大家好，我是StarRocks的Contributor，我叫李书明。今天是“源码实验”的第一期活动，想和大家分享下，“在StarRocks中如何添加内置函数？”

跟编程语言中的函数一样，数据库中的函数执行逻辑也分为3部分：输入参数、计算逻辑、返回结果。那么下面就聊一聊StarRocks中的函数。

在开始之前，先介绍下本次分享的大致内容，总体包含4块内容：

1. 内置函数相关的背景内容，了解下实现一个内置函数相关的概念；
2. 内置函数的实现原理，介绍下实现标量、聚合、窗口、Table函数的基本原理；
3. 介绍下StarRocks代码开发的基本流程；
4. 由我们的社区小姐姐介绍下如何认领函数。

本次分享是面向的是想社区开发者，想通过本次活动能够帮助“新同学”熟悉内置函数的基本原理和开发流程，函数具体实现细节需要同学们自己细细研究，希望通过本次分享，同学们可以上手实践、贡献代码到社区。

函数是整个数据库中高性能计算的关键，因此在实现的过程中，会对StarRocks的类型系统、表达式执行及执行算子都会有更加具体、全面的认识。同时函数、表达式是计算密集型的，因此其对性能的要求非常高，在实现的过程中，会用到矢量化、SIMD及各种各样的Morden C++的使用技巧。函数实现是非常很技巧性的，同时对代码质量要求有比较高的事情，也只有在实现过程中能够体会到成就的快感。欢迎大家踊跃尝试。

下面开始第一部分，内置函数的背景介绍。

1. StarRocks内置函数背景介绍

我们先看下函数的分类，根据输入、输出的对应关系，可以将函数分为三类：

- 1行输入对1行输出的，我们称为ScalarFunction，标量函数。像常见的abs/upper/lower等。
- 多行输入对应1行输出的，我们称为AggregateFunction，聚合函数。像常见的min/max/sum/count。
- 一行输入对应多行输出的，我们称为TableFunction，在PG生态中也叫SetReturnFunction。常见的unnest，将一行array展开为多行。

而除了上述分类方法之外，也可以按照函数的实现来分：

- 内置于数据库执行引擎内部的，用户开箱即用的，我们称为内置函数。
- 而用户可以通过自己的代码实现其自定义的执行逻辑的，同时支持动态加载的，我们称之为user defined function；不同数据库系统支持不同语言的UDF，比如可以用SQL/python/c++/java实现的。而甚至有些引擎开始支持用WASM支持自定义函数。

有同学会问，WindowFunction怎么没有分到？小伙伴们可以想一想，下面的介绍会给出答案。

在StarRocks我们也有内置函数和用户自定义函数，而本次分享的主题就是怎么实现一个内置函数，内置函数只需要一次代码贡献就可以让所有的StarRocks用户用到，是很有成就感的一件事情。除此之外，我们也支持UDF，不过我们目前仅支持Java实现UDF。关于目前StarRocks内置函数及Java UDF的实现可以参考我们的官方文档及Github上的文档介绍。

下面看看怎么样在10行以内实现一个内置函数？左边这个就是仅仅用了7行代码就实现了一个最简单”hello world”的dummy function。这个是一个标量函数，简单来说就两块内容: 上面就是函数实现、下面是函数的注册。代码添加后，编译运行就可以像其他内置函数一样查询了。（在后面开发环节，我将会演示下这个如何具体的编译操作）；

我们先看看StarRocks支持的类型，函数定义时需要确定其输入、输出参数类型。这里介绍了StarRocks内部三个地方的类型，Thrift协议中定义的类型，基本上同用户测暴露的类型一致。以及右侧FE、BE定义的类型映射关系。类型可以分为两大类，Primitive简单类型和Nested类型，复杂的嵌套类型，目前最新的3.0版本中我们都有支持。同学在开发函数时，需要考虑如何处理不同的输入数据类型？其输出类型是一致吗？多个输入类型该怎么处理?

在FE中，类型的定义有几个关键地方:

- PrimivitiveType类定义了FE到Thrift协议类型的映射，
- CataLog.Type是类型实现的抽象类，具体实现有ScalarType/ArrayType/MapType和StructType。在FE的函数注册时，会用到这些类，用来指定函数的输入、输出类型。

在BE中，类型定义在logical_type.h文件中，除了提供跟Thrift协议一一映射的类型，同时还基于C++ traits和宏定义提供了方便类型匹配的模版函数，这样就可以在编译阶段基于Type Dispatch优化根据不同的类型执行特定类型的代码，甚至可以做特定的优化。消除了类型抽象的代价，减少执行开销。但弊端是编译时模版实例化会使得代码膨胀、编译速度也会变慢。

讲完了类型，再说下Column。不同的数据类型映在BE测对应不同的Column。Column是数据在内存中保存的数据结构，这也是矢量化处理的基础，每个Column就是一列数据，多个Column就组成了Chunk。Column是基类，它提供了基本的操作函数，包括添加数据、序列化等操作。其派生类有FixedLengthColumn、BinaryColumn、JsonColumn、ArrayColumn、MapColumn等，而FixedLengthColumn作为模板类，可以根据不同的模板参数展开为固定长度的数据类型Column。

除了数据类型Column，之外需要特别注意的两个常见Column，ConstColumn和NullableColumn，它们都是Column的派生类。

ConstColumn表示这列数据都是相同的，可以只用1行数据表示，但真实长度则通过size来确定，访问任何一行数据，只需要返回第一行数据既可以，这样以便于实现数据的压缩，减少

数据传输。同时在函数处理中，比如很多的标量函数，做特定的优化，只需要处理一行数据即可。

NullableColumn表示该数据Column是有Null值，因此其内部包含了数据Column和Null值Column，其中Null值Column是一个uint_8的定长Column，其中1表示null, 0表示非null。

而在函数实现中，一定要小心，判断下确定输入的Column是普通的Column还是ConstColumn、Nullable，对ConstColumn和NullableColumn一定要特定的处理。

在真正写代码的时候，也建议增加ConstColumn/NullableColumn输入测试。我们遇到过很多这种CornerCase的bug，写代码的时候也要小心。

值得一提的是，嵌套类型（array/map/struct)内部实现也是嵌套了多个Column组成的，相较于普通的数据类型需要展开后才能处理，增加了额外的难度，使用时也需要注意。

最后也多提一句函数和表达式的关系，表达式是查询引擎中更加基础的概念，它不仅包含函数表达式还包含了SQL标准中的非函数表达，比如case when表达式、算数表达式、Predicate表达式等等。而标量函数对应的则是VectorizedFunctionCallExpr表达式。而其他函数，比如AggFunction和TableFunction则是算子实现框架的一部分，并没有通过表达式框架来实现。

最后强调一点，函数、表达式是实现StarRocks高性能的很重要的一部分，也是矢量化执行很重要的一部分，其核心逻辑是按列处理，并在此基础上做更多算法及SIMD优化。因此小伙伴在熟悉代码时，在很多地方针对表达式计算非常巧妙的性能优化。比如CaseWhen表达式，Predicate表达式等。

好，我们在介绍每类函数的实现原理。我们先说标量函数，聚合函数、窗口函数及Table函数我们后面会一一介绍。

2. StarRocks的函数原理

StarRocks的核心代码，分为FE、BE两大块，FE用Java编写，BE则用C++编写。

针对标量函数，我们看下FE测的处理。FE测首先从SQL中将函数解析为FunctionCallExpr，然后再ExpressionAnalyze阶段根据函数名及输入参数类型从函数表中获取对应的函数，并在Fragment分发阶段转换成对应的Thrift Plan。

而在BE测，在表达式初始化阶段，会根据FE生成的thrift plan生成VectorizedFunctionCallExpr，并prepare阶段通过函数id在BE测的函数表中获取相应的函数，并在Open阶段调用函数定义的prepare_func，然后不断的get_next执行函数逻辑，最后Close阶段调用函数定义的close_func，释放状态资源。

多说一句关于函数实现定义中的`FunctionContext`参数，该参数保存了函数声明周期内的状态，便于在各个阶段交互，比如常量参数、参数类型以及内存使用等。举个例子，比如一个函数有多个参数，其中有些参数是常量参数，我们不需要每次调用get_next时候解析这

个常量参数，这个时候就可以在Prepare函数中将函数解析一次，在之后的get_next中一致用这个解析后的参数就可以。同时在close的时候，需要调用函数的close函数销毁。

在be测添加了一个函数接口之后，如何让这个函数可以用起来？针对标量函数，目前基于python脚本封装了自动化注册框架，只需要在functions.py中针对具体的输入、输出类型添加一行就可以完成FE/BE的注册逻辑了。

这个python注册代码包含了这几个字段，包括：函数id（必须保障其唯一性）、函数名、函数的返回类型以及函数的输入类型。同时也包括两个可选部分，如果函数依赖些状态传递，则需要实现prepare_func和close_func，就是刚刚提到的，有常量参数时，可以通过prepare/close函数进行解析处理。

下面可以看下，abs的例子，针对每一个输入类型在这个python文件中都添加了一个函数注册。有些小伙伴会比较以后，为何要这么繁琐，不能一个函数只注册一次吗?在运行时自动根据输入类型判断。这样确实繁琐，目前看还是有不少好处：

- 能够保障FE/BE的输入、输出类型是对齐的；
- 针对特定输入类型指定相应的函数处理，能够减少运行时动态选择的逻辑，可以优化性能。

StarRocks的标量函数支持多个输入参数、也支持变参，在使用时可根据需要灵活定义注册。添加完这个注册代码之后，在编译阶段会将该注册逻辑同时添加到FE和BE的代码中。

在FE启动的时候，就会调用initBuiltins将内置函数注册到内存中，SQL运行期间就可以查到该内置函数的定义了。

而类似，在BE测同样也会维护一个function_id到函数的映射。在函数表达式执行期间会根据function_id查找对应的函数。具体实现在BuiltInFunctions::find_builtin_function定义中。

聚合函数

说完了标量函数，我们再说聚合函数，聚合函数相较于标量函数，稍微复杂是：

- 在分布式调度执行逻辑中，聚合函数一般都会有多个阶段的，先会在本机做预聚合（local aggregate），然后再根据GroupByKey Shuffle之后再做聚合(final agg)。当然对于有distinct的聚合，可能还有3阶段、4阶段，这里就不展开了，有兴趣的同学可以自己试着阅读下SQL生成的Plan。
- 还有一点是聚合函数必须要维护一个函数状态，用于预聚合和最终聚合时计算使用，比如sum agg，需要为维护一个总和值，在update或者merge时都需要更新该状态。函数状态也是函数实现的一部分了。

了解了agg函数的背景之后，我们看看实现一个agg函数所需要实现的几个主要接口。

- update： 这个用于local agg，读取数据、不断地更新中间状态；
- serialize_to_column：在local agg结束以后，需要做Shuffle，这个时候需要将中间状态序列化成可以网络传输的列。比如sum agg，需要把预聚合sum的结果序列化，传输给

下个阶段。这个类型是在注册时指定的，对于sum，不同的输入类型会有不同的序列化类型，比如int对应bigint，float对应double，decimal对应decimal，而avg函数，其状态需要保存count/sum两个值，则需要直接基于BinaryColumn传输的。

- merge： 在final 阶阶段，agg算子的输入是已经预处理过的中间状态，这个时候调用merge函数对中间状态结果进行合并、更新；
- finalize_to_column：当final阶段处理完之后，需要将中间状态生成最终的结果，需要调用final_to_column函数，输出到指定结果类型的列。
- 上面四个接口基本上涵盖了聚合函数的整个生命周期，除此之外还有不少其他的接口，用于不同地方的优化。
- convert_to_serialize_format接口，这个接口的实现是说当local agg聚合时发现预聚合没有太多聚合效果时，group by key比较稀疏，就不多处理了，直接将input的原始数据按照中间状态类型做streaming处理输出结果。目前的这个实现是自适应的，因此实现agg函数的时候也要实现该接口。

说完了agg函数的实现，讲一讲agg函数的注册逻辑。跟标量函数一样，同样需要在FE和BE测都需要注册，但是不一样的是聚合函数目前没有一个框架自动实现，需要开发者分别在FE、BE相应的代码上添加。

现在StarRocks已经内置了大量的聚合函数，同学们也不用担心，可以参考下这些聚合函数的实现代码。

在FE测，当新增一个内置函数式，需要在initAggBuiltins中注册该聚合函数，针对每一种输入、输出类型都需要注册，同时也要生命其中间状态的类型。在BE测，则是有aggregate functions resolver实现的，之前这个文件太大了编译起来很耗时，现在拆成了多个文件实现注册的。比如avg函数的注册方式就在register_avg定义中。

窗口函数

聚合函数讲完了，我们再说一下窗口函数。窗口函数从其是线上也是继承于聚合函数，因为它的处理逻辑也是多行输入返回一行输出结果。但不同的是聚合函数是讲相同的group by key聚合，而窗口函数则是按照窗口范围聚合数据聚合的。我们先了解下窗口函数的窗口是怎么定义的?

窗口函数支持两种Frame类型，Rows和Range，

- Rows是根据行数定义范围的，比如3 。
- Range则是按照数据值的范围定义的，比如range between。默认窗口是Range between unbounded preceding and current。

而窗口函数从分类上可以分为3类：

- 普通聚合函数的，比如count/sum/avg都可以用作窗口函数
- 排序类型：rank/dense_rank/row_number，这类函数通常无界窗口，做全局的排名；

- 分析类型：lead/lag/first_value、last_value；

StarRocks目前也实现了不少对应的窗口函数，在实现过程中也可以参考下实现。

WindowFunction是继承AggreteFunction实现的，但额外增加了针对窗口的实现接口。窗口函数目前实现都是单阶段的，不用考虑分布式问题。其核心接口就是这三个：

- update_batch_single_state_with_frame，每移动一行，根据窗口的范围来更新状态。
- 这里要区分几个概念：
  - Frame: 根据上下届确定的窗口大小，用于窗口计算；
  - PeerGroup: 则是为了更好地区分排序键引入的区别
    - 针对Rows Frame类型同分区键结果相同
    - 对于Range Frame类型则同排序键结果一样。
- get_values：每移动一行，获取(start, end)范围内的聚合结果。
- reset，会在Frame结束后调用。

下面看一个具体的例子（无界窗口），对id列做partition和sort by操作之后然后求sum，针对range类型，其窗口是从partition 开始到相同的sort key结束，所以相同的id会有相同的结果。而对于rows base的窗口，则是每一行输出一个结果，所以看到相同的id_int会有不同的结果。

讲完了窗口函数的实现，窗口函数是如何注册的呢？他的注册逻辑跟聚合函数是一样的。多提一句，除了上述的窗口函数实现之外，针对简单的sum/count/avg我们还实现了一种滑动窗口式的removable cumulative的窗口时间，针对窗口的很多优化可以参考下paper。

TableFunction

最后我们聊一下TableFunction，跟聚合函数一样，table函数也是内嵌于TableFunction算子内部的。其实现逻辑大致分为两块：

- 上游算子push_chunk时，将输入chunk保存至TableFunction状态中；
- 在TableFunction算子pull_chunk输出时，调用TableFunction的process逻辑，并根据State状态处理返回。

TableFunction的定义主要有这几个函数：

- 初始化：初始化TableFunctionState
- process：这也是窗口函数的核心逻辑，执行窗口函数并返回多行数组列及相应的offset
- close：销毁TableFunctionState

目前TableFunction注册也需要手动在FE/BE上添加实现，在FE上，需要在TableFunction::InitBuiltIn方法中添加新的函数签名，注意添加该函数的输入类型和返回类型。同样在BE中也有TableFunctionResolver中注册新的函数签名。目前StarRocks也支持了部分TableFunction，包括，unnest/mutli_unnest/json_each和generate_series等。添加新的内置函数时可以参考下。

3. 内置函数开发

现在基本的内置函数原理已经了解了，小伙伴们是不是蠢蠢欲动，想小试牛刀了？下面聊聊如何真正的开发实践?

先大致说下基本的开发环境。硬件层面，我们一般都是本地+远端服务器配合开发，而且服务器尽量配置高一点，这样编译速度会快点。开发软件上，be一般是vscode或者clion，都可以的，fe因为都是java代码，还是会有Idea神器。基础工具，c++编译gcc/clang都支持，如果条件的话，推荐还是使用clang，编译性能会好点而且错误提示也比较友好。

开发环境配置，首先是在本地/远程的代码clone，（小伙伴本最好先fork下我们的代码，方便后续提交pr)。配置环境的，强烈推荐基于docker环境开发，当然动手能力强的也可以直接在ubuntu/centos服务器上编译源码开发。因为docker环境中把开发依赖的组件都集成好，连三方依赖库也是编译好的。基本上开箱即用，等下我会按照docker环境的方式演示下。

然后编译测试脚本，这里也不一一介绍了。我们官方文档中应该都有介绍。如果小伙伴在使用过程中，发现文档介绍不合理或者不清晰的话，可以即使在官方微信群或者在github上提issue提问或者修复。

而等代码开发好了，怎么提交PR呢?

- 创建一个issue，如果有issue的话就不用了;
- 先把代码Push到自己fork的分支，然后提交到StarRocks的Project中。在提交之后，为了保证代码质量会有很多测试集成流程，小伙伴们在提交之后需要多加留意并耐心等待。

下面我花几分钟简单演示下，开发流程是如何操作的。

于计算密集型查询、或频繁重复使用的计算过程，运用代码生成技术能达到数十倍的性能提升。

4. 函数认领任务

到最后宣传下我们的2023函数认领任务。所有的Functions Tasks都在这个issue里。

- 在认领时，记得@通知下我们的社区同学，告知下你认领了这个函数。
- 在开发过前记得同社区同学即使沟通实现的思路或者函数的行为定等。
- 在开发过程中切记同社区同学一起讨论，有问题也可以在issue或者论坛即使反馈。
- 如果开发后发现有其他时间没法完成了，也没关系，联系我们的社区同学，以便于重新assign。