Indigo-dynamic 개발 가이드 (공개본)

Indigo-Dynamic-Adapter Reference Book

2024-12-20

Contents

1. Indigo-Dynamic-Adapter

주제

◎ Indigo-Dynamic-Adapter 모듈 소개
◎ Indigo-Dynamic-Adapter 모듈 구성
◎ Service 와 Service Chaining
◎ Service-Context
◎ Service-Context 의 생명주기
◎ InterfaceInfo
◎ Error-Handler
◎ Callback
◎ Storage / Provider

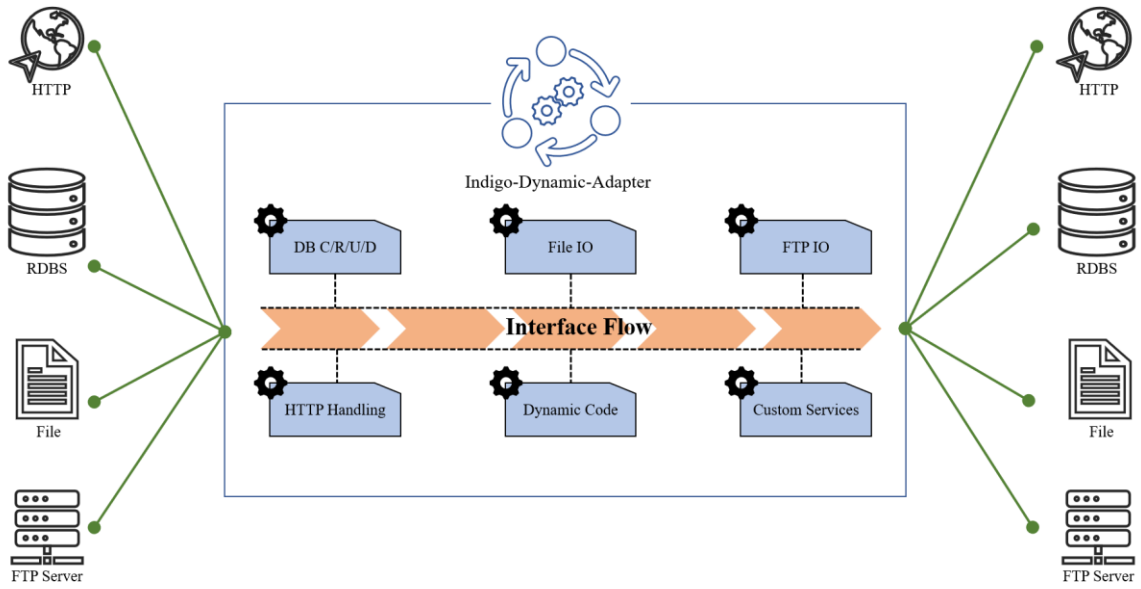
◎ Indigo-Dynamic-Adapter 모듈 소개

Indigo-Dynamic-Adapter 는 INDIGO ESB 솔루션을 기반으로 작동하는 데이터 연계 모듈입니다. 이 모듈은 DB, FTP, 파일, HTTP 등 다양한 프로토콜 간 데이터 연계 프로세스를 지원하며, Spring Bean XML 문법을 활용하여 컴포넌트 조합 방식으로 손쉽게 데이터 연계 프로세스를 개발할 수 있습니다.

Indigo-Dynamic-Adapter 는 유연성, 확장성, 그리고 유지보수의 편리성을 중점으로 하여 설계되었습니다. 인터페이스 업무 담당자는 복잡한 데이터 연계 작업을 간단하게 정의하고, 필요에 따라 프로세스를 확장 및 수정할 수 있습니다.

특히 Dynamic-Code 기능은 Indigo-Dynamic-Adapter 의 가장 핵심적인 기능 중 하나입니다. 이 기능을 통해 웹 콘솔에서 직접 Java 코드를 작성하고 코드를 수동으로 컴파일/빌드/서버 업로드 하는 과정 없이 운영 환경에 바로 배포할 수 있습니다.

이는 개발과 운영 과정을 효율화하고, 다양한 인터페이스 요구 사항에 신속히 대응할 수 있도록 지원합니다.



◎ Indigo-Dynamic-Adapter 모듈 구성

Indigo-Dynamic-Adapter 는 다음과 같은 주요 구성 요소로 이루어져 있습니다.

1. Dispatcher

인터페이스 이벤트를 수신하고, 해당 인터페이스 ID 와 매핑된 프로세스를 호출하는 역할을 수행합니다.

2. Service-Processor

프로세스를 실제로 수행하는 핵심 모듈로, 다음과 같은 기능을 제공합니다.

△Service-Chaining: 사전에 설정된 Service 컴포넌트를 순차적으로 실행하는 과정입니다.

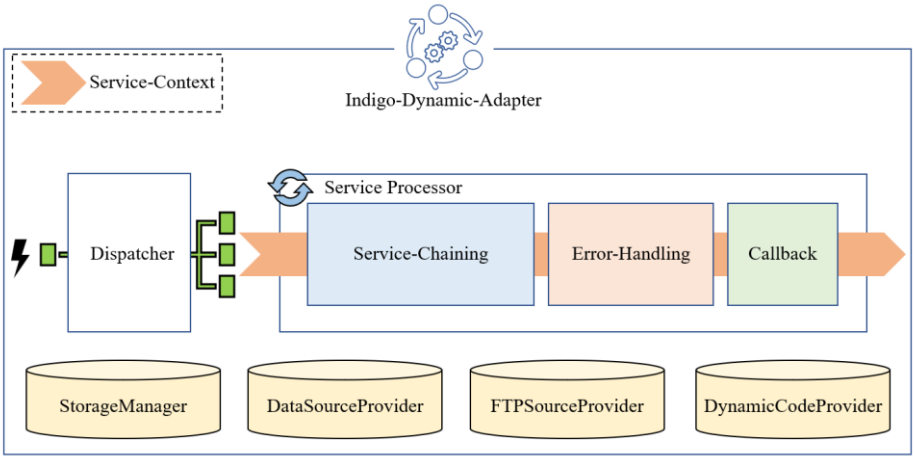
△Error-Handling: 프로세스 수행 중 발생한 에러를 처리하는 과정입니다.

△Callback: 반드시 실행되어야 할 작업으로, Service-Chaining 과 Error-Handling 이 종료된 이후에 설정된 Callback 들을 호출합니다.

△Service-Context: 매 인터페이스 이벤트 발생 시 마다 Dispatcher 에서 새롭게 생성되어 Service-Processor, Error-Handler, Callback 에서 수행되는 과정을 관통하는 객체입니다. Service-Context 는 인터페이스 이벤트 발생 시점부터 종료 시점까지 유지되며, 각 단계의 데이터와 상태를 일관되게 관리하는 중요한 역할을 수행합니다.

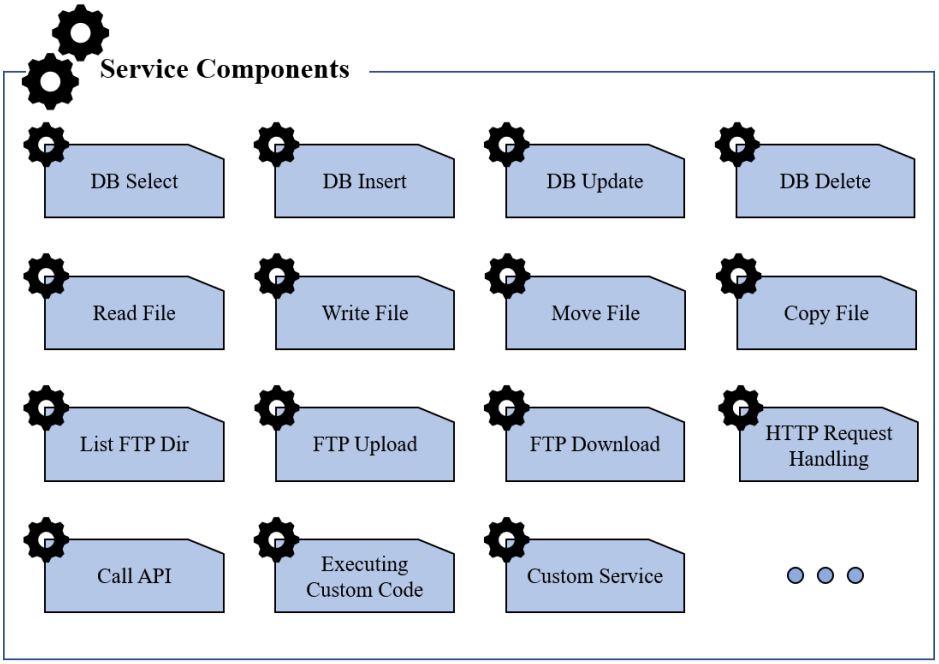
3. 설정 정보

Dispatcher 와 Service-Processor 가 사용하는 인터페이스 정보, 서비스 프로세스 정보, 에러 핸들러, 콜백, FTP Server 또는 DB 접속정보 등의 설정 정보는 어댑터 애플리케이션이 실행되는 동안 static 하게 유지되며 이러한 설정 정보는 Storage 와 Provider 를 통해 접근 및 관리됩니다.

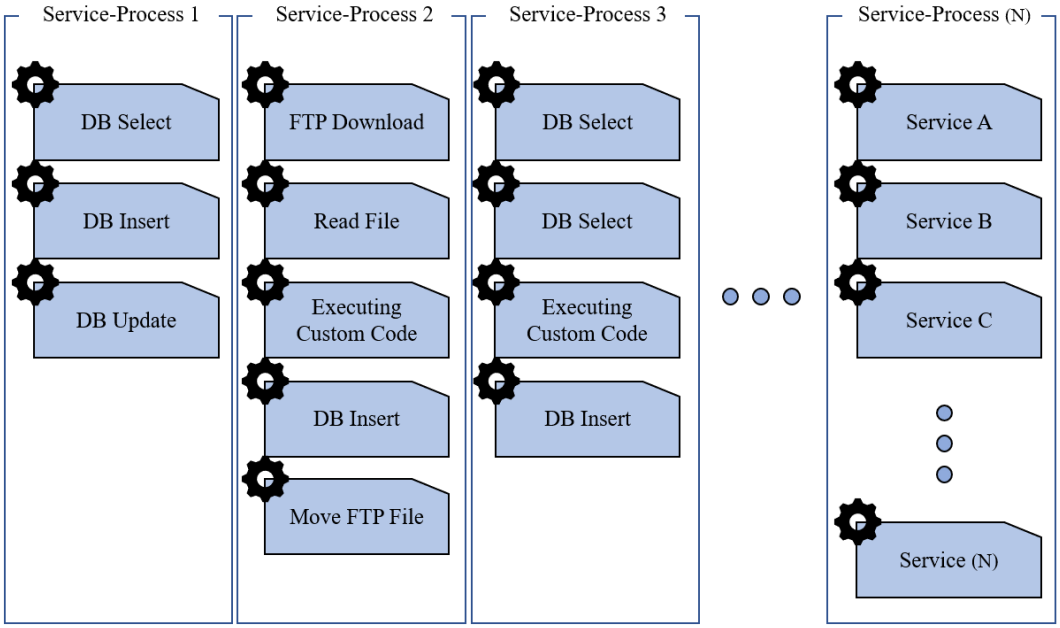


◎ Service 와 Service Chaining

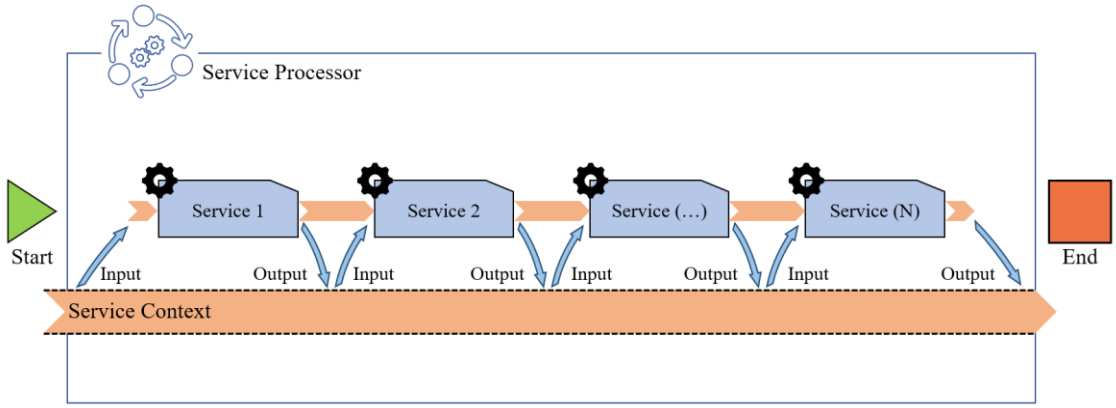
Service 는 고유한 기능을 수행하는 최소 기능 단위입니다. 예를 들어, DB 작업과 관련된 Service 는 쿼리의 종류에 따라 Select, Insert, Update, Delete 등이 있으며, 파일 입출력(File IO)과 관련된 Service 는 ReadFile, WriteFile 등이 있습니다.



이러한 Service 를 특정 작업 순서에 맞게 조합하여 순차적으로 실행하도록 구성한 것을 Services, Service-Strategy, 또는 Service-Process 라고 합니다.



Service-Strategy 의 Services 를 순차적으로 실행하는 과정을 Service-Chaining 이라고 하며, Service-Chaining 을 실행하는 주체는 Service-Processor 입니다.



◎ Service-Context

앞서 말한 Service-Chaining 은 여러 개의 Service 를 특정 순서에 따라 조합하여 실행하는 과정입니다. 각각의 Service 는 특정 기능(예: 데이터베이스 작업, 파일 처리 등)을 수행하는 최소 단위의 컴포넌트로, Service 자신이 수행하는 작업에만 집중합니다.

따라서 개별 Service 는 어떤 데이터가 처리되어야 하는지 또는 프로세스의 전체 맥락에 대해 알지 못하기 때문에 각 Service 에서 사용될 데이터를 전달하고 일관성을 유지하기 위해 Service Context 객체가 사용됩니다.

ServiceContext 는 Service-Chaining 과정에서 Input 과 Output 데이터를 관리하고, 데이터 흐름과 상태를 유지하는 중요한 객체입니다. 주요 기능은 다음과 같습니다:

△데이터 저장 및 전달

각 Service 는 기능 수행을 위해 ServiceContext 로부터 Input 데이터를 받습니다. Service 가 작업을 완료하면, 결과 데이터는 설정된 명칭으로 Output 됩니다. 이러한 Input 및 Output 데이터가 ServiceContext 에 저장되며, 다음 Service 로 데이터를 전달합니다.

△프로세스의 일관성 유지

ServiceContext 는 Service-Chaining 시작 전 Dispatcher 에서 생성되어 전체 프로세스가 완료될 때까지 동일한 객체가 사용됩니다. 이를 통해 Service 간 데이터 교환이 일관되게 이루어집니다.

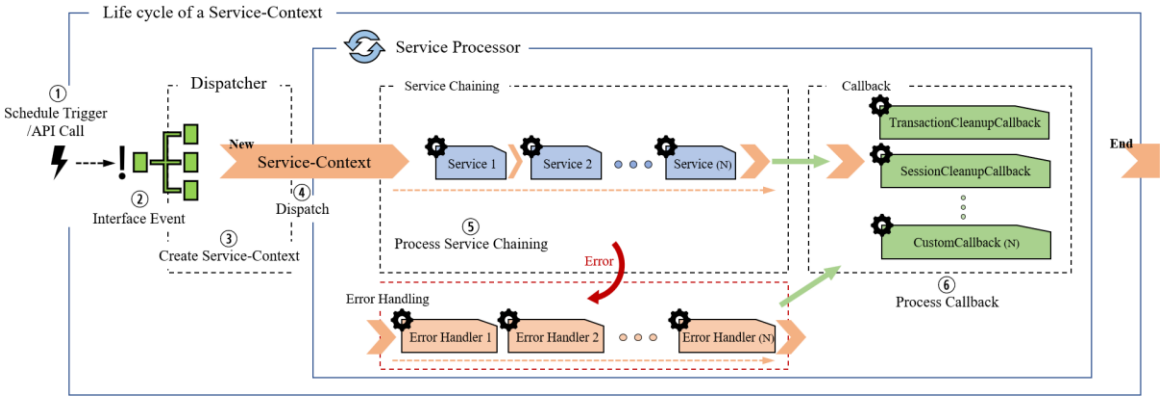
△상태 정보 관리

각 단계에서 발생하는 상태(성공/실패, 서비스, 로그, 시작/종료시간 등)을 저장하고 관리합니다. Service-Processor 와 Error-Handler, Callback 등에서는 이를 활용해 프로세스를 처리합니다.

◎ Service-Context 의 생명주기

Dispatcher 가 인터페이스 이벤트를 수신하면, 새로운 ServiceContext 객체가 생성되고, 초기 데이터(인터페이스 ID, 시작시간 등)가 ServiceContext 에 설정됩니다. 그 다음, Service-Processor 에 ServiceContext 가 전달됩니다.

Service-Processor 는 전달받은 ServiceContext 로부터 인터페이스에 대한 정보를 가져와 그 인터페이스에서 사용될 Service-Strategy 를 찾아 Service-Chaining 을 실행합니다. Service-Chaining 과정에서 실행되는 각 Service 는 전달받은 Input 데이터를 기반으로 작업을 수행하고, 결과로 Output 데이터를 ServiceContext 에 저장합니다. ServiceContext 는 Service-Chaining 과 Error-Handling(에러가 발생하는 경우) Callback 과정을 관통하여 사용된 후 최종적으로 최초에 생성되었던 위치인 Dispatcher 에서 소멸됩니다.



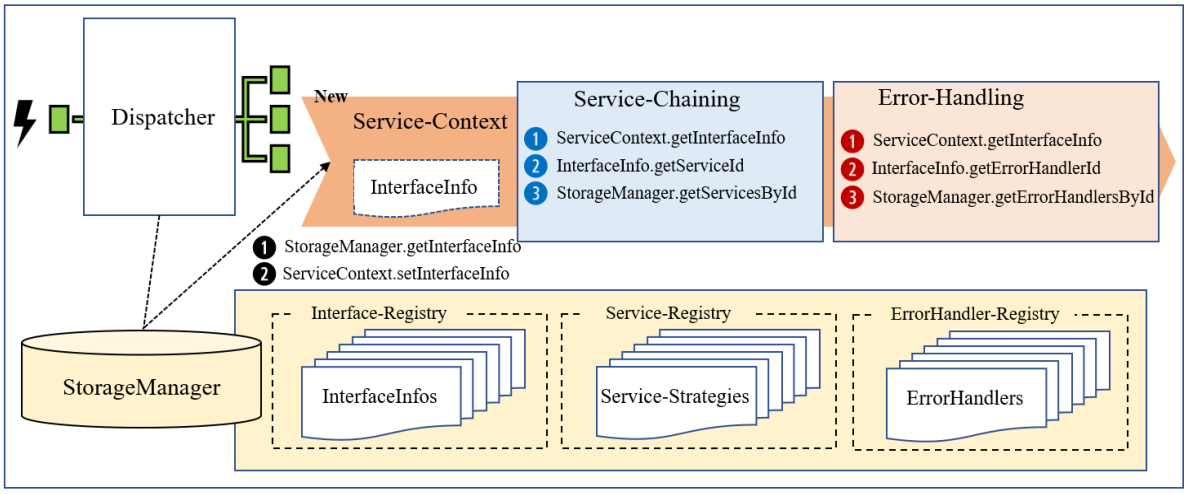
◎ InterfaceInfo

InterfaceInfo 객체는 Indigo-Dynamic-Adapter 에서 Service-Chaining 프로세스를 수행할 때, 해당 프로세스에 대한 핵심 정보를 저장하고 있는 객체입니다. InterfaceInfo 는 하나의 개별 인터페이스와 그 인터페이스가 처리 되어야하는 Service-Strategy 및 Error-Handler 에 대한 정보를 담고 있는 메타데이터의 역할을 하며, Dispatcher 가 ServiceContext 를 생성할 때 참고하는 중요한 정보입니다.

△InterfaceInfo 의 역할

1) 프로세스 정의

InterfaceInfo 는 ServiceProcessor 가 어떤 인터페이스 프로세스를 수행해야 하는지 정의합니다. Dispatcher 가 인터페이스 이벤트 발생 시 전달받은 파라미터 또는 URL 을 사용하여 StorageManager 에서 InterfaceInfo 를 가져와 ServiceContext 에 설정을 하면, ServiceProcessor 는 해당 InterfaceInfo 에 설정된 serviced 를 통해 StorageManager 에서 Service-Strategy 를 검색해 Service-Chaining 을 진행합니다. 또 프로세스 도중 에러가 발생하는 경우에도 InterfaceInfo 에 설정된 errorhandleId 를 통해 StorateManager 에서 적합한 ErrorHandler 를 찾아 에러 처리 프로세스를 진행합니다



2) 데이터 연계 설정 관리

InterfaceInfo 는 다음과 같은 데이터 연계 설정 정보를 포함합니다.

속성	설명	필수	기본값	예시
activated	인터페이스의 활성화 여부		true	true
controllerInterface	해당 인터페이스가 다른 인터페이스를 제어하는 컨트롤 인터페이스인지 여부		false	false
interfaceId	한 어댑터 내에서 인터페이스를 식별하는 고유한 ID	Y		IFD2F_SRC_TGT_001
interfaceName	인터페이스의 이름			DB to FTP 연계 인터페이스
description	인터페이스에 대한 설명			OOO 업무를 위한 데이터연계 시 사용됨
sourceCode	인터페이스의 소스 코드			SRC
targetCode	인터페이스의 타겟 코드			TGT
serviceId	인터페이스가 사용하는 Service-Strategy 의 ID	Y		DB2DB_Sel_Ind_Upd
errorHandlerId	인터페이스 프로세스 도중 오류 발생 시 실행할 Error-Handler 의 ID			EH_DB2DB
querySequence	DB 작업을 위한 쿼리 시퀀스(실행되는 순서대로 입력)			DBSRC${if_id}.SELECT, DBTGT${if_id}.INSERT, DBSRC${if_id}.UPDATE
errorQuerySequence	에러가 발생하는 경우 DB 작업을 위한 쿼리 시퀀스			DBSRC${if_id}.ERROR_UPDATE
txTimeoutSecond	트랜잭션 타임아웃 설정(초)		-1	300
fileTemplates	파일 연계 관련 설정을 위한 템플릿			FileTemplate 설정을 참고
sourceAliases	FileTemplate, FTP 서버, DB Source 등을 참조하기 위한 Alias 매핑 정보			SRC:FRP_SERVER1, TGT:FTP_SERVER2
dynamicCodeSequence	동적 코드 실행을 위한 시퀀스			@{if_id}.MAP_DATA, @{if_id}.CALL_PROCESS
errorDynamicCodeSequence	에러가 발생하는 경우 동적 코드 실행을 위한 시퀀스			COMMON.FILTER_ERROR
loggingWhenNormal	정상 작동 시 로그를 남길지 여부		true	false
loggingWhenError	오류 발생 시 로그를 남길지 여부		true	true
frontHttpUrl	HTTP 기반 인터페이스로 사용되는 경우의 URL			/post/v1/send-data
frontHttpMethod	HTTP 기반 인터페이스로 사용되는 경우의 HTTP Method			POST

△FileTemplate

FileTemplate 은 FTP 나 File 연계 시 사용되는 설정정보가 저장되는 객체입니다. 파일을 어떤 디렉터리에서 작업할 지, 작업 대상 파일명이나 파일명의 패턴은 어떻게 되는지, 작업 대상 파일이 디렉터리인지, 파일인지 등의 설정을 할 수 있습니다. FileTemplate 설정된 정보들은 File 이나 FTP 관련 Service 에서 참조되어 사용됩니다.

속성	설명	기본값	비고
localSendDir	DirectoryType.LOCAL_SEND 과 매핑		작업 대상 파일 경로에 대한 설정이며 Fiile 또는 FTP 관련 Service의 directoryType 설정에서 참조된다.
localReceiveDir	DirectoryType.LOCAL_RECEIVE 과 매핑		//
localTempDir	DirectoryType.LOCAL_TEMP 과 매핑		//
localSuccessDir	DirectoryType.LOCAL_SUCCESS 과 매핑		//
localErrorDir	DirectoryType.LOCAL_ERROR 과 매핑		//
localBackupDir	DirectoryType.LOCAL_BACKUP 과 매핑		//
localMoveDir	DirectoryType.LOCAL_MOVE 과 매핑		//
localCopyDir	DirectoryType.LOCAL_COPY 과 매핑		//
localWriteDir	DirectoryType.LOCAL_WRITE 과 매핑		//
remoteSendDir	DirectoryType.REMOTE_SEND 과 매핑		//
remoteReceiveDir	DirectoryType.REMOTE_RECEIVE 과 매핑		//
remoteTempDir	DirectoryType.REMOTE_TEMP 과 매핑		//
remoteSuccessDir	DirectoryType.REMOTE_SUCCESS 과 매핑		//
remoteErrorDir	DirectoryType.REMOTE_ERROR 과 매핑		//
remoteBackupDir	DirectoryType.REMOTE_BACKUP 과 매핑		//
remoteMoveDir	DirectoryType.REMOTE_MOVE 과 매핑		//
remoteCopyDir	DirectoryType.REMOTE_COPY 과 매핑		//
remoteWriteDir	DirectoryType.REMOTE_WRITE 과 매핑		//
type	작업 대상 파일의 타입	ALL	ALL: 디렉터리 및 파일 / DIRECTORY: 디렉터리 / FILE: 파일
charset	파일의 인코딩	UTF-8	

◎ Error-Handler

Service-Chaining 도중 특정 Service 에서 에러가 발생하면, Service-Chaining 은 즉시 중단됩니다. 이때 Service-Processor 에서 에러가 처리되는 과정은 다음과 같습니다.

1) ServiceContext 에 등록된 InterfaceInfo 객체에서 errorHandlerId 속성을 확인합니다. errorHandlerId 는 해당 인터페이스에 대한 에러 처리를 담당할 Error-Handler 의 ID 를 나타냅니다.

2) StorageManager 의 Error-Handler-Registry 에서 errorHandlerId 에 매핑된 Error-Handler 를 검색합니다. Error-Handler 가 등록되어 있으면, 에러 처리 프로세스를 실행합니다.

매핑된 Error-Handler 는 에러 처리 전략을 통해 프로세스 중 발생한 에러를 관리하며 다음과 같은 에러 처리 전략을 등록할 수 있습니다.

△Rollback: 실패한 작업의 상태를 복구하거나 보정.(DB 롤백 또는 파일 위치 원상복구)
△알림: 관리자에게 에러를 알리는 작업 수행.
△대체 경로: 실패한 작업을 우회하거나 다른 서비스로 대체 실행.

◎ Callback

Callback 은 Service-Chaining 의 성공 실패 여부와 관계없이 항상 실행되는 프로세스를 의미한다. 기본적으로 DB 작업 후 종료되지 않은 트랜잭션을 종료해주는 Callback 과 FTP 서버 작업 후 해제되지 않은 세션을 종료해주는 Callback 이 등록되어 있습니다. 추가적으로 직접 구현한 Callback 클래스를 등록하여 프로세스 수행이력을 항상 남겨주거나 에러 발생 시 알림 전송 등의 프로세스를 구현할 수도 있습니다.

◎ Storage / Provider

DB 접속 정보, FTP Server 접속 정보, InterfaceInfo 설정, Service-Strategy 설정, Error-Handler 설정 등과 같은 static 한 정보는 Storage 와 Provider 역할을 하는 객체에 등록되고 해당 정보에 대한 접근 시에도 이 객체를 통해 접근이 가능하다.

△StorageManager

StorageManager 는 InterfaceInfo 와 Service-Strategies, Error-Handlers 를 저장하며 이에 대한 접근을 관리한다.

△DataSourceProvider

DataSourceProvider 는 DB 에 접속 및 쿼리작업을 대행하는 객체인 QueryExecutor 를 저장하며 관리한다.

△FTPSourceProvider

FTPSourceProvider 는 FTP Server 와 관련된 접속 및 작업을 대행하는 객체인 FTPClientTemplate 을 저장 및 관리한다.

△DynamicCodeProvider

DynamicCodeProvider 는 동적으로 실행가능한 Java 코드인 DynamicCode 의 저장/컴파일/접근을 관리한다.

2. Indigo-Dynamic-Adapter 개발 환경 구성

주제

◎ 프로젝트 및 어댑터 생성
◎ 빈설정
◎ 주요 설정파일 등록 [mybatis-configuration.xml]
◎ 주요 설정파일 등록 [config_CORE.xml]
◎ 주요 설정파일 등록 [config_INTERFACES.xml]
◎ 주요 설정파일 등록 [config_SERVICES.xml]
◎ 주요 설정파일 등록 [DNC_COMMON.dnc]
◎ DB Source 등록
◎ FTP Source 등록

Indigo-Dynamic-Adapter 를 사용하여 인터페이스를 개발하기 전 개발환경을 구성하는 절차를 설명합니다.
절차는 다음과 같습니다.

1) 프로젝트 및 어댑터 생성

2) 주요 설정파일 등록

3) 주요 컴포넌트 등록

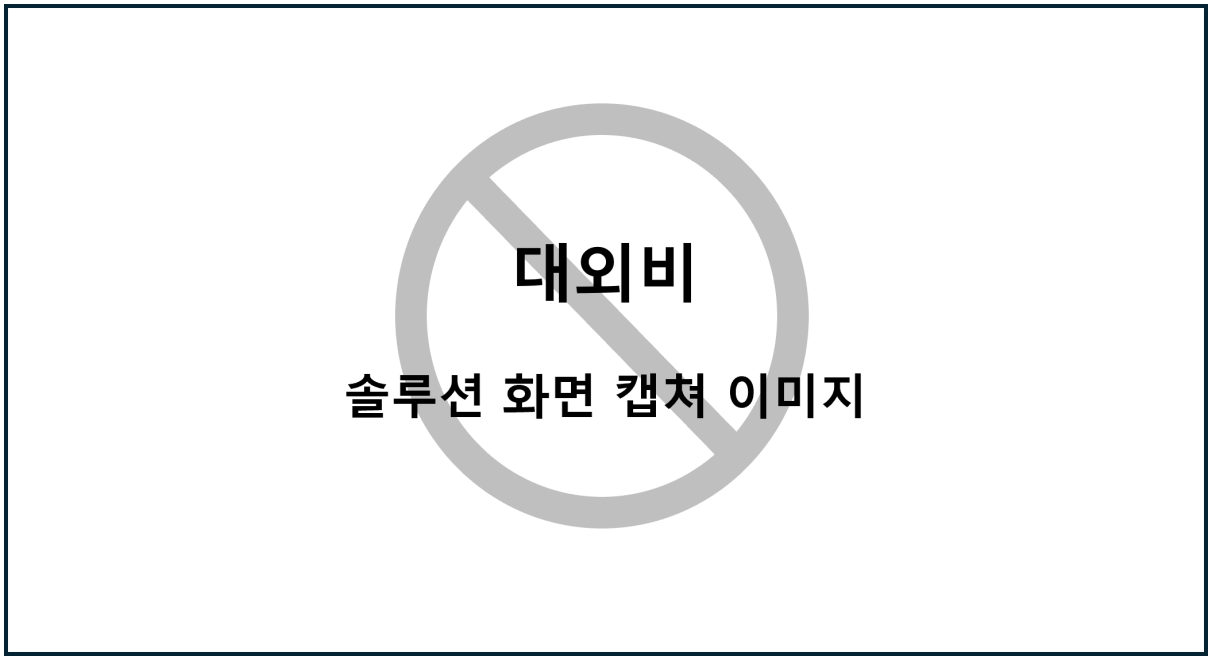
◎ 프로젝트 및 어댑터 생성

△세부절차

1. 프로젝트 생성
2. 어댑터 생성

1. 프로젝트 생성

새로운 프로젝트 관리가 필요한 경우에만 프로젝트 생성을 진행합니다.
▷IMC(웹 관리콘솔) 메인화면의 우측 프로젝트 관리를 클릭하여 이동합니다.
▷화면 우측 상단 추가 버튼을 클릭하여 원하는 프로젝트 이름을 입력한 뒤 저장합니다.



2. 어댑터 생성

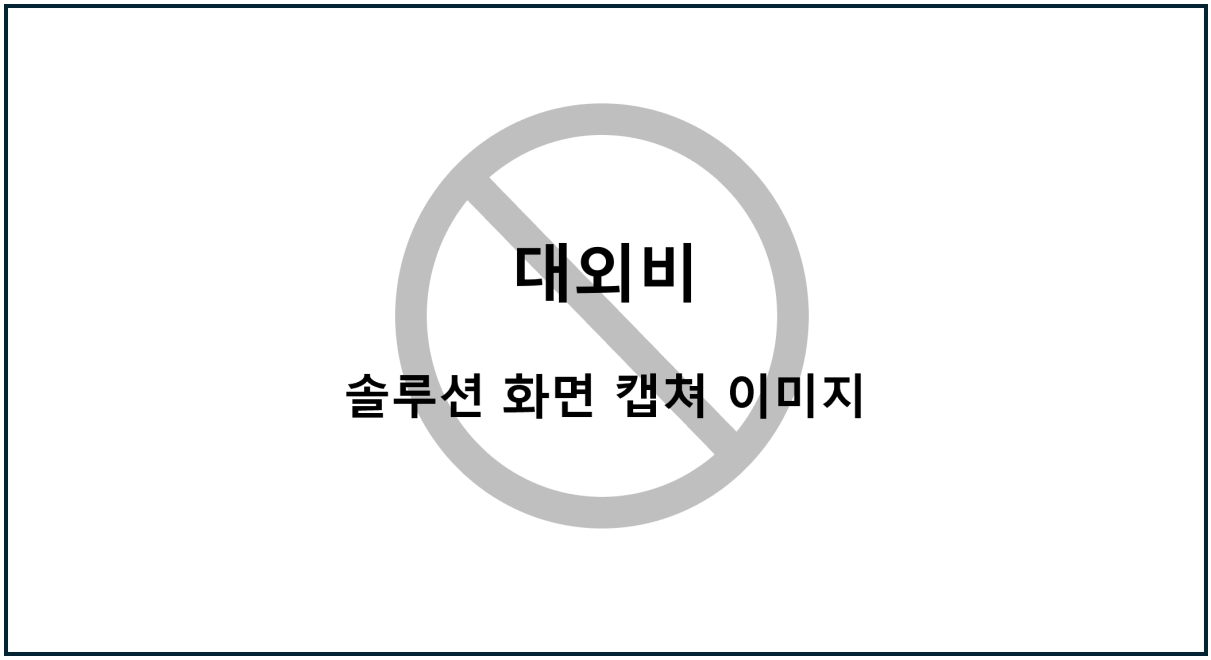
기존 어댑터를 활용하여 인터페이스 개발을 할 수 없는 경우에만 새로운 어댑터를 생성합니다.

▷IMC(웹 관리콘솔) 의 프로젝트 관리 화면에서 어댑터를 생성한 프로젝트를 클릭하여 해당 프로젝트 화면으로 이동합니다.

▷화면 우측 상단 편집 버튼을 클릭한 뒤 추가 버튼을 클릭합니다.

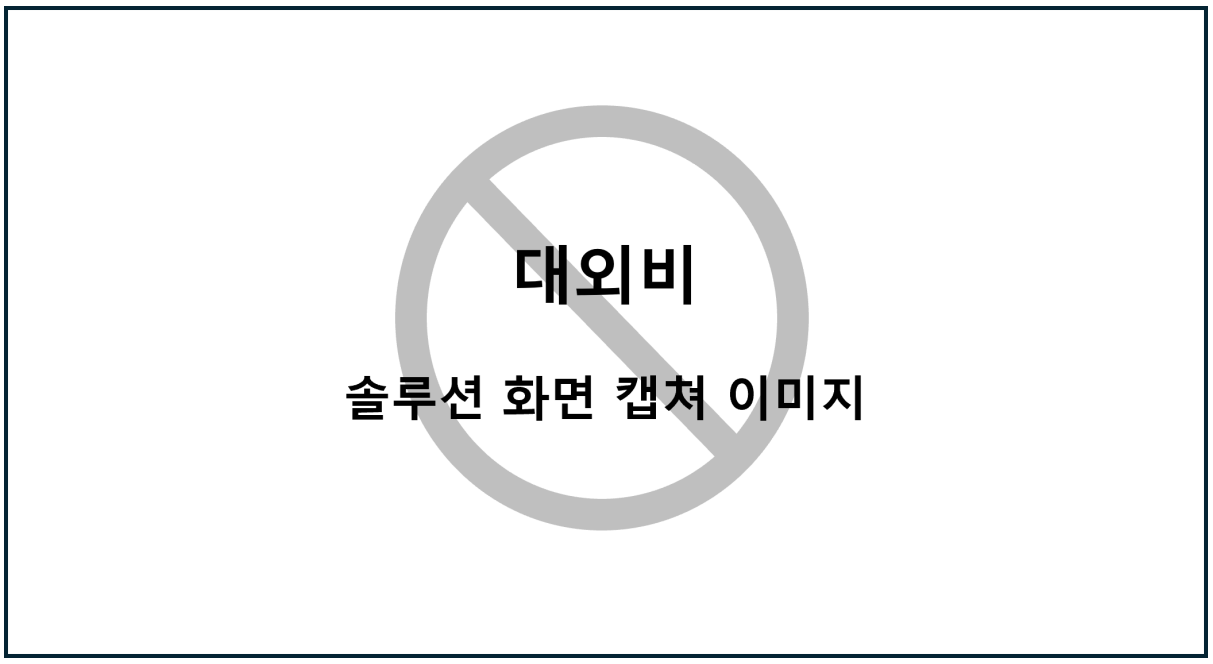
▷어댑터 추가 팝업 화면에서 어댑터 관리자, 이름, 아이디, 포트를 입력한 뒤 기타 정보의 자바 힙 사이즈를(어댑터가 사용할 메모리) 적절하게 합니다.

▷필수 설정정보 입력 및 유효성 검증이 끝나면 저장 버튼을 클릭하여 어댑터를 생성합니다.



◎ 주요 설정파일 등록

어댑터를 생성한 뒤 화면의 목록에서 해당 어댑터를 클릭하여 어댑터 관리 화면으로 이동합니다. 어댑터 관리 화면에서 주요 설정파일 등록 절차와 관련이 있는 메뉴는 빈설정/그 외 설정/컴포넌트 설정 입니다.



△세부절차

1. bean.xml 설정
2. mybatis-configuration.xml 생성
3. config_CORE.xml 생성
4. config_INTERFACES.xml 생성
5. config_SERVICES.xml 생성
6. config_ERROR_HANDLERS.xml 생성
7. DNC_COMMON.dnc 생성 후 기본 Dynamic-Code 등록
8. DB Source 등록
9. FTP Source 등록

인터페이스 개발은 위 세부절차를 따라 주요 설정파일을 등록한 후 진행합니다. 기본설정이 완료된 템플릿 역할을 하는 어댑터를 생성한 뒤 복사하는 방식으로도 사용이 가능합니다.

◎ 빈설정

▷어댑터 관리화면에서 화면 좌측 빈설정을 클릭합니다.

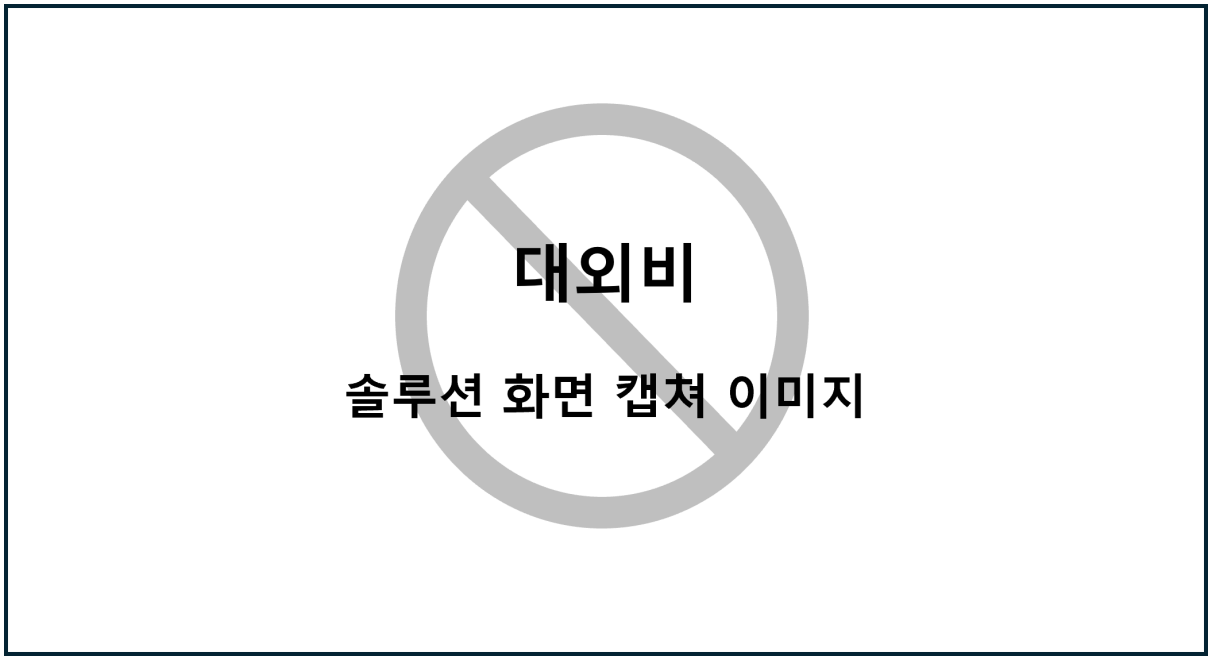
▷빈설정을 클릭했을 때 화면에 표시되는 XML 문서 에디터에서 <beans> 태그와 </beans> 태그 사이에 아래의 두 개의 설정을 등록합니다.

1. config_CORE.xml Import 구문 추가

```
<import resource="classpath:config_CORE.xml"/>
```

2. SimpleDispatcher bean 추가

```
<bean class="mb.dnm.dispatcher.SimpleDispatcher" id="SimpleDispatcher"/>
```



▷저장 버튼을 클릭하여 설정을 저장합니다.

◎ 그 외 설정[mybatis-configuration.xml]

Indigo-Dynamic-Adapter 는 SQL 작업 시 Mybatis 라이브러리를 사용합니다. 따라서 Mybatis 설정 파일을 다음과 같이 등록합니다.

▷어댑터 관리화면의 좌측 메뉴에서 그 외 설정을 클릭합니다.

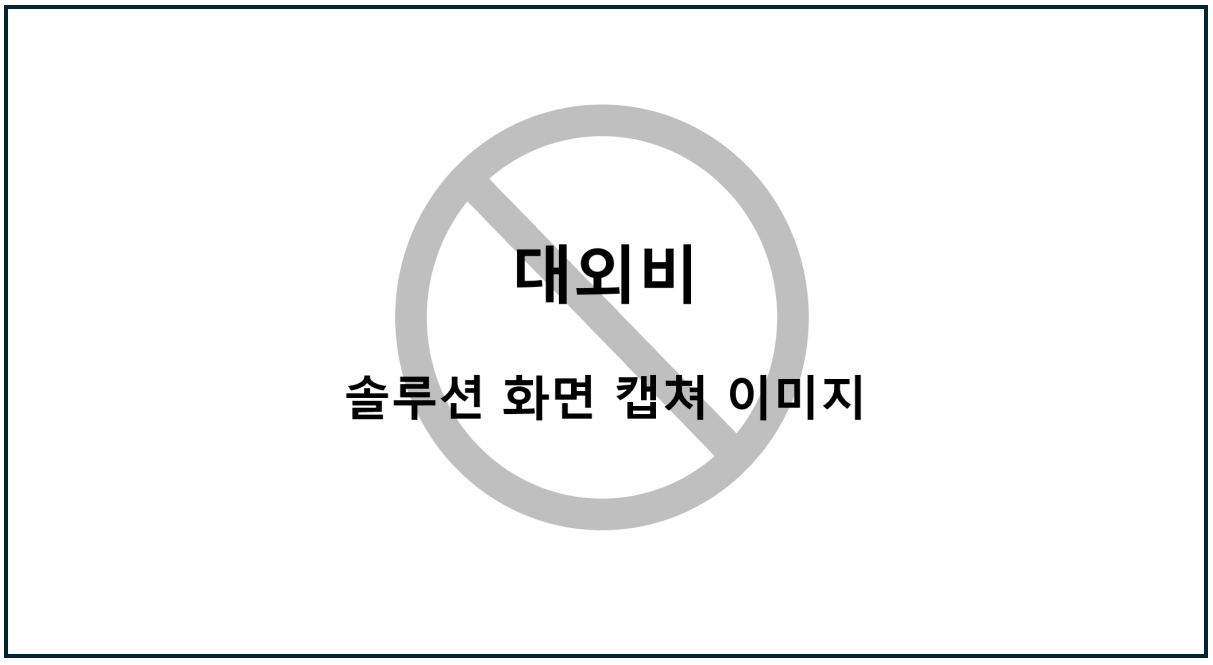
▷그 외 설정 화면 상단의 추가 버튼을 클릭합니다.

▷mybatis-configuration.xml 을 파일명으로 하여 아래 내용을 추가합니다.

파일명	mybatis-configuration.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>

   <settings>
      <setting name="cacheEnabled" value="true" />
      <setting name="lazyLoadingEnabled" value="false" />
      <setting name="multipleResultSetsEnabled" value="true" />
      <setting name="useColumnLabel" value="true" />
      <setting name="useGeneratedKeys" value="false" />
      <setting name="defaultExecutorType" value="BATCH" />
      <setting name="defaultStatementTimeout" value="25000" />
      <setting name="jdbcTypeForNull" value="NULL" />
   </settings>

</configuration>
```

▷저장 버튼을 클릭하여 설정을 저장합니다.



◎ 그 외 설정[config_CORE.xml]

Storage 와 Provider 같은 핵심 컴포넌트를 등록할 설정파일인 config_CORE.xml 를 생성합니다.

▷어댑터 관리화면의 좌측 메뉴에서 그 외 설정을 클릭합니다.

▷그 외 설정 화면 상단의 추가 버튼을 클릭합니다.

▷config_CORE.xml 을 파일명으로 하여 아래 내용을 추가합니다.

파일명	config_CORE.xml

```
<?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-4.0.xsd" >

    <import resource="commons_configure.xml"/>
    <import resource="config_INTERFACES.xml"/>
    <import resource="config_SERVICES.xml"/>
    <import resource="config_ERROR_HANDLERS.xml"/>

    <!-- Storage Manager -->
    <bean class="mb.dnm.storage.StorageManager">
        <property name="defaultInterfaceEnabled" value="true"/>
        <property name="interfaceRegistry" ref="interfaces"/>
        <property name="serviceRegistry" ref="serviceStrategies"/>
        <property name="errorHandlerRegistry" ref="errorHandlers"/>
    </bean>

    <!-- ※DB Access Component -->
    <bean class="mb.dnm.access.db.DataSourceProvider">
        <property name="queryExecutors">
            <list>
            </list>
        </property>
    </bean>

    <!-- ※FTP Access Component -->
    <bean class="mb.dnm.access.ftp.FTPSourceProvider">
        <property name="ftpClients">
            <list>
            </list>
        </property>
    </bean>

    <!-- ※Adapter Web Server -->
    <bean class="mb.dnm.dispatcher.http.HttpDispatcherServer" init-method="start">
        <property name="port"      value="?{어댑터포트+100}"/>
    </bean>

    <!-- ※DynamicCode Repository -->
    <bean class="mb.dnm.access.dynamic.DynamicCodeProvider">
        <property name="compilerThreadCount"  value="5"/>
        <property name="codeLocations"     value="classpath*:*.dnc"/>
    </bean>

</beans>
```

▷위 파일 내용에서 강조 표시된 부분인 ?{어댑터포트+100}를 어댑터 생성 당시 설정했던 port 번호보다 100 큰 번호로 바꿔줍니다.
▷저장 버튼을 클릭하여 설정을 저장합니다.

◎ 그 외 설정[config_INTERFACES.xml]

config_INTERFACES.xml 설정에는 어댑터에서 작동될 인터페이스의 정보 즉, InterfaceInfo 를 등록하는 파일입니다.
▷어댑터 관리화면의 좌측 메뉴에서 그 외 설정을 클릭합니다.
▷그 외 설정 화면 상단의 추가 버튼을 클릭합니다.
▷config_INTERFACES.xml 을 파일명으로 하여 아래 내용을 추가합니다.

파일명	config_INTERFACES.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:util="http://www.springframework.org/schema/util"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.1.xsd
      http://www.springframework.org/schema/util
http://www.springframework.org/schema/util/spring-util-3.1.xsd">

    <util:list id="interfaces">


       <bean class="mb.dnm.storage.InterfaceInfo">
          <property name="interfaceId"          value="INTERFACE_CONTROLLER"/>
          <property name="controllerInterface"   value="true"/>
          <property name="frontHttpUrl"         value="/indigo/adapter/control-interface"/>
          <property name="frontHttpMethod"       value="POST"/>
          <property name="loggingWhenNormal"     value="false"/>
          <property name="loggingWhenError"      value="false"/>
          <property name="serviceId"            value="INTERFACE_CONTROL"/>
       </bean>
    </util:list>

</beans>
```

config_INTERFACES.xml 설정에 기본으로 등록 되어있는 INTERFACE_CONTROLLER 인터페이스는 어댑터에서 인터페이스의 제어(활성화 및 비활성화/작동)을 위한 인터페이스입니다.
▷저장 버튼을 클릭하여 설정을 저장합니다.

◎ 그 외 설정[config_SERVICES.xml]

config_SERVICES.xml 은 인터페이스가 사용할 프로세스인 Service-Strategy 를 등록하는 설정입니다.

▷어댑터 관리화면의 좌측 메뉴에서 그 외 설정을 클릭합니다.

▷그 외 설정 화면 상단의 추가 버튼을 클릭합니다.

▷ config_SERVICES.xml 을 파일명으로 하여 아래 내용을 추가합니다.

파일명	config_SERVICES.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:util="http://www.springframework.org/schema/util"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.1.xsd
      http://www.springframework.org/schema/util
http://www.springframework.org/schema/util/spring-util-3.1.xsd">

    <util:map id="serviceStrategies">

        <entry key="INTERFACE_CONTROL">
            <list>
                <bean class="mb.dnm.service.general.TransformData">
                    <property name="input"            value="$http_body"/>
                    <property name="inputDataType"     value="JSON"/>
                    <property name="outputDataType"    value="MAP"/>
                    <property name="output"           value="$http_body"/>
                </bean>
                <bean class="mb.dnm.service.general.ControlInterfaceActivation">
                    <property name="input"            value="$http_body"/>
                </bean>
                <bean class="mb.dnm.service.http.SetHttpResponseBody">
                    <property name="input"     value="$$COMMAND_RESULT"/>
                </bean>
            </list>
        </entry>

    </util:map>

    <!-- ※Callback for logging -->
    <bean id="custom_logging"  class="dw.loex.callback.CustomLoggingCallback">
        <description>IterationGroup 에서 사용되는 로깅 Callback.</description>
        <property name="dataCountInput"        value="tx_count"/>
        <property name="errorDataCountInput"    value="tx_error_count"/>
        <property name="loggingAdapterUrl"     value="http://127.0.0.1:28901/indigo-
api/post/log"/>
    </bean>

    <bean class="dw.loex.callback.CustomLoggingCallback" init-method="register">
        <description>프로세스 전반적으로 사용되는 로깅 Component.</description>
        <property name="input"               value="no_logging_flag"/>
        <property name="noLoggingWhenInputIs"   value="Y"/>
        <property name="dataCountInput"        value="tx_count"/>
        <property name="errorDataCountInput"    value="tx_error_count"/>
        <property name="loggingAdapterUrl"     value="http://127.0.0.1:28901/indigo-
api/post/log"/>
    </bean>

</beans>
```

config_SERVICES.xml 설정에 기본으로 등록 되어있는 INTERFACE_CONTROL Service 는 어댑터에서 인터페이스의 제어(활성화 및 비활성화/작동)을 위한 Service-Strategy 이며, 등록된 Callback 은 로깅을 남기기 위한 기본 설정입니다.

▷저장 버튼을 클릭하여 설정을 저장합니다.

◎ 그 외 설정[config_ERROR_HANDLERS.xml]

config_ERROR_HANDLERS.xml 은 Service-Chaining 도중 발생하는 에러를 처리할 에러 처리 프로세스를 등록하는 설정입니다.

▷어댑터 관리화면의 좌측 메뉴에서 그 외 설정을 클릭합니다.

▷그 외 설정 화면 상단의 추가 버튼을 클릭합니다.

▷ config_ERROR_HANDLERS.xml 을 파일명으로 하여 아래 내용을 추가합니다.

파일명	config_ERROR_HANDLERS.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:util="http://www.springframework.org/schema/util"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.1.xsd
      http://www.springframework.org/schema/util
http://www.springframework.org/schema/util/spring-util-3.1.xsd">

    <util:map id="errorHandlers">

    </util:map>

</beans>
```

▷저장 버튼을 클릭하여 설정을 저장합니다.

▷저장된 파일의 <util:map> 태그의 내부에 아래과 같은 기본 Error-Handler 3 개를 추가합니다.

설명	DBFILE 연계 시 DB 트랜잭션을 Rollback, 연계서버에 임시저장한 파일을 삭제한다.

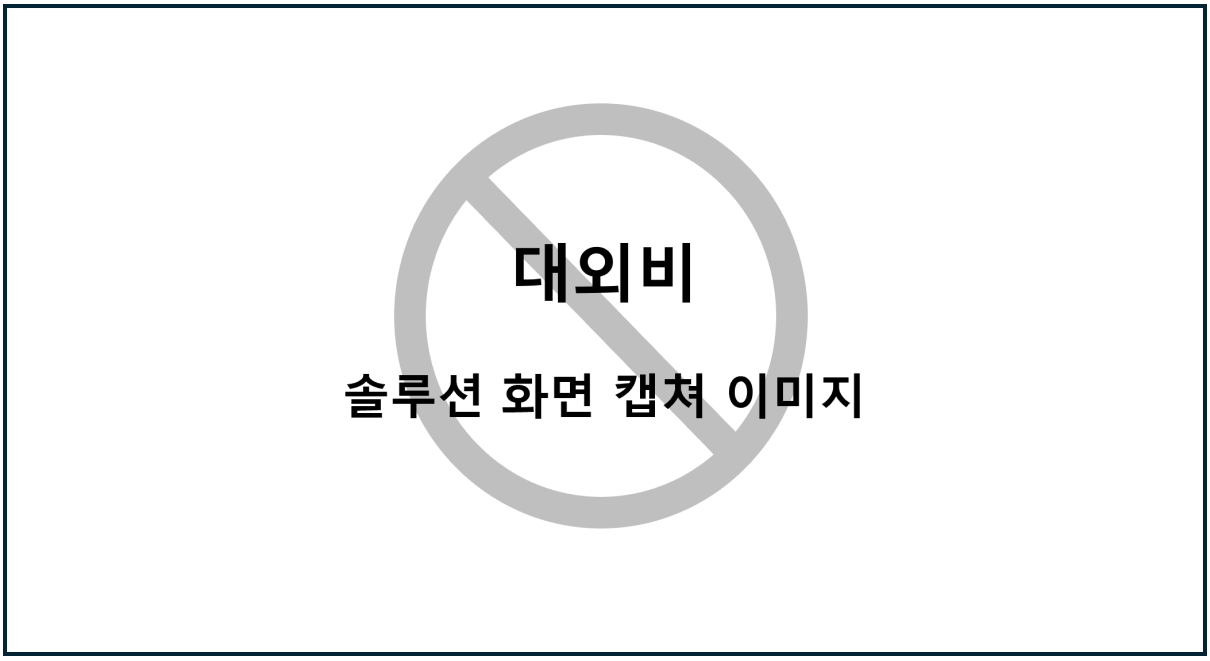
```
<entry key="$EH_DBFILE_ROLLBACK">
    <list>
        <bean class="mb.dnm.exeption.handler.ExecuteServiceErrorHandler">
            <property name="service">
                <bean class="mb.dnm.service.db.Rollback"/>
            </property>
        </bean>
        <bean class="mb.dnm.exeption.handler.ExecuteServiceErrorHandler">
            <property name="service">
                <bean class="mb.dnm.service.file.DeleteFiles">
                    <property name="description"       value="파일을 연계서버(로컬)에서 삭제한다."/>
                    <property name="input"            value="file_loc"/>
                </bean>
            </property>
        </bean>
    </list>
</entry>
```

설명	DB 연계 시 DB 트랜잭션을 Rollback, DB 에러 유형 필터링 후 에러결과를 업데이트 한다.

```
<entry key="$EH_DB2DB">
    <list>
        <bean class="mb.dnm.exeption.handler.ServiceProcessableErrorHandler">
            <property name="ignoreException"          value="true"/>
            <property name="services">
                <list>
                    <bean class="mb.dnm.service.db.Rollback"/>
                    <bean class="mb.dnm.service.dynamic.ExecuteDynamicCode"/>
                    <bean class="mb.dnm.service.db.StartTransaction"/>
                    <bean class="mb.dnm.service.db.Update">
                        <property name="input"        value="data_list"/>
                    </bean>
                    <bean class="mb.dnm.service.db.Commit"/>
                </list>
            </property>
        </bean>
    </list>
</entry>
```

설명	에러 발생 시 로그를 전송합니다.

```
<entry key="$EH_LOGGING">
    <list>
        <bean class="mb.dnm.exeption.handler.ExecuteServiceErrorHandler">
            <property name="service">
                <bean class="mb.dnm.service.general.OutputCustomData">
                    <property name="output"        value="no_logging_flag"/>
                    <property name="customData"    value="N"/>
                </bean>
            </property>
        </bean>
    </list>
</entry>
```



▷위 내용을 모두 등록한 뒤 저장 버튼을 클릭하여 설정을 저장합니다.

◎ 그 외 설정[DNC_COMMON.dnc]

DNC_COMMON.dnc 파일을 생성 후, 모든 인터페이스에서 공통으로 사용할 Dynamic-Code 를 등록할 것입니다.

▷어댑터 관리화면의 좌측 메뉴에서 그 외 설정을 클릭합니다.

▷그 외 설정 화면 상단의 추가 버튼을 클릭합니다.

▷ DNC_COMMON.dnc 을 파일명으로 하여 아래 내용을 추가합니다.

파일명	DNC_COMMON.dnc

```
#namespace: COMMON

데이터베이스 에러 유형을 확인하여 Source DB 에 에러 결과 업데이트를 처리할 지에 대한 여부를 결정한다.

#code_id : MAP_DATA
#{
          Map<String, Object> master data = (Map<String, Object>) getInput(ctx,
"master_data");
          List<Map<String, Object>> detail_data_list = (List<Map<String, Object>>)
getInput(ctx, "detail_data_list");
          String txId = ctx.getTxId();


          log.debug("[{}] Master data 와 Detail data Mapping 시작", txId);
          for (Map<String, Object> detail_data : detail_data_list) {
                    detail_data.put("MASTER", master_data);
          }
          log.debug("[{}] Master data 와 Detail data Mapping 완료", txId);
    };

    List<ErrorTrace> errorList = ctx.getErrorTraces();

    boolean updateSrcTableYn = false;
    for (ErrorTrace error : errorList) {
       String errMsg = error.getMessage();
       if (errMsg != null) {
          errMsg = errMsg.toUpperCase();
          for (String code : ERROR_FILTER) {
             if (errMsg.contains(code)) {
                updateSrcTableYn = true;
                log.info("[{}]Error handling condition is detected. Error code: {}",
ctx.getTxId(), code);
                ctx.setMsg(StringUtil.substring(errMsg, 200));
                break;
             }
          }
           if (updateSrcTableYn) {
              break;
           }
       }
    }

    if (!updateSrcTableYn) {
        ctx.setProcessOn(false);
    }

}#

#code_id : CHECK_ORACLE_DB_ERROR
#{
    final String[] ERROR_FILTER = {
       "ORA-00001", //무결성 제약 조건
       "ORA-01400", //NULL to NOT NULL COLUMN
       "ORA-01407", //NULL to NOT NULL COLUMN
```

```
      "ORA-01438", //Data 길이
      "ORA-01722", //Data 길이
      "ORA-12899"  //Data 길이
   };

   List<ErrorTrace> errorList = ctx.getErrorTraces();

   boolean updateSrcTableYn = false;
   for (ErrorTrace error : errorList) {
      String errMsg = error.getMessage();
      if (errMsg != null) {
         errMsg = errMsg.toUpperCase();
         for (String code : ERROR_FILTER) {
            if (errMsg.contains(code)) {
               updateSrcTableYn = true;
               log.info("[{}]Error handling condition is detected. Error code: {}",
ctx.getTxId(), code);
               ctx.setMsg(StringUtil.substring(errMsg, 200));
               break;
            }
         }
           if (updateSrcTableYn) {
               break;
           }
      }
   }

   if (!updateSrcTableYn) {
       ctx.setProcessOn(false);
   }

}#
```

SAP HANA DB 에러코드 참고 사이트
"https://help.sap.com/docs/HANA_SERVICE_CF/7c78579ce9b14a669c1f3295b0d8ca16/20a78d3275191014b41bae7c4a46d835.html"

```
#code_id : CHECK_HANA_DB_ERROR
#{
   //final String EXCEPTION_CLASS_NAME = "jdbc.exceptions";
   final String[] ERROR_FILTER = {
      "[273]", //not a single character string
      "[267]", //specified length too long for its datatype
      "[274]", //inserted value too large for column
      "[301]", //unique constraint violated
      "[302]", //invalid CHAR or VARCHAR value
      "[303]", //invalid DATE TIME or TIMESTAMP value
      "[339]", //invalid number
      "[359]", //string is too long
   };


   List<ErrorTrace> errorList = ctx.getErrorTraces();

   boolean updateSrcTableYn = false;
   for (ErrorTrace error : errorList) {
      String errMsg = error.getMessage();
```

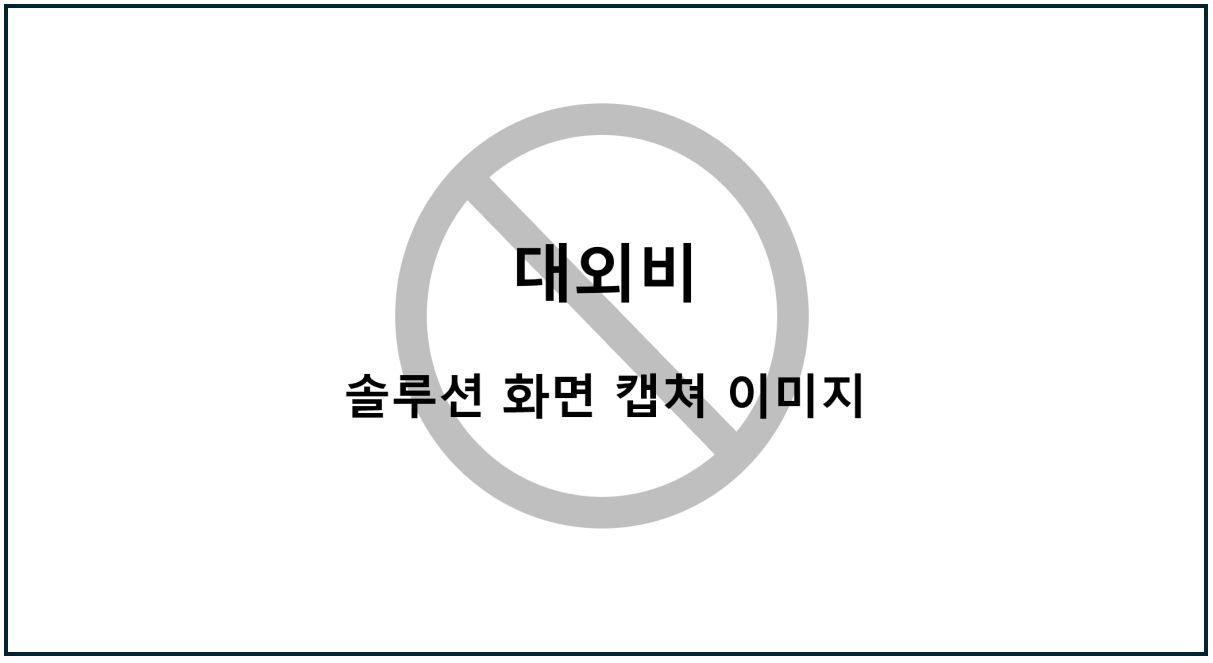
```
      if (errMsg != null) {
        errMsg = errMsg.toUpperCase();

        for (String code : ERROR_FILTER) {
          if (errMsg.contains(code)) {
            updateSrcTableYn = true;
            log.info("[{}]Error handling condition is detected. Error code: {}",
ctx.getTxId(), code);
            ctx.setMsg(StringUtil.substring(errMsg, 0, 199));
            break;
          }
        }
          if (updateSrcTableYn) {
            break;
          }
      }
    }

    if (!updateSrcTableYn) {
       ctx.setProcessOn(false);
    }

}#
```

▷저장 버튼을 클릭하여 설정을 저장합니다.

◎ DB Source 등록[컴포넌트 설정][config_CORE.xml]

인터페이스 대상 시스템의 데이터베이스 접속관련 설정은 어댑터 관리 화면의 컴포넌트 설정과 config_CORE.xml 설정에서 진행합니다.

▷어댑터 관리화면의 좌측 메뉴에서 일반을 클릭합니다.
▷일반 화면의 우측 컴포넌트 설정에서 추가 버튼을 클릭한 후 DB 를 클릭합니다.
▷DB 설정 팝업 화면에 사용한 DB 접속정보를 입력합니다. 팝업 화면의 아이디는 DB DataSource 의 아이디를 의미하며 어댑터 내에서 고유하게 설정해야 합니다. IMC 의 리소스 관리에 등록된 공통 리소스를 활용하는 경우 콤보박스에서 로컬이 아닌 공통을 클릭하여 사용 가능합니다.
▷커넥션 설정의 자동커밋 속성을 false 로 설정합니다.
▷저장 버튼을 클릭하여 컴포넌트 설정을 저장합니다.
▷어댑터에서 접속해야 하는 DB 수만큼 DB 컴포넌트를 추가적으로 등록해줍니다.



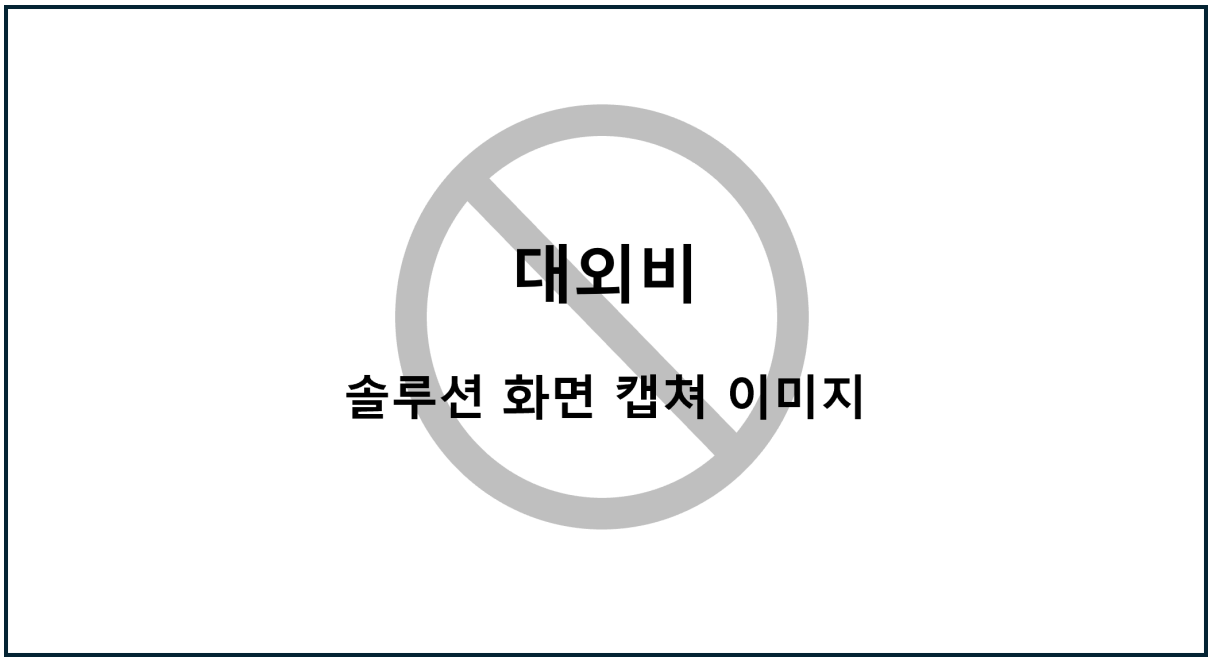
▷어댑터 관리화면의 좌측 메뉴의 그 외 설정에서 config_CORE.xml 를 클릭합니다.

▷config_CORE.xml 설정의 내용중 아래 내용을 찾아 `list` 태그 사이에 ExeuctorTemplate 을 등록해줍니다.

```
17      <!-- ※DB Access Component -->
18      <bean class="mb.dnm.access.db.DataSourceProvider">
19          <property name="queryExecutors">
20              <list>
21
22              </list>
23          </property>
24      </bean>
```

```
<bean class="mb.dnm.access.db.ExecutorTemplate">
    <property name="templateName"       value="?{DB 템플릿명}"/>
    <property name="dataSource"         ref="?{DataSource ID}"/>
    <property name="configLocation"     value="mybatis-configuration.xml"/>
    <property name="mapperLocations"    value="classpath*:SQL_*.xml"/>
</bean>
```

▷강조 표시된 `DB 템플릿명` 은 DataSource 접근 시 사용할 Alias 를 어댑터 내에서 고유하게 설정하여 입력해주고 `DataSource ID` 는 앞서 컴포넌트 설정을 통해 등록했던 DataSource 의 아이디를 입력해줍니다.

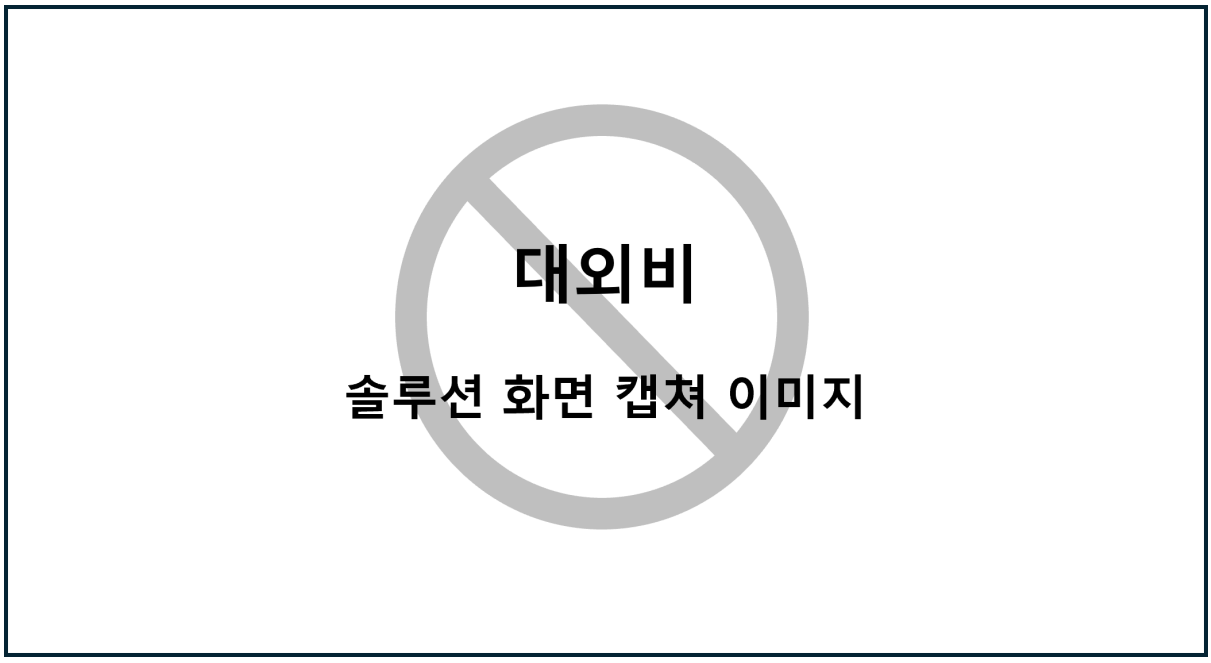


▷ExeuctorTemplate 도 앞서 컴포넌트 설정을 통해 등록했던 DataSource 수만큼 등록해줍니다.
▷저장 버튼을 클릭하여 설정을 저장합니다.

◎ FTP Source 등록[config_CORE.xml]

FTP 서버 접속이 필요한 인터페이스가 있는 경우 config_CORE.xml 설정에 FTP 서버 접속설정 등록을 해주어야 합니다. FTP 서버 접속설정은 다음과 같이 진행합니다.

▷어댑터 관리화면의 좌측 메뉴의 그 외 설정에서 config_CORE.xml 를 클릭합니다.
▷config_CORE.xml 설정의 내용중 아래 내용을 찾아 `list` 태그 사이에 FTPClientTemplate 을 등록해줍니다.



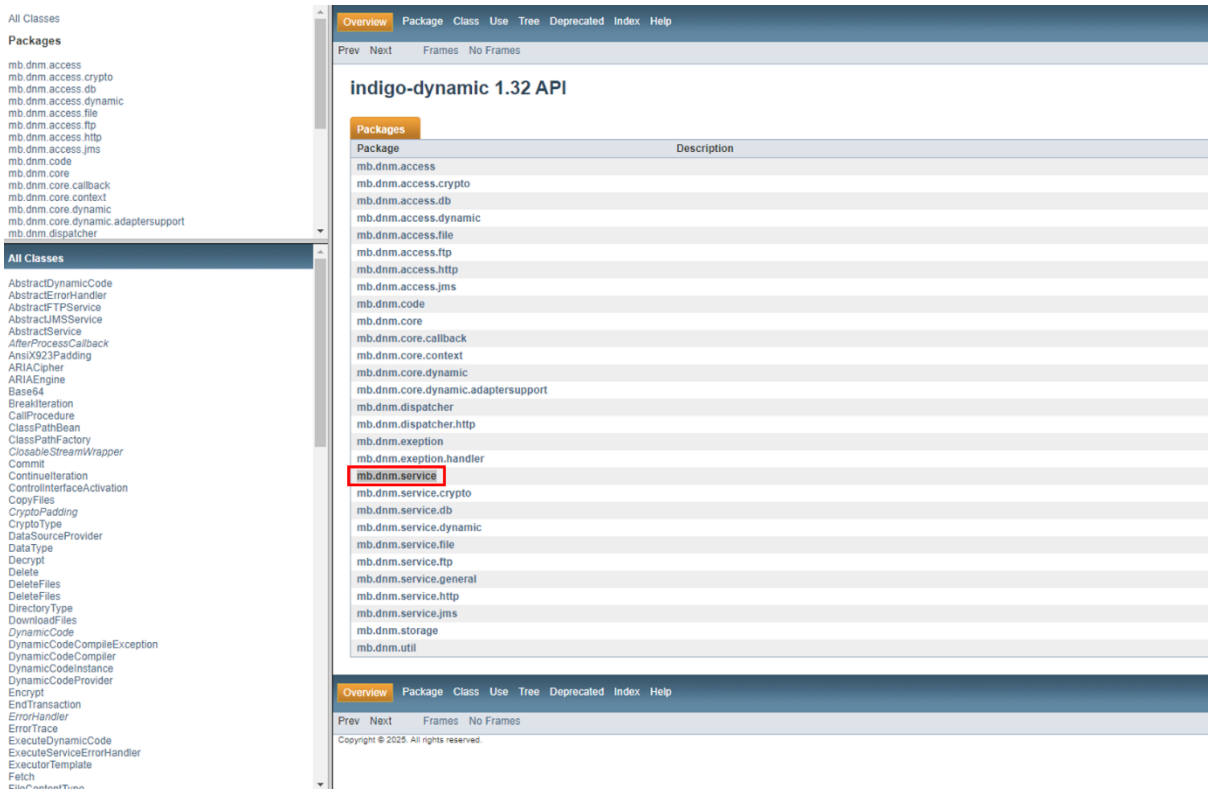

```
<bean class="mb.dnm.access.ftp.FTPClientTemplate">
    <property name="templateName"            value="?{FTP_템플릿명}"/>
    <property name="host"                    value="?{FTP 서버_호스트}"/>
    <property name="port"                    value="?{FTP 서버_포트}"/>
    <property name="user"                    value="?{사용자명}"/>
    <property name="password"                value="?{비밀번호}"/>
    <!-- * MS949, UTF-8, etc -->
    <property name="controlEncoding"         value="?{FTP 서버_인코딩}"/>
    <property name="serverLanguageCode"      value="ko_KR"/>
    <!--* UNIX, UNIX_LTRIM, VMS, WINDOWS, OS/2, OS/400, AS/400, MVS, TYPE: L8, NETWARE, MACOS PETER -->
    <property name="serverKey"               value="?{FTP 서버_운영체제}"/>
    <property name="debugCommandAndReply"    value="false"/>
</bean>
```

▷접속이 필요한 FTP 서버 수만큼 FTPClientTemplate 을 추가한 후 저장 버튼을 클릭하여 설정을 저장합니다.

3. *Service Components*

◎ Service Components

Service Components 의 종류와 속성에 대한 정보는 Indigo-dynamic 1.32 API (Javadoc, 별첨) 의 **mb.dnm.service** 패키지를 참고합니다.

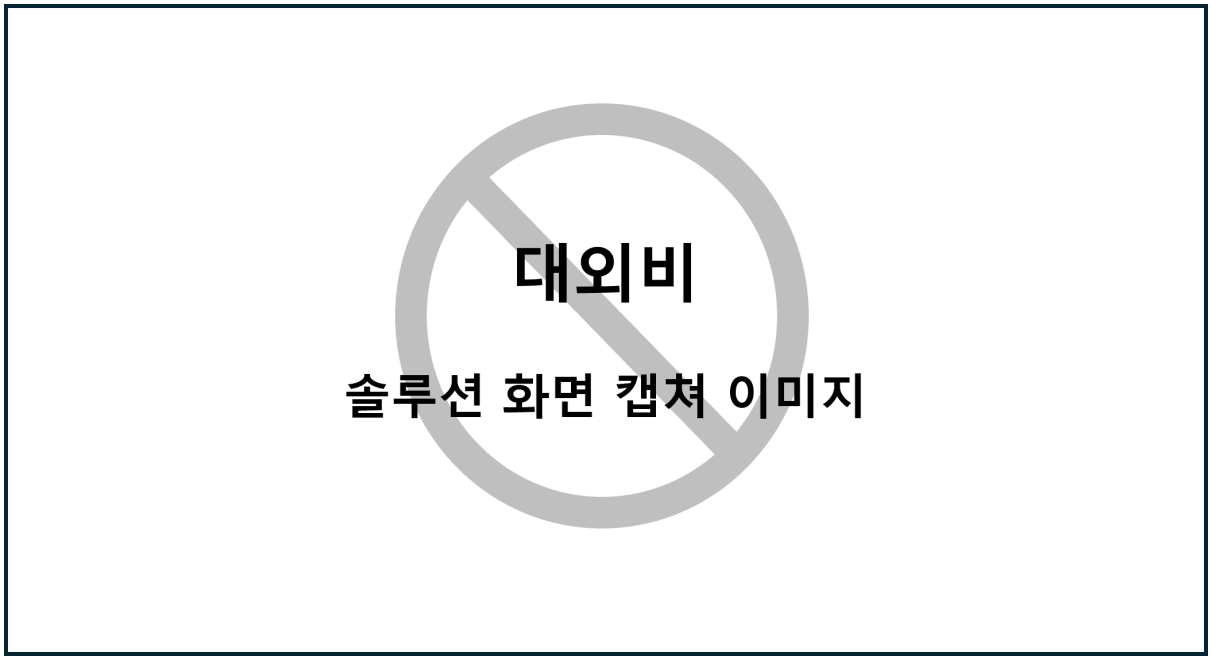


4. 인터페이스 설정 예시

주제

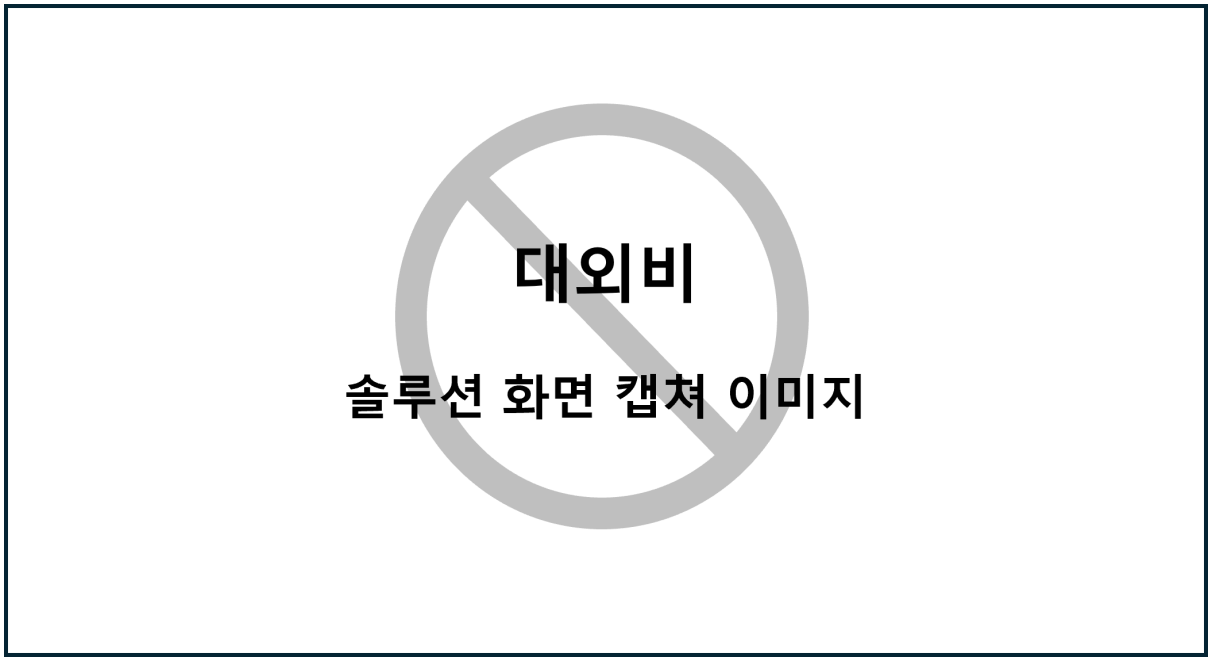
◎ 기본 설정
◎ DB to DB 인터페이스 [기본]
◎ DB to DB 인터페이스 [대용량]
◎ DB to DB 인터페이스 [Master-Detail]
◎ FTP to DB 인터페이스 [기본]
◎ DB to FTP 인터페이스 [기본]
◎ DynamicCode 작성
◎ ScheduleJob 등록

◎ **기본 설정**



DB 서버 및 FTP 서버 구성이 위와 같이 되어있다고 할 때 주요 설정 파일의 설정 예시는 다음과 같다.

▷DB Source 와 DB Target 을 각각 Main/Sub 2 개씩 컴포넌트를 등록한다.(동일한 DB DataSource 를 Main/Sub 2 개씩 등록)



▷config_CORE.xml 의 ExecutorTemplate 에 4 개의 DataSource 를 등록한다.

config_CORE.xml

```
    <!-- ※DB Access Component -->
    <bean class="mb.dnm.access.db.DataSourceProvider">
        <property name="queryExecutors">
            <list>
                <bean class="mb.dnm.access.db.ExecutorTemplate">
                    <property name="templateName"       value="DB_SRC_MAIN"/>
                    <property name="dataSource"         ref="DB_Source_Main"/>
                    <property name="configLocation"     value="mybatis-configuration.xml"/>
                    <property name="mapperLocations"    value="classpath*:SQL_*.xml"/>
                </bean>
                <bean class="mb.dnm.access.db.ExecutorTemplate">
                    <property name="templateName"       value="DB_SRC_SUB"/>
                    <property name="dataSource"         ref="DB_Source_Sub"/>
                    <property name="configLocation"     value="mybatis-configuration.xml"/>
                    <property name="mapperLocations"    value="classpath*:SQL_*.xml"/>
                </bean>
                <bean class="mb.dnm.access.db.ExecutorTemplate">
                    <property name="templateName"       value="DB_TGT_MAIN"/>
                    <property name="dataSource"         ref="DB_Target_Main"/>
                    <property name="configLocation"     value="mybatis-configuration.xml"/>
                    <property name="mapperLocations"    value="classpath*:SQL_*.xml"/>
                </bean>
                <bean class="mb.dnm.access.db.ExecutorTemplate">
                    <property name="templateName"       value="DB_TGT_SUB"/>
                    <property name="dataSource"         ref="DB_Target_Sub"/>
                    <property name="configLocation"     value="mybatis-configuration.xml"/>
                    <property name="mapperLocations"    value="classpath*:SQL_*.xml"/>
                </bean>
            </list>
        </property>
    </bean>
```

▷config_CORE.xml 의 FTPSourceProvider 에 2 개의 FTPClientTemplate 을 등록한다.

config_CORE.xml

```
    <!-- ※FTP Access Component -->
    <bean class="mb.dnm.access.ftp.FTPSourceProvider">
        <property name="ftpClients">
            <list>
                <bean class="mb.dnm.access.ftp.FTPClientTemplate">
                    <property name="templateName"               value="FTP_SRC"/>
                    <property name="host"                       value="10.0.0.111"/>
                    <property name="port"                       value="20"/>
                    <property name="user"                       value="USER"/>
                    <property name="password"                   value="1234"/>
                    <property name="controlEncoding"            value="MS949"/>
                    <property name="serverLanguageCode"         value="ko_KR"/>
                    <property name="serverKey"                  value="WINDOWS"/>
                    <property name="debugCommandAndReply"       value="false"/>
                </bean>
                <bean class="mb.dnm.access.ftp.FTPClientTemplate">
                    <property name="templateName"               value="FTP_TGT"/>
                    <property name="host"                       value="20.0.0.111"/>
                    <property name="port"                       value="20"/>
                    <property name="user"                       value="USER"/>
                    <property name="password"                   value="1234"/>
                    <property name="controlEncoding"            value="MS949"/>
                    <property name="serverLanguageCode"         value="ko_KR"/>
                    <property name="serverKey"                  value="WINDOWS"/>
                    <property name="debugCommandAndReply"       value="false"/>
                </bean>
            </list>
        </property>
    </bean>
```

◎ DB to DB

1. Source DB Select
2. 데이터 존재여부 확인
3. 데이터 개수 확인
4. DB 트랜잭션 시작
5. Target DB Insert
6. Source DB 결과 Update
7. DB 트랜잭션 종료

파일명	config_SERVICES.xml

```
<entry key="$P_DB2DB_Sel_Ins_Upd">
   <list>
      <bean class="mb.dnm.service.db.Select">
         <description>Source DB 에서 연계 대상 데이터를 조회한다.</description>
         <property name="output"           value="data_list"/>
      </bean>
      <bean class="mb.dnm.service.general.StopIfInputIsNullOrEmpty">
         <description>Source DB 에서 조회된 데이터가 없으면 프로세스를 종료한다.</description>
         <property name="input"            value="data list"/>
         <property name="output"           value="no_logging_flag"/>
         <property name="outputValue"       value="Y"/>
      </bean>
      <bean class="mb.dnm.service.general.GetSize">
         <description>Source DB 에서 조회한 데이터의 개수를 확인한다.</description>
         <property name="input"            value="data_list"/>
         <property name="output"           value="tx_count"/>
      </bean>
      <bean class="mb.dnm.service.db.StartTransaction"/>
      <bean class="mb.dnm.service.db.Insert">
         <property name="description"       value="Target DB 에 데이터를 Insert 한다."/>
         <property name="input"            value="data_list"/>
      </bean>
      <bean class="mb.dnm.service.db.Update">
         <property name="description"       value="Source DB 에 성공 결과를 Update 한다."/>
         <property name="input"            value="data_list"/>
      </bean>
      <bean class="mb.dnm.service.db.EndTransaction"/>
   </list>
</entry>
```

파일명	config_INTERFACES.xml

```
<bean class="mb.dnm.storage.InterfaceInfo"  name="DB2DB default">
   <property name="interfaceId"              value="인터페이스 ID"/>
   <property name="interfaceName"            value="인터페이스명"/>
   <property name="sourceCode"               value="송신시스템 코드"/>
   <property name="targetCode"               value="수신시스템 코드"/>
   <property name="loggingWhenNormal"         value="true"/>
   <property name="loggingWhenError"          value="true"/>
   <property name="querySequence"             value="DB_SRC_MAIN$@{if_id}.SELECT,
                                                     DB_TGT_MAIN$@{if_id}.INSERT,
                                                     DB_SRC_MAIN$@{if_id}.SUCCESS_UPDATE"/>
   <property name="errorDynamicCodeSequence"   value="COMMON.CHECK_ORACLE_DB_ERROR"/>
   <property name="errorQuerySequence"         value="DB SRC MAIN$@{if id}.ERROR UPDATE"/>
   <property name="serviceId"                 value="$P_DB2DB_Sel_Ins_Upd"/>
   <property name="errorHandlerId"            value="$EH_DB2DB"/>
</bean>
```

파일명	SQL_인터페이스 ID.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="인터페이스 ID">

    <select id="SELECT" resultType="java.util.HashMap">
      Select 쿼리
    </select>

    <insert id="INSERT">
      Insert 쿼리
    </insert>

    <update id="SUCCESS_UPDATE">
      성공결과 Update 쿼리
    </update>

    <update id="ERROR_UPDATE">
      실패결과 Update 쿼리
    </update>

</mapper>
```

◎ DB to DB [대용량]

1. Source DB 에 Select 쿼리 전송
2. fetchSize 만큼 데이터가 조회될 때까지 대기
3. DB 트랜잭션 시작
4. Target DB Insert
5. Source DB 결과 Update
6. DB 트랜잭션 종료
7. 모든 데이터가 조회될 때까지 2~6 과정 반복

파일명	config_SERVICES.xml

```
<entry key="$P_DB2DB_Sel_Ins_Upd_Large">
   <list>
      <bean class="mb.dnm.service.db.Select">
         <property name="output"           value="tx_count"/>
         <property name="handleResultSet"    value="true"/>
         <property name="resultHandlingSupportFactory">
            <bean class="mb.dnm.access.db.ResultHandlingSupportFactory">
               <property name="fetchedInputName"             value="data_list"/>
               <property name="fetchSize"                   value="1000"/>
               <property name="enforcePassTransactionToContexts"   value="false"/>
               <property name="resultHandlingProcessor">
                  <bean class="mb.dnm.service.general.IterationGroup">
                     <property name="createNewContextEachLoop"      value="true"/>
                     <property name="services">
                        <list>
                           <bean class="mb.dnm.service.db.StartTransaction"/>
                           <bean class="mb.dnm.service.db.Insert">
                              <property name="description"       value="Target DB 에 데이터를 Insert 한다."/>
                              <property name="input"            value="data_list"/>
                           </bean>
                           <bean class="mb.dnm.service.db.Update">
                              <property name="description"       value="Source DB 에 성공 결과를 Update 한다."/>
                              <property name="input"            value="data_list"/>
                           </bean>
                           <bean class="mb.dnm.service.db.Commit"/>
                        </list>
                     </property>
                     <property name="errorHandlers">
                        <list>
                           <bean class="mb.dnm.exeption.handler.ServiceProcessableErrorHandler">
                              <property name="ignoreException"          value="true"/>
                              <property name="services">
                                 <list>
                                    <bean class="mb.dnm.service.db.Rollback">
                                       <property name="description"    value="에러가 발생한 인터페이스의 트랜잭션을
롤백한다."/>
                                    </bean>
                                    <bean class="mb.dnm.service.dynamic.ExecuteDynamicCode">
                                       <property name="codeId"        value="COMMON.CHECK_ORACLE_DB_ERROR"/>
                                    </bean>
                                    <bean class="mb.dnm.service.db.Update">
                                       <property name="description"    value="Source DB 에 처리결과(에러)를 Update
한다."/>
                                       <property name="input"         value="data_list"/>
                                    </bean>
                                 </list>
                              </property>
                           </bean>
                        </list>
                     </property>
                     <property name="callbacks">
                        <description>로깅을 위한 콜백 프로세스</description>
                        <list>
                           <ref bean="custom_logging"/>
                        </list>
                     </property>
                  </bean>
               </property>
            </bean>
         </property>
      </bean>
      <bean class="mb.dnm.service.general.StopIfInputIsNullOrEmpty">
         <property name="input"           value="tx_count"/>
         <property name="output"           value="no_logging_flag"/>
         <property name="outputValue"       value="Y"/>
```

```
        </bean>
    </list>
</entry>
```

파일명	config_INTERFACES.xml

```
<bean class="mb.dnm.storage.InterfaceInfo">
    <property name="interfaceId"              value="인터페이스 ID"/>
    <property name="interfaceName"            value="인터페이스명"/>
    <property name="sourceCode"               value="송신시스템 코드"/>
    <property name="targetCode"               value="수신시스템 코드"/>
    <property name="loggingWhenNormal"         value="true"/>
    <property name="loggingWhenError"          value="true"/>
    <property name="querySequence"             value="DB_SRC_MAIN$@{if_id}.SELECT,
                                                      DB_TGT_MAIN$@{if_id}.INSERT,
                                               DB_SRC_SUB$@{if_id}.SUCCESS_UPDATE"/>
    <property name="errorQuerySequence"
                                          value="DB_SRC_SUB$@{if_id}.ERROR_UPDATE"/>
    <property name="serviceId"                value="$P_DB2DB_Sel_Ins_Upd_Large"/>
</bean>
```

파일명	SQL_인터페이스 ID.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="인터페이스 ID">

    <select id="SELECT" resultType="java.util.HashMap">
      Select 쿼리
    </select>

    <insert id="INSERT">
      Insert 쿼리
    </insert>

    <update id="SUCCESS_UPDATE">
      성공결과 Update 쿼리
    </update>

    <update id="ERROR_UPDATE">
      실패결과 Update 쿼리
    </update>

</mapper>
```

◎ DB to DB [Master-Detail]

1. Source DB 에서 마스터 데이터 Select
2. 데이터 존재여부 확인
3. 마스터 데이터 개수만큼 4 ~ 9 과정을 반복
4. DB 트랜잭션 시작
5. Source DB 디테일 데이터 Select
6. 마스터 데이터와 디테일 데이터를 맵핑
7. Target DB Insert
8. Source DB 마스터 데이터 결과 업데이트
9. DB 트랜잭션 종료

파일명	config_SERVICES.xml

```
<entry key="$P_DB2DB_Master_Detail">
  <list>
     <bean class="mb.dnm.service.general.OutputCustomData">
        <property name="output"        value="no_logging_flag"/>
        <property name="customData"    value="Y"/>
     </bean>
     <bean class="mb.dnm.service.db.Select">
        <description>Source DB 에서 연계 대상 데이터를 조회한다.</description>
        <property name="output"           value="master_data_list"/>
     </bean>
     <bean class="mb.dnm.service.general.StopIfInputIsNullOrEmpty">
        <description>연계 대상 데이터가 없으면 프로세스를 종료한다.</description>
        <property name="input"            value="master_data_list"/>
     </bean>
     <bean class="mb.dnm.service.general.IterationGroup">
        <description>Master 데이터의 수만큼 Detail 데이터를 추출하여 수신 DB 에 데이터를 적재한다.</description>
        <property name="input"                        value="master_data_list"/>
        <property name="iterationInputName"             value="master_data"/>
        <property name="fetchSize"                    value="1"/>
        <property name="createNewContextEachLoop"       value="true"/>
        <property name="services">
           <description>연계대상파일의 수만큼 아래에 정의된 프로세스를 순서대로 반복</description>
           <list>
              <bean class="mb.dnm.service.db.StartTransaction"/>
              <bean class="mb.dnm.service.db.Select">
                 <property name="description"       value="Detail 데이터를 조회한다."/>
                 <property name="input"             value="master_data"/>
                 <property name="output"            value="detail_data_list"/>
              </bean>
              <bean class="mb.dnm.service.general.GetSize">
                 <property name="input"             value="detail_data_list"/>
                 <property name="output"            value="tx_count"/>
              </bean>
              <bean class="mb.dnm.service.dynamic.ExecuteDynamicCode">
                 <property name="codeId"            value="COMMON.MAP_DATA"/>
              </bean>
              <bean class="mb.dnm.service.db.Insert">
                 <property name="description"       value="Target DB 에 Detail 데이터를 Insert 한다."/>
                 <property name="input"             value="detail_data_list"/>
              </bean>
              <bean class="mb.dnm.service.db.Update">
                 <property name="description"       value="Source DB 에 처리결과를 Update 한다."/>
                 <property name="input"             value="master_data"/>
              </bean>
              <bean class="mb.dnm.service.db.Commit"/>
           </list>
        </property>
        <property name="errorHandlers">
           <list>
              <bean class="mb.dnm.exeption.handler.ServiceProcessableErrorHandler">
                 <property name="ignoreException"          value="true"/>
```

```
                    <property name="services">
                        <list>
                            <bean class="mb.dnm.service.db.Rollback"/>
                            <bean class="mb.dnm.service.dynamic.ExecuteDynamicCode">
                                <property name="codeId"        value="COMMON.CHECK_ORACLE_DB_ERROR"/>
                            </bean>
                            <bean class="mb.dnm.service.db.Update">
                                <property name="description"    value="Source DB 에 처리결과(에러)를 Update
한다."/>
                                <property name="input"         value="master_data"/>
                            </bean>
                        </list>
                    </property>
                </bean>
            </list>
        </property>
        <property name="callbacks">
            <description>로깅을 위한 콜백 프로세스</description>
            <list>
                <ref bean="custom_logging"/>
            </list>
        </property>
    </bean>
  </list>
</entry>
```

파일명	config_INTERFACES.xml

```
<bean class="mb.dnm.storage.InterfaceInfo">
    <property name="interfaceId"              value="인터페이스 ID"/>
    <property name="interfaceName"            value="인터페이스명"/>
    <property name="sourceCode"               value="송신시스템 코드"/>
    <property name="targetCode"               value="수신시스템 코드"/>
    <property name="loggingWhenNormal"         value="true"/>
    <property name="loggingWhenError"          value="true"/>
    <property name="querySequence"             value="DB_SRC_MAIN$@{if_id}.SELECT_M,
                                                DB_SRC_MAIN$@{if_id}.SELECT_D,
                                                DB_TGT_MAIN$@{if_id}.INSERT,
                                                DB_SRC_MAIN$@{if_id}.SUCCESS_UPDATE"/>
    <property name="errorQuerySequence"        value="DB_SRC_MAIN$@{if_id}.ERROR_UPDATE"/>
    <property name="serviceId"                value="$P_DB2DB_Master_Detail"/>
    <property name="errorHandlerId"            value="$EH_LOGGING"/>
</bean>
```

파일명	SQL_인터페이스 ID.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="인터페이스 ID">

    <select id="SELECT_M" resultType="java.util.HashMap">
      마스터 Select 쿼리
    </select>

    <select id="SELECT_D" resultType="java.util.HashMap">
      디테일 Select 쿼리
    </select>

    <insert id="INSERT">
```

```
      Insert 쿼리
    </insert>

    <update id="SUCCESS_UPDATE">
      성공결과 Update 쿼리
    </update>

    <update id="ERROR_UPDATE">
      실패결과 Update 쿼리
    </update>

</mapper>
```

◎ FTP to DB

1. InterfaceInfo 의 FileTemplate 설정을 참조하여 Source FTP 서버에서 대상 파일목록을 가져온다.
2. 대상 파일 개수를 확인한다.
3. 파일 쓰기 도중 내용읽기를 방지하기 위해 1 초간 대기한다.
4. 파일목록 개수 만큼 5 ~ 13 과정을 반복한다.
5. Source FTP 서버의 파일을 로컬 서버로 다운로드한다.
6. 다운로드한 파일의 내용을 읽는다.
7. 읽은 파일을 로컬 서버에서 삭제한다.
8. DynamicCode 에 작성된 코드대로 파일을 파싱한다.
9. DB 트랜잭션을 시작한다.
10. 파싱한 데이터를 Target DB Insert
11. Insert 성공한 파일을 FTP 서버의 성공경로로 이동한다.
12. 다음 파일을 처리하기 전 1 초간 일시 대기한다.
13. DB 트랜잭션을 종료한다.

◎ **파일명**	config_SERVICES.xml

```
<entry key="$P_FTP2DB_default">
   <list>
      <bean class="mb.dnm.service.ftp.ListFiles">
         <property name="description"       value="FTP 서버에서 연계할 파일목록을 확인한다."/>
         <property name="sourceAlias"       value="SRC"/>
         <property name="directoryType"     value="REMOTE_SEND"/>
         <property name="output"            value="remote_file_list"/>
      </bean>
      <bean class="mb.dnm.service.general.StopIfInputIsNullOrEmpty">
         <description>연계 대상 파일이 없으면 프로세스를 종료한다.</description>
         <property name="input"             value="remote_file_list"/>
         <property name="output"            value="no_logging_flag"/>
         <property name="outputValue"       value="Y"/>
      </bean>
      <bean class="mb.dnm.service.general.GetSize">
         <description>연계 대상 파일의 개수를 확인한다.</description>
         <property name="input"           value="remote_file_list"/>
         <property name="output"          value="tx_count"/>
      </bean>
      <bean class="mb.dnm.service.general.PauseProcess">
         <description>파일이 쓰여지는 도중 읽는 것을 방지하기 위해 프로세스를 잠시 멈춘다</description>
         <property name="millisecond"       value="1000"/>
      </bean>
      <bean class="mb.dnm.service.general.IterationGroup">
         <description>FTP 서버의 대상파일의 수만큼 IterationGroup 의 services 를 반복한다.</description>
         <property name="input"                          value="remote_file_list"/>
         <property name="iterationInputName"               value="remote_file"/>
         <property name="fetchSize"                      value="1"/>
         <property name="createNewContextEachLoop"         value="true"/>
         <property name="continueDespiteError"             value="true"/>
         <property name="passSessionToContexts"            value="true"/>
         <property name="services">
            <description>연계대상파일의 수만큼 아래에 정의된 프로세스를 순서대로 반복</description>
            <list>
               <bean class="mb.dnm.service.general.OutputCustomData">
                  <description>로그 내용 중 연계 데이터 건수를 지정하기 위한 변수를 output</description>
                  <property name="output"       value="tx_count"/>
                  <property name="castType"     value="int"/>
```

```
                    <property name="customData"     value="1"/>
                </bean>
                <bean class="mb.dnm.service.ftp.DownloadFiles">
                    <property name="description"    value="연계 대상 파일을 Indigo 연계서버(로컬)로
다운로드한다."/>
                    <property name="input"         value="remote_file"/>
                    <property name="sourceAlias"    value="SRC"/>
                    <property name="directoryType"  value="LOCAL_TEMP"/>
                    <property name="output"        value="local_file"/>
                </bean>
                <bean class="mb.dnm.service.file.ReadFile">
                    <property name="description"    value="파일을 읽는다."/>
                    <property name="sourceAlias"    value="SRC"/>
                    <property name="input"         value="local_file"/>
                    <property name="output"        value="file_data"/>
                </bean>
                <bean class="mb.dnm.service.file.DeleteFiles">
                    <property name="description"     value="읽은 파일을 연계서버(로컬)에서 삭제한다."/>
                    <property name="input"          value="local_file"/>
                </bean>
                <bean class="mb.dnm.service.dynamic.ExecuteDynamicCode">
                    <description>파일의 데이터를 파싱 후 file_data 라는 이름으로 매핑된 데이터를 output
한다.</description>
                    <property name="description"     value="연계 대상 파일을 파싱한다."/>
                </bean>
                <bean class="mb.dnm.service.db.StartTransaction"/>
                <bean class="mb.dnm.service.db.Insert">
                    <property name="description"     value="DB 에 데이터를 Insert 한다."/>
                    <property name="input"          value="file_data"/>
                    <property name="output"         value="select_result"/>
                </bean>
                <bean class="mb.dnm.service.ftp.MoveFiles">
                    <property name="description"    value="연계성공한 파일을 FPT 서버의 성공 경로로 파일을
이동한다."/>
                    <property name="input"         value="remote_file"/>
                    <property name="sourceAlias"    value="SRC"/>
                    <property name="directoryType"  value="REMOTE_SUCCESS"/>
                </bean>
                <bean class="mb.dnm.service.general.PauseProcess">
                    <description>다음 파일을 처리하기 전 프로세스를 일시중지한다.</description>
                    <property name="millisecond"     value="1000"/>
                </bean>
                <bean class="mb.dnm.service.db.EndTransaction"/>
            </list>
        </property>
        <property name="errorHandlers">
            <description>에러 발생 시 처리 프로세스 정의</description>
            <list>
                <bean class="mb.dnm.exeption.handler.ExecuteServiceErrorHandler">
                    <property name="service">
                        <bean class="mb.dnm.service.db.Rollback">
                            <property name="description"    value="DB 트랜잭션을 롤백한다."/>
                        </bean>
                    </property>
                </bean>
                <bean class="mb.dnm.exeption.handler.ExecuteServiceErrorHandler">
                    <property name="service">
                        <bean class="mb.dnm.service.file.DeleteFiles">
                            <property name="description"       value="파일을 연계서버(로컬)에서 삭제한다."/>
                            <property name="input"            value="local_file_list"/>
                        </bean>
                    </property>
                </bean>
                <bean class="mb.dnm.exeption.handler.ExecuteServiceErrorHandler">
                    <property name="service">
                        <bean class="mb.dnm.service.ftp.MoveFiles">
                            <property name="description"    value="FPT 서버의 에러경로로 파일을 이동한다."/>
                            <property name="input"         value="remote_file"/>
                            <property name="sourceAlias"    value="SRC"/>
                            <property name="directoryType"  value="REMOTE_ERROR"/>
                        </bean>
                    </property>
                </bean>
                <bean class="mb.dnm.exeption.handler.ExecuteServiceErrorHandler">
```

```
                    <property name="service">
                        <bean class="mb.dnm.service.general.OutputCustomData">
                            <property name="output"        value="no_logging_flag"/>
                            <property name="customData"    value="Y"/>
                        </bean>
                    </property>
                </bean>
            </list>
        </property>
        <property name="callbacks">
            <description>로깅을 위한 콜백 프로세스</description>
            <list>
                <ref bean="custom_logging"/>
            </list>
        </property>
    </bean>
    <bean class="mb.dnm.service.general.OutputCustomData">
        <property name="output"        value="no_logging_flag"/>
        <property name="customData"    value="Y"/>
    </bean>
  </list>
</entry>
```

파일명	config_INTERFACES.xml

```
<bean class="mb.dnm.storage.InterfaceInfo">
   <property name="interfaceId"          value="인터페이스 ID"/>
   <property name="interfaceName"        value="인터페이스명"/>
   <property name="sourceCode"           value="송신시스템 코드"/>
   <property name="targetCode"           value="수신시스템 코드"/>
   <property name="loggingWhenNormal"     value="true"/>
   <property name="loggingWhenError"      value="true"/>
   <property name="sourceAliases"        value="SRC:FTPClient 템플릿 명"/>
   <property name="querySequence"        value="DB_SRC_MAIN$@{if_id}.INSERT"/>
   <property name="dynamicCodeSequence"   value="@{if_id}.PARSING_FILE"/>
   <property name="serviceId"           value="$P_FTP2DB_default"/>
   <property name="fileTemplates">
      <list>
         <bean class="mb.dnm.access.file.FileTemplate">
            <property name="templateName"       value="FTPClient 템플릿 명"/>
            <property name="remoteSendDir"      value="/송신 디렉터리"/>
            <property name="remoteSuccessDir"    value="/송신 디렉터리/succ/@{YYYYMMDD}"/>
            <property name="remoteErrorDir"     value="/송신 디렉터리/err/@{YYYYMMDD}"/>
            <property name="localTempDir"       value="/app/indigo/FILE_WORK/송신시스템
코드/@{if_id}"/>
            <property name="fileNamePattern"    value="파일명 prefix *"/>
            <property name="type"              value="FILE"/>
            <property name="dataType"          value="STRING"/>
            <property name="charset"           value="파일인코딩"/>
         </bean>
      </list>
   </property>
</bean>
```

파일명	SQL_인터페이스 ID.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="인터페이스 ID">

    <insert id="INSERT">
      Insert 쿼리
    </insert>

</mapper>
```

◎ DB to FTP

1. Source DB Select
2. 데이터 존재여부를 확인한다.
3. DynamicCode 를 사용하여 파일내용을 직접 코드로 작성한다.
4. 로컬에 파일 생성 후 내용을 작성한다.
5. DB 트랜잭션 시작
6. Source DB 에 처리결과를 Update
7. FTP 서버에 파일을 업로드한다.
8. DB 트랜잭션 종료
9. 로컬에 생성했던 파일을 삭제

◎ **파일명**	config_SERVICES.xml

```
<entry key="$P_DB2FTP_default">
   <list>
      <bean class="mb.dnm.service.db.Select">
         <description>Source DB 에서 연계 대상 데이터를 조회한다.</description>
         <property name="output"           value="data_list"/>
      </bean>
      <bean class="mb.dnm.service.general.StopIfInputIsNullOrEmpty">
         <description>연계 대상 데이터가 없으면 프로세스를 종료한다.</description>
         <property name="input"            value="data_list"/>
         <property name="output"           value="no_logging_flag"/>
         <property name="outputValue"       value="Y"/>
      </bean>
      <bean class="mb.dnm.service.general.GetSize">
         <description>데이터의 개수를 확인한다.</description>
         <property name="input"            value="data_list"/>
         <property name="output"           value="tx_count"/>
      </bean>
      <bean class="mb.dnm.service.dynamic.ExecuteDynamicCode">
         <description>파일명(file_name) 과 파일내용(file_content), 업데이트
파라미터(update_param)를 output 한다.</description>
         <property name="description"       value="파일명과 내용을 생성한다."/>
      </bean>
      <bean class="mb.dnm.service.file.WriteFile">
         <property name="description"       value="파일을 생성 후 내용을 작성한다."/>
         <property name="sourceAlias"       value="TGT"/>
         <property name="filenameInput"     value="file_name"/>
         <property name="input"            value="file_content"/>
         <property name="directoryType"     value="LOCAL_TEMP"/>
         <property name="output"           value="file_loc"/>
      </bean>
      <bean class="mb.dnm.service.db.StartTransaction"/>
      <bean class="mb.dnm.service.db.Update">
         <description>Select 한 데이터(data_list)를 파라미터로 사용하여 처리결과를 Update
한다.</description>
         <property name="description"       value="Source DB 에 처리결과를 Update 한다."/>
         <property name="input"            value="update_param"/>
      </bean>
      <bean class="mb.dnm.service.ftp.UploadFiles">
         <property name="description"       value="생성한 파일을 FTP 서버에 업로드 한다."/>
         <property name="sourceAlias"       value="TGT"/>
         <property name="input"            value="file_loc"/>
```

```
            <property name="output"            value="remote_file_loc"/>
            <property name="directoryType"      value="REMOTE_RECEIVE"/>
        </bean>
        <bean class="mb.dnm.service.db.EndTransaction"/>
        <bean class="mb.dnm.service.file.DeleteFiles">
            <description>연계 완료 후 로컬에 생성했던 파일을 삭제한다.</description>
            <property name="ignoreError"       value="true"/>
            <property name="input"             value="file_loc"/>
        </bean>
    </list>
</entry>
```

파일명	config_INTERFACES.xml

```
<bean class="mb.dnm.storage.InterfaceInfo"  name="DB2FTP default">
    <property name="interfaceId"          value="인터페이스 ID"/>
    <property name="interfaceName"        value="인터페이스명"/>
    <property name="sourceCode"           value="송신시스템 코드"/>
    <property name="targetCode"           value="수신시스템 코드"/>
    <property name="loggingWhenNormal"     value="true"/>
    <property name="loggingWhenError"      value="true"/>
    <property name="sourceAliases"        value="TGT:JJC"/>
    <property name="querySequence"        value="DB_SRC_MAIN$@{if_id}.SELECT,
DB_SRC_MAIN$@{if_id}.UPDATE"/>
    <property name="dynamicCodeSequence"   value="@{if_id}.MAKE_FILEDATA"/>
    <property name="serviceId"            value="$P_DB2FTP_default"/>
    <property name="errorHandlerId"       value="$EH_DB2FTP"/>
    <property name="fileTemplates">
        <list>
            <bean class="mb.dnm.access.file.FileTemplate">
                <property name="templateName"       value="FTPClient 템플릿 명"/>
                <property name="localTempDir"
value="/app/indigo/FILE_WORK/송신시스템 코드/@{if_id}"/>
                <property name="remoteReceiveDir"    value="/sen/cur"/>
                <property name="charset"           value="파일인코딩"/>
            </bean>
        </list>
    </property>
</bean>
```

파일명	SQL_인터페이스 ID.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="인터페이스 ID">

    <select id="SELECT" resultType="java.util.HashMap">
      Select 쿼리
    </select>

    <update id=" UPDATE">
      결과 Update 쿼리
    </update>

</mapper>
```

◎ DynamicCode 작성법

DynamicCode 는 ExecuteDynamicCode 서비스를 통해 실행된다.

mb.dnm.service.dynamic

Class ExecuteDynamicCode

java.lang.Object
  mb.dnm.service.AbstractService
    mb.dnm.service.dynamic.ExecuteDynamicCode

All Implemented Interfaces:

Serializable, Service

```
public class ExecuteDynamicCode
extends AbstractService
implements Serializable
```

DynamicCode 를 실행한다.

```
<!-- InterfaceInfo의 dynamicCodeSequence에 등록된 순서대로 실행하는 경우-->
<bean class="mb.dnm.service.dynamic.ExecuteDynamicCode"/>

<!-- InterfaceInfo의 dynamicCodeSequence에 등록된 특정 codeId를 지정하여 실행하는 경우-->
<bean class="mb.dnm.service.dynamic.ExecuteDynamicCode">
    <property name="codeId"                value="namespace.codeID"/>
</bean>
```

파일명	DNC_인터페이스 ID.dnc

```
#namespace: DynamicCode 의 namespace(어댑터 내에서 고유해야함)

/*

※ input Object 를 가져오는 법 (1) : getInput(ctx, "파라미터 명");
※ input Object 를 가져오는 법 (2) : ctx.getContextParam("파라미터 명");
※ Object value 를 output 하는 법 (1) : setOutputValue(ctx, "파라미터 명", value);
※ Object value 를 output 하는 법 (2) : ctx.addContextParam("파라미터 명", value);
※ input 또는 output 된 Object 를 삭제하는 법 : ctx.deleteContextParam("파라미터 명");

※ 트랜잭션 ID (인터페이스 트랜잭션 별 고유아이디) : String txId = ctx.getTxId();
※ 인터페이스 시작 시간 : Date startTime = ctx.getStartTime();
※ 인터페이스(각 서비스) 종료 시간 : Date endTime = ctx.getEndTime();

※ Log 남기기: org.slf4j.Logger 인터페이스를 변수명 log 로 사용 가능함
※ Process 를 중단하는 법 : ctx.processOn(false);
※ 메시지를 기록하는 법 : ctx.setMsg(StringBuilder msg); 또는 ctx.setMsg(String msg);
※ 메시지를 가져오는 법 : String msg = ctx.getMsg();
※ Context 정보를 가져오는 법 : Map<String, Object> contextInfoMap =
ctx.getContextInformation();

*/

코드아이디는 동일한 namespace 에서 고유해야한다.
#code_id: 코드아이디_1
#{

    //Java 코드를 작성

}#

#code_id: 코드아이디_2
#{

    //Java 코드를 작성

}#
```

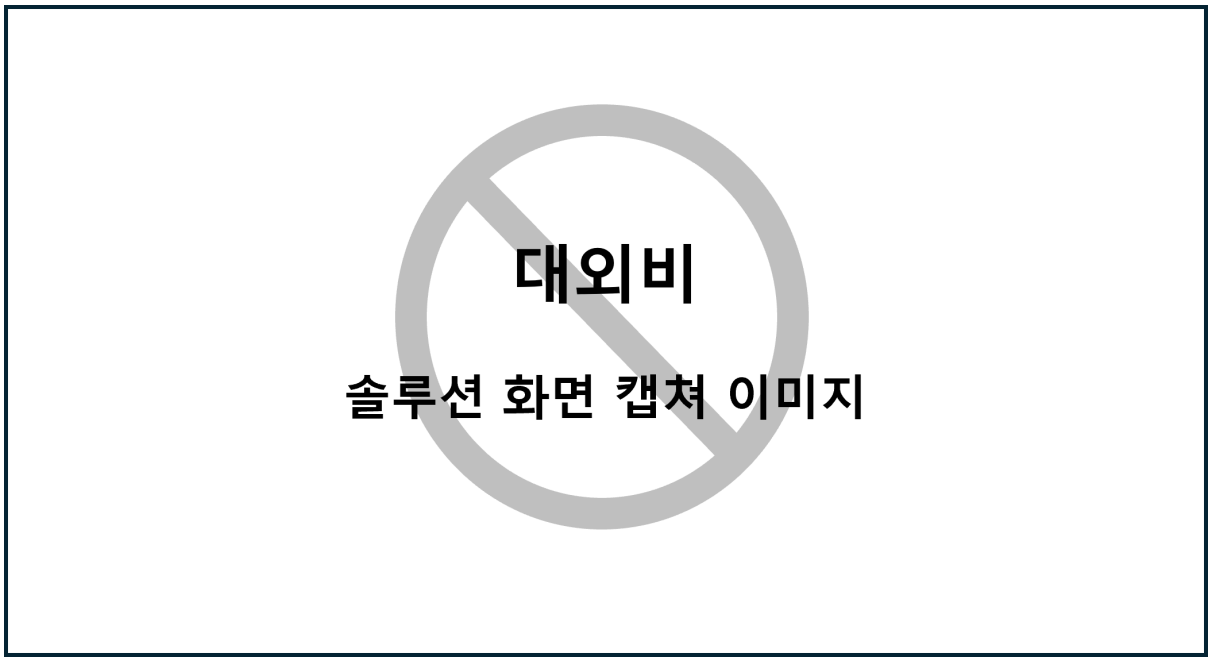
◎ Schedule Job 등록

개발완료한 인터페이스를 주기적인 배치작업으로 실행하기 위해 스케쥴 설정을 할 수 있다. 절차는 다음과 같다.

1. 스케줄 설정

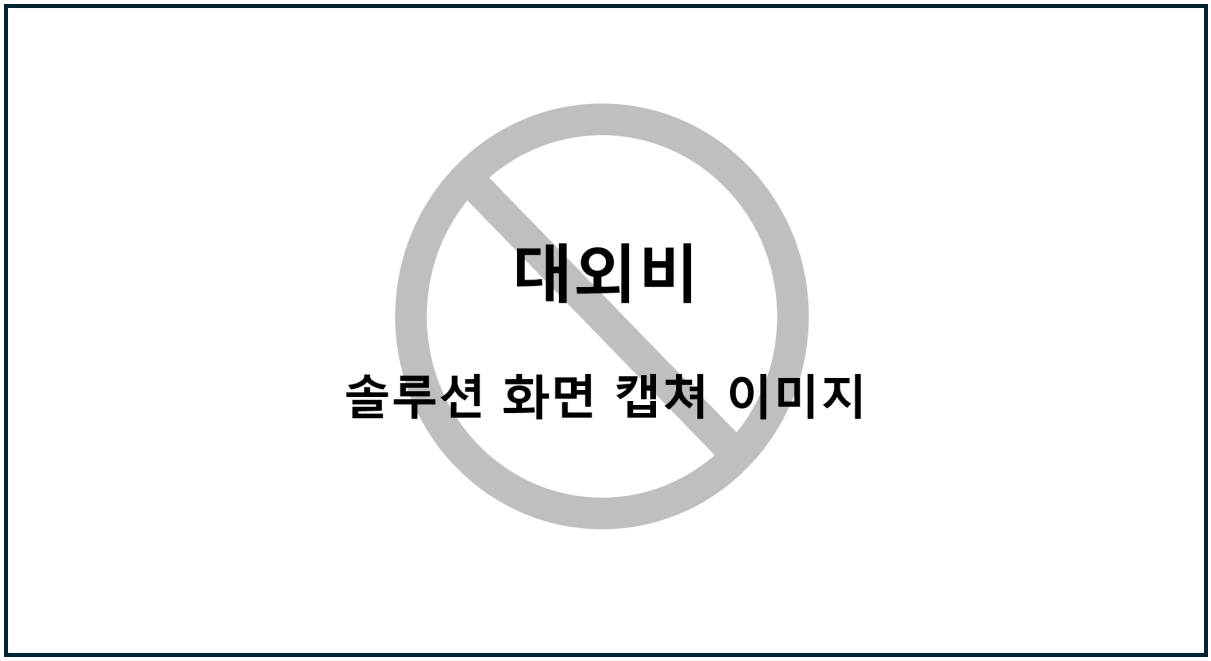
▷IMC(웹 관리콘솔)에서 스케줄 작업을 등록하려는 인터페이스가 포함된 어댑터 관리화면으로 이동합니다.

▷IMC 화면 우측의 컴포넌트 설정-추가-스케줄을 클릭합니다.



▷스케줄 설정화면에서 크론 스케줄 선택합니다.

▷ID 는 scheduler, 목표 객체는 SimpleDispatcher 로 설정한 뒤 Add 버튼을 클릭합니다.



▷실행하려는 인터페이스 ID 를 크론 스케줄 설정 화면의 Arguments 부분에 입력합니다.

▷크론 표현식에 따라 스케줄 시간을 설정 후 저장합니다.