## Sharding-JDBC

## 核心概念

在使用Sharding-JDBC之前，一定是先理解清楚下面几个核心概念。

### 逻辑表

水平拆分的数据库（表）的相同逻辑和数据结构表的总称。例：订单数据根据主键尾数拆分为10张表，分别是t\_order\_0到t\_order\_9，他们的逻辑表名为t\_order。

### 真实表

在分片的数据库中真实存在的物理表。即上个示例中的t\_order\_0到t\_order\_9。

### 数据节点

数据分片的最小单元。由数据源名称和数据表组成，例：ds\_0.t\_order\_0。

### 绑定表

指分片规则一致的主表和子表。例如：t\_order表和t\_order\_item表，均按照order\_id分片，则此两张表互为绑定表关系。绑定表之间的多表关联查询不会出现笛卡尔积关联，关联查询效率将大大提升。举例说明,如果SQL为：

### 广播表

指所有的分片数据源中都存在的表，表结构和表中的数据在每个数据库中均完全一致。适用于数据量不大且需要与海量数据的表进行关联查询的场景，例如：字典表。

## 数据分片

### 分片键

用于分片的数据库字段，是将数据库(表)水平拆分的关键字段。例：将订单表中的订单主键的尾数取模分片，则订单主键为分片字段。 SQL 中如果无分片字段，将执行全路由，性能较差。 除了对单分片字段的支持，Sharding-JDBC 也支持根据多个字段进行分片。

### 分片算法

通过分片算法将数据分片，支持通过=、>=、<=、>、<、BETWEEN和IN分片。 分片算法需要应用方开发者自行实现，可实现的灵活度非常高。

目前提供4种分片算法。 由于分片算法和业务实现紧密相关，因此并未提供内置分片算法，而是通过分片策略将各种场景提炼出来，提供更高层级的抽象，并提供接口让应用开发者自行实现分片算法。

#### 精确分片算法

对应 PreciseShardingAlgorithm，**用于处理使用单一键作为分片键的 = 与 IN 进行分片的场景**。需要配合 StandardShardingStrategy 使用。

#### 范围分片算法

对应 RangeShardingAlgorithm，**用于处理使用单一键作为分片键的 BETWEEN AND、>、<、>=、<=进行分片的场景**。需要配合 StandardShardingStrategy 使用。

#### 复合分片算法

对应 ComplexKeysShardingAlgorithm，用于处理使用多键作为分片键进行分片的场景，包含多个分片键的逻辑较复杂，需要应用开发者自行处理其中的复杂度。需要配合 ComplexShardingStrategy 使用。

#### Hint分片算法

对应 HintShardingAlgorithm，**用于处理通过Hint指定分片值而非从SQL中提取分片值的场景**。需要配合 HintShardingStrategy 使用。

### 分片策略

包含分片键和分片算法，由于分片算法的独立性，将其独立抽离。真正可用于分片操作的是分片键 + 分片算法，也就是分片策略。目前提供 5 种分片策略。

### 标准分片策略

对应 StandardShardingStrategy。提供对 SQ L语句中的 =, >, <, >=, <=, IN 和 BETWEEN AND 的分片操作支持。 StandardShardingStrategy 只支持单分片键，提供 PreciseShardingAlgorithm 和 RangeShardingAlgorithm 两个分片算法。 **PreciseShardingAlgorithm 是必选的**，用于处理 = 和 IN 的分片。 RangeShardingAlgorithm 是可选的，用于处理 BETWEEN AND, >, <, >=, <=分片，如果不配置 RangeShardingAlgorithm，SQL 中的 BETWEEN AND 将按照全库路由处理。

### 复合分片策略

对应 ComplexShardingStrategy。复合分片策略。提供对 SQL 语句中的 =, >, <, >=, <=, IN 和 BETWEEN AND 的分片操作支持。 **ComplexShardingStrategy 支持多分片键**，由于多分片键之间的关系复杂，因此并未进行过多的封装，而是直接将分片键值组合以及分片操作符透传至分片算法，完全由应用开发者实现，提供最大的灵活度。

### 行表达式分片策略

对应 InlineShardingStrategy。使用 Groovy 的表达式，提供对 SQL 语句中的 = 和 IN的分片操作支持，只支持单分片键。 对于简单的分片算法，可以通过简单的配置使用，从而避免繁琐的Java代码开发，如: t\_user\_$->{u\_id % 8} 表示 t\_user 表根据 u\_id 模 8，而分成 8 张表，表名称为 t\_user\_0 到 t\_user\_7。 **可以认为是精确分片算法的简易实现**

### Hint分片策略

对应 HintShardingStrategy。通过 Hint 指定分片值而非从 SQL 中提取分片值的方式进行分片的策略。

### 分布式主键

用于在分布式环境下，生成全局唯一的id。Sharding-JDBC 提供了内置的分布式主键生成器，例如 UUID、SNOWFLAKE。还抽离出分布式主键生成器的接口，方便用户自行实现自定义的自增主键生成器。**为了保证数据库性能，主键id还必须趋势递增，避免造成频繁的数据页面分裂。**

到这里，针对hc\_question\_reply\_record表，使用reply\_wheel\_time作为分片键，按照季度分片的处理就完成了。还有一点要注意的就是，**分库分表之后，查询的时候最好都带上分片键作为查询条件**，否则就会使用全库路由，性能很低。 还有就是Sharing-JDBC对mysql的全文索引支持的不是很好，项目有使用到的地方也要注意一下。总结来说整个过程还是比较简单的，后续碰到其它业务场景，相信大家按照这个思路肯定都能解决的。

## **标准分片策略**

\*\*使用场景\*\*：SQL 语句中有>，>=, <=，<，=，IN 和 BETWEEN AND 操作符，都可以应用此分片策略。

标准分片策略（StandardShardingStrategy），它只支持对单个分片健（字段）为依据的分库分表，并提供了两种分片算法 PreciseShardingAlgorithm（精准分片）和 RangeShardingAlgorithm（范围分片）。

在使用标准分片策略时，精准分片算法是必须实现的算法，用于 SQL 含有 = 和 IN 的分片处理；范围分片算法是非必选的，用于处理含有 BETWEEN AND 的分片处理。

一旦我们没配置范围分片算法，而 SQL 中又用到 BETWEEN AND 或者 like等，那么 SQL 将按全库、表路由的方式逐一执行，查询性能会很差需要特别注意。

接下来自定义实现 精准分片算法 和 范围分片算法。