李翰林

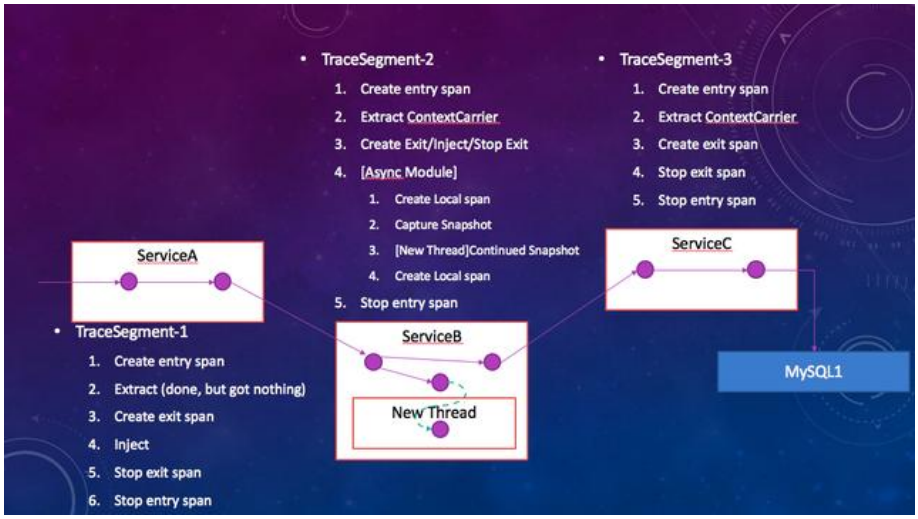
李翰林的技术论文集

浅谈skywalking的TraceSegmentServiceClient

本文参考原文-http://bjbsair.com/2020-03-22/tech-info/5102.html

序

本文主要研究一下skywalking的TraceSegmentServiceClient



TracingContextListener

skywalking-6.6.0/apm-sniffer/apm-agent-core/src/main/java/org/apache/skywalking/apm/agent/core/context/TracingContextListener.java

```
public interface TracingContextListener {
    void afterFinished(TraceSegment traceSegment);
}
```

- TracingContextListener定义了afterFinished方法，其参数为TraceSegment

TraceSegment

skywalking-6.6.0/apm-sniffer/apm-agent-core/src/main/java/org/apache/skywalking/apm/agent/core/context/trace/TraceSegment.java

```
public class TraceSegment {

    private ID traceSegmentId;

    private List<TraceSegmentRef> refs;

    private List<AbstractTracingSpan> spans;

    private DistributedTraceIds relatedGlobalTraces;

    private boolean ignore = false;

    private boolean isSizeLimited = false;

    private final long createTime;
```

```
    public TraceSegment() {
        this.traceSegmentId = GlobalIdGenerator.generate();
        this.spans = new LinkedList<AbstractTracingSpan>();
        this.relatedGlobalTraces = new DistributedTraceIds();
        this.relatedGlobalTraces.append(new NewDistributedTraceId());
        this.createTime = System.currentTimeMillis();
    }

    public void ref(TraceSegmentRef refSegment) {
        if (refs == null) {
            refs = new LinkedList<TraceSegmentRef>();
        }
        if (!refs.contains(refSegment)) {
            refs.add(refSegment);
        }
    }

    public void relatedGlobalTraces(DistributedTraceId distributedTraceId) {
        relatedGlobalTraces.append(distributedTraceId);
    }

    public void archive(AbstractTracingSpan finishedSpan) {
        spans.add(finishedSpan);
    }

    public TraceSegment finish(boolean isSizeLimited) {
        this.isSizeLimited = isSizeLimited;
        return this;
    }

    public ID getTraceSegmentId() {
        return traceSegmentId;
    }

    public int getServiceId() {
        return RemoteDownstreamConfig.Agent.SERVICE_ID;
    }

    public boolean hasRef() {
        return !(refs == null || refs.size() == 0);
    }

    public List<TraceSegmentRef> getRefs() {
        return refs;
    }

    public List<DistributedTraceId> getRelatedGlobalTraces() {
        return relatedGlobalTraces.getRelatedGlobalTraces();
    }

    public boolean isSingleSpanSegment() {
        return this.spans != null && this.spans.size() == 1;
    }

    public boolean isIgnore() {
        return ignore;
    }

    public void setIgnore(boolean ignore) {
        this.ignore = ignore;
    }

    public UpstreamSegment transform() {
        UpstreamSegment.Builder upstreamBuilder = UpstreamSegment.newBuilder();
        for (DistributedTraceId distributedTraceId : getRelatedGlobalTraces()) {
            upstreamBuilder = upstreamBuilder.addGlobalTraceIds(distributedTraceI
        }
        SegmentObject.Builder traceSegmentBuilder = SegmentObject.newBuilder();
        /**
```

```
         * Trace Segment
         */
        traceSegmentBuilder.setTraceSegmentId(this.traceSegmentId.transform());
        // Don't serialize TraceSegmentReference

        // SpanObject
        for (AbstractTracingSpan span : this.spans) {
            traceSegmentBuilder.addSpans(span.transform());
        }
        traceSegmentBuilder.setServiceId(RemoteDownstreamConfig.Agent.SERVICE_ID)
        traceSegmentBuilder.setServiceInstanceId(RemoteDownstreamConfig.Agent.SER
        traceSegmentBuilder.setIsSizeLimited(this.isSizeLimited);

        upstreamBuilder.setSegment(traceSegmentBuilder.build().toByteString());
        return upstreamBuilder.build();
    }

    @Override
    public String toString() {
        return "TraceSegment{" +
            "traceSegmentId='" + traceSegmentId + '\'' +
            ", refs=" + refs +
            ", spans=" + spans +
            ", relatedGlobalTraces=" + relatedGlobalTraces +
            '}';
    }

    public int getApplicationInstanceId() {
        return RemoteDownstreamConfig.Agent.SERVICE_INSTANCE_ID;
    }

    public long createTime() {
        return this.createTime;
    }
}
```

- TraceSegment定义了traceSegmentId、refs、spans、relatedGlobalTraces等属性；它提供了ref、relatedGlobalTraces、archive 、finish、transform等方法

IConsumer

skywalking-6.6.0/apm-commons/apm-datacarrier/src/main/java/org/apache/skywalking/apm/commons/datacarrier/consumer/IConsumer.java

```
public interface IConsumer<T> {
    void init();

    void consume(List<T> data);

    void onError(List<T> data, Throwable t);

    void onExit();
}
```

- IConsumer定义了init、consume、onError、onExit方法

TraceSegmentServiceClient

skywalking-6.6.0/apm-sniffer/apm-agent-core/src/main/java/org/apache/skywalking/apm/agent/core/remote/TraceSegmentServiceClient.java

```
@DefaultImplementor
public class TraceSegmentServiceClient implements BootService, IConsumer<TraceSeg
    private static final ILog logger = LogManager.getLogger(TraceSegmentServiceCl
    private static final int TIMEOUT = 30 * 1000;

    private long lastLogTime;
    private long segmentUplinkedCounter;
    private long segmentAbandonedCounter;
    private volatile DataCarrier<TraceSegment> carrier;
    private volatile TraceSegmentReportServiceGrpc.TraceSegmentReportServiceStub
```

```
    private volatile GRPCChannelStatus status = GRPCChannelStatus.DISCONNECT;

    @Override
    public void prepare() throws Throwable {
        ServiceManager.INSTANCE.findService(GRPCChannelManager.class).addChannelL
    }

    @Override
    public void boot() throws Throwable {
        lastLogTime = System.currentTimeMillis();
        segmentUplinkedCounter = 0;
        segmentAbandonedCounter = 0;
        carrier = new DataCarrier<TraceSegment>(CHANNEL_SIZE, BUFFER_SIZE);
        carrier.setBufferStrategy(BufferStrategy.IF_POSSIBLE);
        carrier.consume(this, 1);
    }

    @Override
    public void onComplete() throws Throwable {
        TracingContext.ListenerManager.add(this);
    }

    @Override
    public void shutdown() throws Throwable {
        TracingContext.ListenerManager.remove(this);
        carrier.shutdownConsumers();
    }

    @Override
    public void init() {

    }

    @Override
    public void consume(List<TraceSegment> data) {
        if (CONNECTED.equals(status)) {
            final GRPCStreamServiceStatus status = new GRPCStreamServiceStatus(fa
            StreamObserver<UpstreamSegment> upstreamSegmentStreamObserver = servi
                @Override
                public void onNext(Commands commands) {
                    ServiceManager.INSTANCE.findService(CommandService.class).rec
                }

                @Override
                public void onError(Throwable throwable) {
                    status.finished();
                    if (logger.isErrorEnable()) {
                        logger.error(throwable, "Send UpstreamSegment to collecto
                    }
                    ServiceManager.INSTANCE.findService(GRPCChannelManager.class)
                }

                @Override
                public void onCompleted() {
                    status.finished();
                }
            });

            try {
                for (TraceSegment segment : data) {
                    UpstreamSegment upstreamSegment = segment.transform();
                    upstreamSegmentStreamObserver.onNext(upstreamSegment);
                }
            } catch (Throwable t) {
                logger.error(t, "Transform and send UpstreamSegment to collector
            }

            upstreamSegmentStreamObserver.onCompleted();

            status.wait4Finish();
```

```
            segmentUplinkedCounter += data.size();
        } else {
            segmentAbandonedCounter += data.size();
        }

        printUplinkStatus();
    }

    private void printUplinkStatus() {
        long currentTimeMillis = System.currentTimeMillis();
        if (currentTimeMillis - lastLogTime > 30 * 1000) {
            lastLogTime = currentTimeMillis;
            if (segmentUplinkedCounter > 0) {
                logger.debug("{} trace segments have been sent to collector.", se
                segmentUplinkedCounter = 0;
            }
            if (segmentAbandonedCounter > 0) {
                logger.debug("{} trace segments have been abandoned, cause by no
                segmentAbandonedCounter = 0;
            }
        }
    }

    @Override
    public void onError(List<TraceSegment> data, Throwable t) {
        logger.error(t, "Try to send {} trace segments to collector, with unexpec
    }

    @Override
    public void onExit() {

    }

    @Override
    public void afterFinished(TraceSegment traceSegment) {
        if (traceSegment.isIgnore()) {
            return;
        }
        if (!carrier.produce(traceSegment)) {
            if (logger.isDebugEnable()) {
                logger.debug("One trace segment has been abandoned, cause by buff
            }
        }
    }

    @Override
    public void statusChanged(GRPCChannelStatus status) {
        if (CONNECTED.equals(status)) {
            Channel channel = ServiceManager.INSTANCE.findService(GRPCChannelMana
            serviceStub = TraceSegmentReportServiceGrpc.newStub(channel);
        }
        this.status = status;
    }
}
```

- TraceSegmentServiceClient实现了BootService、IConsumer、TracingContextListener、GRPCChannelListener接口；其prepare方法往GRPCChannelManager注册自身的channelListener；其boot方法设置lastLogTime，实例化DataCarrier，并设置其consumer为自身；其onComplete方法执行TracingContext.ListenerManager.add(this)；其shutdown方法执行TracingContext.ListenerManager.remove(this)以及carrier.shutdownConsumers()；其consume方法在status为CONNECTED的时候执行upstreamSegmentStreamObserver.onNext(upstreamSegment)、upstreamSegmentStreamObserver.onCompleted()以及status.wait4Finish()；其afterFinished方法执行carrier.produce(traceSegment)；其statusChanged设置serviceStub及status

ConsumerThread

skywalking-6.6.0/apm-commons/apm-datacarrier/src/main/java/org/apache/skywalking/apm/commons/datacarrier/consumer/ConsumerThread.java

```java
public class ConsumerThread<T> extends Thread {
    private volatile boolean running;
    private IConsumer<T> consumer;
    private List<DataSource> dataSources;
    private long consumeCycle;

    ConsumerThread(String threadName, IConsumer<T> consumer, long consumeCycle) {
        super(threadName);
        this.consumer = consumer;
        running = false;
        dataSources = new ArrayList<DataSource>(1);
        this.consumeCycle = consumeCycle;
    }

    /**
     * add whole buffer to consume
     *
     * @param sourceBuffer
     */
    void addDataSource(QueueBuffer<T> sourceBuffer) {
        this.dataSources.add(new DataSource(sourceBuffer));
    }

    @Override
    public void run() {
        running = true;

        final List<T> consumeList = new ArrayList<T>(1500);
        while (running) {
            if (!consume(consumeList)) {
                try {
                    Thread.sleep(consumeCycle);
                } catch (InterruptedException e) {
                }
            }
        }

        // consumer thread is going to stop
        // consume the last time
        consume(consumeList);

        consumer.onExit();
    }

    private boolean consume(List<T> consumeList) {
        for (DataSource dataSource : dataSources) {
            dataSource.obtain(consumeList);
        }

        if (!consumeList.isEmpty()) {
            try {
                consumer.consume(consumeList);
            } catch (Throwable t) {
                consumer.onError(consumeList, t);
            } finally {
                consumeList.clear();
            }
            return true;
        }
        return false;
    }

    void shutdown() {
        running = false;
    }

    /**
     * DataSource is a refer to {@link Buffer}.
     */
```

```
    class DataSource {
        private QueueBuffer<T> sourceBuffer;

        DataSource(QueueBuffer<T> sourceBuffer) {
            this.sourceBuffer = sourceBuffer;
        }

        void obtain(List<T> consumeList) {
            sourceBuffer.obtain(consumeList);
        }
    }
}
```

- ConsumerThread继承了Thread，其run方法会循环执行consume(consumeList)，跳出循环时会再次执行consume(consumeList)，最后执行consumer.onExit()；consume方法会遍历dataSources，执行其dataSource.obtain(consumeList)，然后在consumeList不为空的时候执行consumer.consume(consumeList)方法

ConsumeDriver

skywalking-6.6.0/apm-commons/apm-datacarrier/src/main/java/org/apache/skywalking/apm/commons/datacarrier/consumer/ConsumeDriver.java

```
public class ConsumeDriver<T> implements IDriver {
    private boolean running;
    private ConsumerThread[] consumerThreads;
    private Channels<T> channels;
    private ReentrantLock lock;

    public ConsumeDriver(String name, Channels<T> channels, Class<? extends ICons
        long consumeCycle) {
        this(channels, num);
        for (int i = 0; i < num; i++) {
            consumerThreads[i] = new ConsumerThread("DataCarrier." + name + ".Con
            consumerThreads[i].setDaemon(true);
        }
    }

    public ConsumeDriver(String name, Channels<T> channels, IConsumer<T> prototyp
        this(channels, num);
        prototype.init();
        for (int i = 0; i < num; i++) {
            consumerThreads[i] = new ConsumerThread("DataCarrier." + name + ".Con
            consumerThreads[i].setDaemon(true);
        }

    }

    private ConsumeDriver(Channels<T> channels, int num) {
        running = false;
        this.channels = channels;
        consumerThreads = new ConsumerThread[num];
        lock = new ReentrantLock();
    }

    private IConsumer<T> getNewConsumerInstance(Class<? extends IConsumer<T>> con
        try {
            IConsumer<T> inst = consumerClass.newInstance();
            inst.init();
            return inst;
        } catch (InstantiationException e) {
            throw new ConsumerCannotBeCreatedException(e);
        } catch (IllegalAccessException e) {
            throw new ConsumerCannotBeCreatedException(e);
        }
    }

    @Override
    public void begin(Channels channels) {
```

```
        if (running) {
            return;
        }
        try {
            lock.lock();
            this.allocateBuffer2Thread();
            for (ConsumerThread consumerThread : consumerThreads) {
                consumerThread.start();
            }
            running = true;
        } finally {
            lock.unlock();
        }
    }

    @Override
    public boolean isRunning(Channels channels) {
        return running;
    }

    private void allocateBuffer2Thread() {
        int channelSize = this.channels.getChannelSize();
        /**
         * if consumerThreads.length < channelSize
         * each consumer will process several channels.
         *
         * if consumerThreads.length == channelSize
         * each consumer will process one channel.
         *
         * if consumerThreads.length > channelSize
         * there will be some threads do nothing.
         */
        for (int channelIndex = 0; channelIndex < channelSize; channelIndex++) {
            int consumerIndex = channelIndex % consumerThreads.length;
            consumerThreads[consumerIndex].addDataSource(channels.getBuffer(chann
        }

    }

    @Override
    public void close(Channels channels) {
        try {
            lock.lock();
            this.running = false;
            for (ConsumerThread consumerThread : consumerThreads) {
                consumerThread.shutdown();
            }
        } finally {
            lock.unlock();
        }
    }
}
```

- ConsumeDriver实现了IDriver接口，其ConsumeDriver会创建num个ConsumerThread；其begin方法会执行allocateBuffer2Thread，给每个consumerThread添加dataSource，然后执行consumerThread.start()；其close方法会执行consumerThread.shutdown()

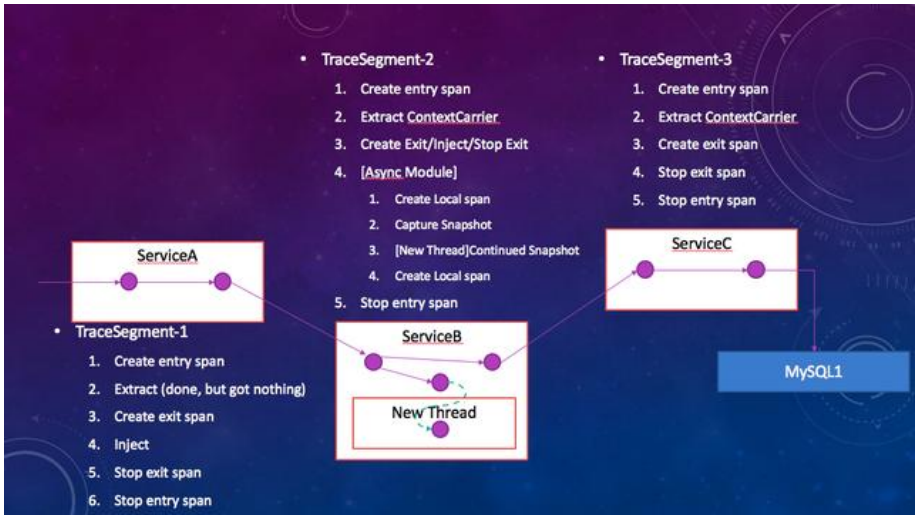
小结

TraceSegmentServiceClient实现了BootService、IConsumer、TracingContextListener、GRPCChannelListener接口；其prepare方法往GRPCChannelManager注册自身的channelListener；其boot方法设置lastLogTime，实例化DataCarrier，并设置其consumer为自身；其onComplete方法执行TracingContext.ListenerManager.add(this)；其shutdown方法执行TracingContext.ListenerManager.remove(this)以及carrier.shutdownConsumers()；其consume方法在status为CONNECTED的时候执行upstreamSegmentStreamObserver.onNext(upstreamSegment)、upstreamSegmentStreamObserver.onCompleted()以及status.wait4Finish()；其afterFinished方法执行carrier.produce(traceSegment)；其statusChanged设置serviceStub及status

doc

- TraceSegmentServiceClient本文参考原文-http://bjbsair.com/2020-03-22/tech-info/5102/
  序
-

本文主要研究一下skywalking的TraceSegmentServiceClient



TracingContextListener

skywalking-6.6.0/apm-sniffer/apm-agent-core/src/main/java/org/apache/skywalking/apm/agent/core/context/TracingContextListener.java

```
public interface TracingContextListener {
    void afterFinished(TraceSegment traceSegment);
}
```

- TracingContextListener定义了afterFinished方法，其参数为TraceSegment

TraceSegment

skywalking-6.6.0/apm-sniffer/apm-agent-core/src/main/java/org/apache/skywalking/apm/agent/core/context/trace/TraceSegment.java

```
public class TraceSegment {

    private ID traceSegmentId;

    private List<TraceSegmentRef> refs;

    private List<AbstractTracingSpan> spans;

    private DistributedTraceIds relatedGlobalTraces;

    private boolean ignore = false;

    private boolean isSizeLimited = false;

    private final long createTime;

    public TraceSegment() {
        this.traceSegmentId = GlobalIdGenerator.generate();
        this.spans = new LinkedList<AbstractTracingSpan>();
        this.relatedGlobalTraces = new DistributedTraceIds();
        this.relatedGlobalTraces.append(new NewDistributedTraceId());
        this.createTime = System.currentTimeMillis();
    }

    public void ref(TraceSegmentRef refSegment) {
        if (refs == null) {
            refs = new LinkedList<TraceSegmentRef>();
        }
        if (!refs.contains(refSegment)) {
            refs.add(refSegment);
        }
```

```
    }

    public void relatedGlobalTraces(DistributedTraceId distributedTraceId) {
        relatedGlobalTraces.append(distributedTraceId);
    }

    public void archive(AbstractTracingSpan finishedSpan) {
        spans.add(finishedSpan);
    }

    public TraceSegment finish(boolean isSizeLimited) {
        this.isSizeLimited = isSizeLimited;
        return this;
    }

    public ID getTraceSegmentId() {
        return traceSegmentId;
    }

    public int getServiceId() {
        return RemoteDownstreamConfig.Agent.SERVICE_ID;
    }

    public boolean hasRef() {
        return !(refs == null || refs.size() == 0);
    }

    public List<TraceSegmentRef> getRefs() {
        return refs;
    }

    public List<DistributedTraceId> getRelatedGlobalTraces() {
        return relatedGlobalTraces.getRelatedGlobalTraces();
    }

    public boolean isSingleSpanSegment() {
        return this.spans != null && this.spans.size() == 1;
    }

    public boolean isIgnore() {
        return ignore;
    }

    public void setIgnore(boolean ignore) {
        this.ignore = ignore;
    }

    public UpstreamSegment transform() {
        UpstreamSegment.Builder upstreamBuilder = UpstreamSegment.newBuilder();
        for (DistributedTraceId distributedTraceId : getRelatedGlobalTraces()) {
            upstreamBuilder = upstreamBuilder.addGlobalTraceIds(distributedTraceI
        }
        SegmentObject.Builder traceSegmentBuilder = SegmentObject.newBuilder();
        /**
         * Trace Segment
         */
        traceSegmentBuilder.setTraceSegmentId(this.traceSegmentId.transform());
        // Don't serialize TraceSegmentReference

        // SpanObject
        for (AbstractTracingSpan span : this.spans) {
            traceSegmentBuilder.addSpans(span.transform());
        }
        traceSegmentBuilder.setServiceId(RemoteDownstreamConfig.Agent.SERVICE_ID)
        traceSegmentBuilder.setServiceInstanceId(RemoteDownstreamConfig.Agent.SER
        traceSegmentBuilder.setIsSizeLimited(this.isSizeLimited);

        upstreamBuilder.setSegment(traceSegmentBuilder.build().toByteString());
        return upstreamBuilder.build();
    }
```

```
    @Override
    public String toString() {
        return "TraceSegment{" +
            "traceSegmentId='" + traceSegmentId + '\'' +
            ", refs=" + refs +
            ", spans=" + spans +
            ", relatedGlobalTraces=" + relatedGlobalTraces +
            '}';
    }

    public int getApplicationInstanceId() {
        return RemoteDownstreamConfig.Agent.SERVICE_INSTANCE_ID;
    }

    public long createTime() {
        return this.createTime;
    }
}
```

- TraceSegment定义了traceSegmentId、refs、spans、relatedGlobalTraces等属性；它提供了ref、relatedGlobalTraces、archive 、finish、transform等方法

IConsumer

skywalking-6.6.0/apm-commons/apm-datacarrier/src/main/java/org/apache/skywalking/apm/commons/datacarrier/consumer/IConsumer.java

```
public interface IConsumer<T> {
    void init();

    void consume(List<T> data);

    void onError(List<T> data, Throwable t);

    void onExit();
}
```

- IConsumer定义了init、consume、onError、onExit方法

TraceSegmentServiceClient

skywalking-6.6.0/apm-sniffer/apm-agent-core/src/main/java/org/apache/skywalking/apm/agent/core/remote/TraceSegmentServiceClient.java

```
@DefaultImplementor
public class TraceSegmentServiceClient implements BootService, IConsumer<TraceSeg
    private static final ILog logger = LogManager.getLogger(TraceSegmentServiceCl
    private static final int TIMEOUT = 30 * 1000;

    private long lastLogTime;
    private long segmentUplinkedCounter;
    private long segmentAbandonedCounter;
    private volatile DataCarrier<TraceSegment> carrier;
    private volatile TraceSegmentReportServiceGrpc.TraceSegmentReportServiceStub
    private volatile GRPCChannelStatus status = GRPCChannelStatus.DISCONNECT;

    @Override
    public void prepare() throws Throwable {
        ServiceManager.INSTANCE.findService(GRPCChannelManager.class).addChannelL
    }

    @Override
    public void boot() throws Throwable {
        lastLogTime = System.currentTimeMillis();
        segmentUplinkedCounter = 0;
        segmentAbandonedCounter = 0;
        carrier = new DataCarrier<TraceSegment>(CHANNEL_SIZE, BUFFER_SIZE);
        carrier.setBufferStrategy(BufferStrategy.IF_POSSIBLE);
        carrier.consume(this, 1);
    }
```

```
    @Override
    public void onComplete() throws Throwable {
        TracingContext.ListenerManager.add(this);
    }

    @Override
    public void shutdown() throws Throwable {
        TracingContext.ListenerManager.remove(this);
        carrier.shutdownConsumers();
    }

    @Override
    public void init() {

    }

    @Override
    public void consume(List<TraceSegment> data) {
        if (CONNECTED.equals(status)) {
            final GRPCStreamServiceStatus status = new GRPCStreamServiceStatus(fa
            StreamObserver<UpstreamSegment> upstreamSegmentStreamObserver = servi
                @Override
                public void onNext(Commands commands) {
                    ServiceManager.INSTANCE.findService(CommandService.class).rec
                }

                @Override
                public void onError(Throwable throwable) {
                    status.finished();
                    if (logger.isErrorEnable()) {
                        logger.error(throwable, "Send UpstreamSegment to collecto
                    }
                    ServiceManager.INSTANCE.findService(GRPCChannelManager.class)
                }

                @Override
                public void onCompleted() {
                    status.finished();
                }
            });

            try {
                for (TraceSegment segment : data) {
                    UpstreamSegment upstreamSegment = segment.transform();
                    upstreamSegmentStreamObserver.onNext(upstreamSegment);
                }
            } catch (Throwable t) {
                logger.error(t, "Transform and send UpstreamSegment to collector
            }

            upstreamSegmentStreamObserver.onCompleted();

            status.wait4Finish();
            segmentUplinkedCounter += data.size();
        } else {
            segmentAbandonedCounter += data.size();
        }

        printUplinkStatus();
    }

    private void printUplinkStatus() {
        long currentTimeMillis = System.currentTimeMillis();
        if (currentTimeMillis - lastLogTime > 30 * 1000) {
            lastLogTime = currentTimeMillis;
            if (segmentUplinkedCounter > 0) {
                logger.debug("{} trace segments have been sent to collector.", se
                segmentUplinkedCounter = 0;
            }
```

```
            if (segmentAbandonedCounter > 0) {
                logger.debug("{} trace segments have been abandoned, cause by no
                segmentAbandonedCounter = 0;
            }
        }
    }

    @Override
    public void onError(List<TraceSegment> data, Throwable t) {
        logger.error(t, "Try to send {} trace segments to collector, with unexpec
    }

    @Override
    public void onExit() {

    }

    @Override
    public void afterFinished(TraceSegment traceSegment) {
        if (traceSegment.isIgnore()) {
            return;
        }
        if (!carrier.produce(traceSegment)) {
            if (logger.isDebugEnable()) {
                logger.debug("One trace segment has been abandoned, cause by buff
            }
        }
    }

    @Override
    public void statusChanged(GRPCChannelStatus status) {
        if (CONNECTED.equals(status)) {
            Channel channel = ServiceManager.INSTANCE.findService(GRPCChannelMana
            serviceStub = TraceSegmentReportServiceGrpc.newStub(channel);
        }
        this.status = status;
    }
}
```

- TraceSegmentServiceClient实现了BootService、IConsumer、TracingContextListener、GRPCChannelListener接口；其prepare方法往GRPCChannelManager注册自身的channelListener；其boot方法设置lastLogTime，实例化DataCarrier，并设置其consumer为自身；其onComplete方法执行TracingContext.ListenerManager.add(this)；其shutdown方法执行TracingContext.ListenerManager.remove(this)以及carrier.shutdownConsumers()；其consume方法在status为CONNECTED的时候执行upstreamSegmentStreamObserver.onNext(upstreamSegment)、upstreamSegmentStreamObserver.onCompleted()以及status.wait4Finish()；其afterFinished方法执行carrier.produce(traceSegment)；其statusChanged设置serviceStub及status

ConsumerThread

skywalking-6.6.0/apm-commons/apm-datacarrier/src/main/java/org/apache/skywalking/apm/commons/datacarrier/consumer/ConsumerThread.java

```
public class ConsumerThread<T> extends Thread {
    private volatile boolean running;
    private IConsumer<T> consumer;
    private List<DataSource> dataSources;
    private long consumeCycle;

    ConsumerThread(String threadName, IConsumer<T> consumer, long consumeCycle) {
        super(threadName);
        this.consumer = consumer;
        running = false;
        dataSources = new ArrayList<DataSource>(1);
        this.consumeCycle = consumeCycle;
    }

    /**
     * add whole buffer to consume
```

```
     *
     * @param sourceBuffer
     */
    void addDataSource(QueueBuffer<T> sourceBuffer) {
        this.dataSources.add(new DataSource(sourceBuffer));
    }

    @Override
    public void run() {
        running = true;

        final List<T> consumeList = new ArrayList<T>(1500);
        while (running) {
            if (!consume(consumeList)) {
                try {
                    Thread.sleep(consumeCycle);
                } catch (InterruptedException e) {
                }
            }
        }

        // consumer thread is going to stop
        // consume the last time
        consume(consumeList);

        consumer.onExit();
    }

    private boolean consume(List<T> consumeList) {
        for (DataSource dataSource : dataSources) {
            dataSource.obtain(consumeList);
        }

        if (!consumeList.isEmpty()) {
            try {
                consumer.consume(consumeList);
            } catch (Throwable t) {
                consumer.onError(consumeList, t);
            } finally {
                consumeList.clear();
            }
            return true;
        }
        return false;
    }

    void shutdown() {
        running = false;
    }

    /**
     * DataSource is a refer to {@link Buffer}.
     */
    class DataSource {
        private QueueBuffer<T> sourceBuffer;

        DataSource(QueueBuffer<T> sourceBuffer) {
            this.sourceBuffer = sourceBuffer;
        }

        void obtain(List<T> consumeList) {
            sourceBuffer.obtain(consumeList);
        }
    }
}
```

- ConsumerThread继承了Thread，其run方法会循环执行consume(consumeList)，跳出循环时会再次执行consume(consumeList)，最后执行consumer.onExit()；consume方法会遍历dataSources，执行其

dataSource.obtain(consumeList)，然后在consumeList不为空的时候执行consumer.consume(consumeList)方法

ConsumeDriver

skywalking-6.6.0/apm-commons/apm-datacarrier/src/main/java/org/apache/skywalking/apm/commons/datacarrier/consumer/ConsumeDriver.java

```
public class ConsumeDriver<T> implements IDriver {
    private boolean running;
    private ConsumerThread[] consumerThreads;
    private Channels<T> channels;
    private ReentrantLock lock;

    public ConsumeDriver(String name, Channels<T> channels, Class<? extends ICons
        long consumeCycle) {
        this(channels, num);
        for (int i = 0; i < num; i++) {
            consumerThreads[i] = new ConsumerThread("DataCarrier." + name + ".Con
            consumerThreads[i].setDaemon(true);
        }
    }

    public ConsumeDriver(String name, Channels<T> channels, IConsumer<T> prototyp
        this(channels, num);
        prototype.init();
        for (int i = 0; i < num; i++) {
            consumerThreads[i] = new ConsumerThread("DataCarrier." + name + ".Con
            consumerThreads[i].setDaemon(true);
        }

    }

    private ConsumeDriver(Channels<T> channels, int num) {
        running = false;
        this.channels = channels;
        consumerThreads = new ConsumerThread[num];
        lock = new ReentrantLock();
    }

    private IConsumer<T> getNewConsumerInstance(Class<? extends IConsumer<T>> con
        try {
            IConsumer<T> inst = consumerClass.newInstance();
            inst.init();
            return inst;
        } catch (InstantiationException e) {
            throw new ConsumerCannotBeCreatedException(e);
        } catch (IllegalAccessException e) {
            throw new ConsumerCannotBeCreatedException(e);
        }
    }

    @Override
    public void begin(Channels channels) {
        if (running) {
            return;
        }
        try {
            lock.lock();
            this.allocateBuffer2Thread();
            for (ConsumerThread consumerThread : consumerThreads) {
                consumerThread.start();
            }
            running = true;
        } finally {
            lock.unlock();
        }
    }

    @Override
```

```
    public boolean isRunning(Channels channels) {
        return running;
    }

    private void allocateBuffer2Thread() {
        int channelSize = this.channels.getChannelSize();
        /**
         * if consumerThreads.length < channelSize
         * each consumer will process several channels.
         *
         * if consumerThreads.length == channelSize
         * each consumer will process one channel.
         *
         * if consumerThreads.length > channelSize
         * there will be some threads do nothing.
         */
        for (int channelIndex = 0; channelIndex < channelSize; channelIndex++) {
            int consumerIndex = channelIndex % consumerThreads.length;
            consumerThreads[consumerIndex].addDataSource(channels.getBuffer(chann
        }

    }

    @Override
    public void close(Channels channels) {
        try {
            lock.lock();
            this.running = false;
            for (ConsumerThread consumerThread : consumerThreads) {
                consumerThread.shutdown();
            }
        } finally {
            lock.unlock();
        }
    }
}
```

- ConsumeDriver实现了IDriver接口，其ConsumeDriver会创建num个ConsumerThread；其begin方法会执行allocateBuffer2Thread，给每个consumerThread添加dataSource，然后执行consumerThread.start()；其close方法会执行consumerThread.shutdown()

小结

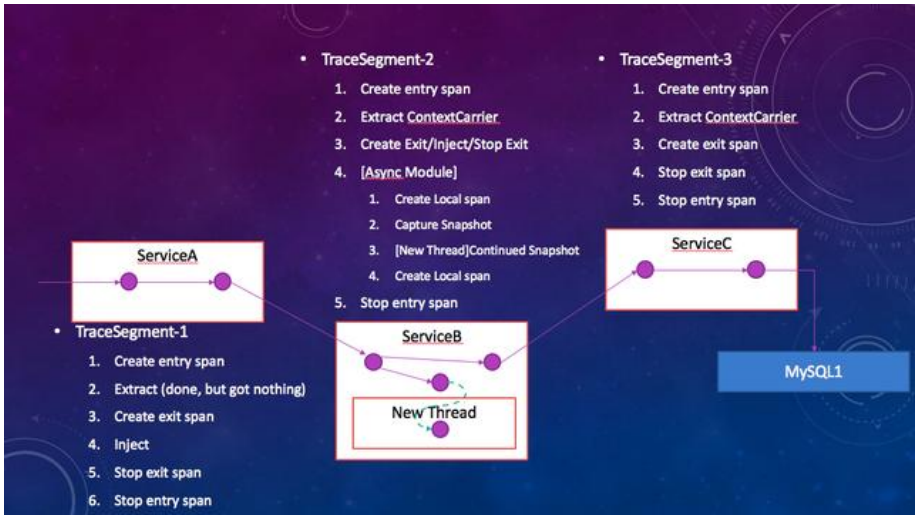
TraceSegmentServiceClient实现了BootService、IConsumer、TracingContextListener、GRPCChannelListener接口；其prepare方法往GRPCChannelManager注册自身的channelListener；其boot方法设置lastLogTime，实例化DataCarrier，并设置其consumer为自身；其onComplete方法执行TracingContext.ListenerManager.add(this)；其shutdown方法执行TracingContext.ListenerManager.remove(this)以及carrier.shutdownConsumers()；其consume方法在status为CONNECTED的时候执行upstreamSegmentStreamObserver.onNext(upstreamSegment)、upstreamSegmentStreamObserver.onCompleted()以及status.wait4Finish()；其afterFinished方法执行carrier.produce(traceSegment)；其statusChanged设置serviceStub及status

doc

- TraceSegmentServiceClient本文参考原文-http://bjbsair.com/2020-03-22/tech-info/5102/
  序

-

本文主要研究一下skywalking的TraceSegmentServiceClient



TracingContextListener

skywalking-6.6.0/apm-sniffer/apm-agent-core/src/main/java/org/apache/skywalking/apm/agent/core/context/TracingContextListener.java

```
public interface TracingContextListener {
    void afterFinished(TraceSegment traceSegment);
}
```

- TracingContextListener定义了afterFinished方法，其参数为TraceSegment

TraceSegment

skywalking-6.6.0/apm-sniffer/apm-agent-core/src/main/java/org/apache/skywalking/apm/agent/core/context/trace/TraceSegment.java

```
public class TraceSegment {

    private ID traceSegmentId;

    private List<TraceSegmentRef> refs;

    private List<AbstractTracingSpan> spans;

    private DistributedTraceIds relatedGlobalTraces;

    private boolean ignore = false;

    private boolean isSizeLimited = false;

    private final long createTime;

    public TraceSegment() {
        this.traceSegmentId = GlobalIdGenerator.generate();
        this.spans = new LinkedList<AbstractTracingSpan>();
        this.relatedGlobalTraces = new DistributedTraceIds();
        this.relatedGlobalTraces.append(new NewDistributedTraceId());
        this.createTime = System.currentTimeMillis();
    }

    public void ref(TraceSegmentRef refSegment) {
        if (refs == null) {
            refs = new LinkedList<TraceSegmentRef>();
        }
        if (!refs.contains(refSegment)) {
            refs.add(refSegment);
        }
    }

    public void relatedGlobalTraces(DistributedTraceId distributedTraceId) {
        relatedGlobalTraces.append(distributedTraceId);
    }
```

```
    public void archive(AbstractTracingSpan finishedSpan) {
        spans.add(finishedSpan);
    }

    public TraceSegment finish(boolean isSizeLimited) {
        this.isSizeLimited = isSizeLimited;
        return this;
    }

    public ID getTraceSegmentId() {
        return traceSegmentId;
    }

    public int getServiceId() {
        return RemoteDownstreamConfig.Agent.SERVICE_ID;
    }

    public boolean hasRef() {
        return !(refs == null || refs.size() == 0);
    }

    public List<TraceSegmentRef> getRefs() {
        return refs;
    }

    public List<DistributedTraceId> getRelatedGlobalTraces() {
        return relatedGlobalTraces.getRelatedGlobalTraces();
    }

    public boolean isSingleSpanSegment() {
        return this.spans != null && this.spans.size() == 1;
    }

    public boolean isIgnore() {
        return ignore;
    }

    public void setIgnore(boolean ignore) {
        this.ignore = ignore;
    }

    public UpstreamSegment transform() {
        UpstreamSegment.Builder upstreamBuilder = UpstreamSegment.newBuilder();
        for (DistributedTraceId distributedTraceId : getRelatedGlobalTraces()) {
            upstreamBuilder = upstreamBuilder.addGlobalTraceIds(distributedTraceI
        }
        SegmentObject.Builder traceSegmentBuilder = SegmentObject.newBuilder();
        /**
         * Trace Segment
         */
        traceSegmentBuilder.setTraceSegmentId(this.traceSegmentId.transform());
        // Don't serialize TraceSegmentReference

        // SpanObject
        for (AbstractTracingSpan span : this.spans) {
            traceSegmentBuilder.addSpans(span.transform());
        }
        traceSegmentBuilder.setServiceId(RemoteDownstreamConfig.Agent.SERVICE_ID)
        traceSegmentBuilder.setServiceInstanceId(RemoteDownstreamConfig.Agent.SER
        traceSegmentBuilder.setIsSizeLimited(this.isSizeLimited);

        upstreamBuilder.setSegment(traceSegmentBuilder.build().toByteString());
        return upstreamBuilder.build();
    }

    @Override
    public String toString() {
        return "TraceSegment{" +
            "traceSegmentId='" + traceSegmentId + '\'' +
            ", refs=" + refs +
```

```
            ", spans=" + spans +
            ", relatedGlobalTraces=" + relatedGlobalTraces +
            '}';
    }

    public int getApplicationInstanceId() {
        return RemoteDownstreamConfig.Agent.SERVICE_INSTANCE_ID;
    }

    public long createTime() {
        return this.createTime;
    }
}
```

- TraceSegment定义了traceSegmentId、refs、spans、relatedGlobalTraces等属性；它提供了ref、relatedGlobalTraces、archive 、finish、transform等方法

IConsumer

skywalking-6.6.0/apm-commons/apm-datacarrier/src/main/java/org/apache/skywalking/apm/commons/datacarrier/consumer/IConsumer.java

```
public interface IConsumer<T> {
    void init();

    void consume(List<T> data);

    void onError(List<T> data, Throwable t);

    void onExit();
}
```

- IConsumer定义了init、consume、onError、onExit方法

TraceSegmentServiceClient

skywalking-6.6.0/apm-sniffer/apm-agent-core/src/main/java/org/apache/skywalking/apm/agent/core/remote/TraceSegmentServiceClient.java

```
@DefaultImplementor
public class TraceSegmentServiceClient implements BootService, IConsumer<TraceSeg
    private static final ILog logger = LogManager.getLogger(TraceSegmentServiceCl
    private static final int TIMEOUT = 30 * 1000;

    private long lastLogTime;
    private long segmentUplinkedCounter;
    private long segmentAbandonedCounter;
    private volatile DataCarrier<TraceSegment> carrier;
    private volatile TraceSegmentReportServiceGrpc.TraceSegmentReportServiceStub
    private volatile GRPCChannelStatus status = GRPCChannelStatus.DISCONNECT;

    @Override
    public void prepare() throws Throwable {
        ServiceManager.INSTANCE.findService(GRPCChannelManager.class).addChannelL
    }

    @Override
    public void boot() throws Throwable {
        lastLogTime = System.currentTimeMillis();
        segmentUplinkedCounter = 0;
        segmentAbandonedCounter = 0;
        carrier = new DataCarrier<TraceSegment>(CHANNEL_SIZE, BUFFER_SIZE);
        carrier.setBufferStrategy(BufferStrategy.IF_POSSIBLE);
        carrier.consume(this, 1);
    }

    @Override
    public void onComplete() throws Throwable {
        TracingContext.ListenerManager.add(this);
    }
```

```
    @Override
    public void shutdown() throws Throwable {
        TracingContext.ListenerManager.remove(this);
        carrier.shutdownConsumers();
    }

    @Override
    public void init() {

    }

    @Override
    public void consume(List<TraceSegment> data) {
        if (CONNECTED.equals(status)) {
            final GRPCStreamServiceStatus status = new GRPCStreamServiceStatus(fa
            StreamObserver<UpstreamSegment> upstreamSegmentStreamObserver = servi
                @Override
                public void onNext(Commands commands) {
                    ServiceManager.INSTANCE.findService(CommandService.class).rec
                }

                @Override
                public void onError(Throwable throwable) {
                    status.finished();
                    if (logger.isErrorEnable()) {
                        logger.error(throwable, "Send UpstreamSegment to collecto
                    }
                    ServiceManager.INSTANCE.findService(GRPCChannelManager.class)
                }

                @Override
                public void onCompleted() {
                    status.finished();
                }
            });

            try {
                for (TraceSegment segment : data) {
                    UpstreamSegment upstreamSegment = segment.transform();
                    upstreamSegmentStreamObserver.onNext(upstreamSegment);
                }
            } catch (Throwable t) {
                logger.error(t, "Transform and send UpstreamSegment to collector
            }

            upstreamSegmentStreamObserver.onCompleted();

            status.wait4Finish();
            segmentUplinkedCounter += data.size();
        } else {
            segmentAbandonedCounter += data.size();
        }

        printUplinkStatus();
    }

    private void printUplinkStatus() {
        long currentTimeMillis = System.currentTimeMillis();
        if (currentTimeMillis - lastLogTime > 30 * 1000) {
            lastLogTime = currentTimeMillis;
            if (segmentUplinkedCounter > 0) {
                logger.debug("{} trace segments have been sent to collector.", se
                segmentUplinkedCounter = 0;
            }
            if (segmentAbandonedCounter > 0) {
                logger.debug("{} trace segments have been abandoned, cause by no
                segmentAbandonedCounter = 0;
            }
        }
    }
```

```
    @Override
    public void onError(List<TraceSegment> data, Throwable t) {
        logger.error(t, "Try to send {} trace segments to collector, with unexpec
    }

    @Override
    public void onExit() {

    }

    @Override
    public void afterFinished(TraceSegment traceSegment) {
        if (traceSegment.isIgnore()) {
            return;
        }
        if (!carrier.produce(traceSegment)) {
            if (logger.isDebugEnable()) {
                logger.debug("One trace segment has been abandoned, cause by buff
            }
        }
    }

    @Override
    public void statusChanged(GRPCChannelStatus status) {
        if (CONNECTED.equals(status)) {
            Channel channel = ServiceManager.INSTANCE.findService(GRPCChannelMana
            serviceStub = TraceSegmentReportServiceGrpc.newStub(channel);
        }
        this.status = status;
    }
}
```

- TraceSegmentServiceClient实现了BootService、IConsumer、TracingContextListener、GRPCChannelListener接口；其prepare方法往GRPCChannelManager注册自身的channelListener；其boot方法设置lastLogTime，实例化DataCarrier，并设置其consumer为自身；其onComplete方法执行TracingContext.ListenerManager.add(this)；其shutdown方法执行TracingContext.ListenerManager.remove(this)以及carrier.shutdownConsumers()；其consume方法在status为CONNECTED的时候执行upstreamSegmentStreamObserver.onNext(upstreamSegment)、upstreamSegmentStreamObserver.onCompleted()以及status.wait4Finish()；其afterFinished方法执行carrier.produce(traceSegment)；其statusChanged设置serviceStub及status

ConsumerThread

skywalking-6.6.0/apm-commons/apm-datacarrier/src/main/java/org/apache/skywalking/apm/commons/datacarrier/consumer/ConsumerThread.java

```
public class ConsumerThread<T> extends Thread {
    private volatile boolean running;
    private IConsumer<T> consumer;
    private List<DataSource> dataSources;
    private long consumeCycle;

    ConsumerThread(String threadName, IConsumer<T> consumer, long consumeCycle) {
        super(threadName);
        this.consumer = consumer;
        running = false;
        dataSources = new ArrayList<DataSource>(1);
        this.consumeCycle = consumeCycle;
    }

    /**
     * add whole buffer to consume
     *
     * @param sourceBuffer
     */
    void addDataSource(QueueBuffer<T> sourceBuffer) {
        this.dataSources.add(new DataSource(sourceBuffer));
    }
```

```
    @Override
    public void run() {
        running = true;

        final List<T> consumeList = new ArrayList<T>(1500);
        while (running) {
            if (!consume(consumeList)) {
                try {
                    Thread.sleep(consumeCycle);
                } catch (InterruptedException e) {
                }
            }
        }

        // consumer thread is going to stop
        // consume the last time
        consume(consumeList);

        consumer.onExit();
    }

    private boolean consume(List<T> consumeList) {
        for (DataSource dataSource : dataSources) {
            dataSource.obtain(consumeList);
        }

        if (!consumeList.isEmpty()) {
            try {
                consumer.consume(consumeList);
            } catch (Throwable t) {
                consumer.onError(consumeList, t);
            } finally {
                consumeList.clear();
            }
            return true;
        }
        return false;
    }

    void shutdown() {
        running = false;
    }

    /**
     * DataSource is a refer to {@link Buffer}.
     */
    class DataSource {
        private QueueBuffer<T> sourceBuffer;

        DataSource(QueueBuffer<T> sourceBuffer) {
            this.sourceBuffer = sourceBuffer;
        }

        void obtain(List<T> consumeList) {
            sourceBuffer.obtain(consumeList);
        }
    }
}
```

- ConsumerThread继承了Thread，其run方法会循环执行consume(consumeList)，跳出循环时会再次执行consume(consumeList)，最后执行consumer.onExit()；consume方法会遍历dataSources，执行其dataSource.obtain(consumeList)，然后在consumeList不为空的时候执行consumer.consume(consumeList)方法

ConsumeDriver

skywalking-6.6.0/apm-commons/apm-datacarrier/src/main/java/org/apache/skywalking/apm/commons/datacarrier/consumer/ConsumeDriver.java

```
public class ConsumeDriver<T> implements IDriver {
    private boolean running;
    private ConsumerThread[] consumerThreads;
    private Channels<T> channels;
    private ReentrantLock lock;

    public ConsumeDriver(String name, Channels<T> channels, Class<? extends ICons
        long consumeCycle) {
        this(channels, num);
        for (int i = 0; i < num; i++) {
            consumerThreads[i] = new ConsumerThread("DataCarrier." + name + ".Con
            consumerThreads[i].setDaemon(true);
        }
    }

    public ConsumeDriver(String name, Channels<T> channels, IConsumer<T> prototyp
        this(channels, num);
        prototype.init();
        for (int i = 0; i < num; i++) {
            consumerThreads[i] = new ConsumerThread("DataCarrier." + name + ".Con
            consumerThreads[i].setDaemon(true);
        }

    }

    private ConsumeDriver(Channels<T> channels, int num) {
        running = false;
        this.channels = channels;
        consumerThreads = new ConsumerThread[num];
        lock = new ReentrantLock();
    }

    private IConsumer<T> getNewConsumerInstance(Class<? extends IConsumer<T>> con
        try {
            IConsumer<T> inst = consumerClass.newInstance();
            inst.init();
            return inst;
        } catch (InstantiationException e) {
            throw new ConsumerCannotBeCreatedException(e);
        } catch (IllegalAccessException e) {
            throw new ConsumerCannotBeCreatedException(e);
        }
    }

    @Override
    public void begin(Channels channels) {
        if (running) {
            return;
        }
        try {
            lock.lock();
            this.allocateBuffer2Thread();
            for (ConsumerThread consumerThread : consumerThreads) {
                consumerThread.start();
            }
            running = true;
        } finally {
            lock.unlock();
        }
    }

    @Override
    public boolean isRunning(Channels channels) {
        return running;
    }

    private void allocateBuffer2Thread() {
        int channelSize = this.channels.getChannelSize();
        /**
```

```
         * if consumerThreads.length < channelSize
         * each consumer will process several channels.
         *
         * if consumerThreads.length == channelSize
         * each consumer will process one channel.
         *
         * if consumerThreads.length > channelSize
         * there will be some threads do nothing.
         */
        for (int channelIndex = 0; channelIndex < channelSize; channelIndex++) {
            int consumerIndex = channelIndex % consumerThreads.length;
            consumerThreads[consumerIndex].addDataSource(channels.getBuffer(chann
        }

    }

    @Override
    public void close(Channels channels) {
        try {
            lock.lock();
            this.running = false;
            for (ConsumerThread consumerThread : consumerThreads) {
                consumerThread.shutdown();
            }
        } finally {
            lock.unlock();
        }
    }
}
```

- ConsumeDriver实现了IDriver接口，其ConsumeDriver会创建num个ConsumerThread；其begin方法会执行allocateBuffer2Thread，给每个consumerThread添加dataSource，然后执行consumerThread.start()；其close方法会执行consumerThread.shutdown()

小结

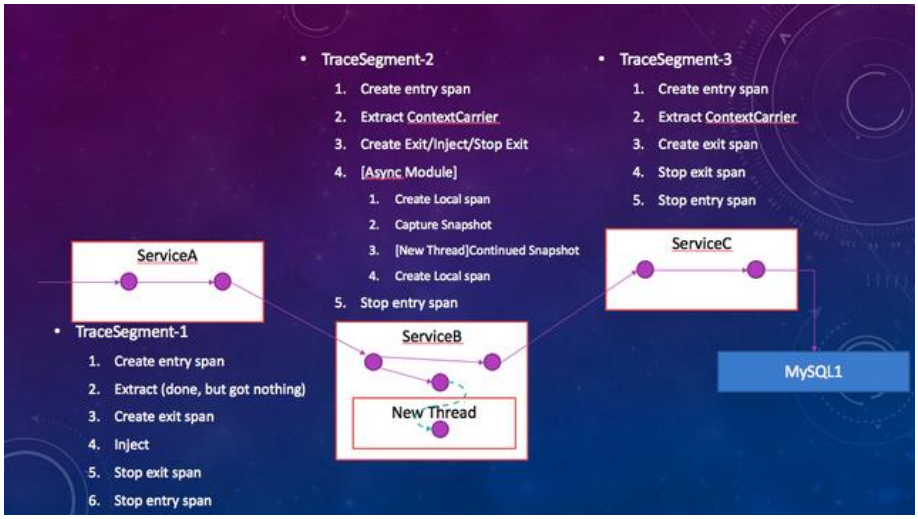
TraceSegmentServiceClient实现了BootService、IConsumer、TracingContextListener、GRPCChannelListener接口；其prepare方法往GRPCChannelManager注册自身的channelListener；其boot方法设置lastLogTime，实例化DataCarrier，并设置其consumer为自身；其onComplete方法执行TracingContext.ListenerManager.add(this)；其shutdown方法执行TracingContext.ListenerManager.remove(this)以及carrier.shutdownConsumers()；其consume方法在status为CONNECTED的时候执行upstreamSegmentStreamObserver.onNext(upstreamSegment)、upstreamSegmentStreamObserver.onCompleted()以及status.wait4Finish()；其afterFinished方法执行carrier.produce(traceSegment)；其statusChanged设置serviceStub及status

doc

- TraceSegmentServiceClient本文参考原文-http://bjbsair.com/2020-03-22/tech-info/5102/
  序

-

本文主要研究一下skywalking的TraceSegmentServiceClient



TracingContextListener

skywalking-6.6.0/apm-sniffer/apm-agent-core/src/main/java/org/apache/skywalking/apm/agent/core/context/TracingContextListener.java

```
public interface TracingContextListener {
    void afterFinished(TraceSegment traceSegment);
}
```

- TracingContextListener定义了afterFinished方法，其参数为TraceSegment

TraceSegment

skywalking-6.6.0/apm-sniffer/apm-agent-core/src/main/java/org/apache/skywalking/apm/agent/core/context/trace/TraceSegment.java

```
public class TraceSegment {

    private ID traceSegmentId;

    private List<TraceSegmentRef> refs;

    private List<AbstractTracingSpan> spans;

    private DistributedTraceIds relatedGlobalTraces;

    private boolean ignore = false;

    private boolean isSizeLimited = false;

    private final long createTime;

    public TraceSegment() {
        this.traceSegmentId = GlobalIdGenerator.generate();
        this.spans = new LinkedList<AbstractTracingSpan>();
        this.relatedGlobalTraces = new DistributedTraceIds();
        this.relatedGlobalTraces.append(new NewDistributedTraceId());
        this.createTime = System.currentTimeMillis();
    }

    public void ref(TraceSegmentRef refSegment) {
        if (refs == null) {
            refs = new LinkedList<TraceSegmentRef>();
        }
        if (!refs.contains(refSegment)) {
            refs.add(refSegment);
        }
    }

    public void relatedGlobalTraces(DistributedTraceId distributedTraceId) {
        relatedGlobalTraces.append(distributedTraceId);
    }
```

```
    public void archive(AbstractTracingSpan finishedSpan) {
        spans.add(finishedSpan);
    }

    public TraceSegment finish(boolean isSizeLimited) {
        this.isSizeLimited = isSizeLimited;
        return this;
    }

    public ID getTraceSegmentId() {
        return traceSegmentId;
    }

    public int getServiceId() {
        return RemoteDownstreamConfig.Agent.SERVICE_ID;
    }

    public boolean hasRef() {
        return !(refs == null || refs.size() == 0);
    }

    public List<TraceSegmentRef> getRefs() {
        return refs;
    }

    public List<DistributedTraceId> getRelatedGlobalTraces() {
        return relatedGlobalTraces.getRelatedGlobalTraces();
    }

    public boolean isSingleSpanSegment() {
        return this.spans != null && this.spans.size() == 1;
    }

    public boolean isIgnore() {
        return ignore;
    }

    public void setIgnore(boolean ignore) {
        this.ignore = ignore;
    }

    public UpstreamSegment transform() {
        UpstreamSegment.Builder upstreamBuilder = UpstreamSegment.newBuilder();
        for (DistributedTraceId distributedTraceId : getRelatedGlobalTraces()) {
            upstreamBuilder = upstreamBuilder.addGlobalTraceIds(distributedTraceI
        }
        SegmentObject.Builder traceSegmentBuilder = SegmentObject.newBuilder();
        /**
         * Trace Segment
         */
        traceSegmentBuilder.setTraceSegmentId(this.traceSegmentId.transform());
        // Don't serialize TraceSegmentReference

        // SpanObject
        for (AbstractTracingSpan span : this.spans) {
            traceSegmentBuilder.addSpans(span.transform());
        }
        traceSegmentBuilder.setServiceId(RemoteDownstreamConfig.Agent.SERVICE_ID)
        traceSegmentBuilder.setServiceInstanceId(RemoteDownstreamConfig.Agent.SER
        traceSegmentBuilder.setIsSizeLimited(this.isSizeLimited);

        upstreamBuilder.setSegment(traceSegmentBuilder.build().toByteString());
        return upstreamBuilder.build();
    }

    @Override
    public String toString() {
        return "TraceSegment{" +
            "traceSegmentId='" + traceSegmentId + '\'' +
            ", refs=" + refs +
```

```
            ", spans=" + spans +
            ", relatedGlobalTraces=" + relatedGlobalTraces +
            '}';
    }

    public int getApplicationInstanceId() {
        return RemoteDownstreamConfig.Agent.SERVICE_INSTANCE_ID;
    }

    public long createTime() {
        return this.createTime;
    }
}
```

- TraceSegment定义了traceSegmentId、refs、spans、relatedGlobalTraces等属性；它提供了ref、relatedGlobalTraces、archive 、finish、transform等方法

IConsumer

skywalking-6.6.0/apm-commons/apm-datacarrier/src/main/java/org/apache/skywalking/apm/commons/datacarrier/consumer/IConsumer.java

```
public interface IConsumer<T> {
    void init();

    void consume(List<T> data);

    void onError(List<T> data, Throwable t);

    void onExit();
}
```

- IConsumer定义了init、consume、onError、onExit方法

TraceSegmentServiceClient

skywalking-6.6.0/apm-sniffer/apm-agent-core/src/main/java/org/apache/skywalking/apm/agent/core/remote/TraceSegmentServiceClient.java

```
@DefaultImplementor
public class TraceSegmentServiceClient implements BootService, IConsumer<TraceSeg
    private static final ILog logger = LogManager.getLogger(TraceSegmentServiceCl
    private static final int TIMEOUT = 30 * 1000;

    private long lastLogTime;
    private long segmentUplinkedCounter;
    private long segmentAbandonedCounter;
    private volatile DataCarrier<TraceSegment> carrier;
    private volatile TraceSegmentReportServiceGrpc.TraceSegmentReportServiceStub
    private volatile GRPCChannelStatus status = GRPCChannelStatus.DISCONNECT;

    @Override
    public void prepare() throws Throwable {
        ServiceManager.INSTANCE.findService(GRPCChannelManager.class).addChannelL
    }

    @Override
    public void boot() throws Throwable {
        lastLogTime = System.currentTimeMillis();
        segmentUplinkedCounter = 0;
        segmentAbandonedCounter = 0;
        carrier = new DataCarrier<TraceSegment>(CHANNEL_SIZE, BUFFER_SIZE);
        carrier.setBufferStrategy(BufferStrategy.IF_POSSIBLE);
        carrier.consume(this, 1);
    }

    @Override
    public void onComplete() throws Throwable {
        TracingContext.ListenerManager.add(this);
    }
```

```
    @Override
    public void shutdown() throws Throwable {
        TracingContext.ListenerManager.remove(this);
        carrier.shutdownConsumers();
    }

    @Override
    public void init() {

    }

    @Override
    public void consume(List<TraceSegment> data) {
        if (CONNECTED.equals(status)) {
            final GRPCStreamServiceStatus status = new GRPCStreamServiceStatus(fa
            StreamObserver<UpstreamSegment> upstreamSegmentStreamObserver = servi
                @Override
                public void onNext(Commands commands) {
                    ServiceManager.INSTANCE.findService(CommandService.class).rec
                }

                @Override
                public void onError(Throwable throwable) {
                    status.finished();
                    if (logger.isErrorEnable()) {
                        logger.error(throwable, "Send UpstreamSegment to collecto
                    }
                    ServiceManager.INSTANCE.findService(GRPCChannelManager.class)
                }

                @Override
                public void onCompleted() {
                    status.finished();
                }
            });

            try {
                for (TraceSegment segment : data) {
                    UpstreamSegment upstreamSegment = segment.transform();
                    upstreamSegmentStreamObserver.onNext(upstreamSegment);
                }
            } catch (Throwable t) {
                logger.error(t, "Transform and send UpstreamSegment to collector
            }

            upstreamSegmentStreamObserver.onCompleted();

            status.wait4Finish();
            segmentUplinkedCounter += data.size();
        } else {
            segmentAbandonedCounter += data.size();
        }

        printUplinkStatus();
    }

    private void printUplinkStatus() {
        long currentTimeMillis = System.currentTimeMillis();
        if (currentTimeMillis - lastLogTime > 30 * 1000) {
            lastLogTime = currentTimeMillis;
            if (segmentUplinkedCounter > 0) {
                logger.debug("{} trace segments have been sent to collector.", se
                segmentUplinkedCounter = 0;
            }
            if (segmentAbandonedCounter > 0) {
                logger.debug("{} trace segments have been abandoned, cause by no
                segmentAbandonedCounter = 0;
            }
        }
    }
```

```
    @Override
    public void onError(List<TraceSegment> data, Throwable t) {
        logger.error(t, "Try to send {} trace segments to collector, with unexpec
    }

    @Override
    public void onExit() {

    }

    @Override
    public void afterFinished(TraceSegment traceSegment) {
        if (traceSegment.isIgnore()) {
            return;
        }
        if (!carrier.produce(traceSegment)) {
            if (logger.isDebugEnable()) {
                logger.debug("One trace segment has been abandoned, cause by buff
            }
        }
    }

    @Override
    public void statusChanged(GRPCChannelStatus status) {
        if (CONNECTED.equals(status)) {
            Channel channel = ServiceManager.INSTANCE.findService(GRPCChannelMana
            serviceStub = TraceSegmentReportServiceGrpc.newStub(channel);
        }
        this.status = status;
    }
}
```

- TraceSegmentServiceClient实现了BootService、IConsumer、TracingContextListener、GRPCChannelListener接口；其prepare方法往GRPCChannelManager注册自身的channelListener；其boot方法设置lastLogTime，实例化DataCarrier，并设置其consumer为自身；其onComplete方法执行TracingContext.ListenerManager.add(this)；其shutdown方法执行TracingContext.ListenerManager.remove(this)以及carrier.shutdownConsumers()；其consume方法在status为CONNECTED的时候执行upstreamSegmentStreamObserver.onNext(upstreamSegment)、upstreamSegmentStreamObserver.onCompleted()以及status.wait4Finish()；其afterFinished方法执行carrier.produce(traceSegment)；其statusChanged设置serviceStub及status

ConsumerThread

skywalking-6.6.0/apm-commons/apm-datacarrier/src/main/java/org/apache/skywalking/apm/commons/datacarrier/consumer/ConsumerThread.java

```
public class ConsumerThread<T> extends Thread {
    private volatile boolean running;
    private IConsumer<T> consumer;
    private List<DataSource> dataSources;
    private long consumeCycle;

    ConsumerThread(String threadName, IConsumer<T> consumer, long consumeCycle) {
        super(threadName);
        this.consumer = consumer;
        running = false;
        dataSources = new ArrayList<DataSource>(1);
        this.consumeCycle = consumeCycle;
    }

    /**
     * add whole buffer to consume
     *
     * @param sourceBuffer
     */
    void addDataSource(QueueBuffer<T> sourceBuffer) {
        this.dataSources.add(new DataSource(sourceBuffer));
    }
```

```
    @Override
    public void run() {
        running = true;

        final List<T> consumeList = new ArrayList<T>(1500);
        while (running) {
            if (!consume(consumeList)) {
                try {
                    Thread.sleep(consumeCycle);
                } catch (InterruptedException e) {
                }
            }
        }

        // consumer thread is going to stop
        // consume the last time
        consume(consumeList);

        consumer.onExit();
    }

    private boolean consume(List<T> consumeList) {
        for (DataSource dataSource : dataSources) {
            dataSource.obtain(consumeList);
        }

        if (!consumeList.isEmpty()) {
            try {
                consumer.consume(consumeList);
            } catch (Throwable t) {
                consumer.onError(consumeList, t);
            } finally {
                consumeList.clear();
            }
            return true;
        }
        return false;
    }

    void shutdown() {
        running = false;
    }

    /**
     * DataSource is a refer to {@link Buffer}.
     */
    class DataSource {
        private QueueBuffer<T> sourceBuffer;

        DataSource(QueueBuffer<T> sourceBuffer) {
            this.sourceBuffer = sourceBuffer;
        }

        void obtain(List<T> consumeList) {
            sourceBuffer.obtain(consumeList);
        }
    }
}
```

- ConsumerThread继承了Thread，其run方法会循环执行consume(consumeList)，跳出循环时会再次执行consume(consumeList)，最后执行consumer.onExit()；consume方法会遍历dataSources，执行其dataSource.obtain(consumeList)，然后在consumeList不为空的时候执行consumer.consume(consumeList)方法

ConsumeDriver

skywalking-6.6.0/apm-commons/apm-datacarrier/src/main/java/org/apache/skywalking/apm/commons/datacarrier/consumer/ConsumeDriver.java

```
public class ConsumeDriver<T> implements IDriver {
    private boolean running;
    private ConsumerThread[] consumerThreads;
    private Channels<T> channels;
    private ReentrantLock lock;

    public ConsumeDriver(String name, Channels<T> channels, Class<? extends ICons
        long consumeCycle) {
        this(channels, num);
        for (int i = 0; i < num; i++) {
            consumerThreads[i] = new ConsumerThread("DataCarrier." + name + ".Con
            consumerThreads[i].setDaemon(true);
        }
    }

    public ConsumeDriver(String name, Channels<T> channels, IConsumer<T> prototyp
        this(channels, num);
        prototype.init();
        for (int i = 0; i < num; i++) {
            consumerThreads[i] = new ConsumerThread("DataCarrier." + name + ".Con
            consumerThreads[i].setDaemon(true);
        }

    }

    private ConsumeDriver(Channels<T> channels, int num) {
        running = false;
        this.channels = channels;
        consumerThreads = new ConsumerThread[num];
        lock = new ReentrantLock();
    }

    private IConsumer<T> getNewConsumerInstance(Class<? extends IConsumer<T>> con
        try {
            IConsumer<T> inst = consumerClass.newInstance();
            inst.init();
            return inst;
        } catch (InstantiationException e) {
            throw new ConsumerCannotBeCreatedException(e);
        } catch (IllegalAccessException e) {
            throw new ConsumerCannotBeCreatedException(e);
        }
    }

    @Override
    public void begin(Channels channels) {
        if (running) {
            return;
        }
        try {
            lock.lock();
            this.allocateBuffer2Thread();
            for (ConsumerThread consumerThread : consumerThreads) {
                consumerThread.start();
            }
            running = true;
        } finally {
            lock.unlock();
        }
    }

    @Override
    public boolean isRunning(Channels channels) {
        return running;
    }

    private void allocateBuffer2Thread() {
        int channelSize = this.channels.getChannelSize();
        /**
```

```
         * if consumerThreads.length < channelSize
         * each consumer will process several channels.
         *
         * if consumerThreads.length == channelSize
         * each consumer will process one channel.
         *
         * if consumerThreads.length > channelSize
         * there will be some threads do nothing.
         */
        for (int channelIndex = 0; channelIndex < channelSize; channelIndex++) {
            int consumerIndex = channelIndex % consumerThreads.length;
            consumerThreads[consumerIndex].addDataSource(channels.getBuffer(chann
        }

    }

    @Override
    public void close(Channels channels) {
        try {
            lock.lock();
            this.running = false;
            for (ConsumerThread consumerThread : consumerThreads) {
                consumerThread.shutdown();
            }
        } finally {
            lock.unlock();
        }
    }
}
```

- ConsumeDriver实现了IDriver接口，其ConsumeDriver会创建num个ConsumerThread；其begin方法会执行allocateBuffer2Thread，给每个consumerThread添加dataSource，然后执行consumerThread.start()；其close方法会执行consumerThread.shutdown()

小结

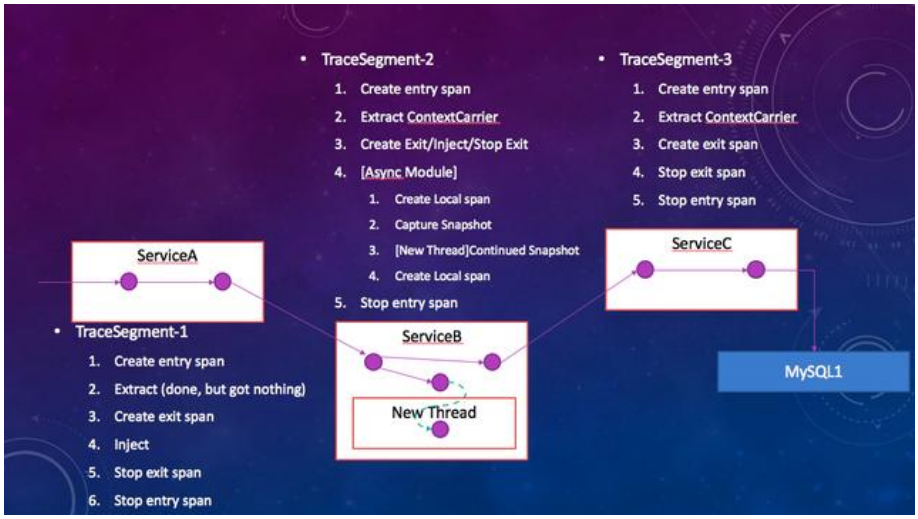
TraceSegmentServiceClient实现了BootService、IConsumer、TracingContextListener、GRPCChannelListener接口；其prepare方法往GRPCChannelManager注册自身的channelListener；其boot方法设置lastLogTime，实例化DataCarrier，并设置其consumer为自身；其onComplete方法执行TracingContext.ListenerManager.add(this)；其shutdown方法执行TracingContext.ListenerManager.remove(this)以及carrier.shutdownConsumers()；其consume方法在status为CONNECTED的时候执行upstreamSegmentStreamObserver.onNext(upstreamSegment)、upstreamSegmentStreamObserver.onCompleted()以及status.wait4Finish()；其afterFinished方法执行carrier.produce(traceSegment)；其statusChanged设置serviceStub及status

doc

- TraceSegmentServiceClient本文参考原文-http://bjbsair.com/2020-03-22/tech-info/5102/
  序

-

本文主要研究一下skywalking的TraceSegmentServiceClient



TracingContextListener

skywalking-6.6.0/apm-sniffer/apm-agent-core/src/main/java/org/apache/skywalking/apm/agent/core/context/TracingContextListener.java

```
public interface TracingContextListener {
    void afterFinished(TraceSegment traceSegment);
}
```

- TracingContextListener定义了afterFinished方法，其参数为TraceSegment

TraceSegment

skywalking-6.6.0/apm-sniffer/apm-agent-core/src/main/java/org/apache/skywalking/apm/agent/core/context/trace/TraceSegment.java

```
public class TraceSegment {

    private ID traceSegmentId;

    private List<TraceSegmentRef> refs;

    private List<AbstractTracingSpan> spans;

    private DistributedTraceIds relatedGlobalTraces;

    private boolean ignore = false;

    private boolean isSizeLimited = false;

    private final long createTime;

    public TraceSegment() {
        this.traceSegmentId = GlobalIdGenerator.generate();
        this.spans = new LinkedList<AbstractTracingSpan>();
        this.relatedGlobalTraces = new DistributedTraceIds();
        this.relatedGlobalTraces.append(new NewDistributedTraceId());
        this.createTime = System.currentTimeMillis();
    }

    public void ref(TraceSegmentRef refSegment) {
        if (refs == null) {
            refs = new LinkedList<TraceSegmentRef>();
        }
        if (!refs.contains(refSegment)) {
            refs.add(refSegment);
        }
    }

    public void relatedGlobalTraces(DistributedTraceId distributedTraceId) {
        relatedGlobalTraces.append(distributedTraceId);
    }
```

```
    public void archive(AbstractTracingSpan finishedSpan) {
        spans.add(finishedSpan);
    }

    public TraceSegment finish(boolean isSizeLimited) {
        this.isSizeLimited = isSizeLimited;
        return this;
    }

    public ID getTraceSegmentId() {
        return traceSegmentId;
    }

    public int getServiceId() {
        return RemoteDownstreamConfig.Agent.SERVICE_ID;
    }

    public boolean hasRef() {
        return !(refs == null || refs.size() == 0);
    }

    public List<TraceSegmentRef> getRefs() {
        return refs;
    }

    public List<DistributedTraceId> getRelatedGlobalTraces() {
        return relatedGlobalTraces.getRelatedGlobalTraces();
    }

    public boolean isSingleSpanSegment() {
        return this.spans != null && this.spans.size() == 1;
    }

    public boolean isIgnore() {
        return ignore;
    }

    public void setIgnore(boolean ignore) {
        this.ignore = ignore;
    }

    public UpstreamSegment transform() {
        UpstreamSegment.Builder upstreamBuilder = UpstreamSegment.newBuilder();
        for (DistributedTraceId distributedTraceId : getRelatedGlobalTraces()) {
            upstreamBuilder = upstreamBuilder.addGlobalTraceIds(distributedTraceI
        }
        SegmentObject.Builder traceSegmentBuilder = SegmentObject.newBuilder();
        /**
         * Trace Segment
         */
        traceSegmentBuilder.setTraceSegmentId(this.traceSegmentId.transform());
        // Don't serialize TraceSegmentReference

        // SpanObject
        for (AbstractTracingSpan span : this.spans) {
            traceSegmentBuilder.addSpans(span.transform());
        }
        traceSegmentBuilder.setServiceId(RemoteDownstreamConfig.Agent.SERVICE_ID)
        traceSegmentBuilder.setServiceInstanceId(RemoteDownstreamConfig.Agent.SER
        traceSegmentBuilder.setIsSizeLimited(this.isSizeLimited);

        upstreamBuilder.setSegment(traceSegmentBuilder.build().toByteString());
        return upstreamBuilder.build();
    }

    @Override
    public String toString() {
        return "TraceSegment{" +
            "traceSegmentId='" + traceSegmentId + '\'' +
            ", refs=" + refs +
```

```
            ", spans=" + spans +
            ", relatedGlobalTraces=" + relatedGlobalTraces +
            '}';
    }

    public int getApplicationInstanceId() {
        return RemoteDownstreamConfig.Agent.SERVICE_INSTANCE_ID;
    }

    public long createTime() {
        return this.createTime;
    }
}
```

- TraceSegment定义了traceSegmentId、refs、spans、relatedGlobalTraces等属性；它提供了ref、relatedGlobalTraces、archive 、finish、transform等方法

IConsumer

skywalking-6.6.0/apm-commons/apm-datacarrier/src/main/java/org/apache/skywalking/apm/commons/datacarrier/consumer/IConsumer.java

```
public interface IConsumer<T> {
    void init();

    void consume(List<T> data);

    void onError(List<T> data, Throwable t);

    void onExit();
}
```

- IConsumer定义了init、consume、onError、onExit方法

TraceSegmentServiceClient

skywalking-6.6.0/apm-sniffer/apm-agent-core/src/main/java/org/apache/skywalking/apm/agent/core/remote/TraceSegmentServiceClient.java

```
@DefaultImplementor
public class TraceSegmentServiceClient implements BootService, IConsumer<TraceSeg
    private static final ILog logger = LogManager.getLogger(TraceSegmentServiceCl
    private static final int TIMEOUT = 30 * 1000;

    private long lastLogTime;
    private long segmentUplinkedCounter;
    private long segmentAbandonedCounter;
    private volatile DataCarrier<TraceSegment> carrier;
    private volatile TraceSegmentReportServiceGrpc.TraceSegmentReportServiceStub
    private volatile GRPCChannelStatus status = GRPCChannelStatus.DISCONNECT;

    @Override
    public void prepare() throws Throwable {
        ServiceManager.INSTANCE.findService(GRPCChannelManager.class).addChannelL
    }

    @Override
    public void boot() throws Throwable {
        lastLogTime = System.currentTimeMillis();
        segmentUplinkedCounter = 0;
        segmentAbandonedCounter = 0;
        carrier = new DataCarrier<TraceSegment>(CHANNEL_SIZE, BUFFER_SIZE);
        carrier.setBufferStrategy(BufferStrategy.IF_POSSIBLE);
        carrier.consume(this, 1);
    }

    @Override
    public void onComplete() throws Throwable {
        TracingContext.ListenerManager.add(this);
    }
```

```
    @Override
    public void shutdown() throws Throwable {
        TracingContext.ListenerManager.remove(this);
        carrier.shutdownConsumers();
    }

    @Override
    public void init() {

    }

    @Override
    public void consume(List<TraceSegment> data) {
        if (CONNECTED.equals(status)) {
            final GRPCStreamServiceStatus status = new GRPCStreamServiceStatus(fa
            StreamObserver<UpstreamSegment> upstreamSegmentStreamObserver = servi
                @Override
                public void onNext(Commands commands) {
                    ServiceManager.INSTANCE.findService(CommandService.class).rec
                }

                @Override
                public void onError(Throwable throwable) {
                    status.finished();
                    if (logger.isErrorEnable()) {
                        logger.error(throwable, "Send UpstreamSegment to collecto
                    }
                    ServiceManager.INSTANCE.findService(GRPCChannelManager.class)
                }

                @Override
                public void onCompleted() {
                    status.finished();
                }
            });

            try {
                for (TraceSegment segment : data) {
                    UpstreamSegment upstreamSegment = segment.transform();
                    upstreamSegmentStreamObserver.onNext(upstreamSegment);
                }
            } catch (Throwable t) {
                logger.error(t, "Transform and send UpstreamSegment to collector
            }

            upstreamSegmentStreamObserver.onCompleted();

            status.wait4Finish();
            segmentUplinkedCounter += data.size();
        } else {
            segmentAbandonedCounter += data.size();
        }

        printUplinkStatus();
    }

    private void printUplinkStatus() {
        long currentTimeMillis = System.currentTimeMillis();
        if (currentTimeMillis - lastLogTime > 30 * 1000) {
            lastLogTime = currentTimeMillis;
            if (segmentUplinkedCounter > 0) {
                logger.debug("{} trace segments have been sent to collector.", se
                segmentUplinkedCounter = 0;
            }
            if (segmentAbandonedCounter > 0) {
                logger.debug("{} trace segments have been abandoned, cause by no
                segmentAbandonedCounter = 0;
            }
        }
    }
```

```
    @Override
    public void onError(List<TraceSegment> data, Throwable t) {
        logger.error(t, "Try to send {} trace segments to collector, with unexpec
    }

    @Override
    public void onExit() {

    }

    @Override
    public void afterFinished(TraceSegment traceSegment) {
        if (traceSegment.isIgnore()) {
            return;
        }
        if (!carrier.produce(traceSegment)) {
            if (logger.isDebugEnable()) {
                logger.debug("One trace segment has been abandoned, cause by buff
            }
        }
    }

    @Override
    public void statusChanged(GRPCChannelStatus status) {
        if (CONNECTED.equals(status)) {
            Channel channel = ServiceManager.INSTANCE.findService(GRPCChannelMana
            serviceStub = TraceSegmentReportServiceGrpc.newStub(channel);
        }
        this.status = status;
    }
}
```

- TraceSegmentServiceClient实现了BootService、IConsumer、TracingContextListener、GRPCChannelListener接口；其prepare方法往GRPCChannelManager注册自身的channelListener；其boot方法设置lastLogTime，实例化DataCarrier，并设置其consumer为自身；其onComplete方法执行TracingContext.ListenerManager.add(this)；其shutdown方法执行TracingContext.ListenerManager.remove(this)以及carrier.shutdownConsumers()；其consume方法在status为CONNECTED的时候执行upstreamSegmentStreamObserver.onNext(upstreamSegment)、upstreamSegmentStreamObserver.onCompleted()以及status.wait4Finish()；其afterFinished方法执行carrier.produce(traceSegment)；其statusChanged设置serviceStub及status

ConsumerThread

skywalking-6.6.0/apm-commons/apm-datacarrier/src/main/java/org/apache/skywalking/apm/commons/datacarrier/consumer/ConsumerThread.java

```
public class ConsumerThread<T> extends Thread {
    private volatile boolean running;
    private IConsumer<T> consumer;
    private List<DataSource> dataSources;
    private long consumeCycle;

    ConsumerThread(String threadName, IConsumer<T> consumer, long consumeCycle) {
        super(threadName);
        this.consumer = consumer;
        running = false;
        dataSources = new ArrayList<DataSource>(1);
        this.consumeCycle = consumeCycle;
    }

    /**
     * add whole buffer to consume
     *
     * @param sourceBuffer
     */
    void addDataSource(QueueBuffer<T> sourceBuffer) {
        this.dataSources.add(new DataSource(sourceBuffer));
    }
```

```
    @Override
    public void run() {
        running = true;

        final List<T> consumeList = new ArrayList<T>(1500);
        while (running) {
            if (!consume(consumeList)) {
                try {
                    Thread.sleep(consumeCycle);
                } catch (InterruptedException e) {
                }
            }
        }

        // consumer thread is going to stop
        // consume the last time
        consume(consumeList);

        consumer.onExit();
    }

    private boolean consume(List<T> consumeList) {
        for (DataSource dataSource : dataSources) {
            dataSource.obtain(consumeList);
        }

        if (!consumeList.isEmpty()) {
            try {
                consumer.consume(consumeList);
            } catch (Throwable t) {
                consumer.onError(consumeList, t);
            } finally {
                consumeList.clear();
            }
            return true;
        }
        return false;
    }

    void shutdown() {
        running = false;
    }

    /**
     * DataSource is a refer to {@link Buffer}.
     */
    class DataSource {
        private QueueBuffer<T> sourceBuffer;

        DataSource(QueueBuffer<T> sourceBuffer) {
            this.sourceBuffer = sourceBuffer;
        }

        void obtain(List<T> consumeList) {
            sourceBuffer.obtain(consumeList);
        }
    }
}
```

- ConsumerThread继承了Thread，其run方法会循环执行consume(consumeList)，跳出循环时会再次执行consume(consumeList)，最后执行consumer.onExit()；consume方法会遍历dataSources，执行其dataSource.obtain(consumeList)，然后在consumeList不为空的时候执行consumer.consume(consumeList)方法

ConsumeDriver

skywalking-6.6.0/apm-commons/apm-datacarrier/src/main/java/org/apache/skywalking/apm/commons/datacarrier/consumer/ConsumeDriver.java

```
public class ConsumeDriver<T> implements IDriver {
    private boolean running;
    private ConsumerThread[] consumerThreads;
    private Channels<T> channels;
    private ReentrantLock lock;

    public ConsumeDriver(String name, Channels<T> channels, Class<? extends ICons
        long consumeCycle) {
        this(channels, num);
        for (int i = 0; i < num; i++) {
            consumerThreads[i] = new ConsumerThread("DataCarrier." + name + ".Con
            consumerThreads[i].setDaemon(true);
        }
    }

    public ConsumeDriver(String name, Channels<T> channels, IConsumer<T> prototyp
        this(channels, num);
        prototype.init();
        for (int i = 0; i < num; i++) {
            consumerThreads[i] = new ConsumerThread("DataCarrier." + name + ".Con
            consumerThreads[i].setDaemon(true);
        }

    }

    private ConsumeDriver(Channels<T> channels, int num) {
        running = false;
        this.channels = channels;
        consumerThreads = new ConsumerThread[num];
        lock = new ReentrantLock();
    }

    private IConsumer<T> getNewConsumerInstance(Class<? extends IConsumer<T>> con
        try {
            IConsumer<T> inst = consumerClass.newInstance();
            inst.init();
            return inst;
        } catch (InstantiationException e) {
            throw new ConsumerCannotBeCreatedException(e);
        } catch (IllegalAccessException e) {
            throw new ConsumerCannotBeCreatedException(e);
        }
    }

    @Override
    public void begin(Channels channels) {
        if (running) {
            return;
        }
        try {
            lock.lock();
            this.allocateBuffer2Thread();
            for (ConsumerThread consumerThread : consumerThreads) {
                consumerThread.start();
            }
            running = true;
        } finally {
            lock.unlock();
        }
    }

    @Override
    public boolean isRunning(Channels channels) {
        return running;
    }

    private void allocateBuffer2Thread() {
        int channelSize = this.channels.getChannelSize();
        /**
```

```
         * if consumerThreads.length < channelSize
         * each consumer will process several channels.
         *
         * if consumerThreads.length == channelSize
         * each consumer will process one channel.
         *
         * if consumerThreads.length > channelSize
         * there will be some threads do nothing.
         */
        for (int channelIndex = 0; channelIndex < channelSize; channelIndex++) {
            int consumerIndex = channelIndex % consumerThreads.length;
            consumerThreads[consumerIndex].addDataSource(channels.getBuffer(chann
        }

    }

    @Override
    public void close(Channels channels) {
        try {
            lock.lock();
            this.running = false;
            for (ConsumerThread consumerThread : consumerThreads) {
                consumerThread.shutdown();
            }
        } finally {
            lock.unlock();
        }
    }
}
```

- ConsumeDriver实现了IDriver接口，其ConsumeDriver会创建num个ConsumerThread；其begin方法会执行allocateBuffer2Thread，给每个consumerThread添加dataSource，然后执行consumerThread.start()；其close方法会执行consumerThread.shutdown()

小结

TraceSegmentServiceClient实现了BootService、IConsumer、TracingContextListener、GRPCChannelListener接口；其prepare方法往GRPCChannelManager注册自身的channelListener；其boot方法设置lastLogTime，实例化DataCarrier，并设置其consumer为自身；其onComplete方法执行TracingContext.ListenerManager.add(this)；其shutdown方法执行TracingContext.ListenerManager.remove(this)以及carrier.shutdownConsumers()；其consume方法在status为CONNECTED的时候执行upstreamSegmentStreamObserver.onNext(upstreamSegment)、upstreamSegmentStreamObserver.onCompleted()以及status.wait4Finish()；其afterFinished方法执行carrier.produce(traceSegment)；其statusChanged设置serviceStub及status

doc

- TraceSegmentServiceClient

« 上一篇： 帮你搞懂Python进程,线程与协程
» 下一篇： 浅谈skywalking的spring-webflux-plugin

posted @ 2020-03-23 23:14 李翰林 阅读(593) 评论(0) 编辑 收藏 举报

编辑推荐:
· 记一次 .NET 某机械臂智能机器人控制系统 MRS CPU 爆高分析
· 技术选型的一点个人思考
· 带团队后的日常思考（四）
· 温故知新：WeakReference了解一下?
· .Net 中异步任务的取消和监控

最新新闻:
· 小米汽车薪酬翻倍挖人，业内：谁不心动？（2021-09-09 21:30）
· 30岁二刷博士，17个月发6篇一作获顶会最佳！现实版人生重开模拟器（2021-09-09 21:20）
· 亲测吃知乎月饼变身“喷射战士”，我给大家科普一下发生甚么事了（2021-09-09 21:00）
· 亚马逊无人收银技术扩围：将于明年登陆旗下全食超市（2021-09-09 19:04）
· 百度投资成立智能科技公司 经营范围含智能机器人的研发等（2021-09-09 17:55）
» 更多新闻...