# Architecture

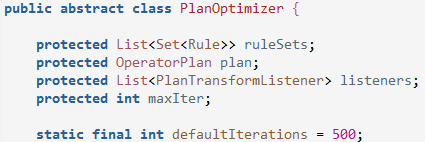
几个关键类/接口的关系：



每个关键类/接口的实现和继承结构在下面各节展开。

# Optimizer

PlanOptimizer 是抽象类，主要和Rule、PlanTransformListener、OperatorPlan打交道。



它接受一个OperatorPlan，即Operators的DAG模型，在optimize()方法里，遍历ruleSet，得到几批Rules，即Set<Rule>。对于每批Rules，调用每个rule.match(plan)来处理传入的OperatorPlan，返回一个匹配成功的List<OperatorPlan> matches，对这些match的plans进行进一步处理。首先获得rule的transformer，然后进行transformer的check()和transform()操作。如果需要Listener操作的，还会遍历listeners，让每个PlanTransformListener监听到transformer进行的transform操作，transformer的reportChanges()方法可以返回他transform操作修改的部分。

代码如下：

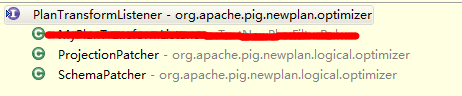


实现类：



## LogicalPlanOptimizer

LogicalPlanOptimizer类是PlanOptimizer的子类

默认加载两个Listener：

Listener的这两个实现在PlanTransformerListener一节具体展开讲述。

初始化的时候会buildRuleSets()，把需要添加的Rule都生成出来，然后校对该Rule是否被强制加入，或被turn off，从而选择性地放入优化规则。以下列举了所有候选的优化规则，会在Rule一节和Transformer一节具体展开讲述：



# Transformer

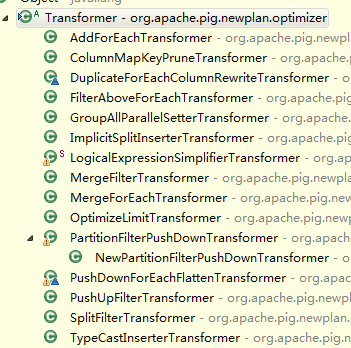
Transformer是抽象类，有三个方法需要子类实现：

check()方法，利用pattern来匹配plan里符合的operator集合，返回match的operator集

transform()方法，具体实施对tree的转换操作

reportChanges()方法，报告tree的哪部分被transform操作过了（只包括被修改了的或增加了的，不包括删除的node），目的是为了让Listener得知，从而可以修改schema或annotation等等。

继承结构如下：



具体Transformer的实现在Rule的各个实现里分析。

# Rule

**public** **abstract** **class** Rule {

**protected** String name = **null**;

**protected** OperatorPlan pattern;

**transient** **protected** OperatorPlan currentPlan;

**private** **transient** Set<Operator> matchedNodes =

**new** **HashSet**<Operator>();

**private** **boolean** mandatory;

**private** **boolean** skipListener = **false**;

Rule已经把 match(OperatorPlan plan)方法的逻辑实现好了。

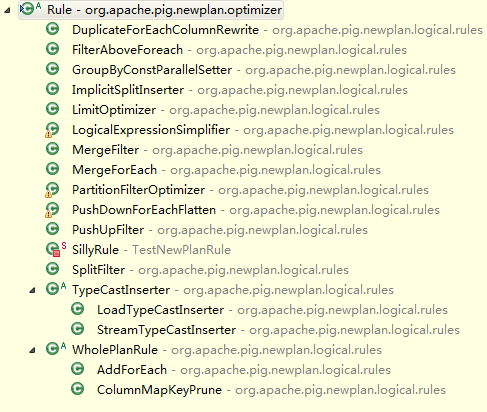
子类需要实现的是buildPattern()方法，来制定各自的”模式”，即pattern变量。

子类还需要实现getNewTransformer()方法来实例化一个transformer，transformer的check()和transform()方法会进一步处理rule匹配的operators。

Rule match()的用途是确保plan的所有子plan都满足该rule的pattern。

实现逻辑比较繁杂。

Rule继承结构



下面按照LogicalPlanOptimizer里的Rule顺序，一一分析每个Rule做的事情，涉及到的transformer和对应的Operator

## **ImplicitSplitInserter**

这个Rule针对整个plan。不制定具体的pattern。作用为隐式插入Spilt操作。

这是一个不能被用户disable的强制Rule，且skip listener。

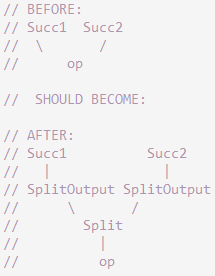
match()方法，遍历plan里的所有operator，只要不是LOSplit或不是LOStore的operator，就要检查是否有两个以上的successor，如果有的话，添加该op到返回集。该Rule用于插入Split操作，所以会过滤掉LOSplit。

buildPattern()返回null，该Rule对全局plan都处理，没有pattern要求。

### ImplicitSplitInserterTransformer

check()方法返回ture

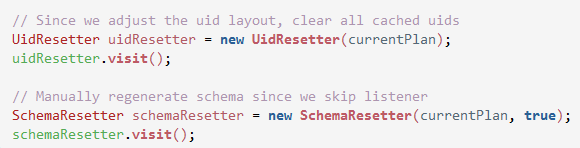
transform()方法，首先再过滤一次不处理的LOSplit和LOStore。要达到的transform效果如下：



处理过程主要是针对Graph符合的根节点，生成新的Split节点，Split节点插入重连成新Graph。



结束后再进行两次Visitor操作，注释说明了两次作用



两个Visitor具体内容参考另一份文档。

reportChanges()返回currentPlan

## DuplicateForEachColumnRewrite

为重复的column重写，让每个column可以具备唯一的uid

buildPattern()是生成一个包含LOForEach operator的LogicalPlan

### DuplicateForEachColumnRewriteTransformer

DuplicateForEachColumnRewriteTransformer

check()方法，略复杂，看不懂

## **PartitionFilterOptimizer**

buildPattern()是生成一个包含LOLoad的plan

## **LogicalExpressionSimplifier**

## **LoadTypeCastInserter**

## **StreamTypeCastInserter**

## **SplitFilter**

## **PushUpFilter**

## **FilterAboveForeach**

## **MergeFilter**

## **PushDownForEachFlatten**

## **ColumnMapKeyPrune**

## **AddForEach**

### AddForEachTransformer

AddForEachTransformer

check()方法，

这些op直接true：LOFilter，LOSort，LOJoin，LOSplitOutput

其他情况：



transform()针对op，做两步：

1. addForeach。
2. addSuccessors。

## **MergeForEach**

## **GroupByConstParallelSetter**

## LimitOptimizer

buildPattern()生产的是LOLimit的OperatorPlan

### OptimizeLimitTransformer

OptimizeLimitTransformer

check()

以下情况Limit不能push up：

LOCogroup，LOFilter，LOSplit，LODistinct，LOJoin

还有一种情况是在LOForEach里，innerPlan只要出现flattern，也不能通过check()

其他都返回true

transform()

先找到叶子节点集的第一个叶子

找到该叶子的所有前辈，取第一个前辈

判断该前辈类型：

LOForEach：move up LOLimit

LOCross/LOUnion：找到LOLimit，对比limit大小，插入新的LOLimit

LOSort：LOSort本身的limit值与这次LOLimit值比较，然后把LOLimit remove掉

LOLoad：LOLoad本身的limit值与这次LOLimit值比较，然后把LOLimit remove掉

LOLimit：LOLimit本身的limit值与这次LOLimit值比较，然后把LOLimit remove掉

LOSplitOutput：找前辈的前辈，如果有LOSort操作，LOSort的limit值重新set，替换新LOSort

# PlanTransformListener

PlanTransformListener监听一个plan被修改后会触发。

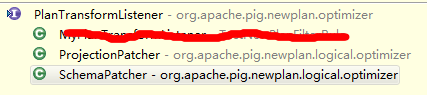
举例：

当一个Rule把一次join里的Filter步骤提前到join操作之间做，那么过滤部分的input schema很可能需要改变，此时一个schema listener就会被触发并执行。

PlanTransformListener是一个接口，需要实现一个方法：

**public** **void** transformed(OperatorPlan fp, OperatorPlan tp) **throws** FrontendException;

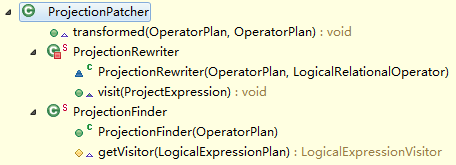
下面具体介绍两个实现类：

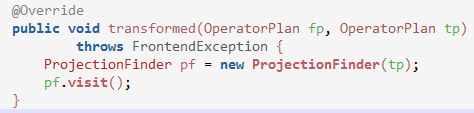


## ProjectionPatcher

作用是在映射操作中修补引用信息

有两个内部静态类





## SchemaPatcher

使用于逻辑执行计划优化过程，当plan被transform了之后修补schema信息。

