

Druid源码导读

张海雷

Druid编程风格

- ▶ 用java编写的函数式编程
- ▶ 使用Guice Module管理

源码结构

- ▶ Druid-api
- ▶ Druid-common
- ▶ Druid-process 索引和查询的核心
- ▶ Druid-server
- ▶ Druid-indexing-service
- ▶ Druid-indexing-Hadoop Hadoop 离线索引实现
- ▶ Extensions-core
- ▶ Extensions-contrib
- ▶ benchmarks



Process-核心引擎

- ▶ 重要分为两部分segment和query
- ▶ Segment中包括Segment的数据结构布局以及编码方式
- ▶ Query中包括不同类型查询的实现

Column

- ▶ “索引” 结构的核心接口
- ▶ 位于 “io.druid.segment.column”包内
- ▶ 不同类型Column的定义

```
/**
 *
 */
public interface Column
{
    public static final String TIME_COLUMN_NAME = "__time";
    public ColumnCapabilities getCapabilities();

    public int getLength();
    public DictionaryEncodedColumn getDictionaryEncoding();
    public RunLengthColumn getRunLengthColumn();
    public GenericColumn getGenericColumn();
    public ComplexColumn getComplexColumn();
    public BitmapIndex getBitmapIndex();
    public SpatialIndex getSpatialIndex();
}
```

data

- ▶ 位于“io.druid.segment.data”包内
- ▶ 索引和存储结构的实现
- ▶ 编码和压缩的实现

存储

- ▶ 定长存储
- ▶ 不定长存储

定长存储

- ▶ 接口IndexedInts、IndexedFloats和IndexedLongs
- ▶ 结合压缩实现，CompressedIndexedLongs
- ▶ 按块压缩 FixedSizeCompressedObjectStrategy

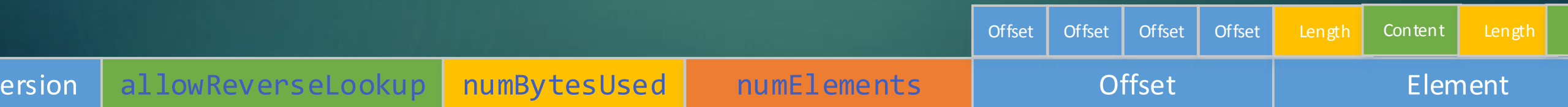
```
@Override
public long get(int index)
{
    final int bufferNum = index / sizePer;
    final int bufferIndex = index % sizePer;

    if (bufferNum != currIndex) {
        loadBuffer(bufferNum);
    }

    return buffer.get(buffer.position() + bufferIndex);
}
```


不定长存储

- 实现类GenericIndexed



编码

- ▶ Long和float采用JDK的编码方式，采用8个字节
- ▶ Int采用多种，JDK原生和变长整数编码
- ▶ Vsize开头的类是变长整数编码
- ▶ 在0.9.2版本中增加对long的新编码方式

serde

- ▶ 位于“io.druid.segment.serde”包中
- ▶ 提供了对各种不同类型的Column和Metric的Ser和De

Filter

- ▶ 位于 “io.druid.segment.filter” 包内
- ▶ 各种Filter逻辑的实现
- ▶ SelectorFilter 获取bitmap index
- ▶ AndFilter bitmap之间的交集

Filter 如何实现? a='1' and b='1'

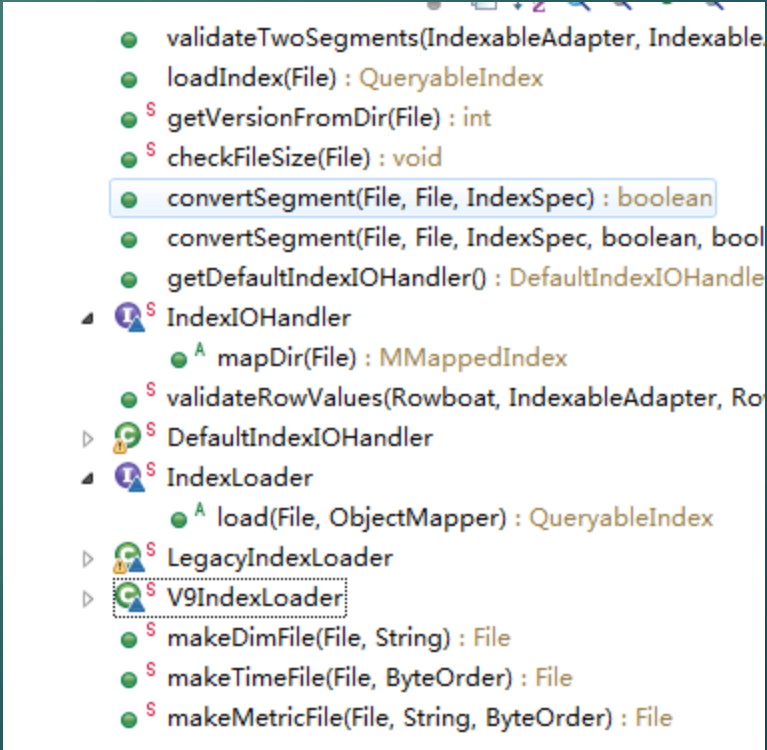
```
7 public interface Filter
8 {
9     public ImmutableBitmap getBitmapIndex(BitmapIndexSelector selector);
10    public ValueMatcher makeMatcher(ValueMatcherFactory factory);
11    public ValueMatcher makeMatcher(ColumnSelectorFactory columnSelectorFactory);
12 }
13
```

Increment Index

- ▶ 实时实现的关键类
- ▶ 内存增量索引，类似LSM-Tree中的memstore
- ▶ 采用map存储，无索引，开启sortFacts，使用SkipListMap
- ▶ 有两种实现OnHeap和OffHeap
- ▶ OffHeap，使用堆外内存存储aggregators，更好地管理内存

IndexIO

- 负责Segment的加载，通过Mmap的方式加载

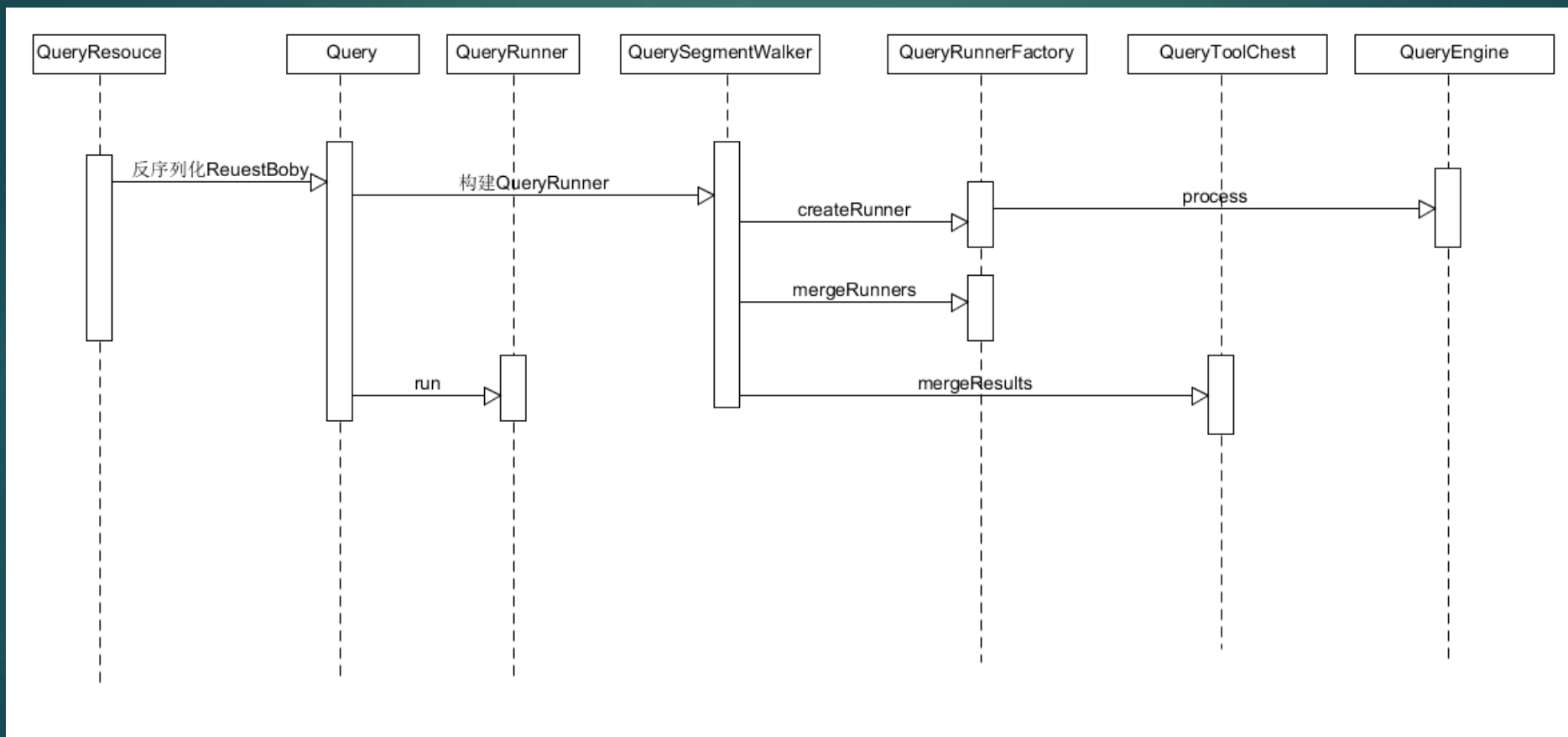


```
● validateTwoSegments(IndexableAdapter, IndexableAdapter) : boolean
● loadIndex(File) : QueryableIndex
● S getVersionFromDir(File) : int
● S checkFileSize(File) : void
● convertSegment(File, File, IndexSpec) : boolean
● convertSegment(File, File, IndexSpec, boolean, boolean) : boolean
● getDefaultIndexIOHandler() : DefaultIndexIOHandler
▲ S IndexIOHandler
  ● A mapDir(File) : MappedIndex
  ● S validateRowValues(Rowboat, IndexableAdapter, Rowboat) : boolean
  ▶ S DefaultIndexIOHandler
  ▲ S IndexLoader
    ● A load(File, ObjectMapper) : QueryableIndex
  ▶ S LegacyIndexLoader
  ▶ S V9IndexLoader
    ● S makeDimFile(File, String) : File
    ● S makeTimeFile(File, ByteOrder) : File
    ● S makeMetricFile(File, String, ByteOrder) : File
```

IndexMerger

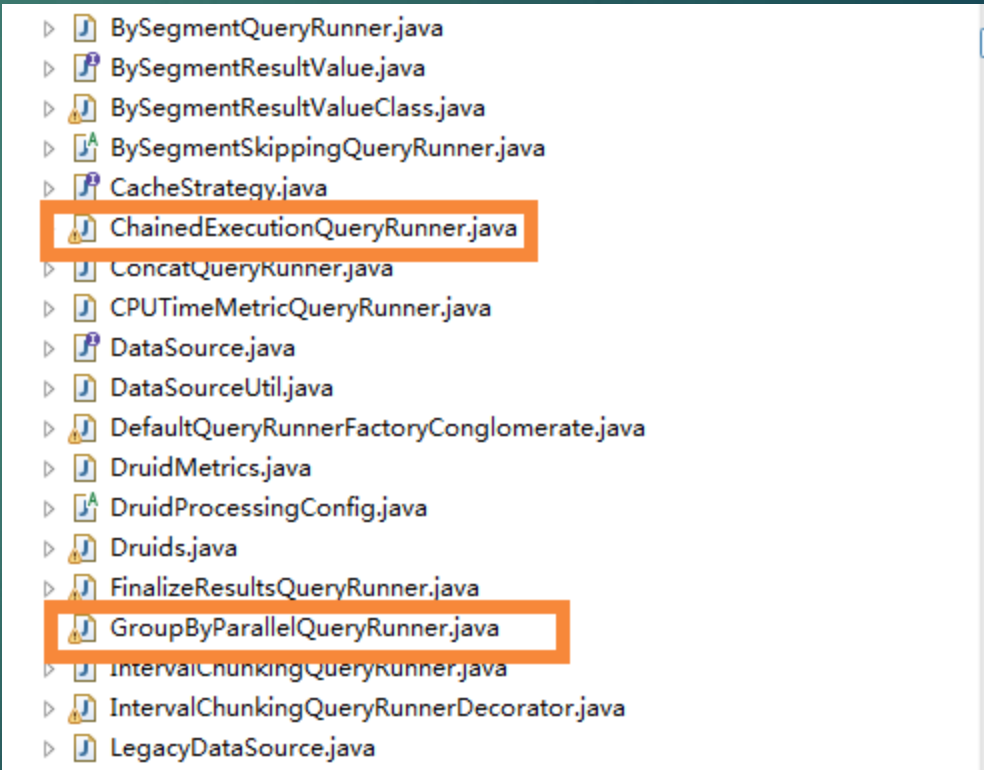
- ▶ 将IncrementalIndex持久化
- ▶ 合并多个持久化的索引成Segment

查询总览



query

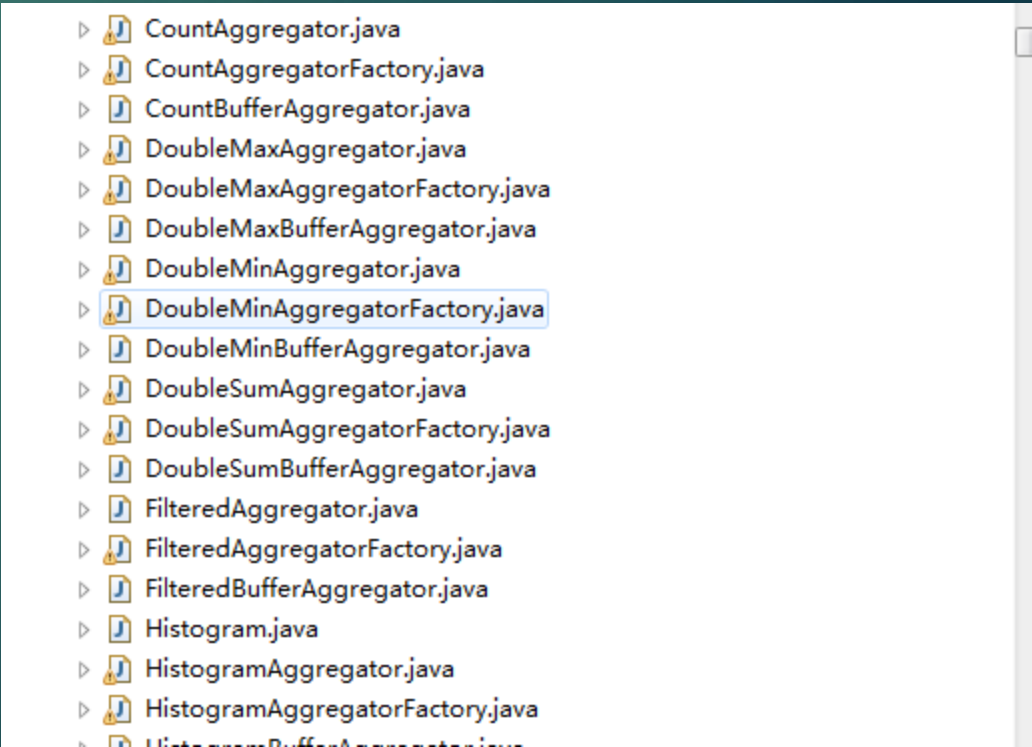
- ▶ 位于“io.druid.query”包内
- ▶ 定义了query通用的接口和类
- ▶ QueryRunner是执行查询逻辑的接口
- ▶ 不同职责的QueryRunner构建成员职链



```
▶ BySegmentQueryRunner.java
▶ BySegmentResultValue.java
▶ BySegmentResultValueClass.java
▶ BySegmentSkippingQueryRunner.java
▶ CacheStrategy.java
▶ ChainedExecutionQueryRunner.java
▶ ConcatQueryRunner.java
▶ CPUMetricQueryRunner.java
▶ DataSource.java
▶ DataSourceUtil.java
▶ DefaultQueryRunnerFactoryConglomerate.java
▶ DruidMetrics.java
▶ DruidProcessingConfig.java
▶ Druids.java
▶ FinalizeResultsQueryRunner.java
▶ GroupByParallelQueryRunner.java
▶ IntervalChunkingQueryRunner.java
▶ IntervalChunkingQueryRunnerDecorator.java
▶ LegacyDataSource.java
```

aggregation

- ▶ 各种Aggregator的实现
- ▶ Aggregator接口提供了基于对象的聚合计算
- ▶ BufferAggregator提供了基于ByteBuffer的聚合计算，一般采用堆外内存
- ▶ 使用AggregatorFactory创建
- ▶ 官方的Aggregator在AggregatorsModule注册



A screenshot of a file explorer window displaying a list of Java files. The files are organized in a tree view with expandable folders. The file 'DoubleMinAggregatorFactory.java' is highlighted with a blue selection box. The list includes various aggregator and factory classes, such as Count, DoubleMax, DoubleMin, DoubleSum, Filtered, and Histogram.

- ▶ CountAggregator.java
- ▶ CountAggregatorFactory.java
- ▶ CountBufferAggregator.java
- ▶ DoubleMaxAggregator.java
- ▶ DoubleMaxAggregatorFactory.java
- ▶ DoubleMaxBufferAggregator.java
- ▶ DoubleMinAggregator.java
- ▶ DoubleMinAggregatorFactory.java
- ▶ DoubleMinBufferAggregator.java
- ▶ DoubleSumAggregator.java
- ▶ DoubleSumAggregatorFactory.java
- ▶ DoubleSumBufferAggregator.java
- ▶ FilteredAggregator.java
- ▶ FilteredAggregatorFactory.java
- ▶ FilteredBufferAggregator.java
- ▶ Histogram.java
- ▶ HistogramAggregator.java
- ▶ HistogramAggregatorFactory.java
- ▶ HistogramBufferAggregator.java

StorageAdapter

- ▶ 根据index和Filter构建Cursor
- ▶ IncrementalIndexStorageAdapter是实时查询的实现
- ▶ QueryableIndexStorageAdapter是bitmap索引查询实现

```
4     }
5
6     final Offset offset;
7     if (filter == null) {
8         offset = new NoFilterOffset(0, index.getNumRows(), descending);
9     } else {
10        final ColumnSelectorBitmapIndexSelector selector = new ColumnSelectorBitmapIndexSelector(
11            index.getBitmapFactoryForDimensions(),
12            index
13        );
14
15        offset = new BitmapOffset(selector.getBitmapFactory(), filter.getBitmapIndex(selector), descending);
16    }
17
18    return Sequences.filter(
19        new CursorSequenceBuilder(
20            index,
21            actualInterval,
22            gran,
23            offset,
24            minDataTimestamp,
25            maxDataTimestamp,
26            descending
27        ).build(),
```

谢谢

