
    public PaymentApi(Lazy<OkHttpClient> lazyClient) {
        this.lazyClient = lazyClient;
    }

    void pay() {
        OkHttpClient client = lazyClient.get();
        //...
    }

}
```

Lazy

```
public class PaymentApi {

    private final Lazy<OkHttpClient> lazyClient;

    @Inject
    public PaymentApi(Lazy<OkHttpClient> lazyClient) {
        this.lazyClient = lazyClient;
    }

    void pay() {
        OkHttpClient client = lazyClient.get();
        //...
    }

}
```

Provider

```
public class PaymentApi {

    private final Provider<OkHttpClient> clientProvider;

    @Inject
    public PaymentApi(Provider<OkHttpClient> clientProvider) {
        this.clientProvider = clientProvider;
    }

    void pay() {
        OkHttpClient client = clientProvider.get();
        //...
    }

}
```

Provider

```
public class PaymentApi {

    private final Provider<OkHttpClient> clientProvider;

    @Inject
    public PaymentApi(Provider<OkHttpClient> clientProvider) {
        this.clientProvider = clientProvider;
    }

    void pay() {
        OkHttpClient client = clientProvider.get();
        //...
    }

}
```

Qualifier

```
@Provides
String providePaymentUrl() {
    return "tinkoff.payment.ru";
}

@Provides
@Named("AnotherUrl")
String providePaymentAnotherUrl() {
    return "tinkoff.payment-another.ru";
}



@Inject
public PaymentApi(@Named("AnotherUrl") String url)
```

Qualifier

```
@Provides
String providePaymentUrl() {
    return "tinkoff.payment.ru";
}

@Provides
@Named("AnotherUrl")
String providePaymentAnotherUrl() {
    return "tinkoff.payment-another.ru";
}



@Inject
public PaymentApi(@Named("AnotherUrl") String url)
```

Qualifier

```
@Qualifier
public @interface CustomQualifier {
}


@Provides
@CustomQualifier
String providePaymentAnotherUrl() {
    return "tinkoff.payment-another.ru";
}


@Inject
public PaymentApi(@CustomQualifier String url)
```

@Subcomponent

```
@Singleton
@Component(modules = AppModule.class)
public interface AppComponent {

    PaymentComponent paymentComponent(PaymentModule paymentModule);

}

@Module
public class AppModule {

    @Provides
    @Singleton
    public OkHttpClient provideOkHttpClient() {
        return new OkHttpClient().newBuilder()
                .connectTimeout(30, TimeUnit.SECONDS)
                .build();
    }
}
```

@Subcomponent

```
@ActivityScope
@Subcomponent(modules = PaymentModule.class)
public interface PaymentComponent {

    void inject(PaymentActivity activity);

}


public class PaymentActivity extends AppCompatActivity {

    @Inject
    OkHttpClient client;

    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        PaymentComponent paymentComponent = App.appComponent
                .paymentComponent(new PaymentModule());
        paymentComponent.inject(this);
    }
}
```

@Subcomponent

```
@ActivityScope
@Subcomponent(modules = PaymentModule.class)
public interface PaymentComponent {

    void inject(PaymentActivity activity);

}


public class PaymentActivity extends AppCompatActivity {

    @Inject
    OkHttpClient client;

    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        PaymentComponent paymentComponent = App.appComponent
                .paymentComponent(new PaymentModule());
        paymentComponent.inject(this);
    }
}
```

Dependent Component

```
@Singleton
@Component(modules = AppModule.class)
public interface AppComponent {

    OkHttpClient provideOkHttpClient();

}
```

```
@ActivityScope
@Component(
        dependencies = AppComponent.class,
        modules = PaymentModule.class
)
public interface PaymentComponent {

    void inject(PaymentActivity activity);

}
```

Dependent Component

```
@Singleton
@Component(modules = AppModule.class)
public interface AppComponent {

    OkHttpClient provideOkHttpClient();

}
```

```
@ActivityScope
@Component(
        dependencies = AppComponent.class,
        modules = PaymentModule.class
)
public interface PaymentComponent {

    void inject(PaymentActivity activity);

}
```

Dependent Component

```
public class PaymentActivity extends AppCompatActivity {

    @Inject
    OkHttpClient client;

    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        PaymentComponent paymentComponent = DaggerPaymentComponent.builder()
                .appComponent(App.appComponent)
                .build();
        paymentComponent.inject(this);
    }
}
```

Dependent Component

```
public class PaymentActivity extends AppCompatActivity {

    @Inject
    OkHttpClient client;

    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        PaymentComponent paymentComponent = DaggerPaymentComponent.builder()
                .appComponent(App.appComponent)
                .build();
        paymentComponent.inject(this);
    }
}
```

Subcomponent vs Dependent Component

https://proandroiddev.com/dagger-2-component-relationships-custom-scopes-8d7e05e70a37

https://google.github.io/dagger/subcomponents

Component Builder

```
@ActivityScope
@Component(modules = PaymentModule.class)
public interface PaymentComponent {

    void inject(PaymentActivity activity);

    @Component.Builder
    interface Builder {

        Builder paymentModule(PaymentModule paymentModule);

        PaymentComponent build();

    }
}
```

@BindsInstance

```
@Component.Builder
interface Builder {

    Builder paymentModule(PaymentModule paymentModule);

    @BindsInstance
    Builder paymentUrl(String url);

    PaymentComponent build();

}
```

@Binds

```
class PaymentApiImpl implements PaymentApi {

    @Inject
    public PaymentApiImpl() {}

}
```

```
interface PaymentApi {
}
```

```
@Module
abstract class PaymentModule {

    @Binds
    abstract PaymentApi providePaymentApi(PaymentApiImpl impl);

}
```

@Provides static

```
@Module
abstract class PaymentModule {

    @Binds
    abstract PaymentApi providePaymentApi(PaymentApiImpl impl);

    @Provides
    static String providePaymentUrl(){
        return "tinkoff.payment.ru";
    }

}
```

Итоги

DI == менее связный и проще тестируемый код

Dagger 2 – наиболее распространенный фрэймворк для реализации DI

Для освоения Dagger 2 необходима практика

@Inject

@Module

@Provides

@Component

@Scope

Материалы

- https://habr.com/post/343248/
- https://startandroid.ru/ru/courses/dagger-2/16-course/dagger2/424-urok-1.html
- https://google.github.io/dagger/users-guide.html
- https://habr.com/post/279125/
- https://proandroiddev.com/dagger-2-android-defeat-the-dahaka-b1c542233efc
- https://google.github.io/dagger/users-guide.html
- https://www.youtube.com/watch?v=plK0zyRLIP8
- https://github.com/square/dagger-intellij-plugin

Спасибо за внимание