CAMEL-ESB 매뉴얼

프로젝트 명	CAMEL-ESB 매뉴얼
문서번호	2018-CAMEL-ESB-매뉴얼-5-1

TORPEDO

TORPEDO CORP.

개정이력

개정번호	내용	시행일자

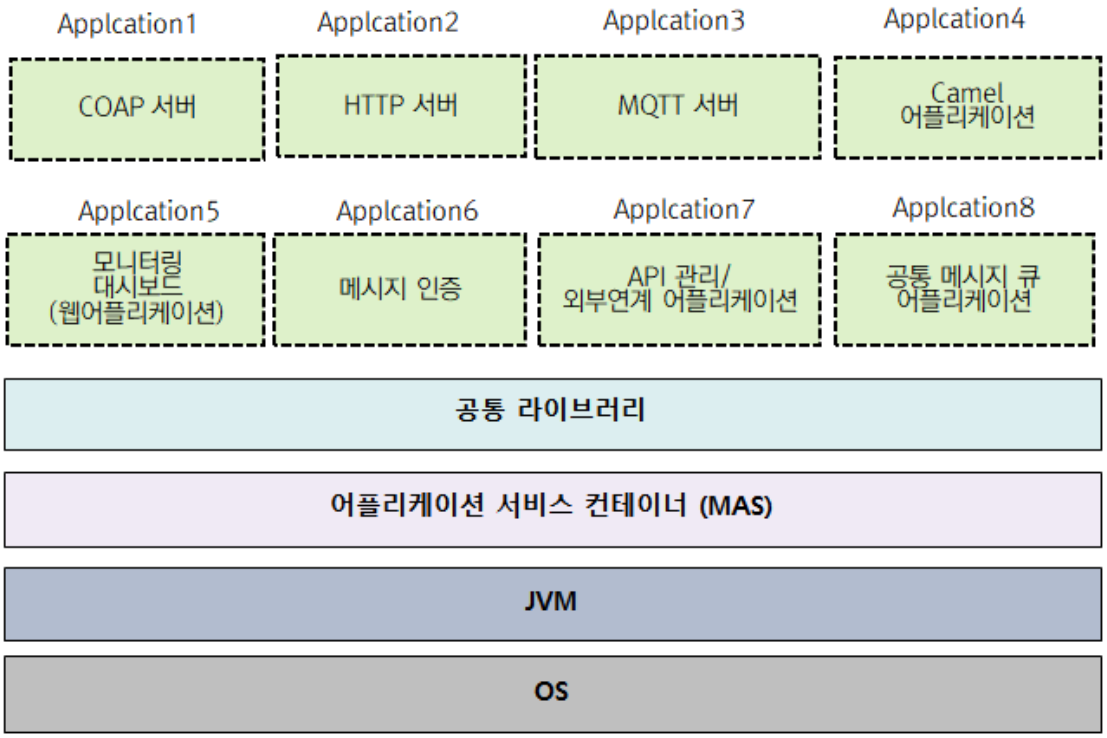
목차

1 Bizframe-MAS 서버

1.1 개념

1.1.1 구조도

Bizframe-MAS는 JVM 기반 하에서 어플리케이션을 운영하기 위한 어플리케이션 컨테이너이다. 일반적인 어플리케이션과 네트워크 어플리케이션, 웹 어플리케이션을 운영할 수 있으며 특히 ESB 운영을 위해서 Camel 어플리케이션을 운영할 수 있다.



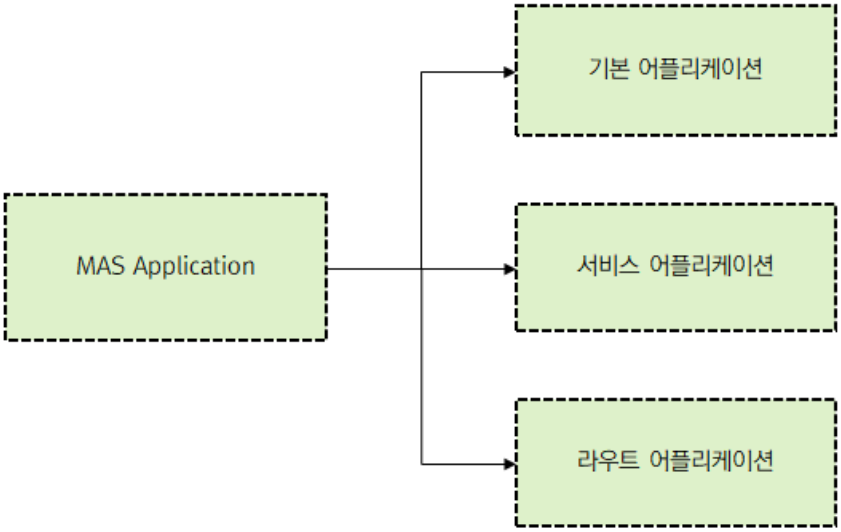
1.1.2 특징

Bizframe-MAS는 다음과 같은 특징을 가진다.

- ✓ JVM 기반 하에서 application 구동 및 관리를 위한 서버
- ✓ Application 초기화/종료 액션 가능
- ✓ Service 시작/중지 액션 가능
- ✓ Application 간 독립적인 클래스로더 사용으로 Application간 독립성 강화
- ✓ 일반 JAVA 클래스를 로딩할 수 있으며 cdi와 jetty는 내장시킴
- ✓ Application 간 메시지 교환을 위해서 route 구조 내장시킴

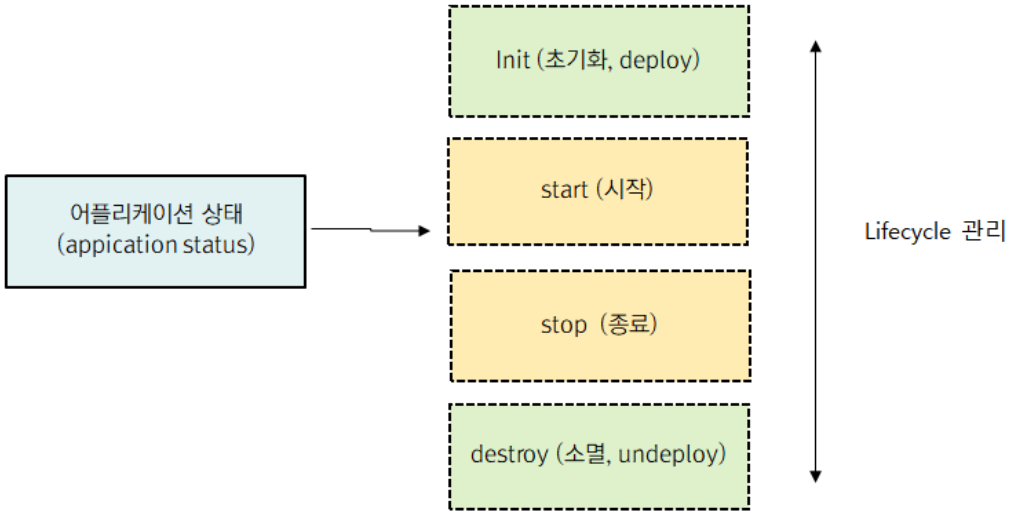
1.1.3 어플리케이션 유형

MAS 어플리케이션은 크게 기본 어플리케이션, 서비스 어플리케이션, 그리고 라우트 어플리케이션으로 나뉘다. 기본 어플리케이션은 초기화 상태, 종료 상태 2가지 상태만을 가진 어플리케이션이고 서비스 어플리케이션은 초기화 상태, 시작 상태, 멈춤 상태, 종료 상태 4개의 상태를 가지는 어플리케이션이다. 라우트 어플리케이션은 어플리케이션 간의 메시지 교환을 위해서 사용된다.



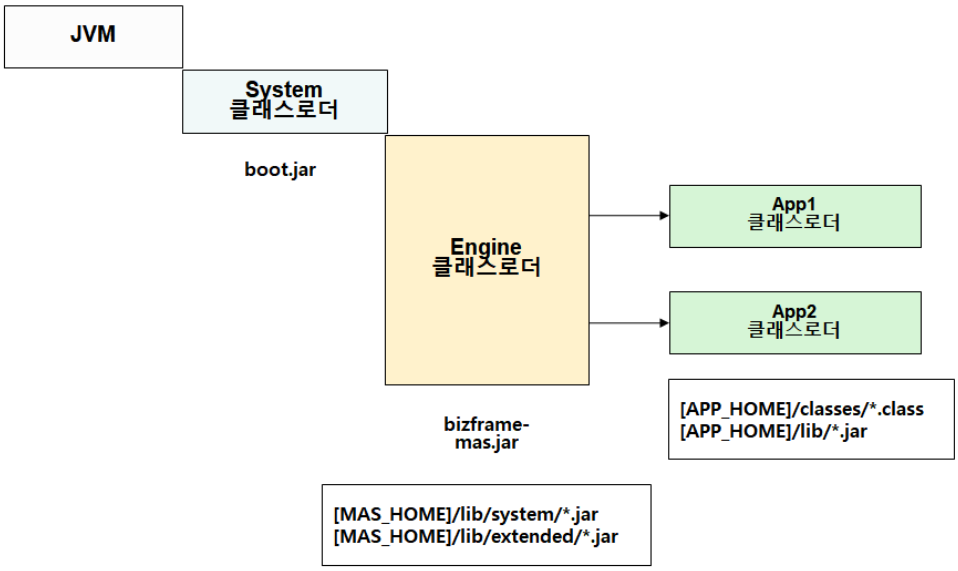
1.1.4 어플리케이션 상태

Bizframe-MAS에 의해서 관리되는 어플리케이션은 초기화 상태(deploy 상태), 시작 상태, 종료 상태, 소멸 상태(undeploy 상태)를 가진다. 이러한 상태 변화는 MAS를 초기 기동하거나 관리자에 의해서 어플리케이션 관리를 수행할 경우 상태의 변화가 생긴다.



1.1.5 클래스로더 관리

Bizframe-MAS는 다중의 어플리케이션을 관리하기 위해서 WAS(Web Application Server)와 비슷한 다층의 클래스로더 관리 구조를 갖는다. MAS 기동시는 System 클래스로더의 boot.jar를 이용하여 올라가며 내부적으로 Engine 클래스로더를 사용하여 [MAS_HOME]/lib/system 하위의 JAR와 [MAS_HOME]/lib/extended 하위의 JAR를 로딩한다. 그리고 어플리케이션들은 Engine클래스로더 하위에서 각각 자기의 클래스로더를 생성하여 JAR파일, class 파일을 로딩하여 어플리케이션을 구동하게 된다.



1.2 설치

1.2.1 서버 사양

항목	권장사양
CPU	dual-core 이상
메모리	4GB이상(JVM 옵션 최소 1GB 설정)
지원 OS	Linux, Windows
JAVA 버전	1.8 이상

1.2.2 서버 설치

BizFrame-MAS 서버의 설치는 배포 압축 파일을 설치하고자 하는 디렉터리에 풀고 실행 파일을 설정하는 것으로 이루어 진다.

1.2.3 Linux에 설치

(1) 주소 디렉터리 서버 설치에 필요한 서버의 환경정보를 설정한다

환경변수	값
주소 디렉터리 서버 HOME 디렉터리	/home/mas
JAVA_HOME	/usr/java16

(실제 설정 값은 사용자 서버의 환경 정보에 따라 설정 한다..)

(2) Bizfram-MAS 서버 HOME 디렉터리($MAS_HOME)에 압축 파일을 해제한다.

```
# gzip –d bizframe-mas-dist-2.0.0.tar.gz
# tar –xvf bizframe-mas-dist-2.0.0.tar
```

(3) $MAS_HOME/bin/로 이동 후 startup.sh 실행, $MAS_HOME/logs 디렉터리에 mas.log 파일을 열어 실행 로그를 확인한다.

(4) esb@fog:~/mas/bin/startup.sh

```
21:00:32.908 [main] INFO  kr.co.bizframe.mas.conf.MasConfig - MasConfig initialize..
21:00:32.908 [main] INFO kr.co.bizframe.mas.conf.MasConfig - MasConfig file=file:/D:/mas_test/bizframe-mas-dist-2.0.0/conf/mas-conf.xml
21:00:32.939 [main] INFO  k.c.b.mas.management.JMXManager - register server
21:00:32.986 [main] INFO  kr.co.bizframe.mas.core.MasServer - mas server is booting..
21:00:33.002 [main] INFO  kr.co.bizframe.mas.conf.MasConfig - MasConfig initialize..
21:00:33.002 [main] INFO kr.co.bizframe.mas.conf.MasConfig - MasConfig file=file:/D:/mas_test/bizframe-mas-dist-2.0.0/conf/mas-conf.xml
21:00:33.002 [main] INFO  kr.co.bizframe.mas.core.MasServer - mas server listen on port=[9004]
21:00:33.439 [main] INFO  kr.co.bizframe.mas.core.MasProcess - mas process id=[24288]
21:00:33.439 [main] INFO  kr.co.bizframe.mas.core.MasEngine - engine conf = EngineDef [id=null, hotDeploy=false, routing=RoutingDef [className=kr.co.bizframe.mas.routing.impl.DefaultRoutingManager, enable=false], applications=null]
21:00:33.439 [main] INFO k.c.b.m.a.ApplicationManager - applications base home=[D:₩mas_test₩bizframe-mas-dist-2.0.0/applications]
21:00:33.439 [main] INFO  kr.co.bizframe.mas.core.MasEngine - BizFrame mas engine started.
```

1.2.4 Windows에 설치

(1) 주소 디렉터리 서버 설치에 필요한 서버의 환경정보를 설정한다

환경변수	값
주소 디렉터리 서버 HOME 디렉터리	C:/mas
JAVA_HOME	C:/java16

(실제 설정 값은 사용자 서버의 환경 정보에 따라 설정 한다..)

(2) 압축프로그램을 이용하여 Bizframe-MAS 서버 HOME 디렉터리(%MAS_HOME%)에 압축 파일을 해제한다.

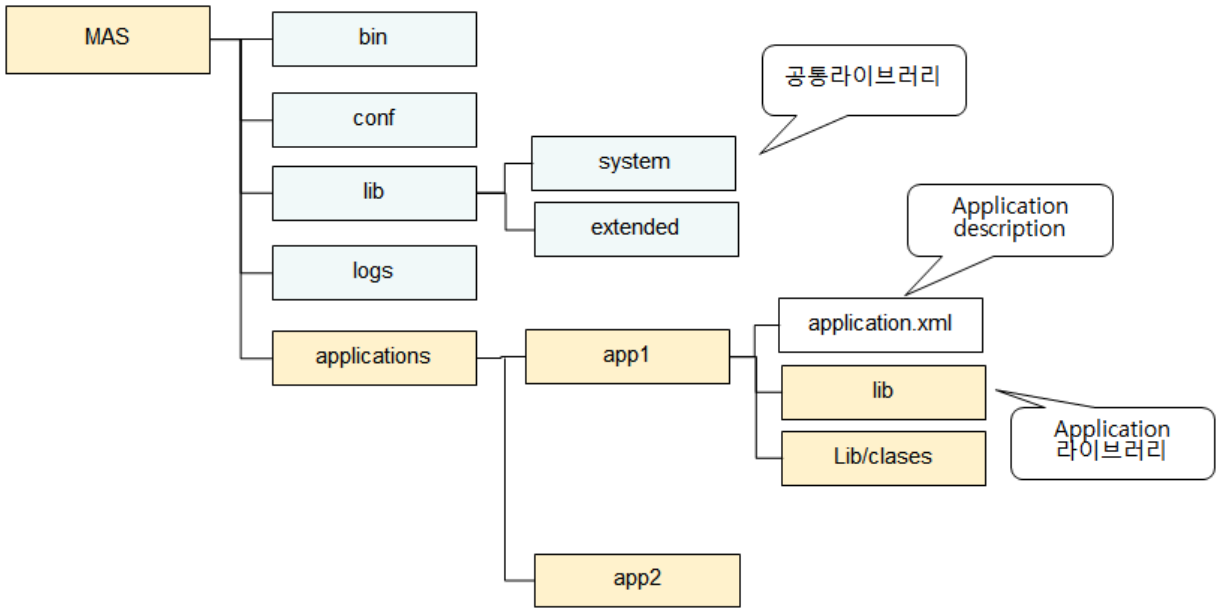
(3) %MAS_HOME%/bin/로 이동 후 startup.sh 실행, %MAS_HOME%/logs 디렉터리에 mas.log 파일을 열어 실행 로그를 확인한다.

1.3 파일 구조

1.3.1 디렉토리 구조

MAS 서버 배포본 압축 파일을 풀면 다음과 같은 디렉터리 구조가 나타난다.

디렉터리	설명
[MAS_HOME]	MAS 서버가 설치된 홈 디렉터리
├ bin	MAS 서버 제어용 실행 파일들 위치
├ conf	MAS 서버 환경 설정 파일들 위치
├ logs	로그 파일 위치. 설정에 의해 변경 가능.
├ lib	MAS 서버 바이너리와 각종 라이브러리 파일(*.jar)들의 위치
└ extends	MAS 추가 확장 라이브러리
└ system	시스템 필수 라이브러리
├ applications	어플리케이션 파일 디렉터리



1.3.2 실행 파일

Bizframe-MAS의 실행 파일들은 [MAS_HOME]/bin 디렉터리 하위에 위치한다.

파일명	계열	설명
startup.bat	윈도우	MAS 시작
startup.sh	유닉스	
shutdown.bat	윈도우	MAS 종료
shutdown.sh	유닉스	
cmd.bat	윈도우	커맨드 라인 명령 실행
cmd.sh	유닉스	
mas.bat	윈도우	MAS 직접 실행 명령
mas.sh	유닉스	

1.3.3 설정 파일

Bizframe-MAS의 설정 파일들은 [MAS_HOME]/conf 디렉터리 하위에 위치한다.

파일명	설명
mas-conf.xml	MAS의 메인 설정파일로 MAS Admin port 설정 및 기본 App 디렉터리 설정 등 기본적인 정보를 설정한다.
Lobback.xml	MAS의 로깅을 설정한다.

1.3.4 라이브러리 파일

- 시스템 필수 라이브러리 - Bizframe-MAS의 시스템 라이브러리 파일은 [MAS_HOME]/lib/system 디렉터리 하위에 위치한다.

파일명	설명
bizframe-mas-2.0.0.jar	MAS 코어 라이브러리
commons-io-2.6.jar	Apache common io 라이브러리
jdom.jar	XML 처리 라이브러리
logback-access-1.1.8.jar **logback-classic-1.1.8.jar** **logback-core-1.1.8.jar** **slf4j-api-1.7.22.jar**	Log 처리 관련 라이브러리

- 확장 라이브러리 - Bizframe-MAS의 실행 파일들은 [MAS_HOME]/lib/extended 디렉터리 하위에 위치한다.

파일명	설명
activation.jar	Java Mail 관련 라이브러리
commons-codec-1.10.jar	Apache common 코덱 관련 라이브러리
commons-exec-1.3.jar	Apache common 명령라인 수행 라이브러리
commons-logging-1.1.3.jar	Apache common 로깅 라이브러리
jna-platform.jar	JNA Native 프로세스 플랫폼별 수행 관련 라이브러리
jna.jar	JNA Native 프로세스 수행 관련 라이브러리
servlet-api-3.1.jar	Java 서블릿 관련 라이브러리
camel	Camel 관련 라이브러리 디렉토리로 하위에 Camel 관련 라이브러리를 위치시킨다.
jetty	Jetty 관련 라이브러리 디렉토리로 하위에 Jetty 관련 라이브러리를 위치시킨다.
activemq	ActiveMQ 관련 라이브러리 디렉토리로 하위에 ActiveMQ 관련 라이브러리를 위치시킨다. (ActiveMQ 미사용시 이 디렉토리는 삭제가능)
spring	spring 관련 라이브러리 디렉토리로 하위에 spring 관련 라이브러리를 위치시킨다.
tomcat	tomcat 관련 라이브러리 디렉토리로 하위에 tomcat 관련 라이브러리를 위치시킨다. (tomcat 미사용시 이 디렉토리는 삭제가능)

1.4 Bizframe-MAS 운영 설정

1.4.1 MAS 설정 (mas-conf.xml)

mas-conf.xml은 MAS 서버의 동작을 설정하는 파일이다.

변수	설명
mas-conf/server/id	MAS 서버의 id
mas-conf/server/port	MAS 서버의 admin port
mas-conf/server/engine/id	MAS 엔진의 id (일반적인 경우 engine 엘리먼트는 한 개며 id는 1로 사용한다.
mas-conf/server/engine/hot-deploy	MAS 엔진이 hot-deploy를 지원할지 여부 (기본 false)

1.4.2 로깅 설정 (logback.xml)

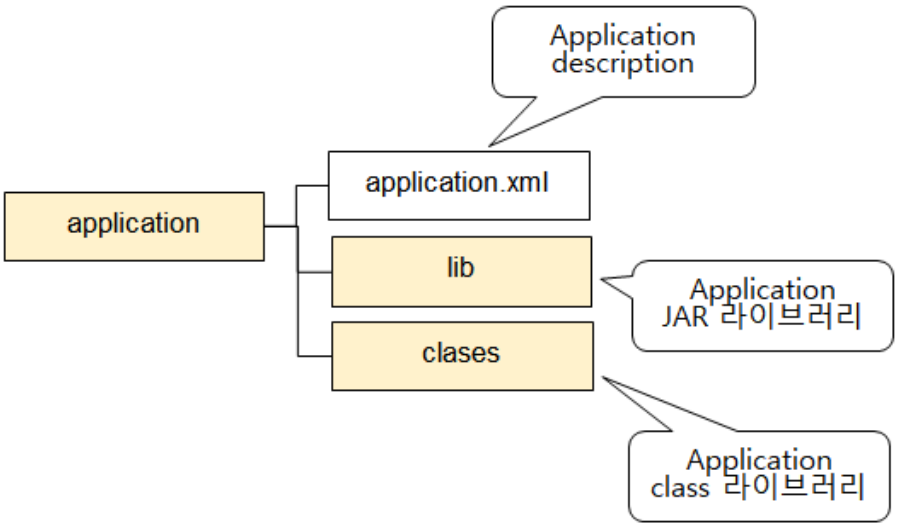
Logback.xml은 MAS 서버 및 어플리케이션에서 생성되는 로그를 설정한다.

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration scan="true" scanPeriod="30 seconds">

    <!--property resource="resource.properties"/-->
        <property name="LOG_DIR" value="../logs" /> //로그 디렉토리 설정

    <appender name="CONSOLE" class="ch.qos.logback.core.ConsoleAppender">
        <encoder>
            <pattern> %d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern> //로그 패턴 설정

        </encoder>
    </appender>
    <appender name="FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
        <file>${LOG_DIR}/mas.log</file>
            <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
            <fileNamePattern>${LOG_DIR}/mas.%d{yyyy-MM-dd}.log</fileNamePattern>
            <maxHistory>30</maxHistory>
        </rollingPolicy>
```

```
        <encoder>
            <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{35} - %msg%n</pattern>
        </encoder>
    </appender>

    <logger name="kr.co.bizframe" level="DEBUG"/>   // MAS의 로그 레벨 설정
    <logger name="org.apache.camel" level="DEBUG"/> // CAMEL의 로그 레벨 설정
    <logger name="org.apache.catalina" level="DEBUG"/> //TOMCAT의 로그 레벨 설정
    <!--logger name="org.eclipse.jetty" level="TRACE"/-->
    <root level="info">
        <appender-ref ref="CONSOLE"/>
        <appender-ref ref="FILE"/>
     </root>
</configuration>
```

1.4.3 어플리케이션 구성 및 설정

MAS 어플리케이션은 다음과 같은 구조를 가진다. Application은 디렉토리 형식으로 구성되거나 디렉토리를 묶은 ZIP 파일 형태 둘다 지원한다. 디렉토리 내에는 어플리케이션 구성을 기술하는 application.xml 파일이 존재한다. lib 디렉토리 하위는 어플리케이션이 사용하는 JAR 파일이 위치하며 classes 디렉토리 하위는 어플리케이션이 사용하는 classes 파일 및 각종 설정 파일이 위치하게 된다.



(1) 어플리케이션 설정 (application.xml)

application.xml

변수	설명
application/id	어플리케이션 id
application/name	어플리케이션 표시 이름
application/load-class	어플리케이션을 구성하는 메인 클래스. 이 클래스가 명시되고 클래스가 존재하여야 어플리케이션이 정상적으로 작동한다.
application/load-sequence	어플리케이션 로딩 순서로 여러 개의 어플리케이션이 존재할 경우 먼저 로딩할 순서를 기술한다. (적은 숫자가 먼저 기동된다)
auto-deploy	MAS 서버 시작 시 해당 어플리케이션이 자동으로 deploy 될지 여부
auto-start	MAS 서버 시작 시 해당 어플리케이션이 자동으로 start 될지 여부
application/parent-first-classloader	상위에 존재하는 클래스로더에서 먼저 클래스를 찾도록 설정함. (기본:false)
application/parent-only-classloader	상위에 존재하는 클래스로더에서만 클래스를 찾도록 설정함. (기본:false)
application/params/param	어플리케이션에 key/value 형태로 파라미터를 전달하도록 설정할 수 있는 설정값

1.5 어플리케이션 구현

1.5.1 어플리케이션 작성

어플리케이션을 구현하기 위해서는 (1) 기본 인터페이스를 구현한 클래스 혹은 (2) 서비스 인터페이스를 구현한 클래스를 작성 후 application.xml에 클래스 이름을 설정하여 주면 된다.

(1) 기본 인터페이스

```
public interface Application {
public void init(ApplicationContext context) throws ApplicationException;
public void destroy(ApplicationContext context) throws pplicationException;
}
```

(2) 서비스 인터페이스

```
public interface Serviceable {
   public void start() throws Exception;
   public void stop() throws Exception;
}
```

1.5.2 Read-made 어플리케이션

Ready-made 어플리케이션은 Bizframe-MAS 내부적으로 먼저 구현되어 있는 어플리케이션을 말한다. 이러한 어플리케이션은 클래스 파일을 어플리케이션 디렉토리 내에 위치하지 않아도 된다.

어플리케이션명	어플리케이션 클래스	설명
JettyApplication	kr.co.bizframe.mas.web.JettyApplication	Jetty 웹어플리케이션
TomcatApplication	kr.co.bizframe.mas.web.TomcatApplication	Tomcat 웹어플리케이션
CamelApplication	kr.co.bizframe.mas.camel.CamelApplication	Camel ESB 어플리케이션

(1) JettyApplication

Jetty 어플리케이션은 Jetty (https://www.eclipse.org/jetty) 웹서버를 MAS 내에서 사용하기 위한 어플리케이션이다. 어플리케이션은 WEB 어플리케이션 형태로 위치하면 된다.

파라미터	값	설명
doc_base	서버 저장 경로(/home/imxs/)	웹어플리케이션 서버 경로
context_path	웹접근 패스 (/imxs)	웹어플리케이션 컨텍스트 패스
port	포트번호(예: 9090)	웹어플리케이션 포트
https	True/false	SSL(TLS) 지원여부
keystoreFile	파일 경로	https를 사용할 경우 키스토어 파일 경로
keystorePass	패스워드	https를 사용할 경우 키스토어 패스워드
truststoreFile	파일 경로	https를 사용할 경우 키스토어 파일 경로
truststorePass	패스워드	https를 사용할 경우 키스토어 패스워드
enable_jmx	True/false	JMX 가능 여부
use_filemappered_buffer	True/false	서버 파일 동적 변경 가능 여부

(2) TomcatApplication

Tomcat 어플리케이션은 Tomcat (https://tomcat.apache.org/) 웹서버를 MAS 내에서 사용하기 위한 어플리케이션이다. 어플리케이션은 WEB 어플리케이션 형태로 위치하면 된다.

파라미터	값	설명
doc_base	서버 저장 경로(/home/imxs/)	웹어플리케이션 서버 경로
context_path	웹접근 패스 (/imxs)	웹어플리케이션 컨텍스트 패스
port	포트번호(예: 9090)	웹어플리케이션 포트
https	True/false	SSL(TLS) 지원여부
keystoreFile	파일 경로	https를 사용할 경우 키스토어 파일 경로
keystorePass	패스워드	https를 사용할 경우 키스토어 패스워드
truststoreFile	파일 경로	https를 사용할 경우 키스토어 파일 경로
truststorePass	패스워드	https를 사용할 경우 키스토어 패스워드

(3) CamelApplication

Camel 어플리케이션은 Camel ESB를 사용하기 위한 어플리케이션이다. Camel 어플리케이션은 내부적으로 spring DSL XML 파일을 라우팅 설정 파일로 사용한다.

파라미터	값	설명
route_xml_file	(기본 : camel-route.xml)	Spring DSL XML 파일 설정
shutdown_timeout	Timeout 시간 값 (단위 초)	Camel shutdown 시 timeout 시간 설정

1.6 커맨드 라인 명령 관리 툴

1.6.1 커맨드 라인 명령 관리 툴

커맨드 라인 명령 관리 툴은 MAS를 설치하고 간단하게 어플리케이션을 조회하고 시작/종료 등을 수행할 수 있는 관리 툴이다. 명령어 기반이 아닌 웹 관리자 화면상에서 조작할 수 있는 관리 툴도 있으나 기본적으로 포함되지는 않는다.

1.6.2 커맨드 라인 명령 관리 툴 시작/종료

커맨드 라인 관리 툴은 기본적으로 MAS를 설치하고 [MAS_HOME]/bin/cmd.bat 혹은 cmd.sh를 실행하여 시작한다. MAS 서버가 정상적으로 작동하고 있을 경우 아래와 같이 프롬프트 상태로 명령을 대기하게 된다.
종료의 경우 quit 명령을 입력한다.

```
D:₩mas_test₩bizframe-mas-dist-2.0.0₩bin>cmd.bat
Start BizFrame MAS command tool
JAVA_HOME : C:₩jdk1.8.0_121
PRODUCT_HOME :  ../
============================================================
====== BizFrame micro application server  v2.0.0  ========
============================================================
cmd>
```

1.6.3 상세 커맨드 라인 관리 툴 명령

커맨드라인 프롬프트 상태에서 help 를 입력시 사용할수 있는 명령 리스트를 조회할 수 있다.

```
D:₩mas_test₩bizframe-mas-dist-2.0.0₩bin>cmd.bat
Start BizFrame MAS command tool
JAVA_HOME : C:₩jdk1.8.0_121
PRODUCT_HOME :  ../
============================================================
====== BizFrame micro application server  v2.0.0  ========
============================================================
cmd> help
 quit - quit command line tool
 help - help command line tool
 connect [host port [timeout]] - connect server
 status - status MAS server
 shutdown - shutdown MAS server
 version - version of MAS server
 startapp [app_id] - start application with app_id
 stopapp [app_id] - stop application with app_id
 deployapp [app_id] - deploy application with app_id
 undeployapp [app_id] - undeploy application with app_id
 removeapp [app_id] - remove application with app_id
 appinfo [app_id] - application info with app_id
 applist  -  application info list
 appdef [app_id]- application def info
 appdeflist - application def list
 refreshappdef - refresh application def list
cmd>
```

명령	설명	상세 파라미터
quit	커맨드 라인 명령 툴 종료	
help	도움말 – 명령어 리스트 출력	
connect	MAS 서버 접속	[host port [timeout]]
status	MAS 서버 상태 출력	
shutdown	MAS 서버 종료	
version	MAS 서버 버전 출력	
startapp	어플리케이션 시작	어플리케이션 id
stopapp	어플리케이션 종료	어플리케이션 id
deployapp	어플리케이션 초기화	어플리케이션 id
undeployapp	어플리케이션 소멸	어플리케이션 id
removeapp	어플리케이션 삭제	어플리케이션 id
appinfo	어플리케이션 정보 출력	어플리케이션 id
applist	어플리케이션 리스트 출력	
appdef	어플리케이션 정의 출력 (application.xml 내용)	어플리케이션 id
appdeflist	어플리케이션 정의 리스트 출력	
refreshappdef	어플리케이션 정의 갱신	어플리케이션 id

2 관리/모니터링

2.1 개념

2.1.1 모니터링 대상

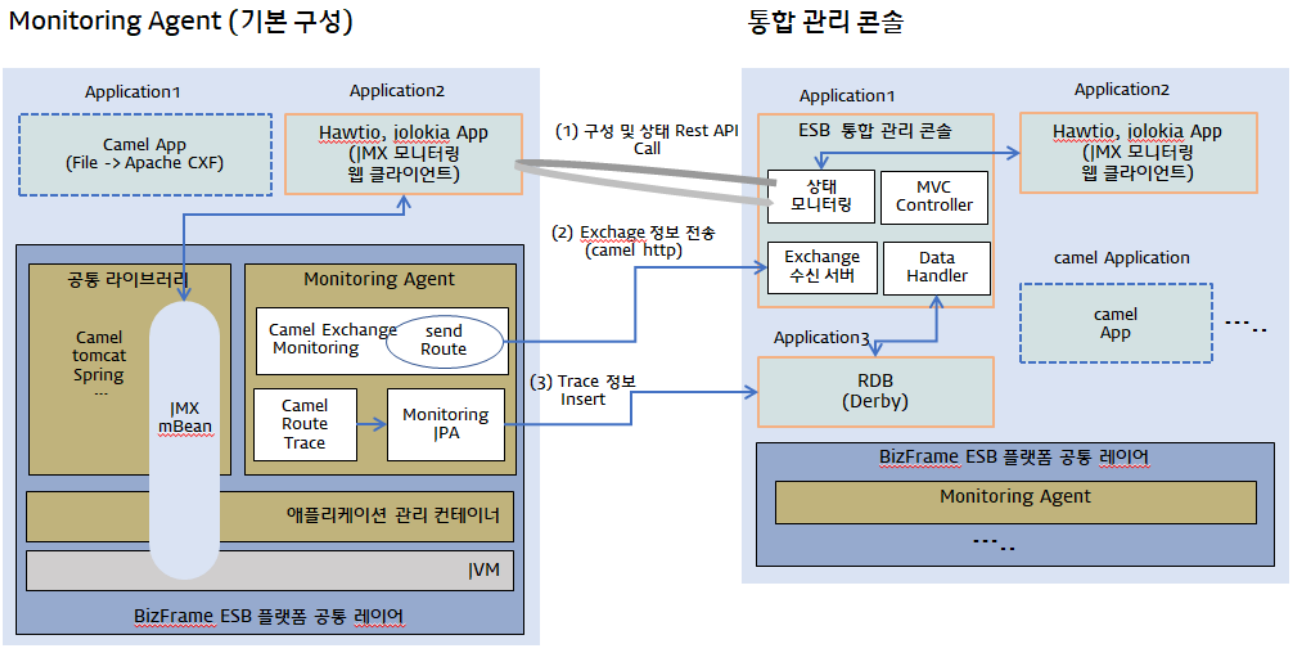
Camel-ESB 플랫폼에서 모니터링 하는 대상은 아래와 같다.

- BizFrame-MAS에 deploy된 어플리케이션 목록 및 상태, 기본 정보
- BizFrame-MAS에 deploy된 어플리케이션 서비스 제어(시작, 중지)
- BizFrame-MAS, Camel 어플리케이션 JMX 모니터링
- Camel Exchange 전송 정보 실시간 모니터링 및 이력관리
- Camel Exchange 단계 몇 Trace 정보 이력관리

2.1.2 구성

Camel-ESB 플랫폼의 관리 및 모니터링을 위한 어플리케이션은 아래와 같다.

- Monitoring Agent : Camel Exchange, Trace 데이터 수집 및 전송
- Hawtio-Jolokia : JMX 모니터링 및 웹 클라이언트
- ESB-mng-console : Camel Exchange, Trace 데이터 처리 및 관리, 통합 웹 관리 콘솔
- Derby : 모니터링 데이터 관리를 위한 RDB



Monitoring Agent는 trace, Exchange, 상태 모니터링 구성(enable/disable)을 설정으로 조정할 수 있음

2.2 Monitoring Agent

Monitoring Agent는 모니터링 대상 Camel-ESB 플랫폼의 [MAS_HOME]/lib/extended/camel-monitoring 디렉터리 하위에 실행 라이브러리 형태로 설치한다.

1) Camel Exchange 정보 수집은 org.apache.camel.support.EventNotifierSupport.notify 인터페이스를 구현함으로 처리된다.

해당 인터페이스는

* ExchangeCreatedEvent
* ExchangeSendingEvent
* ExchangeSentEvent
* ExchangeCompletedEvent
* ExchangeFailedEvent

와 같은 이벤트 발생 시 notify 메서드를 호출하는데 본 플랫폼에서는 Camel route 수행 결과를 포함한 ExchangeCompletedEvent(성공), ExchangeFailedEvent(실패)에 대해서 Exchange정보를 모니터링 한다.

모니터링된 Exchange 정보는 설정 주기(default : 5초) 동안 수집된 후 json형태로 아래와 같은 route설정에 따라 수집 서버로 http 전송된다.

```
<route id="sendTrace" trace="false">
        <from uri="timer://foo?fixedRate=true&amp;period=5000"/>
        <to uri="bean:bizFrameExchangeTracer?method=sendFinishedExchangeInfos"/>
        <choice>
```

```
            <when>
                <simple>${bodyAs(String).length} > 0</simple>
                <log message="in body size : ${bodyAs(String).length}"/>
                <to uri="jetty:http://localhost:8083/trace"/>
                <log message="----------------response body : ${in.body}"/>
                <to uri="bean:bizFrameExchangeTracer?method=clearFinishedExchangeInfos"/>
            </when>
        </choice>
        <onException>
                <exception>java.net.ConnectException</exception>
                <handled><constant>true</constant></handled>
                <process ref="bizFrameExchangeTracerErrorProcessor" />
        </onException>
</route>
```

2) Camel Exchange 단계 별 Trace 정보 수집은 camel에 내장된 org.apache.camel.processor.interceptor.Tracer를 확장하여 본 플랫폼에 필요한 데이터 모델 재 정의 후 처리된다.

Trace interceptor에 따라 Exchange in/out 단계 별로 trace된 데이터는 JPA 인터페이스로 직접 DB에 저장된다. (Camel-ESB 에서는 Derby를 네트워크 형태로 구성하여 사용)
JPA 인터페이스에 따라 데이터를 저장하기 위해서는 [MAS_HOME]/conf/META-INF/persistence.xml에 JPA 구현체인 hibernate 설정이 필요하다. (별도의 DB에서 모니터링 데이터를 관리하기 위해서는 아래 설정을 수정)

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             version="1.0">

  <persistence-unit name="bizframeTracer" transaction-type="RESOURCE_LOCAL">

<class>kr.co.bizframe.esb.camel.monitoring.tracer.BizFrameJpaTraceEventMessage</class>
    <properties>
      <property name="hibernate.dialect"
value="kr.co.bizframe.esb.camel.monitoring.tracer.FixedDerbyDialect"/>
      <property name="hibernate.connection.driver_class"
value="org.apache.derby.jdbc.ClientDriver"/>
      <property name="hibernate.connection.url"
```

```
value="jdbc:derby://localhost:3333/./../data/db/trace;create=true"/>
      <property name="hibernate.hbm2ddl.auto" value="update"/>
         <property name="hibernate.connection.username" value="app"/>
         <property name="hibernate.connection.password" value="app"/>

      <!-- debugging flags -->
      <property name="hibernate.show_sql" value="true"/>
      <property name="hibernate.format_sql" value="true"/>
    </properties>
  </persistence-unit>
</persistence>
```

3) 모니터링 설치

모니터링 구성에 대한 설정은 camel-monitoring.properties 파일에 정의한다.([MAS_HOME]/conf/camel-monitoring.properties)

명령	설명	상세 파라미터
tracer.enabled	Trace 모니터링 여부	true(default), false
tracer.exclude.routes	Trace 모니터링 제외 camel routeId	
tracer.jpa	Trace 데이터 JPA 처리 여부	true(default), false
tracer.log.level	Trace 정보 log 레벨	TRACE, DEBUG, INFO, WARN, ERROR, OFF(default)
trace.exchange.enable	Exchange 모니터링 여부	true(default), false
trace.exchange.exclude.routes	Exchange 모니터링 제외 camel routeId	
trace.exchange.timer.period	수집된 Exchange 데이터 전송 주기	단위 : sec 초, default : 60
trace.exchange.server.endpoint	수집된 Exchange 데이터 전송 서버	jetty:http://[host]:[port]/trace

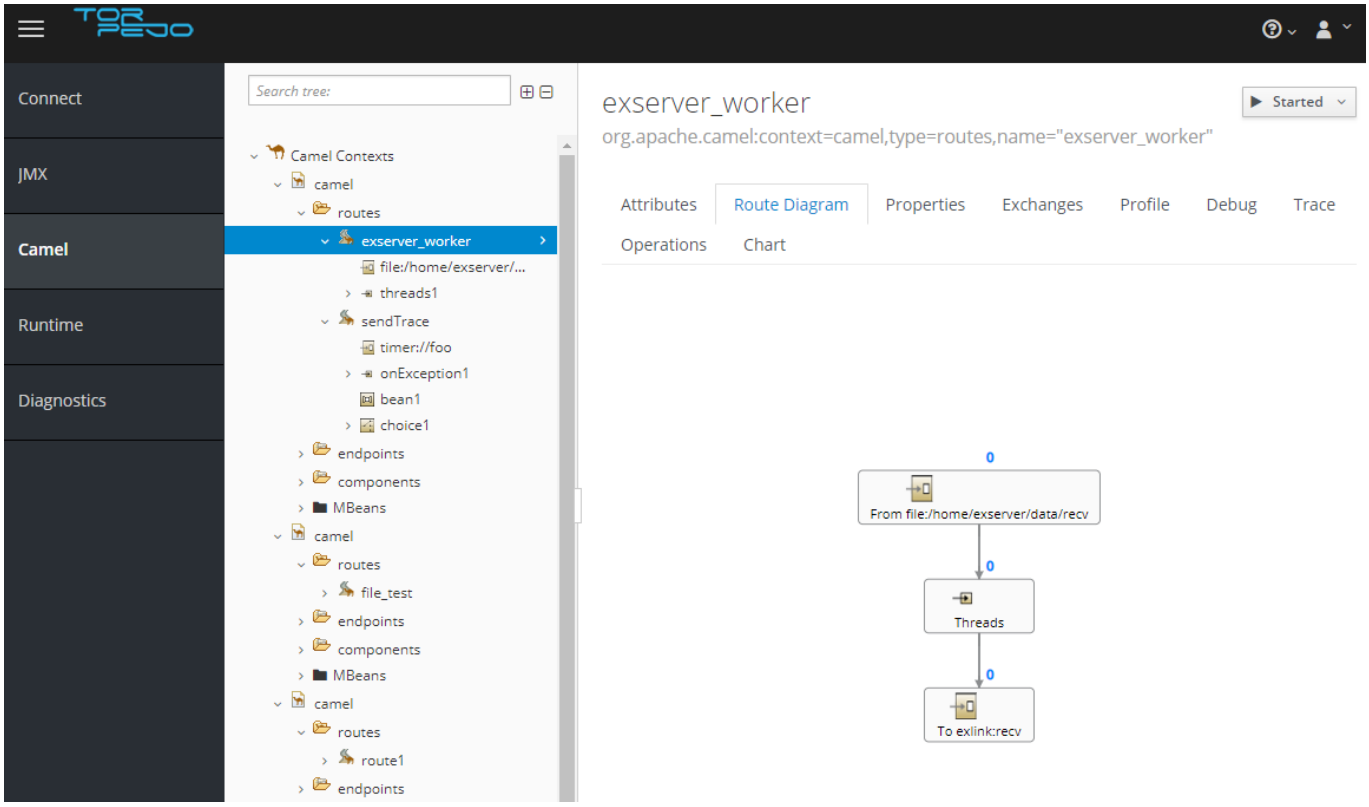
모니터링 구성을 실제 Camel 어플리케이션에 적용하기 위해서는 spring DSL XML 라우팅 설정 파일인 camel-route.xml의 context 하위에 모니터링 설정을 JAVA DSL로 구현한 RouteBuilder 를 반드시 import시켜야 한다.

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="
        http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
        http://camel.apache.org/schema/spring
http://camel.apache.org/schema/spring/camel-spring.xsd">

       <!-- monitoring -->
       <bean id="monitoringRouteBuilder"
class="kr.co.bizframe.esb.camel.monitoring.MonitoringRouteBuilder"/>
       <!-- monitoring -->

       <camelContext id="bean-test" xmlns="http://camel.apache.org/schema/spring">

             <!-- monitoring -->
             <routeBuilder ref="monitoringRouteBuilder"/>
             <!-- monitoring -->

             <route id="http-client">
               <from uri="file:c:/data/camel/input?noop=true&amp;delay=5000"/>
               <to uri="http://localhost:9011/myService/"/>
          </route>
       </camelContext>
</beans>
```

2.3 Hawtio-Jolokia

JMX 인터페이스를 구현한 어플리케이션의 모니터링을 위해 JMX-HTTP 컨버터(REST API제공) Jolokia 와 JMX 웹 클라이언트인 Hawtio를 적용하였음
[MAS_HOME]/applications/hawtio 웹 어플리케이션 설치

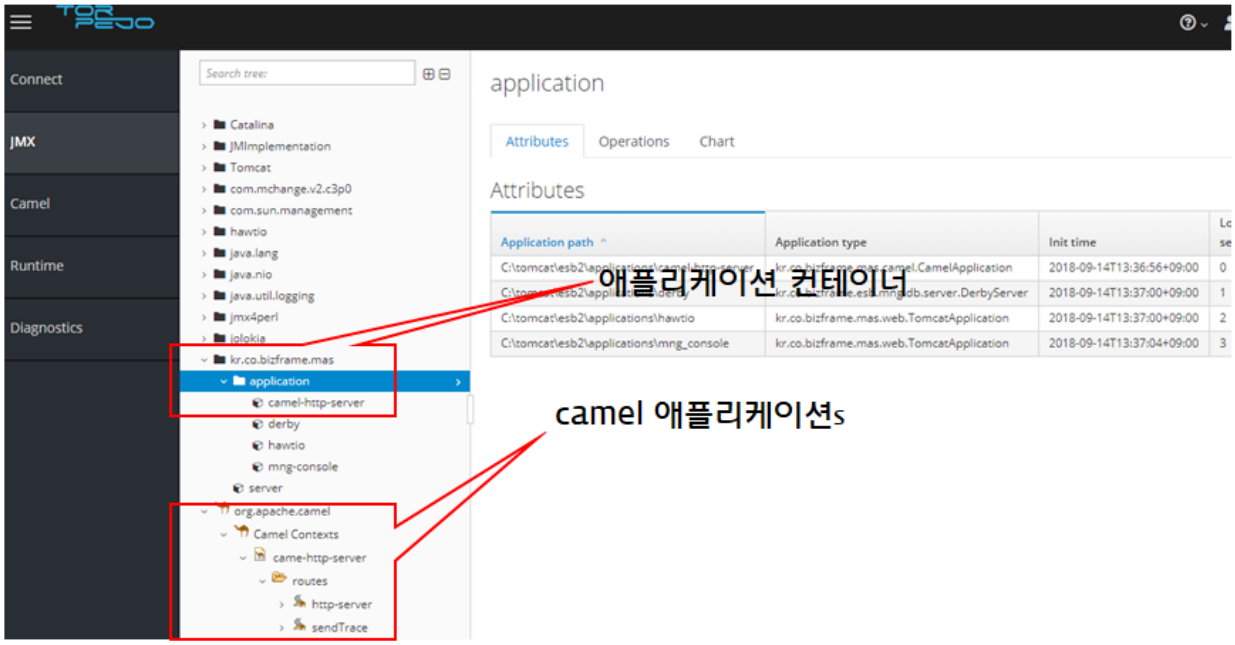
Hawtio-Jolokia를 통해 camel, bizframe-mas의 주요 정보 및 오퍼레이션을 처리할 수 있음
Camel 어플리케이션의 경우 Hawtio camel 구현체에 따라 Route Diagram 및 trace 등 다양한 세부 기능을 구현



BizFrame-MAS는 어플리케이션 목록 및 상태, 제어기능을 JMX인터페이스로 제공

JMX 각 항목이나 제어는 JMX 외부 API 접근 URL (→ jolokia)

ex : http://[ip]/:[port]/hawtio/jolokia/read/kr.co.bizframe.mas:type=application,*로 제공되어 웹 또는 그 외 클라이언트에서 호출 가능



2.4 ESB-mng-console

ESB-mng-console은

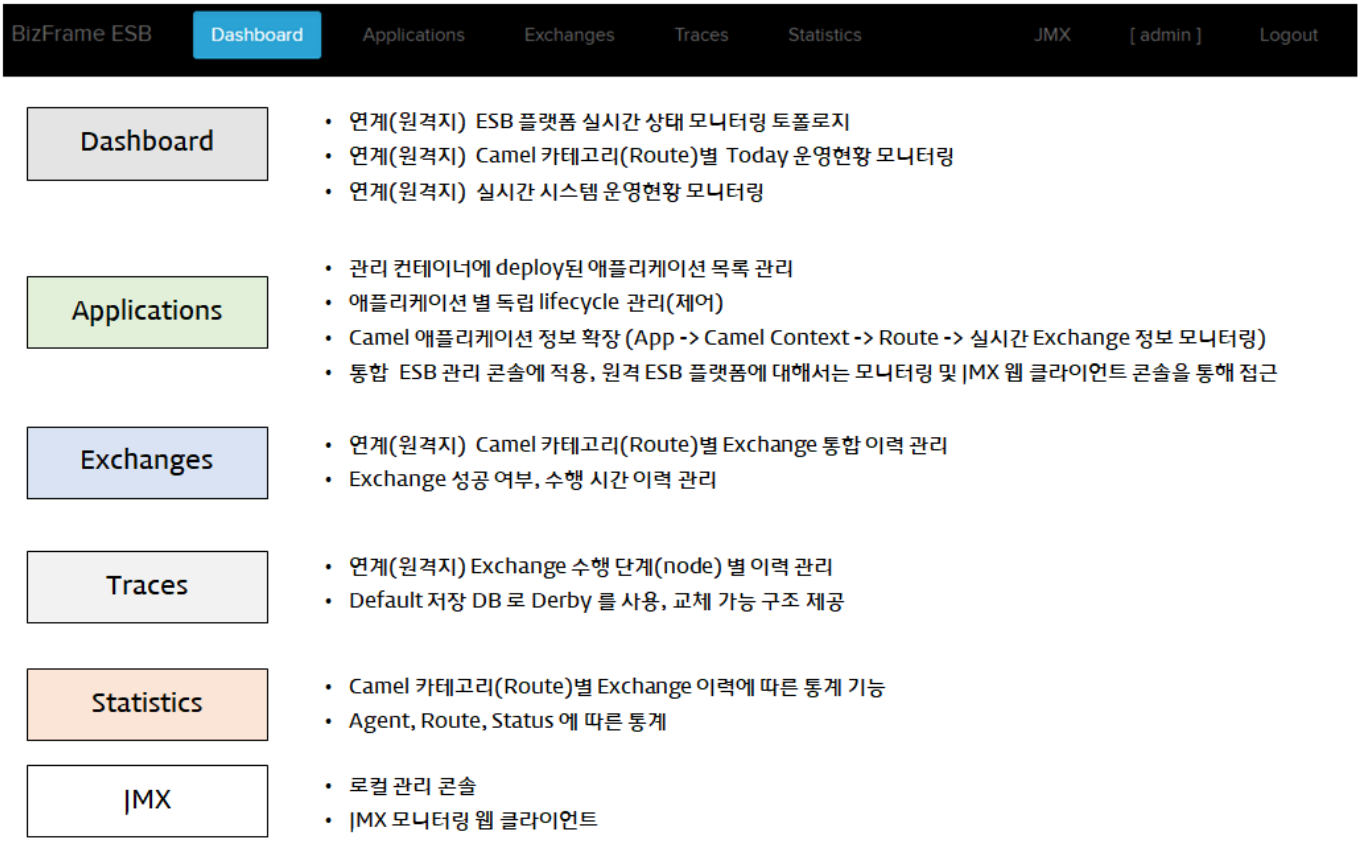
- Monitoring Agent가 수집 전송한 모니터링 데이터를 저장 관리하고
- JMX 모니터링을 통해 어플리케이션 상태 조회 및 제어
- 원격지 Camel-ESB를 통합 모니터링 할 수 있는 실시간 토폴로지

기능을 제공하는 웹 어플리케이션으로

[MAS_HOME]/applications/mng-console, derby 웹 어플리케이션 설치

관리콘솔에서 사용하는 데이터 관리를 위해 derby 어플리케이션(네트워크 설정) 동반 설치한다.

관리 메뉴는 Dashboard, Applications, Exchange, Traces, Statistics 로 구성되고 주요 기능은 아래와 같다.



2.4.1 대시보드

관리자가 구성한 연계 대상(원격지) 토폴로지에서 실시간 ESB 플랫폼의 상태(어플리케이션, 라우트)를 모니터링 함으로 전체적인 시스템 현황을 한 눈에 파악할 수 있다.
또한 연계(원격지) ESB 플랫폼의 상세 속성 및 제어 등 JMX 웹 모니터링을 통해 하나의 관리 콘솔에서 ESB 연계 통합 모니터링을 구성할 수 있다.

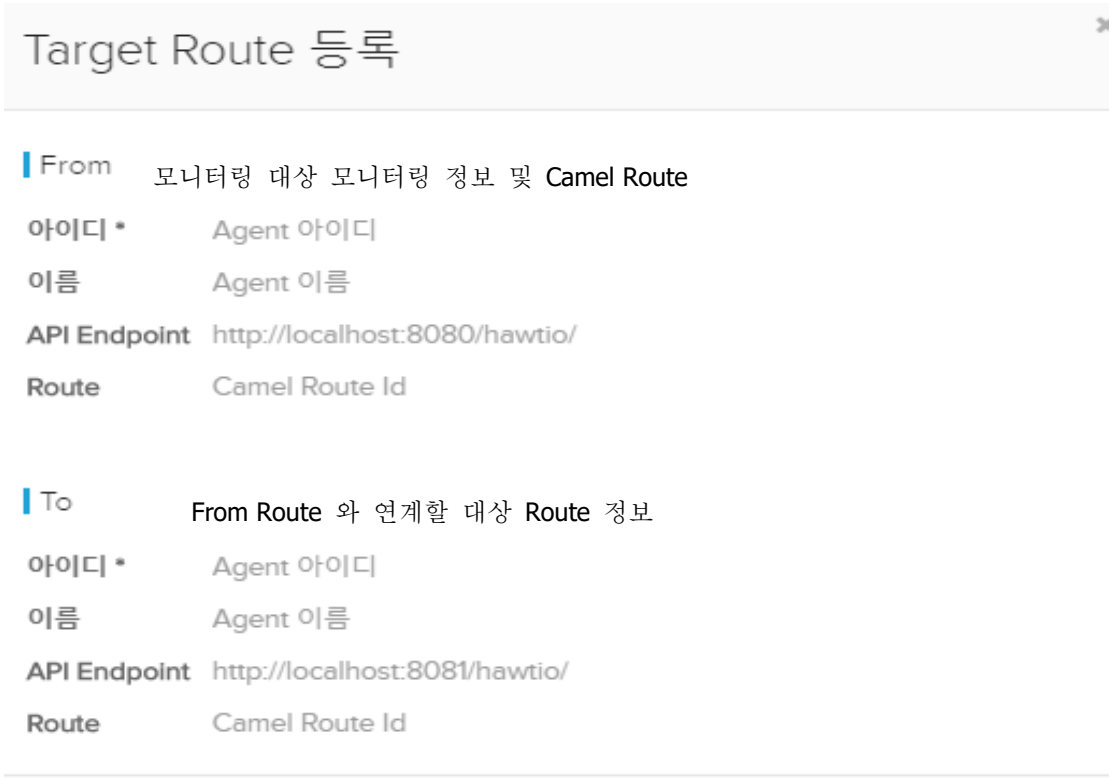
토폴로지 단위는 ESB 플랫폼의 camel 어플리케이션의 route 단위로 구성된다.

Camel 실시간 모니터링 대상은 오른쪽 상단의 

등록 버튼 클릭 후 모니터링 할 Route 정보(FROM)및 연관 Route 정보(TO)를 함께 등록하여 구성한다.
연계 관계는 FROM -> TO 방향으로 설정한다.

- 입력되는 agentId는 BizFrame-MAS, [MAS_HOME]/conf/mas-conf.xml의 server id 값으로 ESB 연계 토폴로지 구성 시 각 ESB 플랫폼의 agentId는 고유해야 한다.
- 입력되는 이름은 토폴로지 상에서 ESB 플랫폼을 나타낸다.

- 입력되는 API Endpoint는 JMX 모니터링을 위한 Hawtio-Jolokia 웹 어플리케이션 URL 로서 http://[host]:[port]/hawtio(웹 컨텍스트 이름) 해당 값을 입력해야 만 원격으로 어플리케이션, Route 상태를 모니터링 할 수 있다. (추후 API를 통해 속성 및 제어기능을 확장할 수 있다.)
- 입력되는 Route 값은 Camel 어플리케이션 Context 하위의 route id로 하나의 업무 단위로 처리한다.

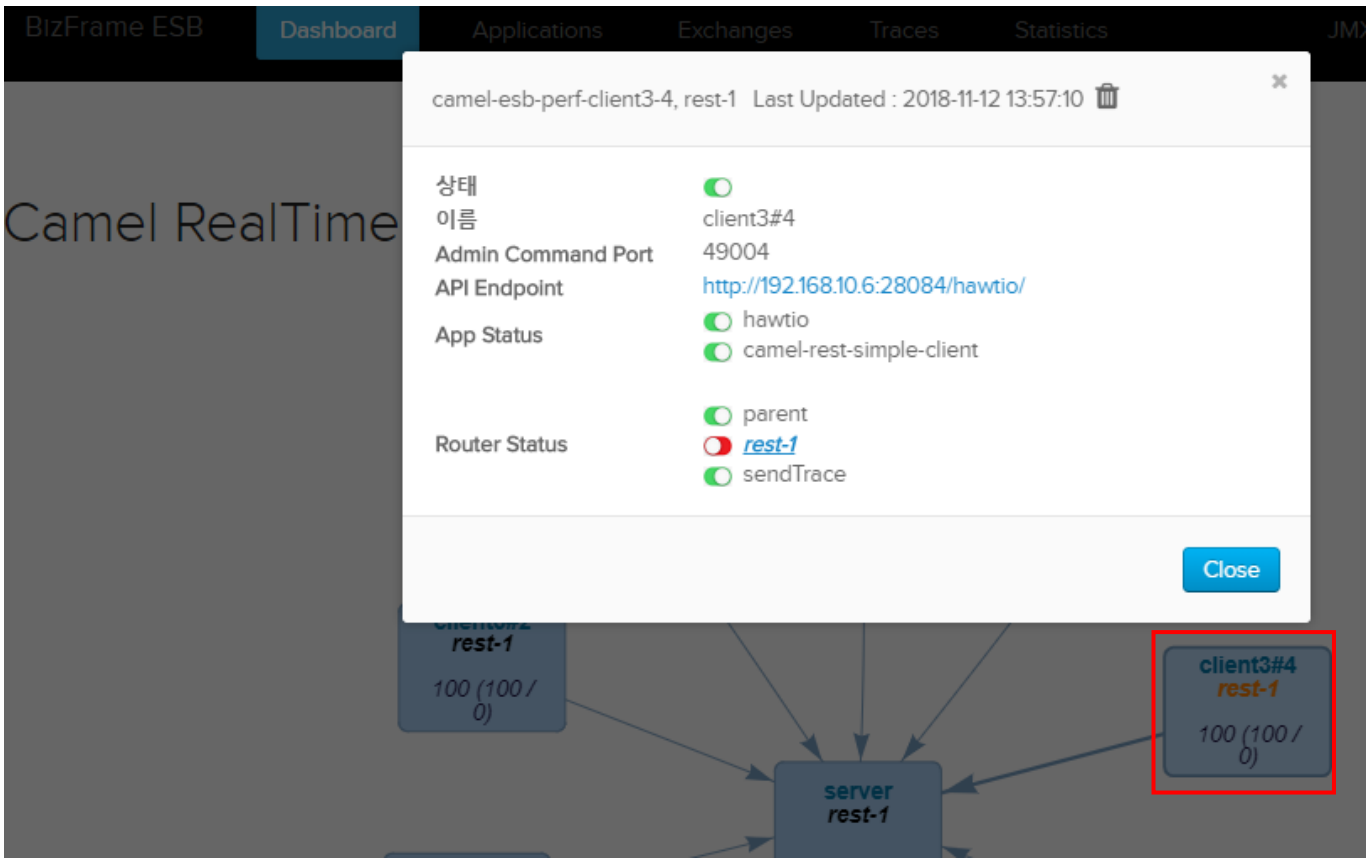


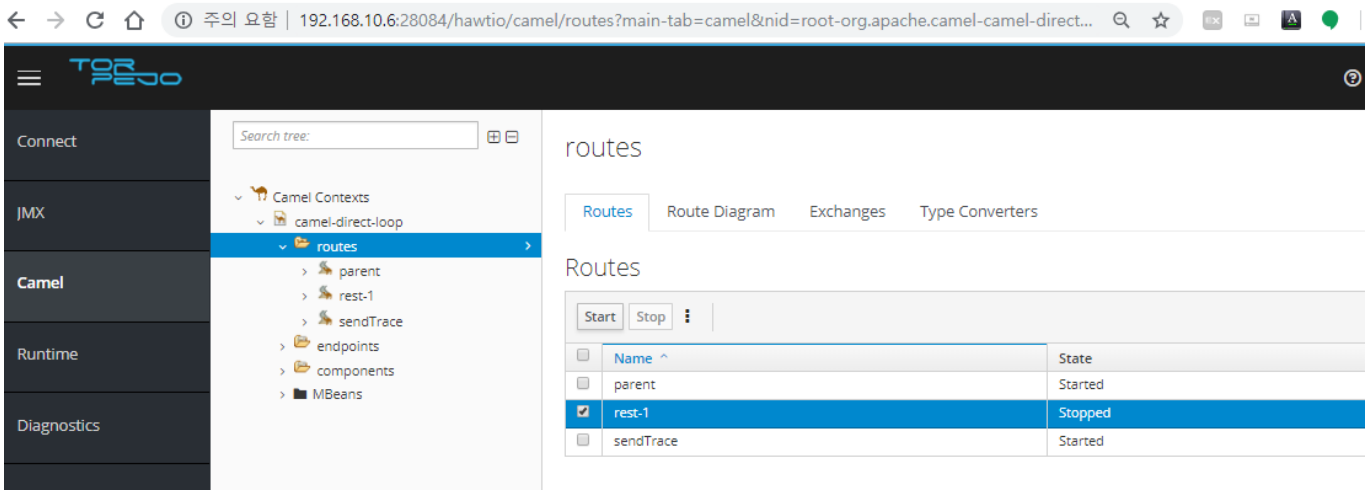
Target Route 등록을 마치면 토폴로지 상에 Route 모니터링 단위가 추가된다.
Route 단위는 아래와 같은 세부 정보를 실시간 모니터링 한다.

라우터 아이디는 색을 통해 상태를 즉시 인지할 수 있다.
에이전트는 이름으로 표시되고 이름 값이 없으면 ID로 표시된다.

정보를 클릭하여 기존에 등록된 정보를 수정 할 수 있다.
수정 시에는 agentId 와 routeId를 반드시 입력해야 한다.
또한 기 등록된 모니터링 대상 From Route에 To 정보를 입력하면 ESB 연계 관계를 추가로 등록할 수 있다.
(FROM : TO = 1: N)

추가된 Route 단위를 클릭하면 해당 ESB 플랫폼의 구성 및 상태 값, 상세 정보를 확인할 수 있다.
(단, 실시간 상태 모니터링은 API URL를 입력해야만 가능함)

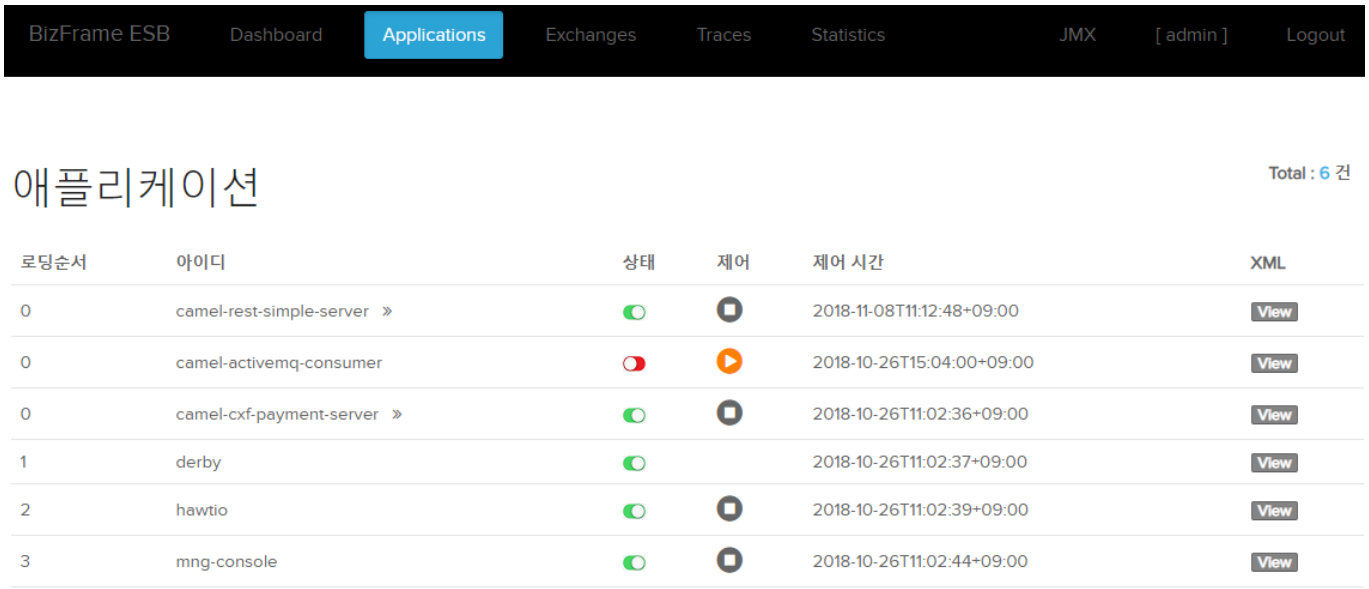




API Endpoint "http://192.168.10.6:28084/hawtio/" JMX 모니터링 웹 클라이언트에서 rest-1의 상태가 'Stopped' 임을 확인할 수 있다. 상태 조회 및 operation 도 가능하다.

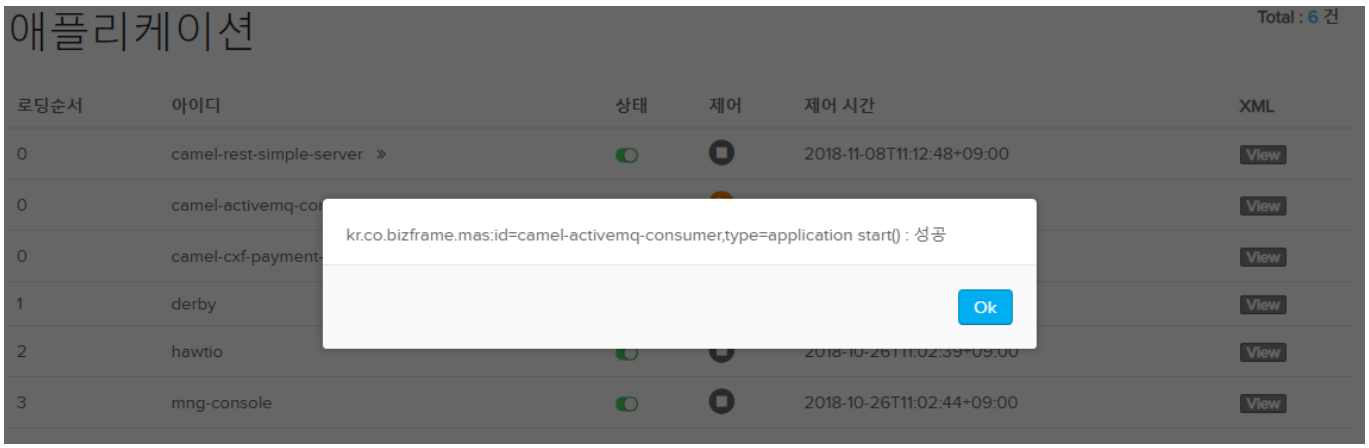
2.4.2 Applications

어플리케이션 관리 컨테이너 BizFrame-MAS에 deploy된 어플리케이션 목록 및 상태를 모니터링 할 수있다. 어플리케이션 별로 lifecycle 관리(제어) 가 가능하고 Camel 어플리케이션일 경우 Camel Tree 구조에 따른 상세 정보(App -> Camel Context -> Route -> 실시간 Exchange 정보 모니터링)를 확장하여 조회할 수 있다.

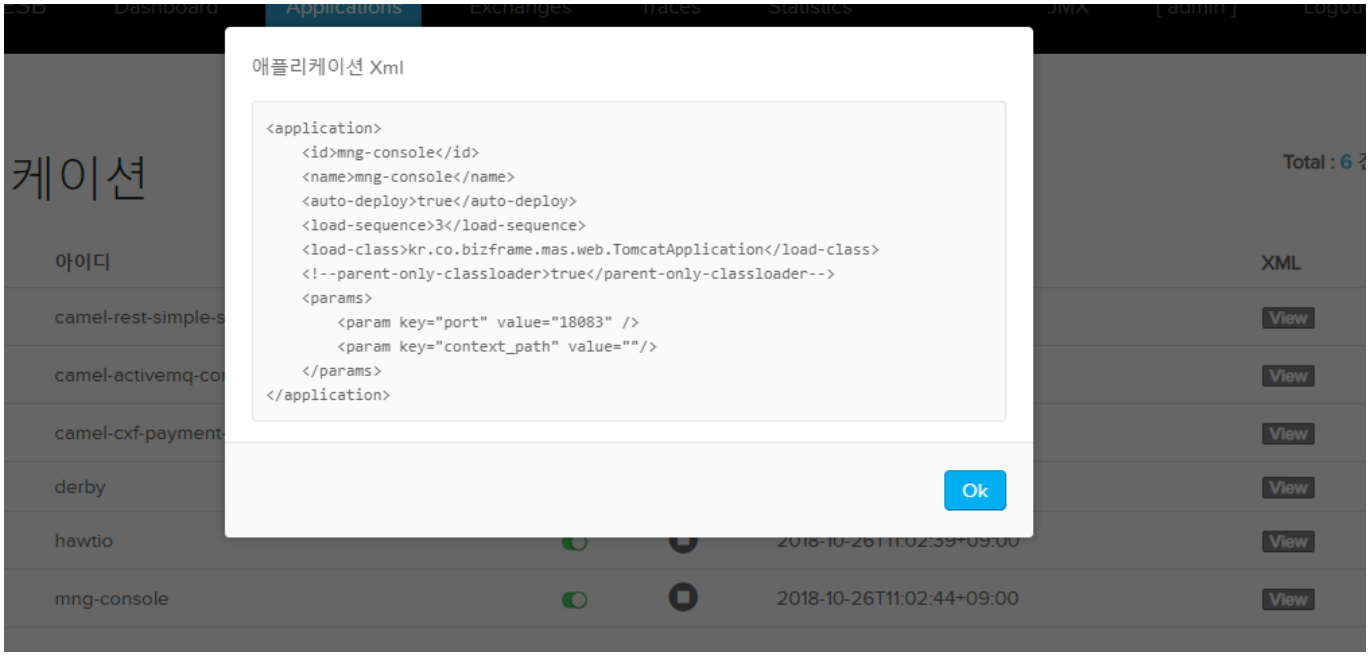


BizFrame-MAS의 애플리케이션들은 로딩 시 순서를 설정할 수 있고(EX: mng-console ESB 웹 관리 콘솔은 프로그램 초기화 시 DB 데이터 조회가 필수이므로 Derby 애플리케이션이 로딩 된 후 로딩 되어야 한다.)
설정된 로딩 순서에 따라 관리된다.
각 어플리케이션의 상태를 조회할 수 있고 어플리케이션 별로 제어가 가능하다.

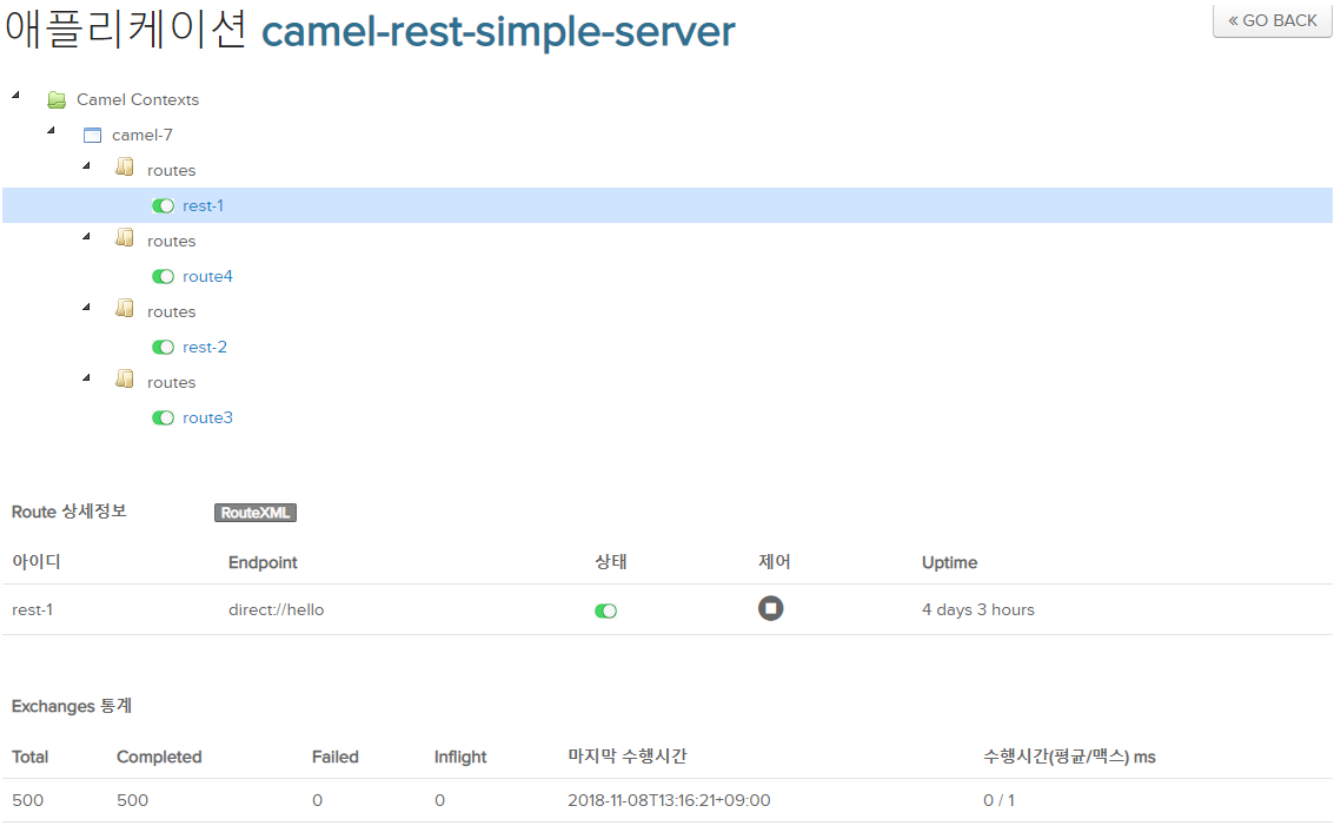
Camel 어플리케이션이 재 시작되면 Camel context는 새로 생성되고 이전 데이터는 사라진다.
이 기능을 사용하면 어플리케이션을 전체 shutdown하지 않고 해당 어플리케이션 중지 → camel-route.xml 수정 → 어플리케이션 재시작 flow에 따라 업무를 동적 수정할 수 있다. (EX. Config 조정 및 process 추가 등)



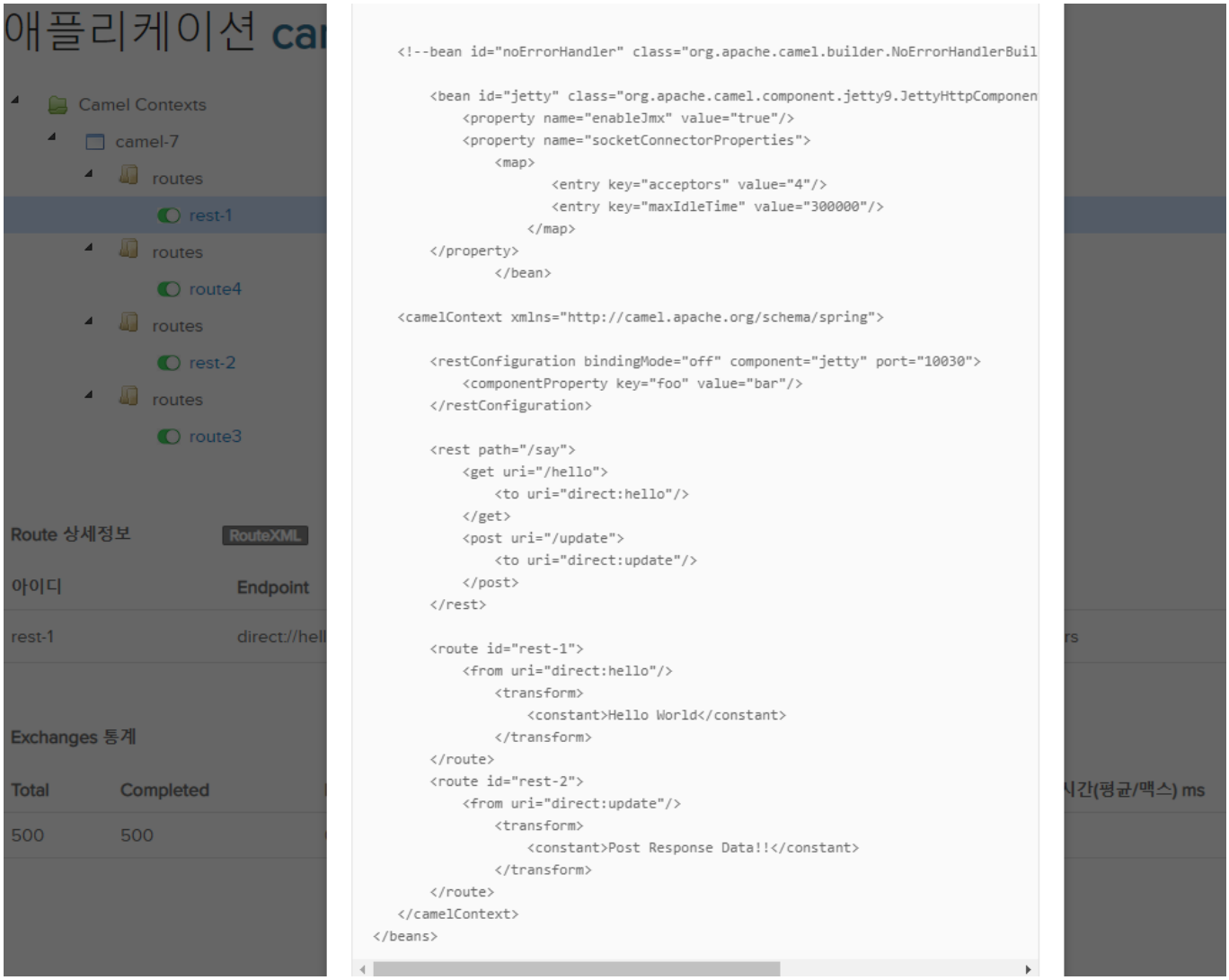
어플리케이션의 마지막 제어시간을 확인 할 수 있고 XML 버튼 클릭 시 BizFrame-MAS의 어플리케이션 설정 XML 값을 확인 할 수 있다.



어플리케이션 목록 중 '>>' 가 표시된 Camel 어플리케이션은 클릭 후 상세 정보를 확인 할 수 있다.



'RouteXML' 버튼 클릭 시 Camel 어플리케이션에서 설정한 camel-route.xml 전문을 확인할 수 있다.



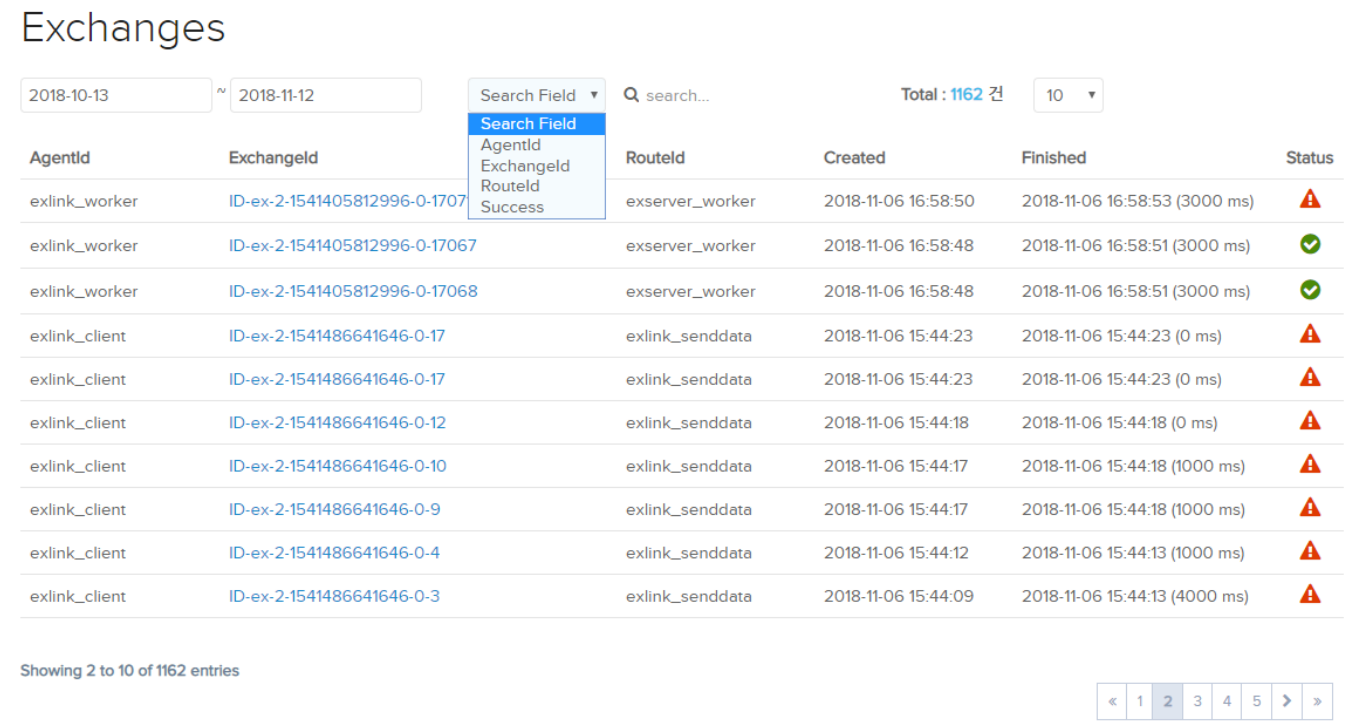
Camel-route xml 내용에 따라 Camel 어플리케이션 'camel-rest-simple-server' 는 camelContex 하위에 4개의 route로 구성되어있음을 Tree 형태로 확인 할 수 있다.
(camel-route.xml 에는 Tree구조에서는 표현 안된 Bean, Config 등 세부 정보를 확인할 수 있다.)

Camel-contexts Tree 구조에서 Route ID를 클릭하면 라우트 별 세부 정보를 확인할 수 있고
상태 제어 또한 가능하다.
Route를 중지시킬 경우 서비스는 잠시 중지되므로 서비스 운영 중 어플리케이션을 종료 없이 동적으로 서비스를 운영할 수 있다.
서비스 가동 후 Route Exchange 통계를 확인할 수 있다. 해당 정보는 서비스가 재가동(Camel Context restart) 되면 Reset 된다.

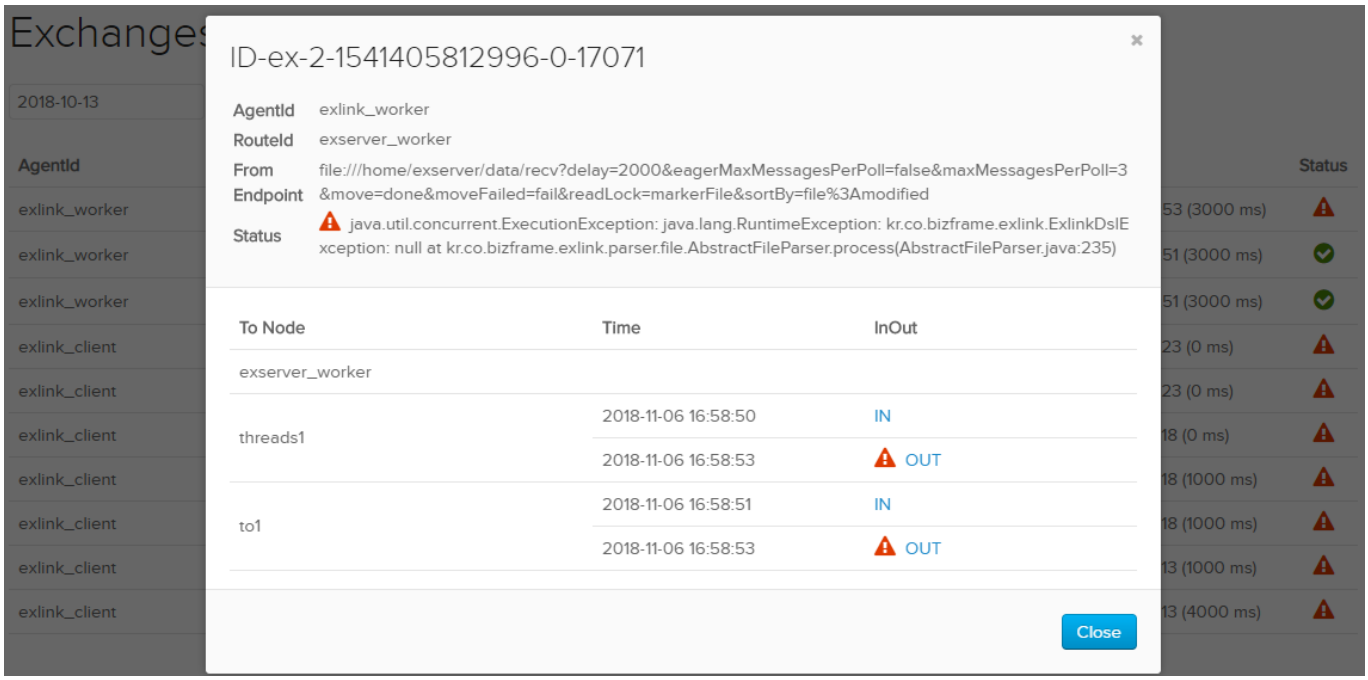
2.4.3 Exchanges

ESB 플랫폼에 deploy된 Camel 어플리케이션의 Route 수행 시 발생하는 Exchange 정보를 수집 관리한다.
연계(원격지) ESB 플랫폼의 Exchange정보를 통합 관리하여 연계흐름을 모니터링 할 수 있다.

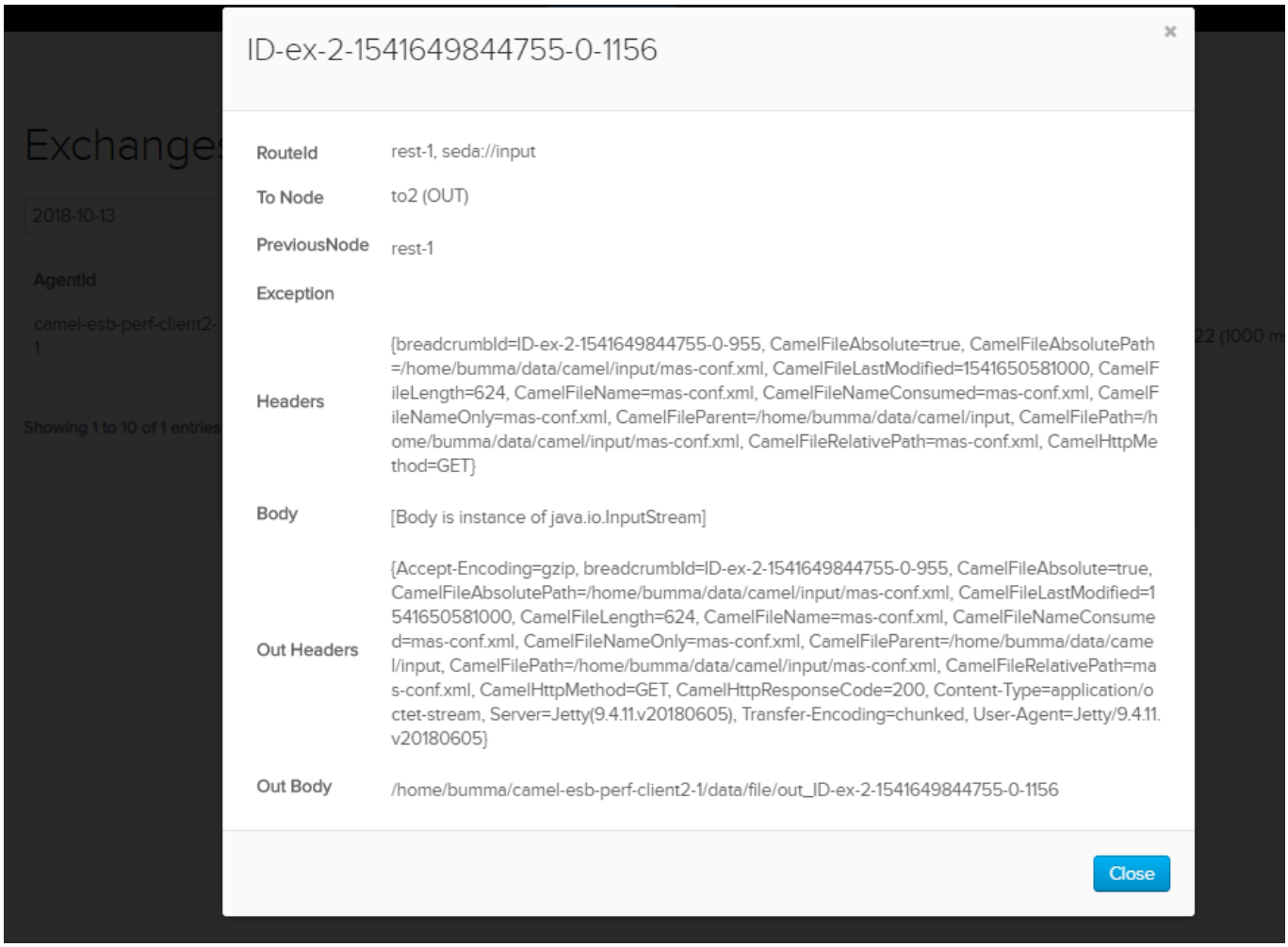
- Exchange 정보는 성공/실패 상태를 가지고 처리 단계 별 Trace 정보와 연결될 수 있다.
- 날짜, AgentId, RouteId, ExchangeId, 상태의 검색 조건을 가진다.



ExchangeId를 클릭하면 상세 정보를 확인 할 수 있다.



단계 별 InOut을 클릭하면 Camel 메시지를 조회할 수 있다.



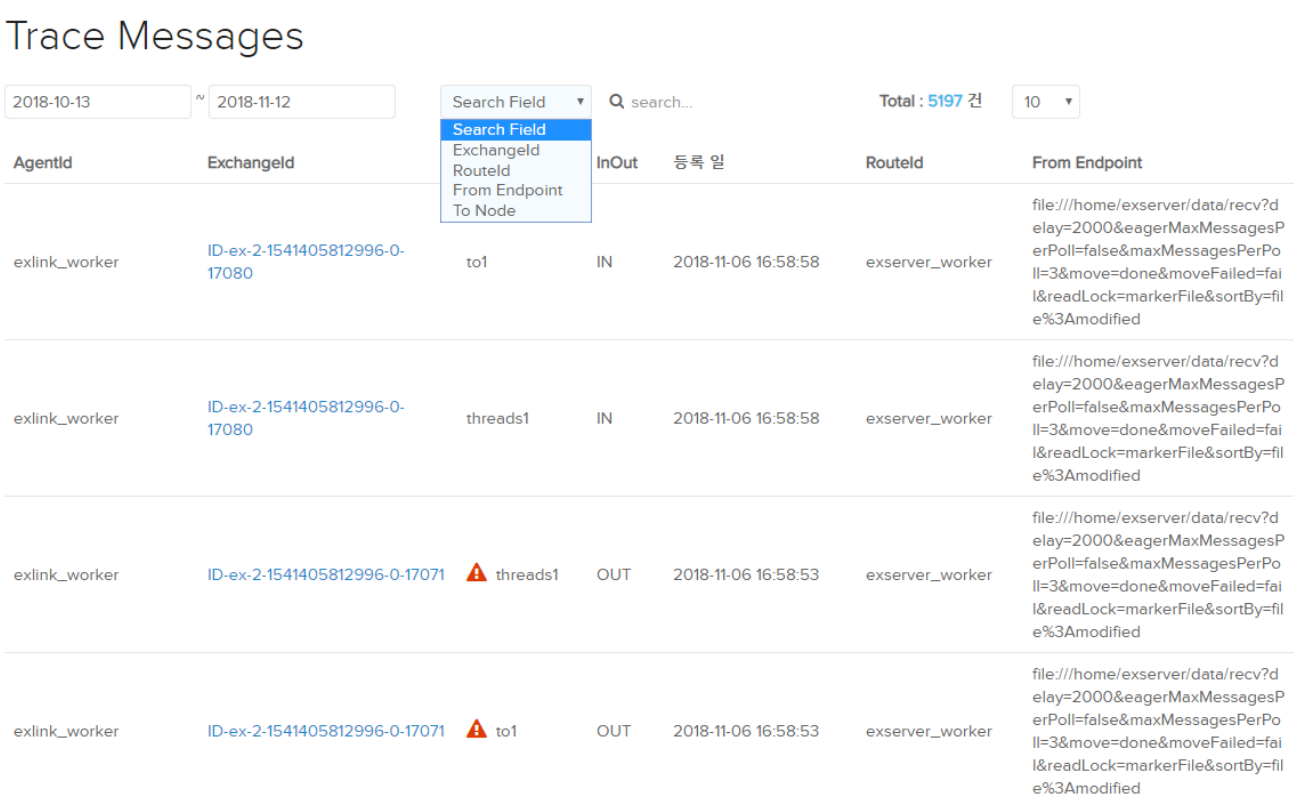
Camel Route가 수행되면 ExchangeId에 따른 Exchange가 발생하고 내부 처리 단계에 따라 해당 ExchangeId를 가진 Camel Message가 생성된다. 각 단계 별 Camel Message는 Trace되어 관리된다.
Exchange 메뉴에서는 Route 수행 흐름에 따른 UI를 제공하여 Exchange → 단계 별 Message 흐름정보 → Message 정보를 한눈에 확인 할 수 있다

.

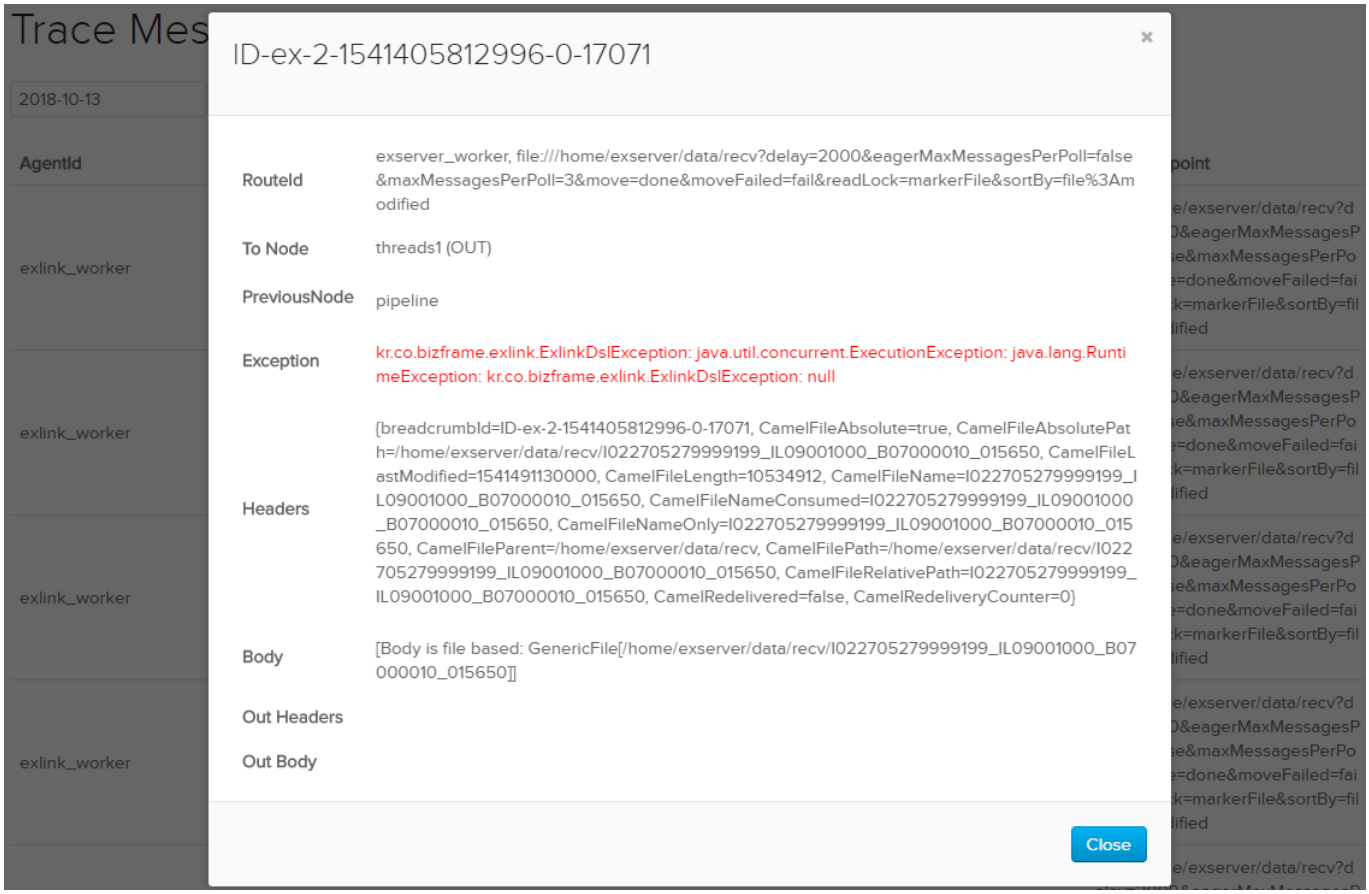
2.4.4 Traces

Camel Route 단계별 Trace 정보를 수집 관리한다.
연계(원격지) ESB 플랫폼의 Trace 정보를 통합 관리하여 연계흐름을 모니터링 할 수 있다.

- Trace 정보는 실패 시 Exception 정보를 상위 ExchangeId를 가진다.
- 날짜, RouteId, ExchangeId, From Endpoint, ToNode 상태의 검색 조건을 가진다.



ExchangeId를 클릭 시 Camel 메시지 정보를 확인할 수 있다.



Trace 정보는 Camel body 포맷이 Stream 일 경우에만 file로 저장하여 관리한다.
해당 정책은 monitoring-agent에서 수정 가능한다.

2.4.5 Statistic

연계(원격지) ESB 플랫폼에서 수집한 Exchange 통계 정보를 관리한다.
날짜별 AgentId, RouteId, Status 값에 따른 통계 집계 기능을 제공한다.