https://medium.com/androiddevelopers/a-pragmatic-guide-to-hilt-with-kotlin-a76859c324a1

Прагматичное руководство по Hilt с Kotlin

Простой способ использовать внедрение зависимостей в вашем приложении для Android

[Филип Станис](https://medium.com/@fstanis?source=post_page-----a76859c324a1--------------------------------)

[Филип Станис](https://medium.com/@fstanis?source=post_page-----a76859c324a1--------------------------------)

следить

[13 ноя.](https://medium.com/androiddevelopers/a-pragmatic-guide-to-hilt-with-kotlin-a76859c324a1?source=post_page-----a76859c324a1--------------------------------) · Читать 6 мин.

[Hilt](https://developer.android.com/training/dependency-injection/hilt-android) - это новая библиотека внедрения зависимостей, построенная на основе [Dagger,](https://developer.android.com/training/dependency-injection/dagger-basics) которая упрощает ее использование в приложениях Android. Это руководство демонстрирует основные функции с помощью нескольких фрагментов кода, которые помогут вам начать использовать Hilt в своем проекте.

Настройка Hilt

Чтобы настроить Hilt в своем приложении, сначала следуйте руководству по [настройке Gradle Build](https://dagger.dev/hilt/gradle-setup) .

После установки всех зависимостей и плагинов добавьте в свой Applicationкласс аннотацию, @HiltAndroidAppчтобы использовать Hilt. Вам не нужно ничего делать или иным образом напрямую вызывать его.

Определение и внедрение зависимостей

Когда вы пишете код, использующий внедрение зависимостей, нужно подумать о двух основных компонентах:

1. Классы, у которых есть зависимости, которые вы хотите внедрить.
2. Классы, которые можно вводить как зависимости.

Они не исключают друг друга, и во многих случаях ваш класс является одновременно инъекционным и имеет зависимости.

Сделайте зависимость инъекционной

Чтобы сделать что-то в Hilt для инъекций, вы должны указать Hilt, как создать экземпляр этой штуки. Эти инструкции называются *привязками* .

Есть три способа определить привязку в Hilt.

1. Аннотируйте конструктор с помощью @Inject
2. Использование @Bindsв модуле
3. Использование @Providesв модуле

⮕ **Аннотируйте конструктор с помощью@Inject**

Любой класс может иметь аннотированный конструктор, @Injectкоторый делает его доступным в качестве зависимости в любом месте вашего проекта.

⮕ **Использование модуля**

Два других способа создания инъекций в Hilt включают использование модулей.

Модуль [Hilt](https://dagger.dev/hilt/modules) можно рассматривать как набор «рецептов», которые сообщают Hilt, как создать экземпляр чего-то, что не имеет конструктора - например, интерфейса или системной службы.

Кроме того, в ваших тестах любой модуль можно заменить другим модулем. Это упрощает, например, замену реализаций интерфейса на моки.

Модули устанавливаются в [компонент Hilt,](https://dagger.dev/hilt/components.html) указанный с помощью @InstallInаннотации. Я объясню это более подробно позже.

*Вариант 1: используйте @Bindsдля создания привязки для интерфейса*

Если вы хотите использовать OatMilkв своем коде Milkпо запросу, создайте абстрактный метод внутри модуля и аннотируйте его с помощью @Binds. Обратите внимание, что это OatMilkдолжно быть само по себе инъекционным, чтобы это работало, чего вы можете добиться, аннотируя его конструктор с помощью @Inject.

*Вариант 2: используйте @Providesдля создания заводской функции*

Если экземпляр не может быть построен напрямую, вы можете создать поставщика. Провайдер - это фабричная функция, возвращающая экземпляр объекта.

Примером этого является системная служба, ConnectivityManagerкоторую необходимо получить из контекста.

По Contextумолчанию объект является инъекционным, если вы аннотируете его с помощью @ApplicationContextили @ActivityContext.

Внедрить зависимость

Как только ваши зависимости станут инъекционными, вы можете внедрить их с помощью Hilt двумя способами.

1. Как параметры конструктора
2. Как поля

**⮕ В качестве параметров конструктора**

Если конструктор отмечен значком @Inject, Hilt вводит все параметры в соответствии с привязками, которые вы определили для этих типов.

**⮕ Как поля**

Если класс является точкой входа, указанной здесь с помощью @AndroidEntryPointаннотации (подробнее об этом в следующем разделе), @Injectвводятся все поля, помеченные значком.

Поля, помеченные значком, @Injectдолжны быть общедоступными. Также удобно сделать их так, lateinitчтобы они не допускали значения NULL, так как их начальное значение до внедрения null.

Обратите внимание, что внедрение зависимостей в виде полей полезно только тогда, когда в вашем классе должен быть конструктор без параметров, например Activity. В большинстве случаев вместо этого вы захотите внедрить через параметры конструктора.

Другие важные концепции

Точка входа

Помните, как я сказал, что во многих случаях ваш класс создается путем внедрения, *и в* него вводятся зависимости? В некоторых случаях у вас будет класс, который не был *создан* посредством внедрения зависимостей, но все же в него были внедрены зависимости. Хорошим примером этого являются действия, которые обычно создаются платформой Android, а не Hilt.

Эти классы являются [*точками входа*](https://dagger.dev/hilt/entry-points.html) в граф зависимостей Hilt, и Hilt необходимо знать, что у них есть зависимости, которые необходимо внедрить.

**⮕ Android точка входа**

Большинство ваших точек входа будут одной из так называемых [точек входа Android](https://dagger.dev/hilt/android-entry-point.html) :

* Деятельность
* Фрагмент
* Посмотреть
* обслуживание
* BroadcastReceiver

Если это так, добавьте к нему аннотацию @AndroidEntryPoint.

**⮕ Другие точки входа**

Большинству приложений когда-либо нужны только точки входа Android, но если вы взаимодействуете с библиотеками, отличными от Dagger, или компонентами Android, которые еще не поддерживаются в Hilt, вам может потребоваться создать собственную точку входа для доступа к графику Hilt вручную. Вы можете узнать больше о [превращении произвольных классов в точки входа](https://dagger.dev/hilt/entry-points.html) .

ViewModel

A ViewModel- это особый случай: он не создается напрямую, поскольку платформа должна их создавать, но также не является точкой входа Android. Вместо этого ViewModelиспользуйте специальную @ViewModelInjectаннотацию, которая позволяет Hilt вводить в них зависимости, когда они создаются с использованием by viewModels(), аналогично тому, как это @Injectработает для других классов.

Если вам нужен доступ к сохраненному состоянию в вашем ViewModel, введите a [SavedStateHandle](https://developer.android.com/reference/androidx/lifecycle/SavedStateHandle)в качестве параметра конструктора, добавив @Assistedаннотацию.

Для использования @ViewModelInjectвам нужно добавить еще несколько зависимостей. Дополнительные сведения см. В разделе « [Интеграция Hilt и Jetpack»](https://developer.android.com/training/dependency-injection/hilt-jetpack) .

Составные части

Каждый модуль устанавливается внутри [компонента Hilt](https://dagger.dev/hilt/components.html) , указанного с помощью . Компонент модуля в основном используется для предотвращения случайного внедрения зависимости в неправильное место. Например, это предотвратит использование привязок и поставщиков в аннотированном модуле в действии.@InstallIn(*<component>*)@InstallIn(ServiceComponent.class)

Кроме того, привязка может быть привязана к компоненту, в котором находится модуль. Это подводит меня к ...

Области применения

По умолчанию привязки не имеют области действия. В приведенном выше примере это означает, что каждый раз, когда вы вводите Milk, вы получаете новый экземпляр OatMilk. Если вы добавите @ActivityScopedаннотацию, вы включите привязку к ActivityComponent.

Теперь, когда ваш модуль задан, Hilt создаст только один для OatMilkкаждого экземпляра действия. Кроме того, этот OatMilkэкземпляр будет привязан к жизненному циклу этого действия - он будет создан при onCreate()вызове действия и уничтожен при onDestroy()вызове действия.

В этом случае оба milkи moreMilkбудут указывать на один и тот же OatMilkэкземпляр. Однако, если у вас несколько экземпляров LatteActivity, у каждого из них будет свой экземпляр OatMilk.

Соответственно, другие зависимости, внедренные в это действие, имеют ту же область видимости, и поэтому они тоже будут использовать один и тот же экземпляр OatMilk:

Область действия зависит от компонента, в котором установлен ваш модуль, например, @ActivityScopedможет применяться только к привязкам внутри модуля, который установлен внутри ActivityComponent.

Область также определяет жизненный цикл внедренных экземпляров: в этом случае единственный экземпляр, Milkиспользуемый Fridgeи LatteActivityсоздается, когда onCreate()вызывается LatteActivity- и уничтожается в его onDestroy(). Это также означает, что наша Milkкоманда не «переживет» изменение конфигурации, так как это потребует вызова onDestroy()активности. Вы можете преодолеть это, используя область действия с более длительным жизненным циклом, например @ActivityRetainedScope.

Список доступных областей действия, компонентов, которым они соответствуют, и соответствующих жизненных циклов, которым они следуют, см. В разделе [Компоненты Hilt](https://dagger.dev/hilt/components.html) .

Внедрение провайдера

Иногда вам нужен более прямой контроль над созданием внедренных экземпляров. Например, вы можете захотеть внедрить один или несколько экземпляров чего-либо только тогда, когда это необходимо, в зависимости от вашей бизнес-логики. В этом случае вы можете использовать [dagger.Provider](https://dagger.dev/api/latest/dagger/Provider.html).

Внедрение провайдера можно использовать независимо от того, какова зависимость и как вы ее вводите. Все, что можно ввести, можно обернуть внутрь, Provider<…>чтобы вместо этого использовалось внедрение поставщика.

Фреймворки внедрения зависимостей (такие как Dagger и [Guice](https://github.com/google/guice) ) традиционно ассоциируются с большими и сложными проектами. Однако, поскольку Hilt легко начать работу и прост в настройке, он объединяет всю мощь Dagger в пакете, который может быть легко использован любым типом приложений, независимо от того, насколько мала или велика кодовая база, в которой оно находится.

Если вы хотите узнать больше о Hilt, о том, как он работает, и о других функциях, которые могут оказаться полезными, перейдите на [его официальный сайт,](https://dagger.dev/hilt/) где вы можете найти более подробный обзор и справочную документацию.