Translog目录结构

每个分片对应一个Translog，每个InternalEngine包含一个Translog

InternalEngine保存Translog的代数（generation），相关数据保存于commit metadata

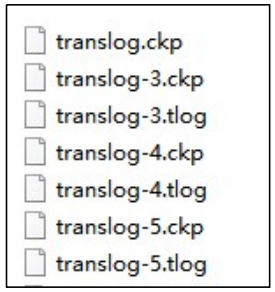
TRANSLOG_GENERATION_KEY指向尚未执行Lucene commit的所有Operation

1、checkpoint文件大小为4K，正好为1个block(8 个扇区), ES使用这种设计来保证checkpoint文件读写的原子性。
2、主要用于存储translog的operation数目、目前translog的代数等元数据

每次Translog提交对应一个TRANSLOG_UUID_KEY

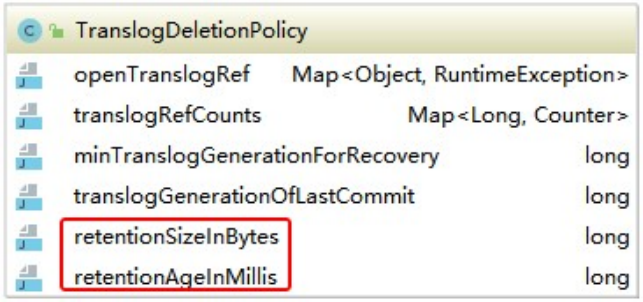
index.translog.generation_threshold_size控制Translog最大大小，默认64MB

每个打开的只读Translog文件translog-{generation}.tlog, 都对应了一个checkpoint文件translog-{generation}.ckp



translog读流程：
1、打开translog-{g}.tlog
2、复制translog.ckp到translog-{g}.ckp
3、打开translog-{g}.ckp

Translog 读取



InternalEngine

创建Translog删除策略（最大文件大小与保留时间）

```
final TranslogDeletionPolicy translogDeletionPolicy = new TranslogDeletionPolicy(
        engineConfig.getIndexSettings().getTranslogRetentionSize().getBytes(),
        engineConfig.getIndexSettings().getTranslogRetentionAge().getMillis()
);
```

打开Translog

```
translog = openTranslog(engineConfig, translogDeletionPolicy, engineConfig.getGlobalCheckpointSupplier());
```

从Lucene的segment info中读取userData

```
return commit == null ? Lucene.readSegmentInfos(directory) : Lucene.readSegmentInfos(commit);
```

```
final Map<String, String> commitUserData = store.readLastCommittedSegmentsInfo().getUserData();
```

使用Translog.TTRANSLOG_UUID_KEY取出Translog uuid

```
commitUserData.get(Translog.TRANSLOG_UUID_KEY);
```

打开Translog

```
return new Translog(translogConfig, translogUUID, translogDeletionPolicy, globalCheckpointSupplier,
    engineConfig.getPrimaryTermSupplier());
```

打开Checkpoint文件，定位当前和下一个Translog文件

```
Checkpoint checkpoint = readCheckpoint(location);
Path nextTranslogFile = location.resolve(getFilename( generation: checkpoint.generation + 1));
Path currentCheckpointFile = location.resolve(getCommitCheckpointFileName(checkpoint.generation));
```

按generation递减顺序依次打开所有translog文件,递减存储

```
// the list of translog readers is guaranteed to be in order of translog generation
private final List<TranslogReader> readers = new ArrayList<>();
```

Translog Checkpoint

Checkpoint
--- 文件结构

translog.ckp

Checkpoint	
offset	long
numOps	int
generation	long
minSeqNo	long
maxSeqNo	long
globalCheckpoint	long
minTranslogGeneration	long
trimmedAboveSeqNo	long
INITIAL_VERSION	int
VERSION_6_0_0	int
CURRENT_VERSION	int
CHECKPOINT_CODEC	String
V3_FILE_SIZE	int
V2_FILE_SIZE	int
V1_FILE_SIZE	int

```
final long offset;
final int numOps;
final long generation;
final long minSeqNo;
final long maxSeqNo;
final long globalCheckpoint;
final long minTranslogGeneration;
final long trimmedAboveSeqNo;
```

translog大小(bytes)
包含的operation数量
目前的代数
最小的operation seqNo
最大的operation seqNo
最近的全局检查点
低于此代数的tlog文件会被忽略
seqNo大于此值的operation会被忽略，使用-2可禁用此功能

```
// start with 1, just to recognize there was some
// magic serialization logic before
private static final int INITIAL_VERSION = 1;
// introduction of global checkpoints
private static final int VERSION_6_0_0 = 2;
// introduction of trimmed above seq#
private static final int CURRENT_VERSION = 3;
private static final String CHECKPOINT_CODEC = "ckp";
```

>= 6.4.0									
CodecHeader	offset	num ops	generation	minSeq#	maxSeq#	global-ckp	min-gener	trim-Abov	footer
12 bytes	8 bytes	4 bytes	8 bytes	8 bytes	8 bytes	8 bytes	8 bytes	8 bytes	16 bytes

>= 6.0.0								
CodecHeader	offset	num ops	generation	minSeq#	maxSeq#	global-ckp	min-gener	footer
12 bytes	8 bytes	4 bytes	8 bytes	8 bytes	8 bytes	8 bytes	8 bytes	16 bytes

>= 5.0.0				
CodecHeader	offset	num ops	generation	footer
12 bytes	8 bytes	4 bytes	8 bytes	16 bytes

Translog Checkpoint 读取

Checkpoint::read(path)
--- 读translog checkpoint

```
public static Checkpoint read(Path path) throws IOException {
    try (Directory dir = new SimpleFSDirectory(path.getParent())) {
        try (IndexInput indexInput = dir.openInput(path.getFileName().toString(), IOContext.DEFAULT)) {
            // We checksum the entire file before we
            CodecUtil.checksumEntireFile(indexInput);
            final int fileVersion = CodecUtil.checkHeader(indexInput, CHECKPOINT_CODEC, INITIAL_VERSION, CURRENT_VERSION);
            if (fileVersion == INITIAL_VERSION) {
                assert indexInput.length() == V1_FILE_SIZE : indexInput.length();
                return Checkpoint.readCheckpointV5_0_0(indexInput);
            } else if (fileVersion == VERSION_6_0_0) {
                assert indexInput.length() == V2_FILE_SIZE : indexInput.length();
                return Checkpoint.readCheckpointV6_0_0(indexInput);
            } else {
                assert fileVersion == CURRENT_VERSION : fileVersion;
                assert indexInput.length() == V3_FILE_SIZE : indexInput.length();
                return Checkpoint.readCheckpointV6_4_0(indexInput);
            }
```

读取header和body，CRC32::update

读取codec footer预存的校验码，与update结果比较

Translog Checkpoint： Codec Head

version == 1； V1_FILE_SIZE = 48 bytes； 5.0.0<=es<6.0.0

version == 2,； V1_FILE_SIZE = 80 bytes,； 6.0.0<=es<6.4.0

version == 3,； V1_FILE_SIZE = 88 bytes,； es>=6.4.0

9 + "ckp"len
CodecHeader
12 bytes

(4B)magic number	(VInt 1B)next str len	(3B)codec	(4B)version
0x3fd76c17	00000011	"ckp"	1<=ver<=3

```
// reads a checksummed checkpoint introduced in ES 5.0.0
static Checkpoint readCheckpointV5_0_0(final DataInput in) throws IOExceptio
    final long offset = in.readLong();
    final int numOps = in.readInt();
    final long generation = in.readLong();
    final long minSeqNo = SequenceNumbers.NO_OPS_PERFORMED;
    final long maxSeqNo = SequenceNumbers.NO_OPS_PERFORMED;
    final long globalCheckpoint = SequenceNumbers.UNASSIGNED_SEQ_NO;
    final long minTranslogGeneration = -1;
    final long trimmedAboveSeqNo = SequenceNumbers.UNASSIGNED_SEQ_NO;
    return new Checkpoint(offset, numOps, generation, minSeqNo, maxSeqNo,
        globalCheckpoint, minTranslogGeneration, trimmedAboveSeqNo);
```

```
static Checkpoint readCheckpointV6_4_0(final DataInput in) throws IOException
    final long offset = in.readLong();
    final int numOps = in.readInt();
    final long generation = in.readLong();
    final long minSeqNo = in.readLong();
    final long maxSeqNo = in.readLong();
    final long globalCheckpoint = in.readLong();
    final long minTranslogGeneration = in.readLong();
    final long trimmedAboveSeqNo = in.readLong();
    return new Checkpoint(offset, numOps, generation, minSeqNo, maxSeqNo,
        globalCheckpoint, minTranslogGeneration, trimmedAboveSeqNo);
```

Translog 读取

Translog::recoverFromFiles
--- 依次打开所有Translog

获取写锁

```
try (ReleasableLock lock = writeLock.acquire()) {
```

从Checkpoint读取最小引用代数minGenerationToRecoverFrom

```
minGenerationToRecoverFrom = checkpoint.minTranslogGeneration;
```

打开最新的tlog文件

```
String checkpointTranslogFile = getFilename(checkpoint.generation);
foundTranslogs.add(openReader(location.resolve(checkpointTranslogFile), checkpoint));
```

依次打开其余的tlog文件,最小代数为minGenerationToRecoverFrom

```
for (long i = checkpoint.generation - 1; i >= minGenerationToRecoverFrom; i--) {
    Path committedTranslogFile = location.resolve(getFilename(i));
final TranslogReader reader = openReader(committedTranslogFile,
    Checkpoint.read(location.resolve(getCommitCheckpointFileName(i))));
foundTranslogs.add(reader);
```

```
Collections.reverse(foundTranslogs);
```

删除被忽略的tlog和ckp文件

检查translog.ckp与translog-{max-generation}.ckp是否相等

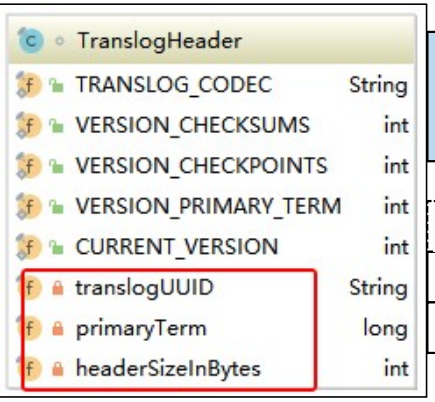
```
Path commitCheckpoint = location.resolve(getCommitCheckpointFileName(checkpoint.generation));
if (Files.exists(commitCheckpoint)) {
    Checkpoint checkpointFromDisk = Checkpoint.read(commitCheckpoint);
    if (checkpoint.equals(checkpointFromDisk) == false) {
        throw new IllegalStateException("Checkpoint file " + commitCheckpoint.getFileName() +
```

TranslogHeader

9 + "translog".len
CodecHeader
17 bytes

translogHeader与checkpointHeader结构一致，存储codec与tlog版本

(4B)magic number	(VInt 1B)next str len	(8B)codec	(4B)version
0x3fd76c17	00001000	"translog"	1<=ver<=3



translogHeader整体大小的计算：
9 + "translog".len + uuidLen(4) + uuid.len + pTerm(8) + checksum(4)

9 + "translog".len			
CodecHeader	uuidLen	uuid	pTerm
17 bytes	4 bytes	uuid.len bytes	8 bytes

checksum
4 bytes

TranslogReader::open(channel, path, checkpoint, uuid)
--- 打开translog-{generation}.tlog文件

读取translogHeader

TranslogHeader::read(uuid, path, channel)
--- 读取translog-{generation}.tlog文件Header

读取当前tlog文件的uuid

读取uuid length（一个int）

```
final int uuidLen = in.readInt();
```

读取uuid

```
final BytesRef uuid = new BytesRef(uuidLen);
uuid.length = uuidLen;
in.read(uuid.bytes, uuid.offset, uuid.length);
```

读取primary term（主分片重新分配的次数）

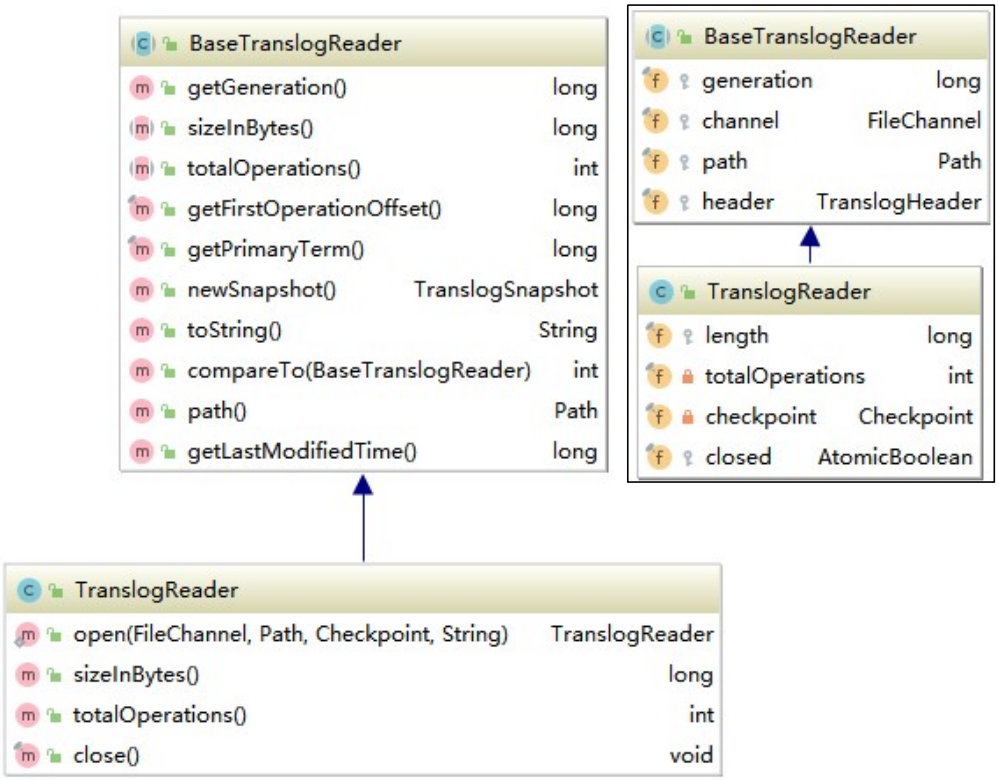
```
primaryTerm = in.readLong();
```

根据已经读取的字节生成checksum

读取预存的checksum（一个int）

```
long expectedChecksum = in.getChecksum();
long readChecksum = Integer.toUnsignedLong(in.readInt());
if (readChecksum != expectedChecksum) {
    throw new TranslogCorruptedException(in.getSource(),
```

TranslogReader



TranslogReader
--- 提供对tlog文件的各类操作

long getGeneration()

从checkpoint获取tlog的generation

long sizeInBytes()

从checkpoint获取offset值, 整个tlog大小

int totalOperations()

从checkpoint获取numOps值

long getFirstOperationOffset()

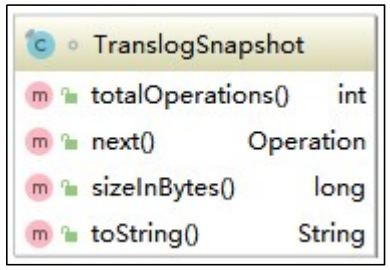
获取tlog文件中header的大小(bytes)，header之后即为operation

int newSnapshot()

TranslogSnapshot封装了与Operation读取有关的操作

```
public TranslogSnapshot newSnapshot() {
    return new TranslogSnapshot( reader: this, sizeInBytes());
}
```

TranslogSnapshot



TranslogHeader	OperationSize	Operation	OperationSize	Operation
headerSizeInBytes	4 bytes	k bytes	4 bytes	k bytes

operation size

position

new position

```
protected Translog.Operation readOperation() throws IOException {
    final int opSize = readSize(reusableBuffer, position);
    reuse = checksummedStream(reusableBuffer, position, opSize, reuse);
    Translog.Operation op = read(reuse);
    position += opSize;
    readOperations++;
    return op;
}
```

TranslogSnapshot
--- 提供对Operation的读取操作

Operation next()

获取下一条Operation

```
public Translog.Operation next() throws IOException {
    while (readOperations < totalOperations) {
        final Translog.Operation operation = readOperation();
        if (operation.seqNo() <= checkpoint.trimmedAboveSeqNo ||
            checkpoint.trimmedAboveSeqNo == SequenceNumbers.UNASSIGNED_SEQ_NO) {
            return operation;
        }
        skippedOperations++;
    }
    return null;
```

忽略一部分Operation

Operation readOperation()

读取下一条Operation

读取一个Int, IntValue + 4 作为需读取的Operation Size (bytes)

+4的目的是算上OperationSize字段的大小(从position开始读)

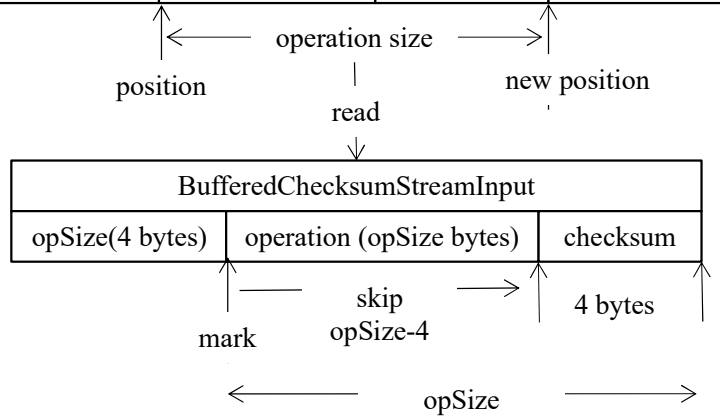
从position处读取OperationSize个字节

读取并构造Operation

更新position， position += operationSize

TranslogSnapshot：读取Operation

TranslogHeader	OperationSize	Operation	OperationSize	Operation
headerSizeInBytes	4 bytes	k bytes	4 bytes	k bytes



BaseTranslogReader

Operation read(BufferedChecksumStreamInput)

读取下一条Operation

```
final Translog.Operation op = Translog.readOperation(inStream);
if (op.primaryTerm() > getPrimaryTerm() && getPrimaryTerm() != SequenceNumbers.UNASSIGNED_PRIMARY_TERM) {
    throw new TranslogCorruptedException(
```

Translog

Operation readOperation(BufferedChecksumStreamInput)

从buffer中读取Operation大小

```
final int opSize = in.readInt();
```

重置checksumBuffer的digest，因为opSize不是Operation一部分

```
in.resetDigest();
```

CRC校验Operation数据

```
in.mark(opSize);
in.skip( numBytes: opSize - 4);
verifyChecksum(in);
in.reset();
```

```
long expectedChecksum = in.getChecksum();
long readChecksum = Integer.toUnsignedLong(in.readInt());
if (readChecksum != expectedChecksum) {
```

Translog.Operation.readOperation(in)

```
operation = Translog.Operation.readOperation(in);
verifyChecksum(in);
```

校验了两次checksum，针对网络数据传输不支持mark功能，从而不能预先校验checksum的情况

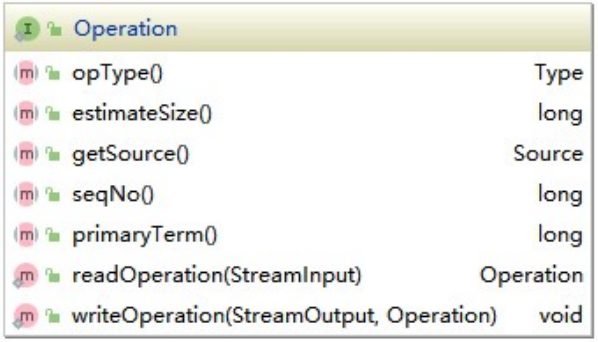
TranslogSnapshot：读取Operation

BufferedChecksumStreamInput		
opSize(4 bytes)	operation (opSize bytes)	checksum
	type(1 bytes)	OP

Translog.Operation
Operation readOperation(BufferedChecksumStreamInput)
从buffer中读取Operation
读取Operation类型（一个字节）
`Translog.Operation.Type type = Translog.Operation.Type.fromId(input.readByte());`
1->CREATE; 2->INDEX; 3->DELETE; 4->NO_OP
根据类型，选择不同的Operation实现来读取数据

```
switch (type) {
    case CREATE:
        // the de-serialization
    case INDEX:
        return new Index(input);
    case DELETE:
        return new Delete(input);
    case NO_OP:
        return new NoOp(input);
```

读取IndexOperation



Translog.Index implements Operation
Index(StreamInput)
读取format（VInt，1字节）
读取文档id（string）
读取文档type（string）
读取文档source（bytesRef）
format < FORMAT_NO_PARENT, 读取文档_parent（optional string）
读取文档version（Long，8字节）
format < FORMAT_NO_VERSION_TYPE, 读取文档_version_type（1字节）
读取文档autoGeneratedIdTimestamp（Long，8字节）
读取文档seqNo（Long，8字节）
读取文档primaryTerm（Long，8字节）

BufferedChecksumStreamInput		
opSize(4 bytes)	operation (opSize bytes)	checksum
	type(1 bytes) OP	

IndexOperation								
format	id	type	source	_parent	version	ver_type	seqNo	pTerm
1 byte	string	string	byteRef	string	8 bytes	1 byte	8 bytes	8 bytes

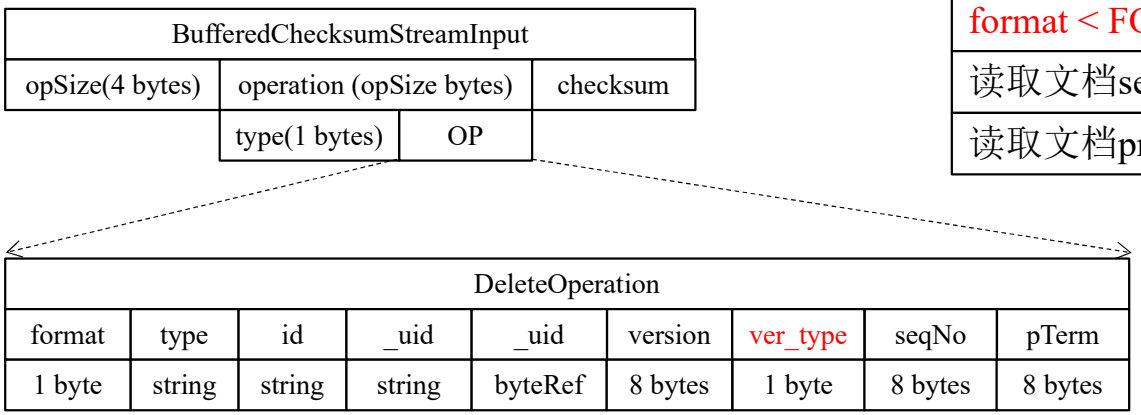
string	
size	content
VInt	~size*2 bytes

optional string		
exits	size	content
1byte	VInt	~size*2 bytes

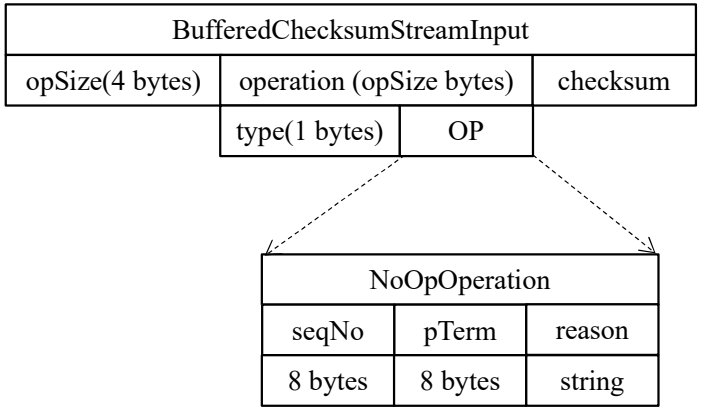
bytesRef	
size	content
VInt	size bytes

读取DeleteOperation

Translog.Delete implements Operation
Delete(StreamInput)
读取format（VInt，1字节）
读取文档type（string）
读取文档id（string），type内id唯一
读取文档_uid Term(field[string], bytes[bytesRef])，index内_uid唯一
_uid = type#id，存于Lucene的_uid字段中,可实现根据type和id检索文档
读取文档version（long）
format < FORMAT_NO_VERSION_TYPE, 读取文档_version_type（1字节）
读取文档seqNo（Long，8字节）
读取文档primaryTerm（Long，8字节）



读取NoOperation



Translog.NoOp implements Operation
NoOp(StreamInput)
读取文档seqNo（Long，8字节）
读取文档primaryTerm（Long，8字节）
读取season（string）

Translog：添加Operation

```
class Index implements Operation
private void write(final StreamOutput out) throws IOException {
    final int format = out.getVersion().onOrAfter(Version.V_7_0_0) ?
        SERIALIZATION_FORMAT : FORMAT_6_0;
    out.writeVInt(format);
    out.writeString(id);
    out.writeString(type);
    out.writeBytesReference(source);
    out.writeOptionalString(routing);
    if (format < FORMAT_NO_PARENT) {
        out.writeOptionalString(null); // _parent
    }
    out.writeLong(version);
    if (format < FORMAT_NO_VERSION_TYPE) {
        out.writeByte(VersionType.EXTERNAL.getValue());
    }
    out.writeLong(autoGeneratedIdTimestamp);
    out.writeLong(seqNo);
    out.writeLong(primaryTerm);
}
```

```
totalOffset += data.length();
```

```
minSeqNo = SequenceNumbers.min(minSeqNo, seqNo);
maxSeqNo = SequenceNumbers.max(maxSeqNo, seqNo);
```

```
return new Translog.Location(generation, offset, data.length());
```

TranslogReader::add(operation)
--- 添加一个Operation

将operation写入缓存

跳过4字节，预留用于写operation size

```
final ReleasableBytesStreamOutput out = new ReleasableBytesStreamOutput(bigArrays)
try {
    final long start = out.position();
    out.skip(Integer.BYTES);
```

写Operation，计算checksum，写checksum

```
Translog.Operation.writeOperation(out, op);
long checksum = out.getChecksum();
out.writeInt((int) checksum);
```

获取刚写入的Operation size

```
final long end = out.position();
final int operationSize = (int) (end - Integer.BYTES - start);
```

将Operation size写入预留空间中

```
out.seek(start);
out.writeInt(operationSize);
out.seek(end);
```

获取刚写入缓存的数据

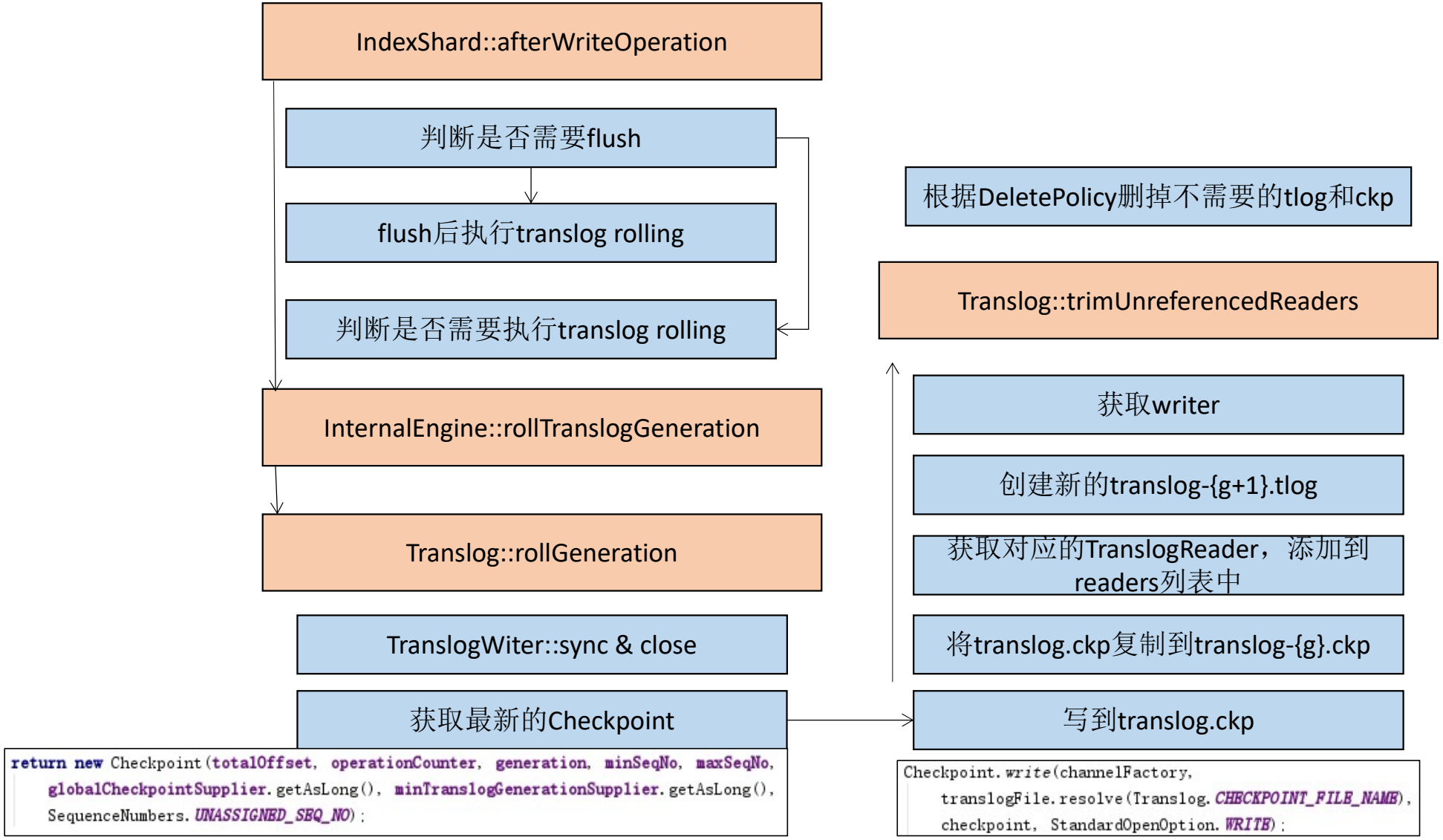
```
final ReleasablePagedBytesReference bytes = out.bytes();
```

将数据写入tlog文件中

```
current.add(bytes, operation.seqNo());
```

更新Translog offset、minSeqNo、maxSeqNo

Translog：commit



StreamInput/Output：String

c <= 0000,0000,0111,1111(0x007F)

buffer[offset++] = (byte)c

0-7

c > 0000,0111,1111,1111(0x07FF)

buffer[offset++] = 1110,0000 | c >> 12 & 0000,1111

1110=14

buffer[offset++] = 1000,0000 | c >> 6 & 0011,1111

buffer[offset++] = 1000,0000 | c >> 0 & 0011,1111

0111,1111(0x007F) < c <= 0111,1111,1111(0x07FF)

buffer[offset++] = 1100,0000 | c >> 6 & 0001,1111

1100=12
1101=13

buffer[offset++] = 1000,0000 | c >> 0 & 0011,1111

c >> 4

case 0-7(<=01111111)

input

output

```
for (int i = 0; i < charCount; i++) {
    final int c = readByte() & 0xff;
    switch (c >> 4) {
        case 0:
        case 1:
        case 2:
        case 3:
        case 4:
        case 5:
        case 6:
        case 7:
            buffer[i] = (char) c;
            break;
        case 12:
        case 13:
            buffer[i] = ((char) ((c & 0x1F) << 6 | readByte() & 0x3F));
            break;
        case 14:
            buffer[i] = ((char) ((c & 0x0F) << 12 | (readByte() & 0x3F) << 6 |
                (readByte() & 0x3F) << 0));
            break;
        default:
            throw new IOException("Invalid string; unexpected character: " + c +
                " hex: " + Integer.toHexString(c));
    }
}
```

```
if (c <= 0x007F) {
    buffer[offset++] = ((byte) c);
} else if (c > 0x07FF) {
    buffer[offset++] = ((byte) (0xE0 | c >> 12 & 0x0F));
    buffer[offset++] = ((byte) (0x80 | c >> 6 & 0x3F));
    buffer[offset++] = ((byte) (0x80 | c >> 0 & 0x3F));
} else {
    buffer[offset++] = ((byte) (0xC0 | c >> 6 & 0x1F));
    buffer[offset++] = ((byte) (0x80 | c >> 0 & 0x3F));
}
```