Apache Spark 自定义优化规则：Custom Optimizer Rule

在《Apache Spark 自定义优化规则：Custom Strategy》文章中我们介绍了如何自定义策略，策略是用在逻辑计划转换到物理计划阶段。而本文将介绍如何自定义逻辑计划优化规则，主要用于优化逻辑计划，和前文不一样的地方是，逻辑优化规则只是等价变换逻辑计划，也就是 Logic Plan -> Login Plan，这个是在应用策略前进行的。

假设我们有以下计算逻辑：

```
val df = spark.range(10).toDF("counts")
val iteblogDF = df.selectExpr("1 * counts ")
println(iteblogDF.queryExecution.optimizedPlan.numberedTreeString)
```

我们有一张表，里面只有名为 counts 的一列，我们对表的每行都乘以 1，现在我们使用 iteblogDF.queryExecution.optimizedPlan.numberedTreeString
来看下上面表达式的优化之后的逻辑计划：

```
00 Project [(1 * id#0L) AS (1 * counts)#4L]
01 +- Range (0, 10, step=1, splits=Some(2))
```

我们可以发现，对于每行的 counts，Spark 都会计算 counts *

1，如果在数据量很大的情况下，这种计算的开销很大的。

实现自己的优化规则

上面程序 Spark 其实得到的不是最优的执行计划，而且我们也知道，1 乘以任何数都是原来的数，根据这个规律，我们可以自定义逻辑优化规则：

```
package com.iteblog.example2

import org.apache.spark.sql.catalyst.expressions.{Literal, Multiply}
import org.apache.spark.sql.catalyst.plans.logical.LogicalPlan
import org.apache.spark.sql.catalyst.rules.Rule

object MultiplyOptimizationRule extends Rule[LogicalPlan] {
  def apply(plan: LogicalPlan): LogicalPlan = plan transformAllExpressions {
    case Multiply(left, right) if right.isInstanceOf[Literal]
      && right.asInstanceOf[Literal].value.asInstanceOf[Long] == 1 =>
      left
    case Multiply(left, right) if left.isInstanceOf[Literal]
      && left.asInstanceOf[Literal].value.asInstanceOf[Long] == 1 =>
      right
  }
}
```

可以看到，我们一样使用 Scala 的模式匹配来匹配我们要替换的东西，如果符合上面规则，那么就直接返回乘以1之后的值。现在我们通过 spark.experimental.extraOptimizations 来注册刚刚写好的规则：

```
spark.experimental.extraOptimizations = MultiplyOptimizationRule :: Nil
val df = spark.range(10).toDF("counts")
val iteblogDF = df.selectExpr("1 * counts ")
println(iteblogDF.queryExecution.optimizedPlan.numberedTreeString)
```

现在程序得到的优化逻辑计划如下：

```
00 Project [id#0L AS (1 * counts)#4L]
01 +- Range (0, 10, step=1, splits=Some(2))
```

对应的各个阶段计划：

```
== Parsed Logical Plan ==
'Project [(1 * 'counts) AS (1 * counts)#4]
+- Project [id#0L AS counts#2L]
   +- Range (0, 10, step=1, splits=Some(2))

== Analyzed Logical Plan ==
(1 * counts): bigint
Project [(cast(1 as bigint) * counts#2L) AS (1 * counts)#4L]
+- Project [id#0L AS counts#2L]
   +- Range (0, 10, step=1, splits=Some(2))

== Optimized Logical Plan ==
Project [id#0L AS (1 * counts)#4L]
+- Range (0, 10, step=1, splits=Some(2))

== Physical Plan ==
*(1) Project [id#0L AS (1 * counts)#4L]
+- *(1) Range (0, 10, step=1, splits=2)
```

可见，程序已经去掉了乘以1的东西。另外，可插拔的逻辑优化规则是从 Spark 2.0 开始引入的，参见 SPARK-9843，他也是实验性的功能，后面可能会发生变化。