Nifi

Luồng chạy của flowfile trên code sẽ như nào ? Từ processor đến flowfile ?

I. Tài liệu tham khảo :

https://nifi.apache.org/docs/nifi-docs/html/developer-guide.html

II. Một số class, funtionc, interface cần biết

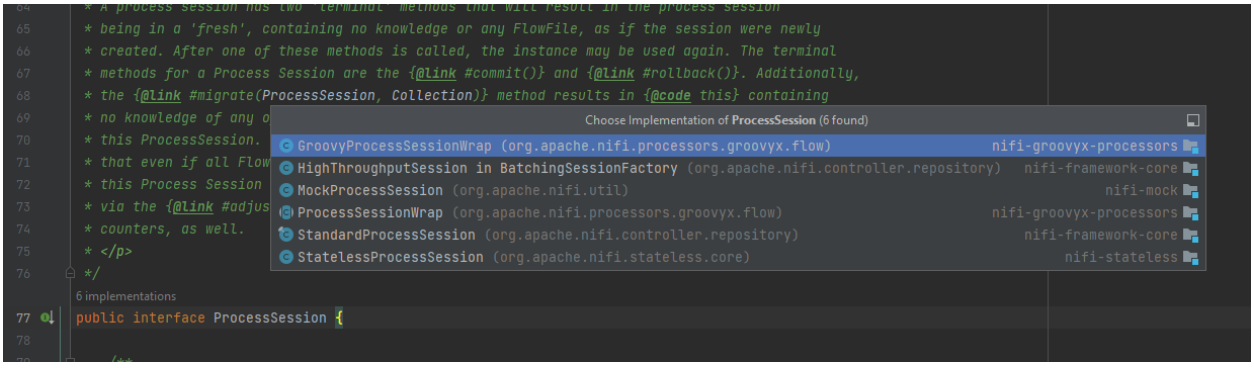
`nifi-framework-core`

filepath: .\nifi-1.11.4\nifi-nar-bundles\nifi-framework-bundle\nifi-framework\nifi-framework-core

Bộ khung core của Nifi, là folder chứa code quản lý flowfile, loadbalance, Scheduler và thực hiện các action, thêm, sửa, xóa processor trên nifi

`interface ProcessSession`

filepath: .\nifi-1.11.4\nifi-api\src\main\java\org\apache\nifi\processor\ProcessSession.java



ProcessSession, thường được gọi đơn giản là một "session", cung cấp một cơ chế để có thể created, get , destroyed, examined, cloned, and transferred FlowFiles sang các

Processor khác. Ngoài ra, ProcessSession cung cấp cơ chế tạo các phiên bản sửa đổi của FlowFiles, bằng cách thêm hoặc xóa các thuộc tính hoặc bằng cách sửa đổi nội dung của FlowFile. ProcessSession cũng đưa ra cơ chế phát ra các Sự kiện chứng minh nhằm cung cấp khả năng theo dõi dòng dõi và lịch sử của FlowFile. Sau khi các thao tác được thực hiện trên một hoặc nhiều FlowFiles, một ProcessSession có thể được cam kết hoặc được khôi phục.

1 số hàm quan trọng sử dụng nhiều của ProcessSession

Khởi tạo 1 ProcessSession trong onTrigger :

```
@Override
public void onTrigger(final ProcessContext context, final ProcessSession session) throws ProcessException {
```

- `session.create()` : Khởi tạo mới 1 flowfile, thường được sử dụng trong các processor khởi đầu của 1 luồng, để khởi tạo flowfile mới

```
for (T entity : entities) {
    // Create the FlowFile for this path.
    final Map<String, String> attributes = createAttributes(entity, context);
    FlowFile flowFile = session.create();
```

- `session.putAllAttributes()`: Hàm này sử dụng để khởi tạo attribute mới cho flowfile. Khởi tạo 1 biến attributes kiểu dữ liệu Map theo cặp key, value để lưu thông tin attribute sau đó dùng `putAllAttributes()` để add các attribute vào flowfile

```
for (T entity : entities) {
    // Create the FlowFile for this path.
    final Map<String, String> attributes = createAttributes(entity, context);
    FlowFile flowFile = session.create();
    flowFile = session.putAllAttributes(flowFile, attributes);
```

- `session.transfer()`: Hàm này sử dụng để chuyển các flowfile sau khi sử lý xong đến các relationship

```java
for (T entity : entities) {
    // Create the FlowFile for this path.
    final Map<String, String> attributes = createAttributes(entity, context);
    FlowFile flowFile = session.create();
    flowFile = session.putAllAttributes(flowFile, attributes);
    session.transfer(flowFile, REL_SUCCESS);
    flowfilesCreated++;
```

- `session.get()` : Lấy thông tin từ các flowfile phía trước, các thông tin như flowfile date create, flowfile attribute… Thường được sử dụng ở các flowfile **InputRequirement**

```java
public void onTrigger(final ProcessContext context, final ProcessSession session) throws ProcessException {
    FlowFile flowFile = session.get();
    if (flowFile == null) {
        return;
    }
```

- `session.commit()`: commit session, thường được sử dụng sau mỗi phép biến đổi của flowfile. lưu lại thông tin, attribute, content của flowfile. Nếu Processor của bạn kế thừa `AbstractProcessor` hoặc sử dụng interface `ProcessSession` bạn có thể bỏ qua sử dụng `session.commit()` chúng sẽ tự động gọi hàm commit or rollback sau mỗi transaction.

  Ngược lại nếu bạn sử dụng `sessionFactory.createSession()` để tạo session thì bạn phải dùng `session.commit()`

```java
        // transfer
        session.transfer(flowFile, REL_FAILURE)
        session.commit()
```

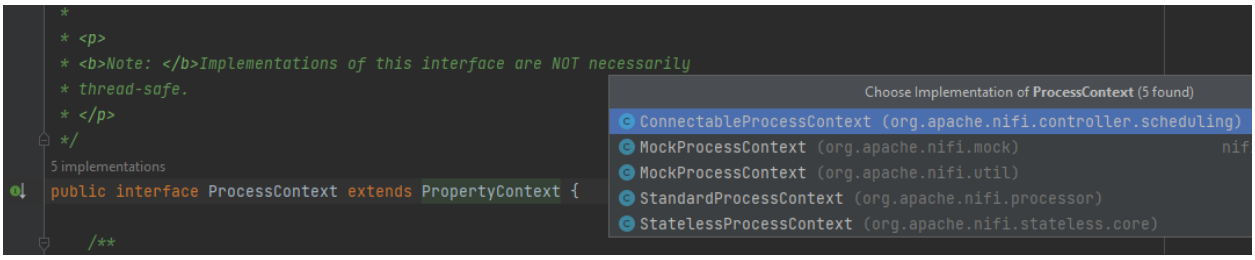
- `session.rollback()`: Thường được sử dụng để cath exception của flowfile lỗi rollback flowfile ở vị trí commit

```java
private void handleException(final HttpServletRequest request, final HttpServletResponse response,
                             final ProcessSession session, String foundSubject, final Throwable t) throws IOException {
    session.rollback();
    logger.error( msg: "Unable to receive file from Remote Host: [{}] SubjectDN [{}] due to {}", new Object[]{request.getRemoteHost(), foundSubject, t});
    response.sendError(HttpServletResponse.SC_INTERNAL_SERVER_ERROR, t.toString());
}
```

- `session.penalize()`: báo log lỗi , show ra FlowFile đã gặp phải sự cố hoặc lỗi trong quá trình xử lý, khiến nó bị trì hoãn hoặc không thể transfer trong luồng dữ liệu.

```
        return;
    } catch (final ProcessException | IOException e) {
        closeConnection = true;
        getLogger().error( msg: "Failed to fetch content for {} from filename {} on remote host {}:{} due to {}; routing to comms.failure",
                new Object[]{flowFile, filename, host, port, e.toString()}, e);
        session.transfer(session.penalize(flowFile), REL_COMMS_FAILURE);
        return;
    }
```

`interface ProcessContext`



`ProcessContext` : là cầu nối giữ Processor và framework. Nó cung cấp thông tin về cách Processor hiện được cấu hình và cho phép Processor thực hiện các Framework-specific tasks.

- `context.getProperty()` : Lấy thông tin `PropertyValue` của Processor, example: process ListFTP có property Listing Strategy, khi gọi hàm này sẽ lấy ra `PropertyValue`

```
final String listingStrategy = context.getProperty(LISTING_STRATEGY).getValue();
if (BY_TIMESTAMPS.equals(listingStrategy)) {
```

- `context.getMaxConcurrentTasks()`: Trả về thông tin config concurrent task trong Processor

```
config.setMaxConnections(context.getMaxConcurrentTasks());
```

- `context.getControllerServiceLookup()`: Trả về thông tin Controller Service Processor đang sử dụng

```
serviceLookup = context.getControllerServiceLookup();
```

Component lifecycle

NiFi API cung cấp quản lý lifecycle của Processor, ControllerServices.. theo Java Annotations.

`@OnScheduled` : là 1 chú thích được sử dụng để đánh dấu một phương thức trong một Processor.

phương thức được đánh dấu bằng chú thích này sẽ được khởi chạy trước khi chạy hàm OnTrigger, thường được dùng cho việc chuẩn bị trước khi chạy processor. Chỉ chạy hàm Ontrigger khi chạy hàm này thành công

Ví dụ như hàm dưới đây, được sử dụng để khởi tạo EntityTracker của Nifi ListFTP stragery

```java
@OnScheduled
public void initListedEntityTracker(ProcessContext context) {
    final boolean isTrackingEntityStrategy = BY_ENTITIES.getValue().equals(context.getProperty(LISTING_STRATEGY).getValue());
    if (listedEntityTracker != null && (resetEntityTrackingState || !isTrackingEntityStrategy)) {
        try {
            listedEntityTracker.clearListedEntities();
        } catch (IOException e) {
            throw new RuntimeException("Failed to reset previously listed entities due to " + e, e);
        }
    }
    resetEntityTrackingState = false;

    if (isTrackingEntityStrategy) {
        if (listedEntityTracker == null) {
            listedEntityTracker = createListedEntityTracker();
        }
    } else {
        listedEntityTracker = null;
    }
}
```

scheduledState

là trạng thái lập lịch của processor:

`DISABLED` *: Không thể lập lịch processor để chạy*

`STOPPED` *: Có thể được lập lịch để chạy nhưng hiện tại thì không*

`RUNNING` *: Đã được lập lịch để chạy*

`STARTING` *: Prepare for running*

`STOPPING` : *đang chuyển trạng thái từ running sang stop*

III. Luồng chạy flowfile

Tình huống giả định 1 luồng get file processor ListFTP đến PUTHDFS diễn ra như nào trong code ?

`ListFTP.java`

filepath: .\nifi-1.11.4\nifi-nar-bundles\nifi-standard-bundle\nifi-standard-processors\src\main\java\org\apache\nifi\processors\standard\ListFTP.java

Trong này class ListFTP `ListFTP` được kế thừa từ class `ListFileTransfer`

class `ListFTP` mục đích khởi tạo Processor và khai báo các property sử dụng để list File trên FTP folder.

```
public class ListFTP extends ListFileTransfer {

    @Override
    protected List<PropertyDescriptor> getSupportedPropertyDescriptors() {
        final PropertyDescriptor port = new PropertyDescriptor.Builder().fromPropertyDescriptor(UNDEFAULTED_PORT).defaultValue("21").build();

        final List<PropertyDescriptor> properties = new ArrayList<>();
        properties.add(LISTING_STRATEGY);
        properties.add(HOSTNAME);
        properties.add(port);
        properties.add(USERNAME);
        properties.add(FTPTransfer.PASSWORD);
        properties.add(REMOTE_PATH);
        properties.add(DISTRIBUTED_CACHE_SERVICE);
```

`ListFileTransfer.java`

filepath: .\nifi-1.11.4\nifi-nar-bundles\nifi-standard-bundle\nifi-standard-processors\src\main\java\org\apache\nifi\processors\standard\ListFileTransfer.java

`ListFileTransfer` Mục đích chính của class này là thực hiện listfile trong folder FTP và tạo attribute được lấy từ flowfile qua hàm `createAttributes`

Class này được kế thừa từ `AbstractListProcessor`

```
2 usages  2 inheritors
public abstract class ListFileTransfer extends AbstractListProcessor<FileInfo> {
    public static final PropertyDescriptor HOSTNAME = new PropertyDescriptor.Builder()
        .name("Hostname")
```

`createAttributes` :

tạo attributes cho flowfile

```java
@Override
protected Map<String, String> createAttributes(final FileInfo fileInfo, final ProcessContext context) {
    final Map<String, String> attributes = new HashMap<>();
    final DateFormat formatter = new SimpleDateFormat(ListFile.FILE_MODIFY_DATE_ATTR_FORMAT, Locale.US);
    attributes.put(getProtocolName() + ".remote.host", context.getProperty(HOSTNAME).evaluateAttributeExpressions().getValue());
    attributes.put(getProtocolName() + ".remote.port", context.getProperty(UNDEFAULTED_PORT).evaluateAttributeExpressions().getValue());
    attributes.put(getProtocolName() + ".listing.user", context.getProperty(USERNAME).evaluateAttributeExpressions().getValue());
    attributes.put(ListFile.FILE_LAST_MODIFY_TIME_ATTRIBUTE, formatter.format(new Date(fileInfo.getLastModifiedTime())));
    attributes.put(ListFile.FILE_PERMISSIONS_ATTRIBUTE, fileInfo.getPermissions());
    attributes.put(ListFile.FILE_OWNER_ATTRIBUTE, fileInfo.getOwner());
    attributes.put(ListFile.FILE_GROUP_ATTRIBUTE, fileInfo.getGroup());
    attributes.put(ListFile.FILE_SIZE_ATTRIBUTE, Long.toString(fileInfo.getSize()));
    attributes.put(CoreAttributes.FILENAME.key(), fileInfo.getFileName());
    final String fullPath = fileInfo.getFullPathFileName();
    if (fullPath != null) {
        final int index = fullPath.lastIndexOf( str: "/");
        if (index > -1) {
            final String path = fullPath.substring(0, index);
            attributes.put(CoreAttributes.PATH.key(), path);
        }
```

`List File`:

thực hiện list file trên FTP folder

```java
4 usages  1 override
@Override
protected List<FileInfo> performListing(final ProcessContext context, final Long minTimestamp) throws IOException {
    final FileTransfer transfer = getFileTransfer(context);
    final List<FileInfo> listing;
    try {
        listing = transfer.getListing();
    } finally {
        IOUtils.closeQuietly(transfer);
    }

    if (minTimestamp == null) {
        return listing;
    }

    final Iterator<FileInfo> itr = listing.iterator();
    while (itr.hasNext()) {
        final FileInfo next = itr.next();
        if (next.getLastModifiedTime() < minTimestamp) {
            itr.remove();
        }
```

`AbstractListProcessor.java`

Class chính để thực hiện tạo flowfile của ListFTP.

Trong Class này chứa hàm OnTrigger(hàm khởi chạy chính của processor). Ở đây tùy vào `listingStrategy` được chọn. dùng `context.getProperty( LISTING_STRATEGY ).getValue()`

để lấy giá trị của `listingStrateg`. Sau đó sẽ ListFTP theo 2 phương thức `BY_TIMESTAMPS` hoặc `BY_ENTITIES` lấy từ giá trị trên

```java
@Override
public void onTrigger(final ProcessContext context, final ProcessSession session) throws ProcessException {

    final String listingStrategy = context.getProperty(LISTING_STRATEGY).getValue();
    if (BY_TIMESTAMPS.equals(listingStrategy)) {
        listByTrackingTimestamps(context, session);

    } else if (BY_ENTITIES.equals(listingStrategy)) {
        listByTrackingEntities(context, session);

    } else {
        throw new ProcessException("Unknown listing strategy: " + listingStrategy);
    }
}
```

Như bạn đã biết hàm OnTrigger khởi chạy processor khi có Trigger, Vật cái gì Trigger hàm onTrigger ?

Hãy tìm hiểu dưới dây

`StandardProcessScheduler.java` và `StandardProcessorNode.java`

`StandardProcessScheduler` Có trách nhiệm lên lịch cho processor

`StandardProcessScheduler` filepath: .\nifi-1.11.4\nifi-nar-bundles\nifi-framework-bundle\nifi-framework\nifi-framework-core\src\main\java\org\apache\nifi\controller

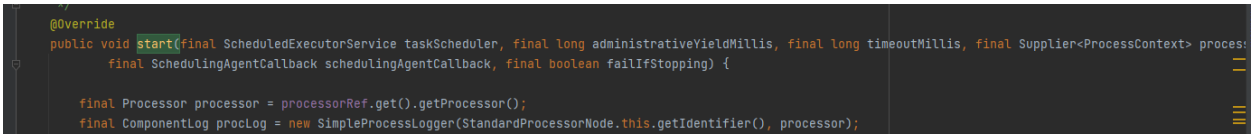
`StandardProcessorNode` filepath : .\nifi-1.11.4\nifi-nar-bundles\nifi-framework-bundle\nifi-framework\nifi-framework-core-api\src\main\java\org\apache\nifi\controller\ProcessorNode.java

---

`StandardProcessScheduler` `schedule()` :

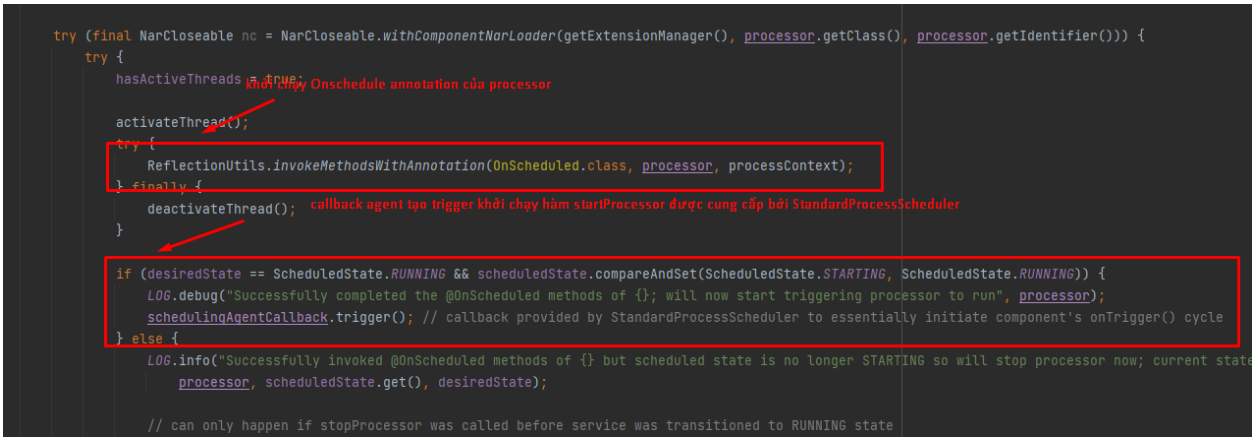
lập lịch cho processor

`StandardProcessorNode` `start()`: khi start processor `StandardProcessorNode` khởi chạy hàm start() có trách nhiệm chuyển đổi trạng thái lập lịch processor(ScheduleState) từ stopped sang starting



`StandardProcessorNode` `initiateStart()` : Hàm khởi chạy processor, chạy Onschedule Annotation của processor, sau khi chạy xong, trigger chạy Ontrigger funtionc của

processor



`StandardProcessScheduler` `startProcessor()` :

Khởi chạy hàm Ontrigger của processor

`FlowController.java`

Filepath: ./nifi-1.11.4\nifi-nar-bundles\nifi-framework-bundle\nifi-framework\nifi-framework-core\src\main\java\org\apache\nifi\controller\FlowController.java

Quản lý tất cả thông tin, lưu trữ của flowfile, schedule, loadbalance.

Dưới đây tạo FlowController theo mode của Nifi standalone hoặc cluster.

```java
public static FlowController createStandaloneInstance(
        final FlowFileEventRepository flowFileEventRepo,
        final NiFiProperties properties,
        final Authorizer authorizer,
        final AuditService auditService,
        final StringEncryptor encryptor,
        final BulletinRepository bulletinRepo,
        final VariableRegistry variableRegistry,
        final FlowRegistryClient flowRegistryClient,
        final ExtensionManager extensionManager) {

    return new FlowController(
            flowFileEventRepo,
            properties,
            authorizer,
            auditService,
            encryptor,
            /* configuredForClustering */ false,
            /* NodeProtocolSender */ null,
            bulletinRepo,
            /* cluster coordinator */ null,
            /* heartbeat monitor */ null,
            /* leader election manager */ null,
            /* variable registry */ variableRegistry,
            flowRegistryClient,
            extensionManager);
}
```

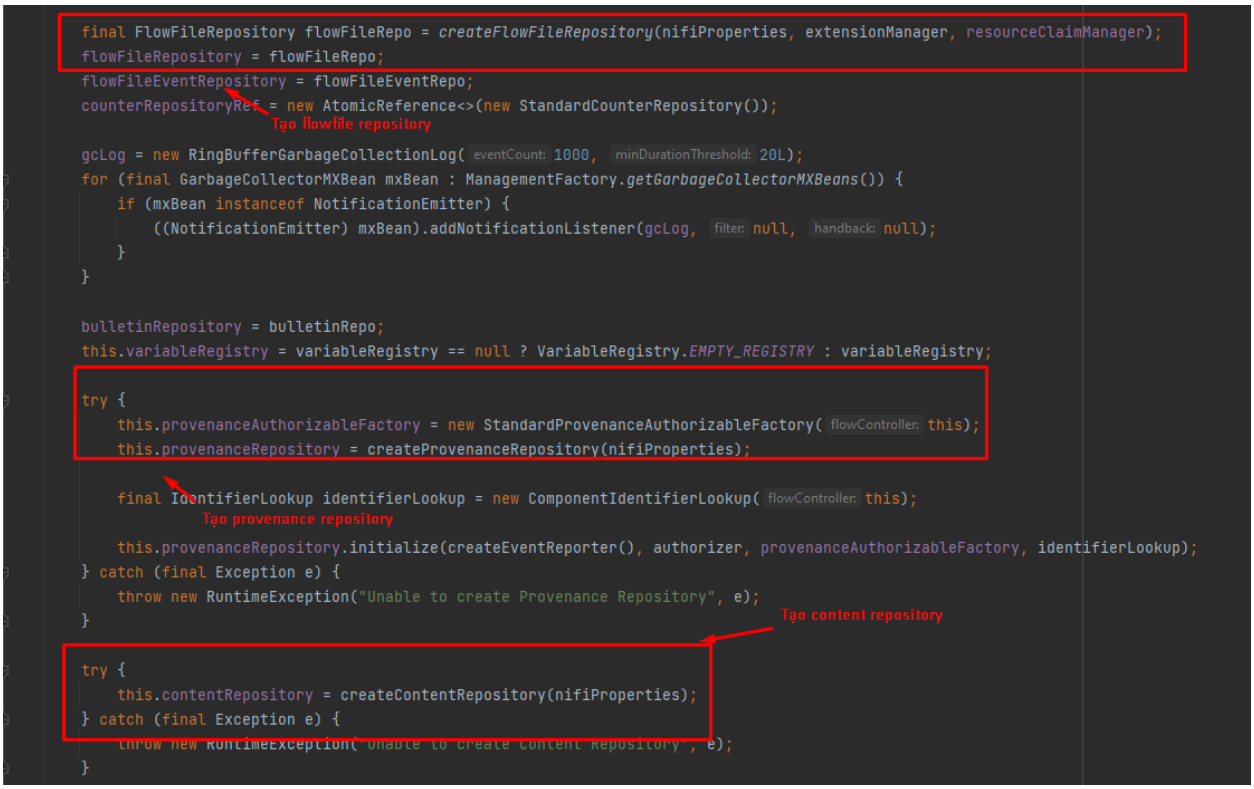
```java
public static FlowController createClusteredInstance(
        final FlowFileEventRepository flowFileEventRepo,
        final NiFiProperties properties,
        final Authorizer authorizer,
        final AuditService auditService,
        final StringEncryptor encryptor,
        final NodeProtocolSender protocolSender,
        final BulletinRepository bulletinRepo,
        final ClusterCoordinator clusterCoordinator,
        final HeartbeatMonitor heartbeatMonitor,
        final LeaderElectionManager leaderElectionManager,
        final VariableRegistry variableRegistry,
        final FlowRegistryClient flowRegistryClient,
        final ExtensionManager extensionManager) {

    final FlowController flowController = new FlowController(
            flowFileEventRepo,
            properties,
            authorizer,
            auditService,
            encryptor,
            /* configuredForClustering */ true,
            protocolSender,
            bulletinRepo,
            clusterCoordinator,
            heartbeatMonitor,
            leaderElectionManager,
            variableRegistry,
            flowRegistryClient,
            extensionManager);

    return flowController;
}
```

Ở đây flowcontroller sẽ tạo các `ProvenanceRepository` `ContentRepository` `flowFileEventRepository`

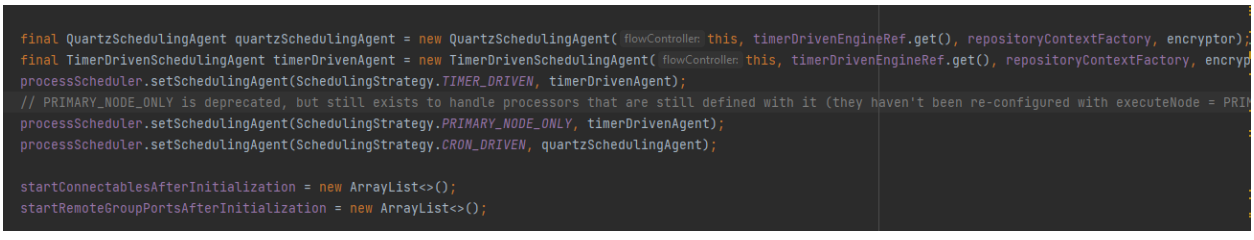
`ProvenanceRepository` là nơi lưu trữ tất cả dữ liệu sự kiện xuất xứ của flowfile. Khi khởi chạy nifi sẽ đc lưu ở folder ${NIFI_HOME}/provenance_repository/

`ContentRepository` là nơi lưu trữ dữ liệu trong flowfile. Khi khởi chạy nifi sẽ đc lưu ở folder ${NIFI_HOME}/content_repository/

`flowFileRepository` là nơi theo dõi trạng thái của các flowfile đang hoạt động trong luồng. Khi khởi chạy nifi sẽ đc lưu ở folder ${NIFI_HOME}/flowfile_repository/



Bên cạnh đó khởi tạo `processScheduler` mục đích lên lịch, set các Scheduling Agent.



`final QuartzSchedulingAgent quartzSchedulingAgent = new QuartzSchedulingAgent(this, timerDrivenEngineRef.get(), repositoryContextFactory, encryptor)` : khởi tạo cron driven cho

nifi processor

`final TimerDrivenSchedulingAgent timerDrivenAgent = new TimerDrivenSchedulingAgent(this, timerDrivenEngineRef.get(), repositoryContextFactory, encryptor, this.nifiProperties);` : khởi tạo timer driven cho nifi processor

Tạo kết nối đến các zookeeper server để chạy mode cluster, loadbalance

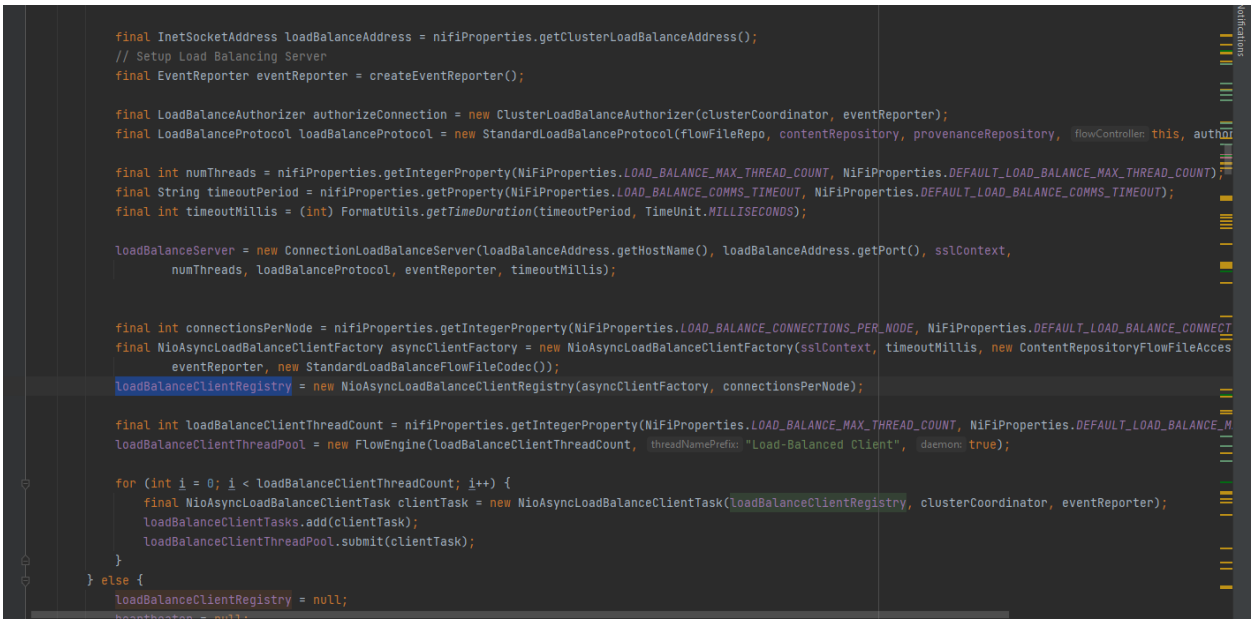
```
// Initialize the Embedded ZooKeeper server, if applicable
if (nifiProperties.isStartEmbeddedZooKeeper() && configuredForClustering) {
    try {
        zooKeeperStateServer = ZooKeeperStateServer.create(nifiProperties);
        zooKeeperStateServer.start();
    } catch (final IOException | ConfigException e) {
        throw new IllegalStateException("Unable to initailize Flow because NiFi was configured to start an Embedded Zookeeper server but failed to do so", e);
    }
} else {
    zooKeeperStateServer = null;
}
```

Nếu được `configuredForClustering` : config cho mod cluster:

Bầu chọn coordinator :

```
    if (configuredForClustering) {
        heartbeater = new ClusterProtocolHeartbeater(protocolSender, clusterCoordinator, leaderElectionManager);

        // Check if there is already a cluster coordinator elected. If not, go ahead
        // and register for coordinator role. If there is already one elected, do not register until
        // we have connected to the cluster. This allows us to avoid becoming the coordinator with a
        // flow that is different from the rest of the cluster (especially an empty flow) and then
        // kicking everyone out. This way, we instead inherit the cluster flow before we attempt to be
        // the coordinator.
        LOG.info("Checking if there is already a Cluster Coordinator Elected...");
        final String clusterCoordinatorAddress = leaderElectionManager.getLeader(ClusterRoles.CLUSTER_COORDINATOR);
        if (StringUtils.isEmpty(clusterCoordinatorAddress)) {
            LOG.info("It appears that no Cluster Coordinator has been Elected yet. Registering for Cluster Coordinator Role.");
            registerForClusterCoordinator( participate: true);
        } else {
            // At this point, we have determined that there is a Cluster Coordinator elected. It is important to note, though,
            // that if we are running an embedded ZooKeeper, and we have just restarted the cluster (at least the nodes that run the
            // embedded ZooKeeper), that we could possibly determine that the Cluster Coordinator is at an address that is not really
            // valid. This is because the latest stable ZooKeeper does not support "Container ZNodes" and as a result the ZNodes that
            // are created are persistent, not ephemeral. Upon restart, we can get this persisted value, even though the node that belongs
            // to that address has not started. ZooKeeper/Curator will recognize this after a while and delete the ZNode. As a result,
            // we may later determine that there is in fact no Cluster Coordinator. If this happens, we will automatically register for
            // Cluster Coordinator through the StandardFlowService.
            LOG.info("The Election for Cluster Coordinator has already begun (Leader is {}). Will not register to be elected for this role until after connecting "
                    + "to the cluster and inheriting the cluster's flow.", clusterCoordinatorAddress);
            registerForClusterCoordinator( participate: false);
        }
```

Thực hiện loadbalance theo `loadBalanceRegistry`



```
        final InetSocketAddress loadBalanceAddress = nifiProperties.getClusterLoadBalanceAddress();
        // Setup Load Balancing Server
        final EventReporter eventReporter = createEventReporter();

        final LoadBalanceAuthorizer authorizeConnection = new ClusterLoadBalanceAuthorizer(clusterCoordinator, eventReporter);
        final LoadBalanceProtocol loadBalanceProtocol = new StandardLoadBalanceProtocol(flowFileRepo, contentRepository, provenanceRepository,  flowController: this, authori

        final int numThreads = nifiProperties.getIntegerProperty(NiFiProperties.LOAD_BALANCE_MAX_THREAD_COUNT, NiFiProperties.DEFAULT_LOAD_BALANCE_MAX_THREAD_COUNT);
        final String timeoutPeriod = nifiProperties.getProperty(NiFiProperties.LOAD_BALANCE_COMMS_TIMEOUT, NiFiProperties.DEFAULT_LOAD_BALANCE_COMMS_TIMEOUT);
        final int timeoutMillis = (int) FormatUtils.getTimeDuration(timeoutPeriod, TimeUnit.MILLISECONDS);

        loadBalanceServer = new ConnectionLoadBalanceServer(loadBalanceAddress.getHostName(), loadBalanceAddress.getPort(), sslContext,
                numThreads, loadBalanceProtocol, eventReporter, timeoutMillis);


        final int connectionsPerNode = nifiProperties.getIntegerProperty(NiFiProperties.LOAD_BALANCE_CONNECTIONS_PER_NODE, NiFiProperties.DEFAULT_LOAD_BALANCE_CONNECT
        final NioAsyncLoadBalanceClientFactory asyncClientFactory = new NioAsyncLoadBalanceClientFactory(sslContext, timeoutMillis, new ContentRepositoryFlowFileAcces
                eventReporter, new StandardLoadBalanceFlowFileCodec());
        loadBalanceClientRegistry = new NioAsyncLoadBalanceClientRegistry(asyncClientFactory, connectionsPerNode);

        final int loadBalanceClientThreadCount = nifiProperties.getIntegerProperty(NiFiProperties.LOAD_BALANCE_MAX_THREAD_COUNT, NiFiProperties.DEFAULT_LOAD_BALANCE_M
        loadBalanceClientThreadPool = new FlowEngine(loadBalanceClientThreadCount,  threadNamePrefix: "Load-Balanced Client",  daemon: true);

        for (int i = 0; i < loadBalanceClientThreadCount; i++) {
            final NioAsyncLoadBalanceClientTask clientTask = new NioAsyncLoadBalanceClientTask(loadBalanceClientRegistry, clusterCoordinator, eventReporter);
            loadBalanceClientTasks.add(clientTask);
            loadBalanceClientThreadPool.submit(clientTask);
        }
    } else {
        loadBalanceClientRegistry = null;
        heartbeater = null;
```