Room 中的数据库关系

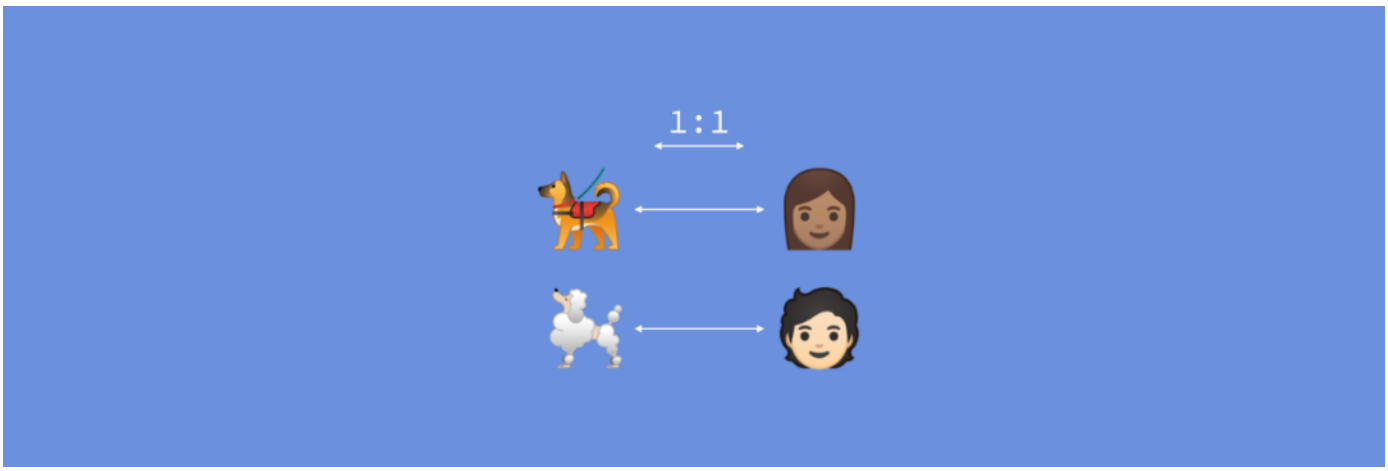
谷歌开发者
已认证的官方帐号

已关注

2 人赞同了该文章

设计一个关系型数据库很重要的一部分是将数据拆分成具有相关关系的数据表，然后将数据以符合这种关系的逻辑方式整合到一起。从Room 2.2的稳定版开始，我们可利用一个@Relation注解来支持表之间所有可能出现的关系: 一对一、一对多和多对多。

一对一关系



一对一关系

假设我们生活在一个每个人只能拥有一只狗，且每只狗只能有一个主人的 “悲惨世界” 中，这就是一对一关系。如果要以关系型数据库的方式来反应它的话，我们可以创建两张表:Dog表和Owner表，其中Dog表通过 owner id 来引用Owner表中的数据，或者Owner表通过 dog id 来引用Dog表中的数据。

```
@Entity
data class Dog(
    @PrimaryKey val dogId: Long,
    val dogOwnerId: Long,
    val name: String,
    val cuteness: Int,
    val barkVolume: Int,
    val breed: String
)

@Entity
data class Owner(@PrimaryKey val ownerId: Long, val name: String)
```

假设我们想在一个列表中展示所有的狗和它们的主人，我们需要创建一个DogAndOwner类:

```
data class DogAndOwner(
    val owner: Owner,
    val dog: Dog
)
```

为了在 SQLite 中进行查询，我们需要 1) 运行两个查询: 一个获取所有的主人数据，一个获取所有的狗狗数据，2) 根据 owner id 来进行数据的关系映射。

```
SELECT * FROM Owner

SELECT * FROM Dog WHERE dogOwnerId IN (ownerId1, ownerId2, …)
```

要在 Room 中获取一个List<DogAndOwner>，我们不需要自己去实现上面说的查询和映射，只需要使用@Relation注解。

在我们的示例中，由于Dog有了 owner 的信息，我们给 dog 变量增加@Relation注解，指定父级 (这里对应Owner) 上的ownerId列对应dogOwnerId:

```
data class DogAndOwner(
    @Embedded val owner: Owner,
    @Relation(
         parentColumn = "ownerId",
         entityColumn = "dogOwnerId"
    )
    val dog: Dog
)
```

现在我们的Dao类可被简化成:

```
@Transaction
@Query("SELECT * FROM Owner")

fun getDogsAndOwners(): List<DogAndOwner>
```

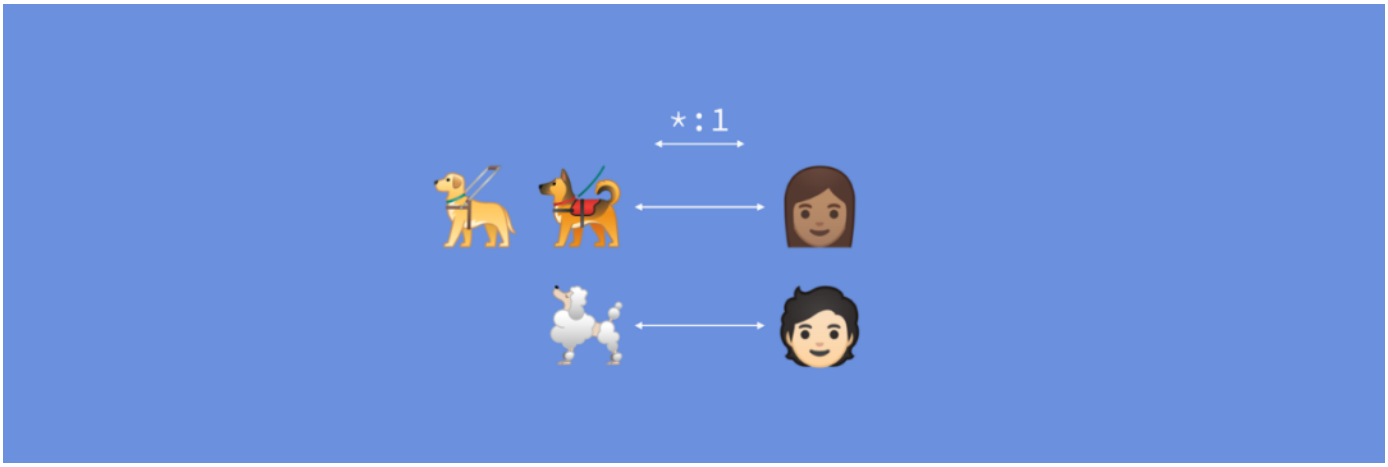
注意: 由于 Room 会默默的帮我们运行两个查询请求，因此需要增加@Transaction注解来确保这个行为是原子性的。

- Dao

- @Transaction

Transaction | Android 开发者 | Android Developers
developer.android.google.cn

一对多关系



一对多关系

再假设，一个主人可以养多只狗狗，现在上面的关系就变成了一对多关系。我们之前定义的数据库schema 并不需要改变，仍然使用同样的表结构，因为在 “多” 这一方的表中已经有了关联键。

现在，要展示狗和主人的列表，我们需要创建一个新的类来进行建模:

```
data class OwnerWithDogs(
    val owner: Owner,
    val dogs: List<Dog>
)
```

为了避免运行两个独立的查询，我们可以在Dog和Owner中定义一对多的关系，同样，还是在List<Dog>前增加@Relation注解。

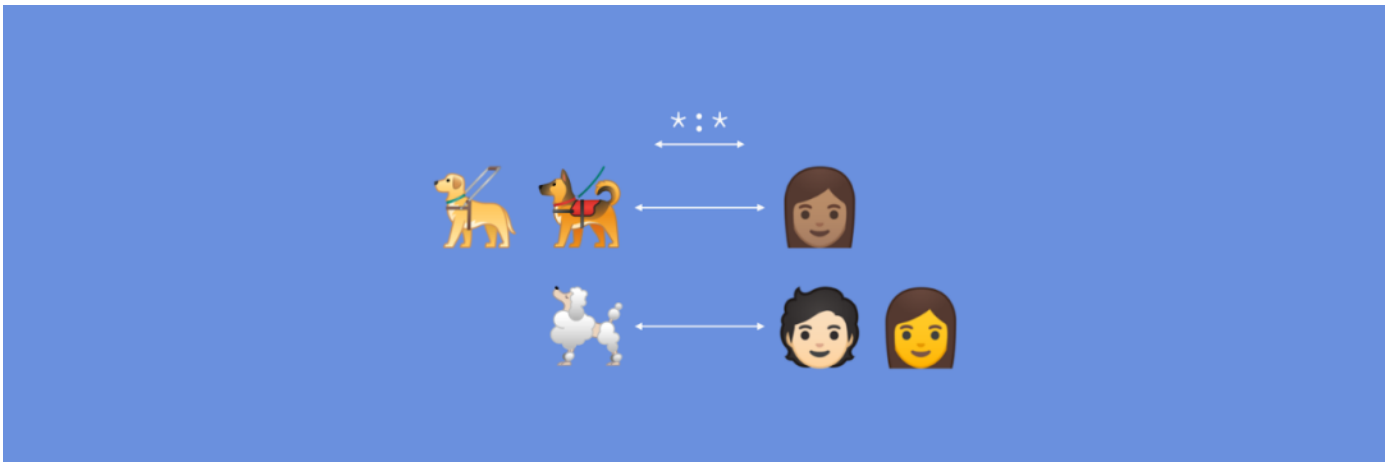
```
data class OwnerWithDogs(
     @Embedded val owner: Owner,
     @Relation(
          parentColumn = "ownerId",
          entityColumn = "dogOwnerId"
     )
     val dogs: List<Dog>
)
```

现在，Dao类又变成了这样:

```
@Transaction
@Query("SELECT * FROM Owner")

fun getDogsAndOwners(): List<OwnerWithDogs>
```

多对多关系



多对多关系

现在，继续假设我们生活在一个完美的世界中，一个人可以拥有多只狗，每只狗可以拥有多个主人。要对这个关系进行映射，之前的Dog和Owner表是不够的。由于一只狗狗可以有多个主人，我们需要在同一个 dog id 上能够匹配多个不同的 owner id。由于dogId是Dog表的主键，我们不能直接在Dog表中添加同样 id 的多条数据。为了解决这个问题，我们需要创建一个associative表 (也被称为连接表)，这个表来存储 (dogId,ownerId) 的数据对。

```
@Entity(primaryKeys = ["dogId", "ownerId"])
data class DogOwnerCrossRef(
    val dogId: Long,
    val ownerId: Long
)
```

associative

https://en.wikipedia.org/wiki/Associative_entity
en.wikipedia.org

如果现在我们想要获取到所有的狗狗和主人的数据，也就是List<OwnerWithDogs>，仅需要编写两个 SQLite 查询，一个获取到所有的主人数据，另一个获取Dog和DogOwnerCrossRef表的连接数据。

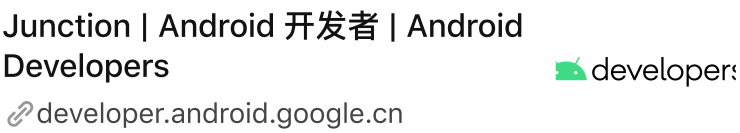
```
SELECT * FROM Owner
SELECT
     Dog.dogId AS dogId,
     Dog.dogOwnerId AS dogOwnerId,
     Dog.name AS name,
     _junction.ownerId
FROM
     DogOwnerCrossRef AS _junction
INNER JOIN Dog ON (_junction.dogId = Dog.dogId)

WHERE _junction.ownerId IN (ownerId1, ownerId2, …)
```

要通过 Room 来实现这个功能，我们需要更新OwnerWithDogs数据类，并告诉 Room 要使用DogOwnerCrossRef这个连接表来获取Dogs数据。我们通过使用Junction引用这张表。

```
data class OwnerWithDogs(
    @Embedded val owner: Owner,
    @Relation(
         parentColumn = "ownerId",
         entityColumn = "dogId",
         associateBy = Junction(DogOwnerCrossRef::class)
    )
    val dogs: List<Dog>
)
```

Junction



在我们的 Dao 中，我们需要从 Owners 中选择并返回正确的数据类:

```
@Transaction
@Query("SELECT * FROM Owner")

fun getOwnersWithDogs(): List<OwnerWithDogs>
```

更高阶的数据库关系用例

当使用@Relation注解时，Room 会默认从所修饰的属性类型推断出要使用的数据库实体。例如，到目前为止我们用@Relation修饰了Dog (或者是List<Dog>)，Room 就会知道如何去对该类进行建模，以及知道要查询的到底是哪一行数据。

如果您想让该查询返回一个不同的类，比如Pup这样不是一个数据库实体但是包含了一些字段的对象。我们可以在@Relation注解中指定要使用的数据库实体:

```
data class Pup(
     val name: String,
     val cuteness: Int = 11
)
data class OwnerWithPups(
     @Embedded val owner: Owner,
     @Relation(
          parentColumn = "ownerId",
          entity = Dog::class,
          entityColumn = "dogOwnerId"
     )
     val dogs: List<Pup>
)
```

如果我们只想从数据库实体中返回特定的列，您需要通过在@Relation中的 projection 属性中定义要返回哪些列。例如，假如我们只想获取OwnerWithDogs数据类中所有狗的名字，由于我们需要用到List<String>，Room 不能推断出这些字符串是对应于狗的品种呢还是狗的名字，因此我们需要在 projection 属性中指名。

```
data class OwnerWithDogs(
     @Embedded val owner: Owner,
     @Relation(
          parentColumn = "ownerId",
          entity = Dog::class,
          entityColumn = "dogOwnerId",
          projection = ["name"]
     )
     val dogNames: List<String>
)
```

如果您想在dogOwnerId和ownerId中定义更严格的关系，而不管您所创建的是什么，您可以通过在字段中使用ForeignKey来做到。记住，SQLite 中的外键会创建索引，并且会在更新或者删除表中数据时做级联操作。因此您要根据实际情况来判断是否使用外键功能。

- ForeignKey

ForeignKey | Android 开发者 | Android Developers
developer.android.google.cn

- SQLite 中的外键

SQLite Foreign Key Support
sqlite.org

不管您是要使用一对一，一对多还是多对多关系，Room 都会为您提供@Relation注解来解决问题。您可以在我们的 Android Dev Summit '19 的一个演讲中了解有关Room 2.2 的更多新功能:

https://v.qq.com/x/page/o3017dleair.html
v.qq.com

- @Relation

Relation | Android 开发者 | Android Developers
developer.android.google.cn

- Room 2.2 的更多新功能

Room | Android 开发者 | Android Developers
developer.android.google.cn

点击这里**进一步了解 Room**

编辑于 03-06

知乎（Android 应用）　Android 开发