CAMEL-ESB 활용 가이드북

프로젝트 명	CAMEL-ESB 활용 가이드북
문서번호	2018-CAMEL-ESB-활용가이드북-10-2

TORPEDO
TORPEDO CORP.

개정이력

개정번호	내용	시행일자

목차

1 Apache Camel Route

1. Route 기본 구조

Route(<camelContext>) 안에 다양한 components 사용하여 EIP 구현

Spring XML / Java DSL 이용하여 route 설정

[Spring 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-2.5.xsd
        http://camel.apache.org/schema/spring http://camel.apache.org/schema/spring/camel-spring.xsd">

    <bean id="jetty" class="org.apache.camel.component.jetty9.JettyHttpComponent9">
    </bean>

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

        <route>
            <from uri="file:d:/camel/src" />
            <to uri="file:///d:/camel/output" />
        </route>
    </camelContext>

 </beans>
```

[Java DSL]

```
import org.apache.camel.builder.RouteBuilder;

public class routeBuilder extends RouteBuilder {

    public void configure() {

            from("file:d:/camel/src")
                .to("file:d:/camel/output");
    }
}
```

org.apache.camel.builder.RouterBuilder extends + configure 메소드 override하여 구현

configure() 안에 from, to 설정하여 route 설정

※ 이 가이드북에서는 Spring XML 이용하여 설명

```
<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">

    <bean id="jetty" class="org.apache.camel.component.jetty9.JettyHttpComponent9">
    </bean>

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

Route 안에서 필요한 library, bean 정의

CXF 등 다른 component 이용할 때는 필요한 library 추가

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

    <route>
        <from uri="file:d:/camel/src" />
        <to uri="file:///d:/camel/output" />
    </route>
</camelContext>
```

camelContext는 Apache Camel에서의 runtime system

<camelContext> 안에서 routes, component 정의, 연결하여 사용

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

    <route>
        <from uri="file:d:/camel/src" />
        <to uri="file:///d:/camel/output" />
    </route>
</camelContext>
```

<camelContext> 하위 <route> 이용하여 정의

<route> 안에 from(starting 디렉토리), to(target 디렉토리) 설정

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

    <route>
        <from uri="file:d:/camel/src?delay=2000" />
        <to uri="file:///d:/camel/output" />
    </route>
</camelContext>
```

Option 사용할 때는 물음표(?) 이용하여 경로와 option 구분

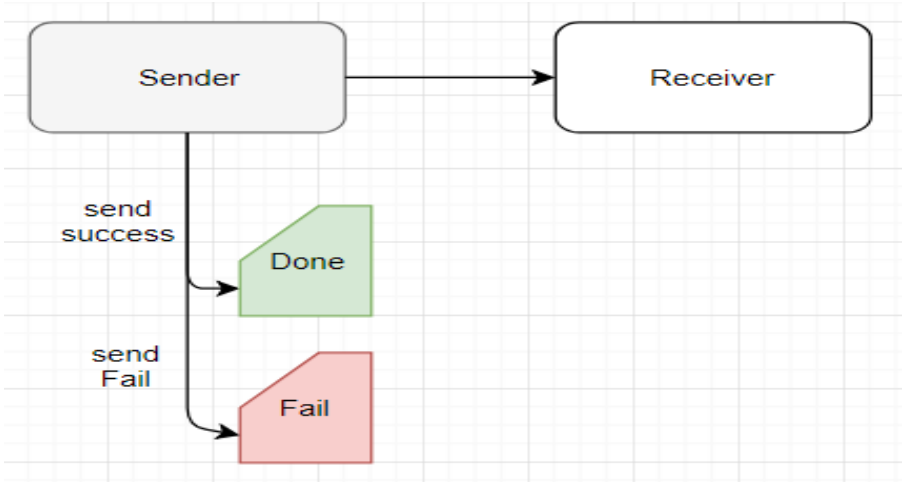
```
<?xml version="1.0" encoding="UTF-8"?>

<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">

   <bean id="jetty" class="org.apache.camel.component.jetty9.JettyHttpComponent9">
   </bean>

   <camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
       <route>
         <from uri="file:d:/camel/src?sortBy=file:length;file:modified&amp;delay=2000"/>
         <to uri="file:///d:/camel/output"/>
       </route>

       <route>
         <from uri="file:d:/camel/input?delay=2000"/>
         <to uri="file:///d:/camel/final"/>
       </route>
   </camelContext>

</beans>
```

2개 이상의 <route> 설정 가능

Option 2개 이상 사용할 때는 & 이용하여 구분

2 File Component

2.1 move·moveFailed



2.1.1 move

Consumer쪽에서 사용가능

File processing 이후 이동할 file 경로 설정

(default) processing 이후 .done subdirectory로 file 이동

```
<?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">

    <bean id="jetty" class="org.apache.camel.component.jetty9.JettyHttpComponent9">
    </bean>

    <camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
        <route>
          <from uri="file:d:/camel/src?move=success&amp;delay=2000"/>
          <to uri="file://d:/camel/output"/>
        </route>
    </camelContext>

 </beans>
```

[route 설명]

from(input) 폴더 = d:/camel/src

processing 후 이동할 폴더 = d:/camel/src/success

to 폴더 = d:/camel/output

from 폴더에 존재하는 file을 polling하여 processing한 file들은 src 밑의 success 폴더로 이동, to uri인 d:/camel/output으로 복사

2.1.2 moveFail

Consumer쪽에서 사용가능

Processing 과정 중 문제가 생겨 fail 되었을 때 이동할 directory 설정

(default) .error subdirectory로 file 이동

```
<?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">

    <bean id="jetty" class="org.apache.camel.component.jetty9.JettyHttpComponent9">
    </bean>

    <camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
        <route>
          <from uri="file:d:/camel/src?moveFailed=fail&amp;delay=2000"/>
          <to uri="file://d:/camel/output"/>
        </route>
    </camelContext>

 </beans>
```

[route 설명]

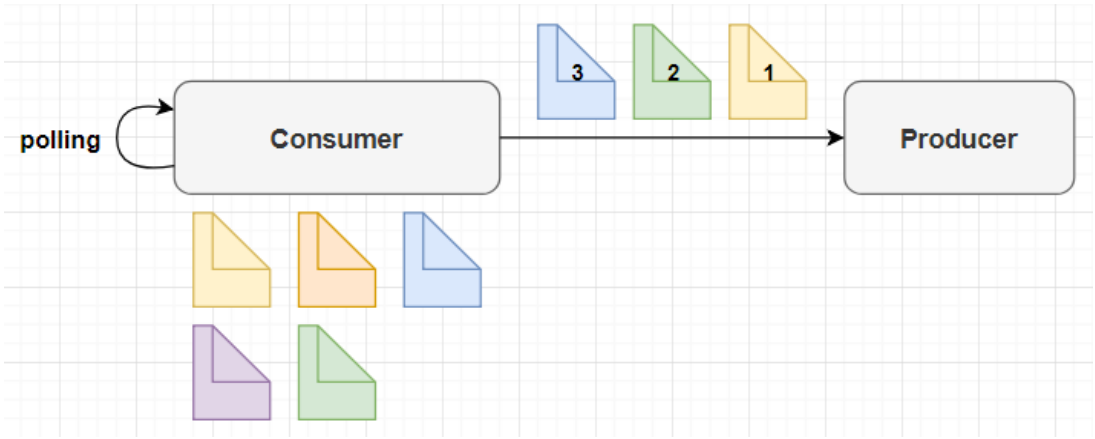
from(input) 폴더 = d:/camel/src

processing 과정 중 error가 발생하게 되면 이동할 폴더 = d:/camel/src/fail

to 폴더 = d:/camel/output

from 폴더에 존재하는 file들을 polling하여 processing하는 과정 중 문제가 생겨 processing을 올바르게 마치지 못한 file들은 src 하위 fail 폴더로 이동

2.2 sortBy·sorter



2.2.1 sortBy

Consumer쪽에서만 사용가능

File Language 이용하여 설정

```
<?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">

    <bean id="jetty" class="org.apache.camel.component.jetty9.JettyHttpComponent9">
    </bean>

    <camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
        <route>
          <from uri="file:d:/camel/src?sortBy=file:length&amp;delay=2000"/>
          <to uri="file://d:/camel/output"/>
        </route>
    </camelContext>

 </beans>
```

[route 설명]

d:/camel/src 폴더의 file들을 polling하되, file 크기 이용하여 polling 순서 결정

```
<?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">

    <bean id="jetty" class="org.apache.camel.component.jetty9.JettyHttpComponent9">
    </bean>

    <camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
        <route>
          <from uri="file:d:/camel/src?sortBy=file:length;file:modified&amp;delay=2000"/>
          <to uri="file://d:/camel/output"/>
        </route>
    </camelContext>

 </beans>
```

[route 설명]

d:/camel/src 폴더에서 file을 polling하되, file 크기로 polling 순서 결정, 만약 file 크기가 같다면 2차 순위인 file 최근 수정날짜로 polling 순서 결정

· syntax: "sortBy=group1;group2;group3; …."

· 각 group마다 세미콜론(;) 이용하여 구분

· 2개 이상의 group을 같이 설정함으로써 차선 순위 설정

· reverse

- polling order 역순으로 진행
- 각 group마다 개별적으로 reverse 선언 가능
- (주의) reverse 사용할 때는 맨 앞에 선언

```
<?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">

    <bean id="jetty" class="org.apache.camel.component.jetty9.JettyHttpComponent9">
    </bean>

    <camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
        <route>
          <from uri="file:d/camel/src?sortBy=file:name&amp;delay=2000" />
          <to uri="file://d:/camel/output/"/>
        </route>

        <route>
          <from uri="file:d/camel/src?sortBy=reverse:file:name&amp;delay=2000" />
          <to uri="file://d:/camel/output/"/>
        </route>
    </camelContext>

 </beans>
```

[route 설명]

1번째 route: file 이름 이용하여 sorting (A…Z순)

2번째 route: file 이름 역순으로 sorting (Z…A순)

※ File Language

file component에서 file header, body에 존재하는 information 이용하여 작업할 때 사용

Simple language에 통합(Camel 2.2 이후) → <simple> 구문에 file language 사용 가능

ex)

file:name: 파일 이름

file:length: 파일 크기

file:modified: 파일 최근 수정 날짜

file:ext: 파일 확장자

date:file:yyyyMMdd : 파일 날짜(date)

※ Simple Language

${} 이용

Variables, operator support 이용하여 표현

ⅰ. Variables

Message 안에 있는 body, header content 이용

In 접두사, ${} 이용하여 접근 ex) in.body, ${body}, in.header, ${in.header} …

File language를 simple language로 사용 가능

ex)

camelId: CamelContext 이름

in.body : input body

out.body : output body

in.header : input header

out.header : output header

ⅱ. Operator support

Single operator만 허용

왼쪽 변수는 ${}로 묶어야 한다.

오른쪽 변수는 ``로 묶은 String, null value, constant value, ${ } 이용한 expression 모두 사용 가능

Operator 양쪽에는 공백이 존재해야 한다.

ex)

==: 동등 연산자

<= , >=, >, < : 비교 연산자

contains, not contains: value 포함 유무

starts with: 왼쪽 변수가 오른쪽 value로 시작하는지 판단 유무

ends with: 왼쪽 변수가 오른쪽 value로 끝나는지 판단 유무

example:)

${in.header.fileName} ends with 'sample'

"ends with" 각 양쪽에는 공백이 존재해야 한다.

Input header에 있는 filename 변수가 sample로 끝나는지에 대해서 알 수 있는 simple language

2.2.2 sorter

File이 processed 되기 전 Camel에서 sorting할 수 있도록 제공

java.util.Comparator<org.apache.camel.component.file.GenericFile> 이용하여 구현

Consumer에서만 사용 가능

File 중 특정 부분만 추출하여 sorting하고자 할 때 유용

```
package kr.co.bizframe.camel;

import java.util.Comparator;
import org.apache.camel.component.file.GenericFile;

public class myComparator<T> implements Comparator<GenericFile<T>> {

    @Override
    public int compare(GenericFile<T> f1, GenericFile<T> f2){

        String file1=f1.getFileName();
        String file2=f2.getFileName();

        return file1.compareTo(file2);
    }
}
```

Route에서 sorter를 이용하기 위한 class

file간 비교를 통해 sorting(=polling 순서) 결정할 method 정의

```
import java.util.Comparator;
import org.apache.camel.component.file.GenericFile;

public class myComparator<T> implements Comparator<GenericFile<T>> {

    @Override
    public int compare(GenericFile<T> f1, GenericFile<T> f2){

        String file1=f1.getFileName();
        String file2=f2.getFileName();

        return file1.compareTo(file2);
    }
}
```

java.util.Comparator 클래스를 implements + compare 메소드 override하여 구현

org.apache.camel.component.file.GenericFile : Camel에서 사용되는 file content를 사용하는데 필요한 package

(주의) sortBy와 sorter option을 같이 사용할 수 없다.

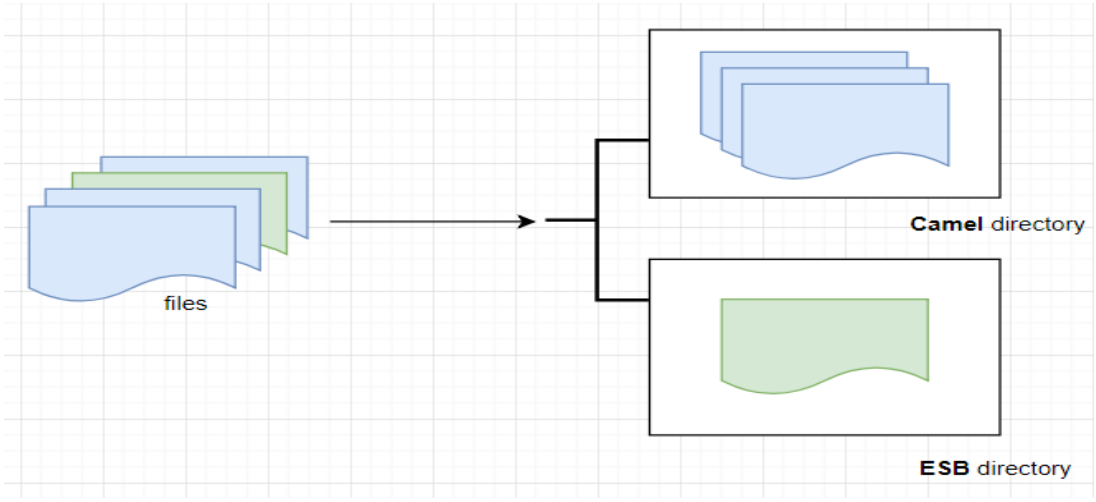
```
<?xml version="1.0" encoding="UTF-8"?>

 <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">

    <bean id="jetty" class="org.apache.camel.component.jetty9.JettyHttpComponent9">
    </bean>

    <bean id="myComparator" class="kr.co.bizframe.camel.myComparator" />

    <camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
        <route>
          <from uri="file://d:/camel/input/src?sorter=#myComparator&amp;delay=2000" />
          <to uri="file://d:/camel/output" />
        </route>
    </camelContext>

 </beans>
```

[route 설명]

Comparator를 이용한 sorting class를 "myComparator"라는 이름으로 bean 등록

Route에 # annotation 붙여 등록한 bean 사용하겠다 정의

2.3 Content Based Router (choice·when·otherwise)



Message의 내용에 따라 multiple point로 routing

<choice>: 조건문이 나온다는 것을 Camel에게 알려준다.

<when>: Java에서의 if문

<otherwise>: Java에서의 else문

(주의) <otherwise> 없이 <when>만 단독으로 사용한다면 오류 생길 가능성 有

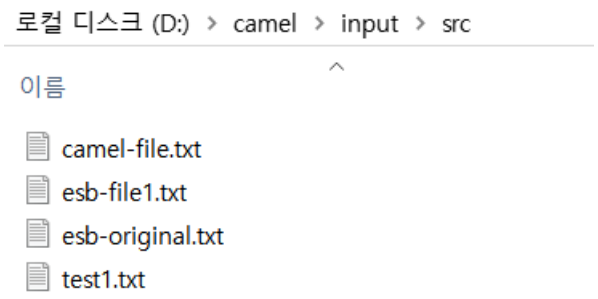
```
<?xml version="1.0" encoding="UTF-8"?>

 <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">

    <bean id="jetty" class="org.apache.camel.component.jetty9.JettyHttpComponent9">
    </bean>

    <camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
        <route>
          <from uri="file:d:/camel/input/src?delay=2000"/>
          <choice>
            <when>
                <simple>${in.header.CamelFileName} starts with 'esb'</simple>
                <to uri="file://d:/camel/output/filter"/>
            </when>
            <otherwise>
                <to uri="file://d:/camel/output/nonFilter"/>
            </otherwise>
          </choice>
        </route>
    </camelContext>

 </beans>
```

[route 설명]

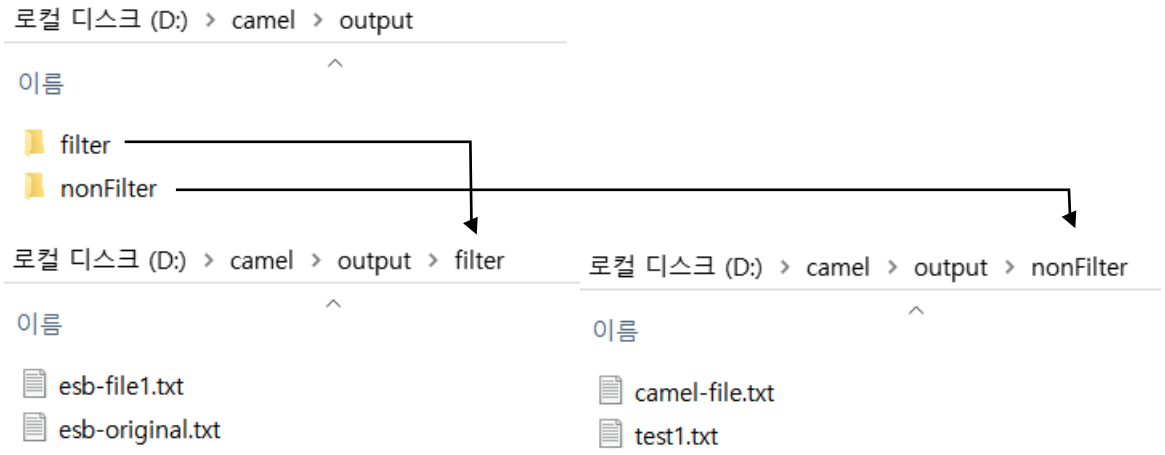
file name이 esb로 시작하는 files → d:/camel/output/filter로 이동

esb로 시작하지 않는 그 외 files → d:/camel/output/nonFilter로 이동

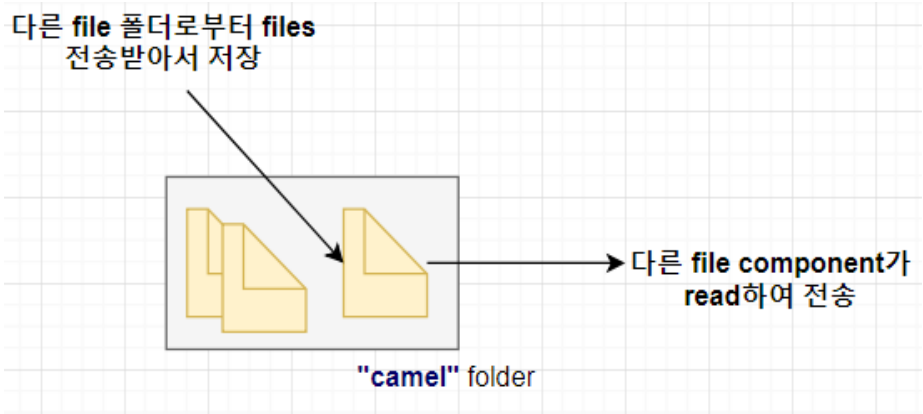
<전송 전>



<전송 후>



2.4 readLock



하나의 file에 두 개 이상의 component가 접근하여 polling 하려는 경우

ex)

1번째 file component - 다른 폴더로부터 file을 읽어와 "camel" 폴더에 저장

2번째 file component – "camel" 폴더로부터 file을 읽어 다른 폴더로 전송

➔ 하나의 폴더에 2개 이상의 component가 접근하면서 같은 파일에 대해 동시에 진행하게 되면 error 발생 가능성 有

Consumer쪽에서만 사용가능

· Option

none (default): readLock 설정 X

markerFile: marker file 이용하여 processing 진행 중인 file은 lock, FTP component에서는 사용 X

changed: file 크기/수정날짜가 작성되는 timestamp 이용하여 file의 현재 사용 유무 판단, 다른 option에 비해 속도가 느리지만 가장 안전

```
<?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">

    <bean id="jetty" class="org.apache.camel.component.jetty9.JettyHttpComponent9">
    </bean>

    <camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
        <route>
          <from uri="file:d:/camel/input/src?delay=2000"/>
          <to uri="file://d:/camel/output"/>
        </route>
        <route>
          <from uri="file:d:/camel/output?readLock=markerFile&amp;delay=2000"/>
          <to uri="file://d:/camel/output2"/>
        </route>
    </camelContext>

 </beans>
```

[route 설명]

·1번째 route

from = d:/camel/input/src

to = d:/camel/output

·2번째 route

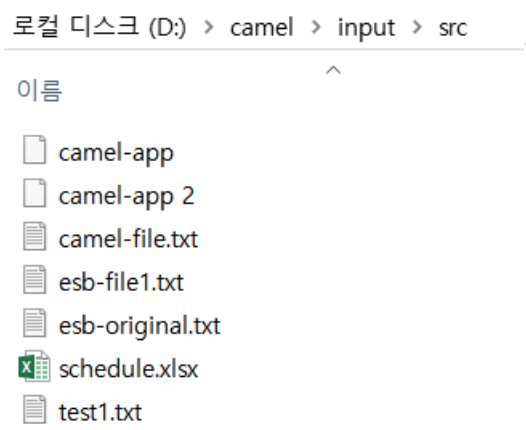
from = d:/camel/output

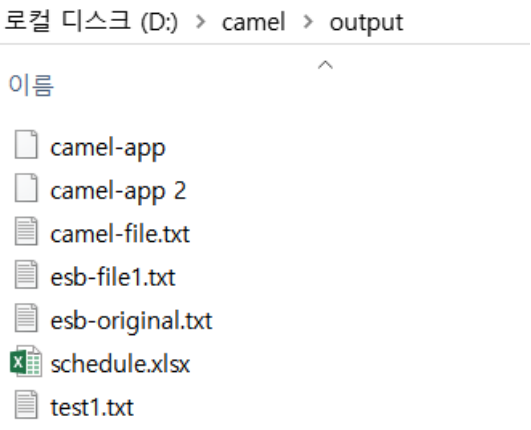
to = d:/camel/output2

→ 1번째 route의 to와 2번째 route의 from이 서로 동일한 위치, 1번째 route에서 file을 write하는 도중 2번째 route에서 같은 파일을 read하려면 문제 발생 가능성 有

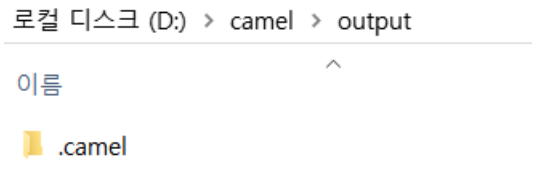
⇒ "readLock=markerFile" 이용하여 현재 processing 되고 있는 file은 lock 상태로 유지해 다른 작업할 수 없도록 설정

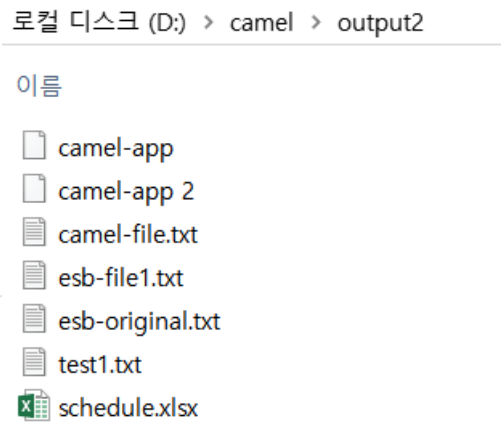
<1번째 route 전송 전/후>





<2번째 route 전송 전/후>





2.5 Filter

특정 파일만 추출

org.apache.camel.component.file.GenericFileFilterFiler 이용하여 구현

```
package kr.co.bizframe.camel;

import org.apache.camel.component.file.GenericFileFilter;
import org.apache.camel.component.file.GenericFile;

public class myFilter implements GenericFileFilter{

    public boolean accept(GenericFile pathname) {

        return !pathname.getFileName().startsWith("camel");
    }
}
```

GenericFileFilter implements + accept() 메소드 override하여 구현

true/false를 return 함으로써 현재 검사하는 file을 skip할 것인지, polling할 것인지 결정

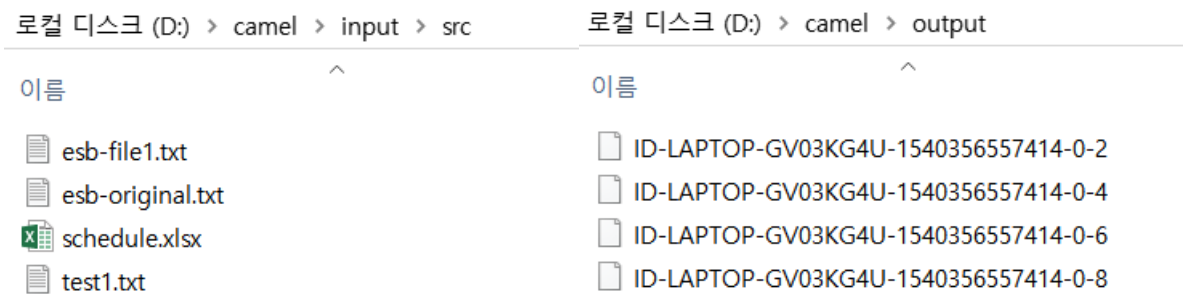
```
<?xml version="1.0" encoding="UTF-8"?>

 <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">

    <bean id="jetty" class="org.apache.camel.component.jetty9.JettyHttpComponent9">
    </bean>

    <bean id="myFilter" class="kr.co.bizframe.camel.myFilter" />

    <camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
        <route>
          <from uri="file://d:/camel/input/src?filter=#myFilter&amp;delay=2000" />
          <to uri="file://d:/camel/output" />
        </route>
    </camelContext>

 </beans>
```

org.apache.camel.component.file.GenericFileFilterFiler 이용하여 구현한 filter class를 bean으로 정의

정의한 bean을 # notation 이용하여 사용

2.6 HTTP 통신

2.6.1 Sender/Receiver간 파일 이름 동일하게 전송



아무런 option 없이 HTTP 통신을 하게 된다면 file 전송에는 문제가 없지만 기존의 file 이름이 그대로 유지되지 않는다.

HTTP에서 정해진 형식대로 전송

→ 해결: <setHeader> 이용하여 file name 지정

<Sender route>

```
<?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">

    <bean id="jetty" class="org.apache.camel.component.jetty9.JettyHttpComponent9">
    </bean>

    <camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
        <route>
          <from uri="file:d:/camel/input/src?delay=2000"/>
          <setHeader headerName="fileName">
            <simple>${in.header.CamelFileName}</simple>
          </setHeader>
          <to uri="http://localhost:10040/myService/" />
        </route>
    </camelContext>

 </beans>
```

```
<setHeader headerName="fileName">
  <simple>${in.header.CamelFileName}</simple>
</setHeader>
```

Message header에 "filename"이라는 이름을 가진 변수에 파일 이름(in.header.CamelFileName)을 넣어준다.

headerName은 사용자가 원하는 이름 설정

Simple language, file language 이용하여 원하는 값을 insert

<Receiver route>

```
<?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">

    <bean id="jetty" class="org.apache.camel.component.jetty9.JettyHttpComponent9">
    </bean>

    <camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
        <route>
          <from uri="jetty://http://localhost:10040/myService/" />
          <to uri="file://d:/camel/output/src?fileName=${in.header.fileName}"/>
        </route>
    </camelContext>

 </beans>
```

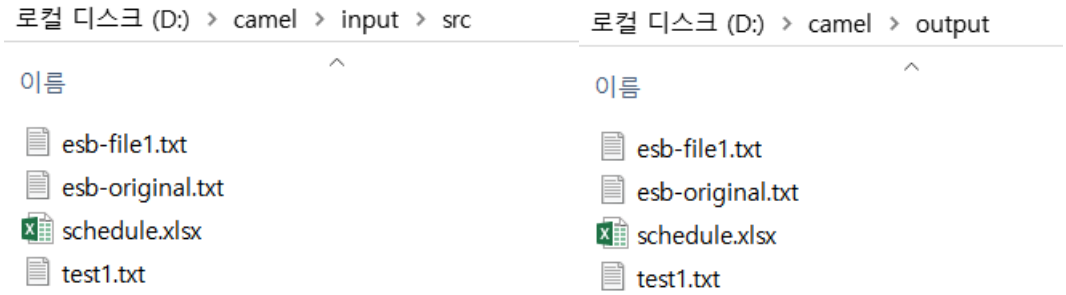
```
<to uri="file:d:/camel/output/src?fileName=${in.header.fileName}"/>
```

filename 옵션 이용하여 filename 지정

Header 안에 "filename"라는 이름을 가진 변수 value를 가져온다.

ex) consumer 쪽에서 fileName이 아닌 "camels"라는 이름으로 넣었다면 in.header.camels이라고 작성해야 한다.

<전송 전/후>

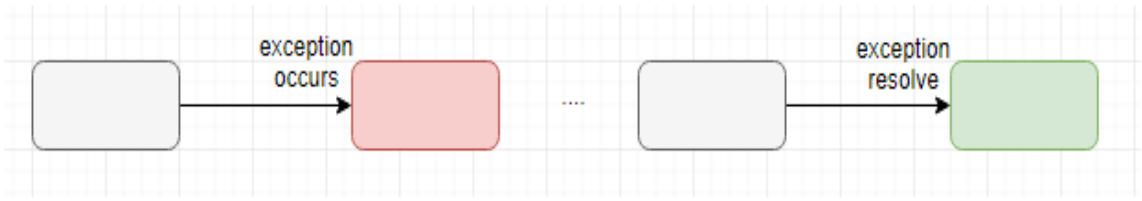


3 Error Handling

3.1 Error handling

3.1.1 Type

1-1. Recoverable error

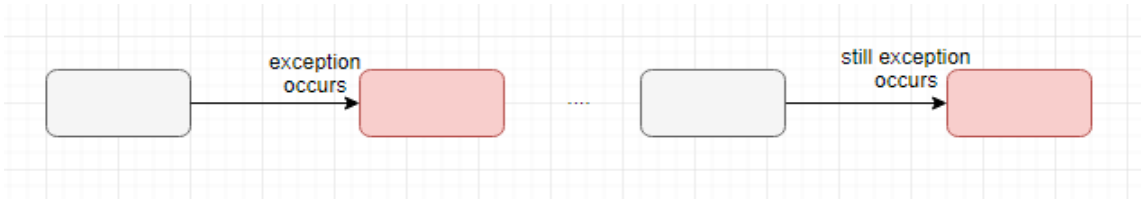


일시적인 오류

전송 시도를 여러 번 했을 때 해결될 수 있는 error

ex) network exception (java.io.IOException)

1-2. Irrecoverable error



시간이 지나 여러 번 전송해도 계속해서 일어나는 error

ex) SQLException

2. Non-transactional

Camel의 error handling 이용하여 처리할 수 있는 error type

3.1.2 Error 발생하는 시점

Camel에서 **routing** / message를 **processing** 하는 과정에서 발생

3.1.3 Error Handing 범위

1. Global

<camelContext>에 적용

<camelContext> 안에 존재하는 모든 routes에서 사용 가능

```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring" errorHandlerRef="myDefault">
    <errorHandler id="myDefault" type="DefaultErrorHandler">
        <redeliveryPolicy maximumRedeliveries="3" retryAttemptedLogLevel="WARN" />
    </errorHandler>

    <route>
      <from uri="file:d:/camel/input/src?delay=2000"/>
      <to uri="http://localhost:10040/myService/" />
    </route>

</camelContext>
```

2. Route

<route>에 적용

단 하나의 route 안에서만 적용

```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
    <errorHandler id="myDefault" type="DefaultErrorHandler">
        <redeliveryPolicy maximumRedeliveries="3" retryAttemptedLogLevel="WARN" />
    </errorHandler>

    <route errorHandlerRef="myDefault">
      <from uri="file:d:/camel/input/src?delay=2000"/>
      <to uri="http://localhost:10040/myService/" />
    </route>

</camelContext>
```

errorHandlerRef 통하여 errorhandler 설정하여 사용

3.2 Try/Catch/Finally

<doTry>, <doCatch>, <doFinally>

→ 앞쪽에 'do'를 붙여서 Java 문법과의 혼동 방지

Camel 2.0 이후 버전부터 사용 가능

Try/catch/finally 사용하게 된다면, 기존 camel의 error handler인 DefaultErrorHandler, onException과 같이 사용할 수 X

```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-2.5.xsd
        http://camel.apache.org/schema/spring http://camel.apache.org/schema/spring/camel-spring.xsd">

    <bean id="jetty" class="org.apache.camel.component.jetty9.JettyHttpComponent9">
    </bean>

    <camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
        <route>
          <from uri="file://d:/camel/input/src?move=done&amp;moveFailed=fail&amp;delay=2000" />
          <doTry>
            <to uri="http://localhost:9098/myService/" />
            <doCatch>
                <exception>java.lang.Exception</exception>
                <to uri="file://d:/camel/output/exception" />
            </doCatch>
            <doFinally>
                <to uri="file://d:/camel/output/finally" />
            </doFinally>
          </doTry>
        </route>
    </camelContext>

 </beans>
```

[route 설명]

route 수행하는 과정 중, java.lang.Exception이 발생하게 되면 to 경로인 http://localhost:9098/myService가 아닌 d:/camel/output/exception으로 이동, 이후 최종적으로 <doFinally> 실행되어 d:/camel/output/finally 폴더로도 이동

```xml
<doTry>
    <to uri="http://localhost:9098/myService/" />
    <doCatch>
        <exception>java.lang.Exception</exception>
        <to uri="file://d:/camel/output/exception" />
    </doCatch>
    <doFinally>
        <to uri="file://d:/camel/output/finally" />
    </doFinally>
</doTry>
```

<doTry> 구문 안에 <doCatch>, <doFinally> 구문을 넣어야 한다.

```xml
<doTry>
  <to uri="http://localhost:9098/myService/" />
  <doCatch>
      <exception>java.lang.Exception</exception>
      <to uri="file://d:/camel/output/exception" />
  </doCatch>
  <doFinally>
      <to uri="file://d:/camel/output/finally" />
  </doFinally>
</doTry>
```

<doCatch> 안에 한 개 이상의 exception 정의 가능

<doFinally>은 생략 가능

<전송 전>

로컬 디스크 (D:) > camel > input > src

이름

esb-file1.txt
esb-original.txt
schedule.xlsx

<routing 중 java.lang.Exception 발 생 후 전송>

로컬 디스크 (D:) > camel > output

이름

exception
finally

로컬 디스크 (D:) > camel > output > exception

이름

esb-file1.txt
esb-original.txt
schedule.xlsx

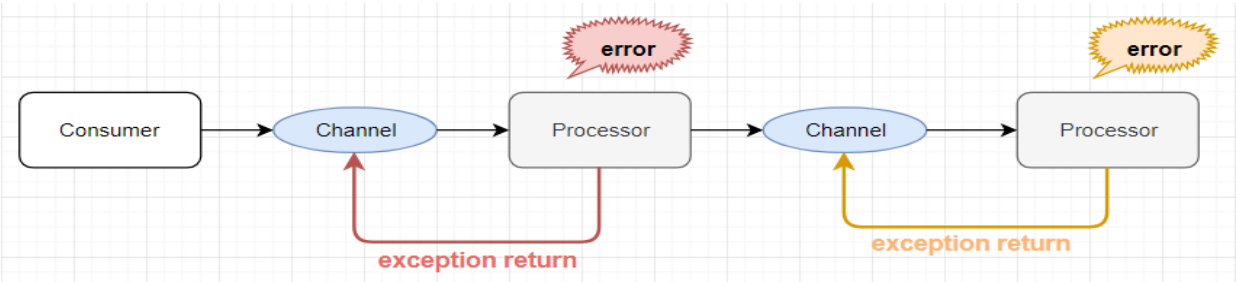
로컬 디스크 (D:) > camel > output > finally

이름

esb-file1.txt
esb-original.txt
schedule.xlsx

3.3 DefaultErrorHandler

3.3.1 Caller에게 exception 알려주는 시점



Route에서 각 node 사이마다 channel 존재

Exception이 발생한다면, exception이 발생한 processor 전 부분에서 가장 가까이에 있는 channel에서 exception을 알려준다.

3.3.2 DefaultErrorHandler

Camel 2.0의 default error handler

· 특징

- Error handling X
- 재전송 X
- Dead letter queue X

(default) routing 과정에서 exception이 발생한다면, 재전송이나 error handling 없이 caller에게 exception을 알려준 후 routing 바로 종료

```
<?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">

    <bean id="jetty" class="org.apache.camel.component.jetty9.JettyHttpComponent9">
    </bean>

    <camelContext id="camel" errorHandlerRef="myDefault" xmlns="http://camel.apache.org/schema/spring">
        <errorHandler id="myDefault" type="DefaultErrorHandler">
        </errorHandler>

        <route>
          <from uri="file://d:/camel/input/src?move=done&amp;moveFailed=fail&amp;delay=2000" />
          <to uri="http://localhost:9098/myService/" />
        </route>
    </camelContext>

 </beans>
```

[route 설명]

"myDefault"라는 이름의 DefaultErrorHandler 정의

<camelContext>에서 errorHandlerRef="myDefault" 설정

→ 모든 routes에 사용할 수 있는 context-scope 단위로 설정

3.3.3 Redelivery

Option	Type	Default	Description
MaximumRedeliveries	int	0	재전송 횟수 (-1 : 성공할 때까지 무한 재전송)
RedeliveryDelay	long	1000	재전송 시도 사이의 delay 시간 (고정값)
MaximumRedeliveryDelay	long	60000	상한 redelivery 시간
AsyncDelayRedelivery	boolean	False	Camel의 asynchronous delayed redelivery 사용 여부
BackoffMultiplier	double	2.0	Starting delay 시간을 제외한 재전송 delay 시간
CollisionAvoidanceFactor	double	0.15	Random delay offset
DelayPattern	String	-	Group 별로 고정된 delay 시간
RetryAttemptedLogLevel	LoggingLevel	DEBUG	재전송 시도가 실행되었을 때 log level
RetriesExhaustedLogLevel	LoggingLevel	ERROR	재전송 시도가 실패되었을 때 log level
LogStackTrace	boolean	true	모든 재전송 시도가 실패되었을 때의 stackTrace

			logged 여부
LogRetryStackTrace	boolean	false	Delivery가 실패했을 때의 stackTrace logged 여부
LogRetryAttempted	boolean	true	재전송 시도 logged 여부
LogExhausted	boolean	true	모든 재전송이 실패되었을 때의 logged 여부
LogHandled	boolean	false	Handled exception의 logged 여부

(※ 표 : DefaultErrorHandler를 포함한 모든 error handler의 redelivey option으로 사용 가능)

```
<?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">

    <bean id="jetty" class="org.apache.camel.component.jetty9.JettyHttpComponent9">
    </bean>

    <camelContext id="camel" errorHandlerRef="myDefault" xmlns="http://camel.apache.org/schema/spring">
        <errorHandler id="myDefault" type="DefaultErrorHandler">
            <redeliveryPolicy maximumRedeliveries="3" retryAttemptedLogLevel="WARN" />
        </errorHandler>

        <route>
          <from uri="file://d:/camel/input/src?move=done&amp;moveFailed=fail&amp;delay=2000" />
          <to uri="http://localhost:9098/myService/" />
        </route>
    </camelContext>

 </beans>
```

[route 설명]

Type은 DefaultErrorHandler, "myDefault"라는 이름을 가진 error handler 정의

재전송에 관하여 최대 재전송 횟수=2, 재전송 시도에 대한 log level은 WARN 단계로 설정

3.3.4 Exchange header

Camel에서 재전송할 때만 존재하는 정보

Header	Type	Description
Exchange.REDELIVERY_COUNTER	int	현재 재전송 시도 횟수
Exchange.REDELIVERED	boolean	Exchange가 재전송 되고 있는지 여부
Exchange.REDELIVERY_EXHAUSTED	boolean	Exchange의 exhausted 여부

3.4 onException

3.4.1 onException

특정 exception에 대하여 처리하고 싶을 때 정의

<onException> 구문으로 이용

주로 Error handler와 같이 결합하여 사용

```
<?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">

    <bean id="jetty" class="org.apache.camel.component.jetty9.JettyHttpComponent9">
    </bean>

    <camelContext id="camel" errorHandlerRef="myDefault" xmlns="http://camel.apache.org/schema/spring">
        <errorHandler id="myDefault" type="DefaultErrorHandler">
            <redeliveryPolicy maximumRedeliveries="3" retryAttemptedLogLevel="WARN" />
        </errorHandler>
                                                                                    defaultErrorHandler
        <onException>
            <exception>java.io.IOException</exception>
            <to uri="file://d:/camel/output/exception" />
        </onException>                                                              onException

        <route>
          <from uri="file://d:/camel/input/src?move=done&amp;moveFailed=fail&amp;delay=2000" />
          <to uri="http://localhost:9098/myService/" />
        </route>
    </camelContext>

 </beans>
```

[route 설명]

DefaultErrorHandler, onException 모두 정의

java.io.IOException 제외한 exception 발생 → DefaultErrorHandler인 "myDefault"에서 처리

java.io.IOException 발생 → <onException>에서 처리

1. 원리

발생한 exception을 <onException> 구문과 비교할 때, exception의 상속 구조대로 밑→위로 올라가면서 하나씩 비교해 일치하는 exception이 있는지 비교

발생한 exception에 대해 상속 구조에 포함된 onException이 2개 이상 존재한다면, 발생하는 exception과 정확하게 일치하는 것을 먼저 handling

완전히 일치하는 것이 없다면, "gap detection" 이용하여 실행할 onException 판단

※ Gap detection

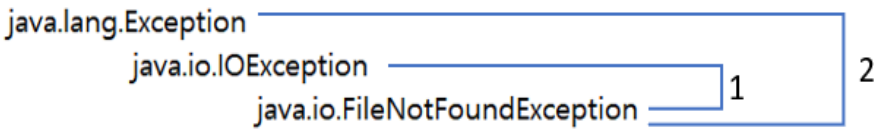
발생한 exception과 route에 안에서 정의한 exception 사이의 gap을 계산하여 가장 낮은 gap을 차지하는 <onException>을 선택

ex)

[camel-route에서 정의한 <onException>]

```
<onException>
    <exception>java.io.IOException</exception>
    <to uri="file://d:/camel/output/exception/route_1"/>
</onException>

<onException>
    <exception>java.lang.Exception</exception>
    <to uri="file://d:/camel/output/exception/route_2"/>
</onException>
```

If) route에서 processing 중 FileNotFoundException 발생



① FileNotFoundException과 정확하게 일치하는 <onException> 존재 X

⇒ "gap detection"으로 결정

② java.io.FileNotFoundException과 java.io.IOException과의 gap은 1, java.lang.Exception는 gap=2

③ java.io.IOException과의 gap이 더 작기 때문에 java.io.IOException 정의된 <onException> 실행

2. 특징

하나의 <onException>에 여러 개의 exception 정의 가능

```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring" errorHandlerRef="myDefault">
    <errorHandler id="myDefault" type="DefaultErrorHandler">
        <redeliveryPolicy maximumRedeliveries="3" retryAttemptedLogLevel="WARN" />
    </errorHandler>

    <onException>
        <exception>java.io.IOException</exception>
        <exception>java.sql.SQLException</exception>
        <to uri="file://d:/camel/output/exception"/>
    </onException>

    <route>
        <from uri="file:d:/camel/input/src?delay=2000"/>
        <to uri="http://localhost:10040/myService/" />
    </route>
</camelContext>
```

<onException>은 다른 error handler보다 우선 순위가 높다.

ex) defaultErrorHandler와 java.lang.Exception을 감지하는 onException, 2개가 정의 되어 있는데 java.lang.Exception이 발생한다면 <onException> 구문이 실행

3.4.2 Redelivery (redeliveryPolicy)

<onException>에서는 재전송에 대한 default는 0으로 설정(재전송 X)

재전송하고자 한다면 redeliveryPolicy를 통해 재전송 설정

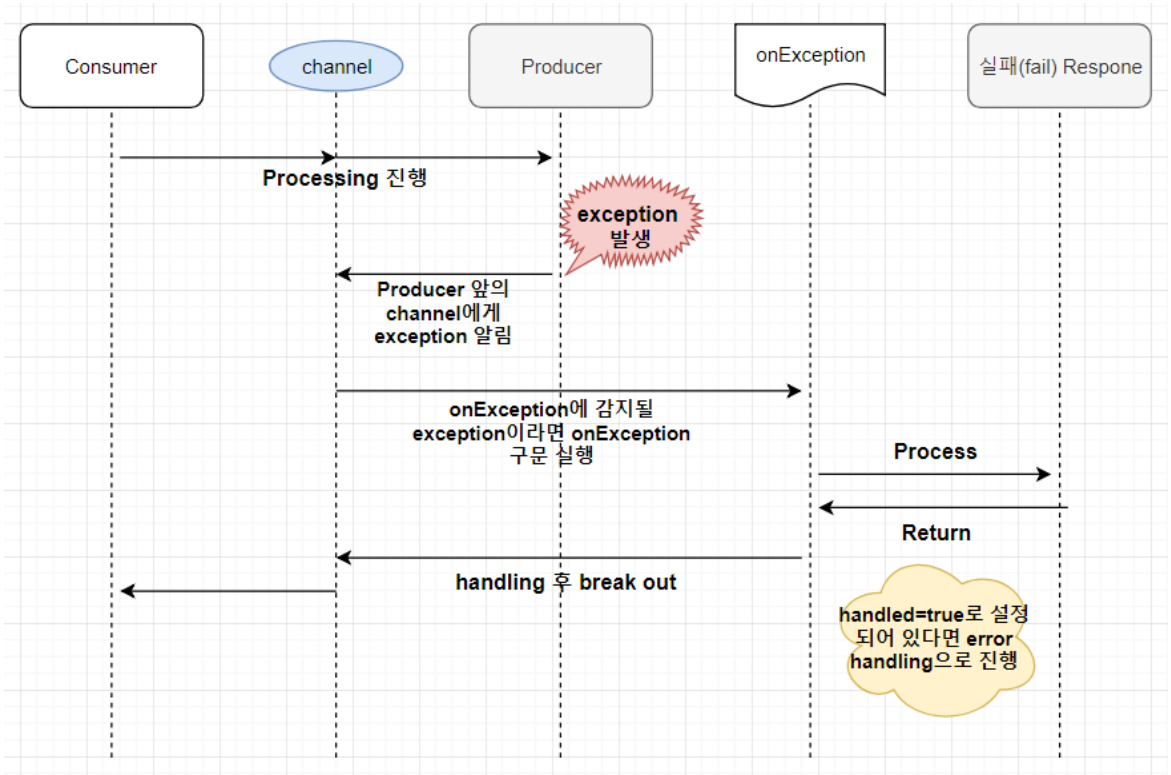
```
<?xml version="1.0" encoding="UTF-8"?>

 <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">

    <bean id="jetty" class="org.apache.camel.component.jetty9.JettyHttpComponent9">
    </bean>

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

        <onException>
            <exception>java.io.IOException</exception>
            <redeliveryPolicy maximumRedeliveries="3" retryAttemptedLogLevel="WARN" />
            <to uri="file://d:/camel/output/exception" />
        </onException>

        <route>
          <from uri="file://d:/camel/input/src?move=done&amp;moveFailed=fail&amp;delay=2000" />
          <to uri="http://localhost:9098/myService/" />
        </route>
    </camelContext>

 </beans>
```

3.4.3 <handled>·<continued>

1. <handled>

(default) handled=false

"handled=true"로 설정하게 되면 Camel이 제공하는 것이 아닌, 원하는 custom response를 caller에게 보낼 수 있다.



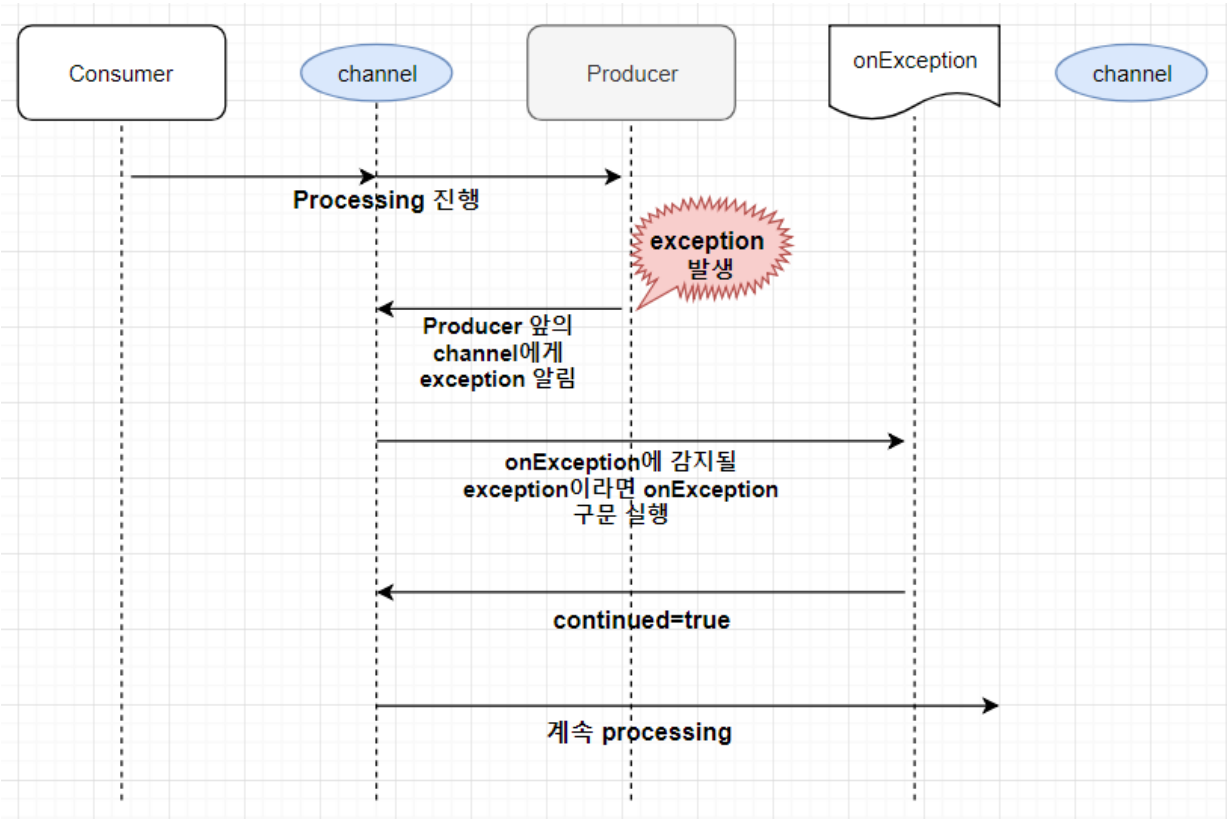
```
<?xml version="1.0" encoding="UTF-8"?>

 <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">

    <bean id="jetty" class="org.apache.camel.component.jetty9.JettyHttpComponent9">
    </bean>

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

        <onException>
            <exception>java.io.IOException</exception>
            <handled><constant>true</constant></handled>
            <to uri="file://d:/camel/output/exception" />
        </onException>

        <route>
          <from uri="file://d:/camel/input/src?move=done&amp;moveFailed=fail&amp;delay=2000" />
          <to uri="http://localhost:9098/myService/" />
        </route>
    </camelContext>

 </beans>
```

(주의) Spring XML에서 "handled=true/false"을 사용하려면 <constant> 구문으로 묶어서 사용해야 한다.

2. <continued>

"continued=true"를 사용하게 된다면 exception이 발생하더라도 무시하고 기존 routing 진행

<onException>을 쓰지 않고 continue 기능을 사용하고 싶다면, try/catch/finally을 이용해야 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">

    <bean id="jetty" class="org.apache.camel.component.jetty9.JettyHttpComponent9">
    </bean>

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

        <onException>
            <exception>java.io.IOException</exception>
            <continued><constant>true</constant></continued>
            <to uri="file://d:/camel/output/exception" />
        </onException>

        <route>
          <from uri="file://d:/camel/input/src?move=done&amp;moveFailed=fail&amp;delay=2000" />
          <to uri="http://localhost:9098/myService/" />
        </route>
    </camelContext>

 </beans>
```

(주의) 하나의 onException에 handled와 continued를 같이 사용할 수 없다.

3.4.4 <onWhen>

<onWhen>과 <onException>을 같이 사용한다면, exception 발생과 더불어 <onWhen> 조건에 맞는 case에만 실행

```
<?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">

    <bean id="jetty" class="org.apache.camel.component.jetty9.JettyHttpComponent9">
    </bean>

    <bean id="userService" class="kr.co.bizframe.camel.rest.UserService" />

    <camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
        <errorHandler id="myDefault" type="DefaultErrorHandler">
            <redeliveryPolicy maximumRedeliveries="3" retryAttemptedLogLevel="WARN" />
        </errorHandler>

        <onException>
            <exception>java.io.IOException</exception>
            <onWhen>
                <simple>${header.size} == null</simple>
            </onWhen>
            <to uri="file://d:/camel/output/exception" />
        </onException>

    </camelContext>

 </beans>
```

[route 설명]

java.io.IOException 발생했을 때, header에서의 size가 null일 때만 <onException> 구문 실행

(java.io.IOException이 발생하여도 header.size가 null이 아니라면 실행 X)

4 Thread·Concurreny

4.1 Concurrency

1. Concurrency

Concurrency = multitasking

Camel route에서 여러 messages들을 동시에 수행할 수 있도록 제공

2. Concurrency 구현 방법

i. . ParallelProcessing

Parallel processing 지원하는 EIP : multicast, recipient, splitter, list, wire tap, error handler …

parallelProcessing=true로 활성화하여 사용

ii. Thread pool

· 종류

가) Cached thread pool

Max thread 개수 X

→ 새로운 task가 들어왔는데 thread pool에서 사용할 수 있는 thread가 존재하지 않는다면, 새로운 thread를 생성하여 진행

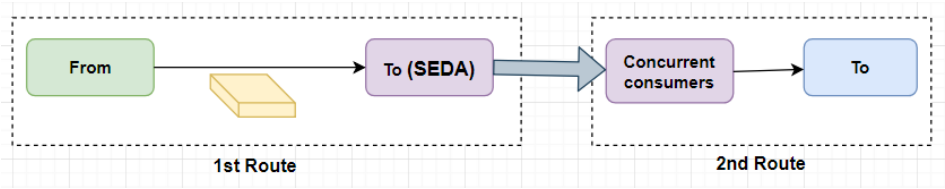
장점 : Parallel processing보다 속도가 빠르다.

단점 : thread pool의 상한선이 정해져 있지 않아 과부하 문제 발생 가능성 多

나) Fixed thread pool

thread 개수 제한한 multi threads

iii. SEDA component



Routes 사이에 internal memory queues 이용

concurrentConsumers 이용하여 multiple concurrent threads 이용

```
<?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">

    <bean id="jetty" class="org.apache.camel.component.jetty9.JettyHttpComponent9">
    </bean>

    <bean id="userService" class="kr.co.bizframe.camel.rest.UserService" />

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

        <route>
          <from uri="file:d:/camel/input/src?delay=2000"/>
          <to uri="seda:move" />
        </route>

        <route>
            <from uri="seda:move?concurrentConsumers=10" />
            <to uri="file:///d:/camel/output" />
        </route>
    </camelContext>

 </beans>
```

[route 설명]

1번째 route : 1개의 thread를 통해 processing

2번째 route : concurrentConsumers 통해 thread 개수 설정 → multitasking

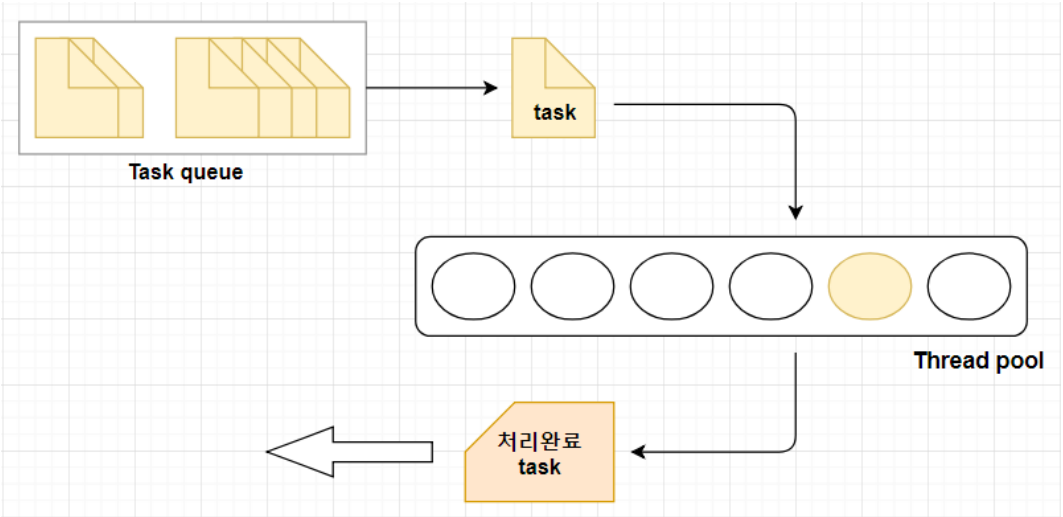
: 1번째 route로부터 들어온 task를 multi thread로 작업

(주의) concurrentConsumers는 고정된 thread pool 개수 사용

→ 고정된 thread 개수가 항상 processing 기다린다. (낭비 가능성 有)

4.2 Thread pools

1. Thread pools



Task queue로부터 task 꺼내어 thread pool에 넣고 실행

단일 thread가 아닌 multiple threads를 사용하여, component는 앞선 exchange를 기다리는 시간 없이 다른 processing을 계속 진행하여 효율성 ↑

2. Option

Option	Default value	Description
poolSize	10	Thread pool 개수
maxPoolSize	20	최대 thread pool 개수
keepAlivetime	60	종료된 후 thread 유지 시간
maxQueueSize	1000	Pool exhausted 되기 전 task queue가 넣을 수 있는 task 개수
rejectedPolicy	CallerRuns	(option) CallerRuns/Abort/DiscardOlddest, Discard

· rejectedPolicy

- CallerRuns: Thread에 가장 최근에 들어온 task부터 실행

  부작용: 가장 최근에 들어온 task 끝나기 전에는 다른 tasks가 들어오는 것을 막는다.

- Abort: 현재 진행하던 task를 중지하고 exception 발생

- DiscardOlddest: task queue에 존재하는 task에서 오래된 순서부터 버린다.

- Discard: task queue에 존재하는 tasks를 모두 버린다.

3. Creating/Using thread pools

Thread pool 생성 방법: <threadPool>, <threadPoolProfile>

ⅰ. <threadPool> 이용

```
<?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">

    <bean id="jetty" class="org.apache.camel.component.jetty9.JettyHttpComponent9">
    </bean>

    <bean id="myFilter" class="kr.co.bizframe.camel.myFilter" />

    <camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
        <threadPool id="EsbPool" threadName="ESB" poolSize="5" maxPoolSize="20" maxQueueSize="250" />

        <route>
          <from uri="file://d:/camel/input/src?filter=#myFilter&amp;delay=2000" />
          <threads executorServiceRef="EsbPool">
            <to uri="file://d:/camel/output" />
          </threads>
        </route>
    </camelContext>

 </beans>
```

ii. <threadPoolProfile> 이용

```
<?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">

    <bean id="jetty" class="org.apache.camel.component.jetty9.JettyHttpComponent9">
    </bean>

    <bean id="myFilter" class="kr.co.bizframe.camel.myFilter" />

    <camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
        <threadPoolProfile id="EsbPool" poolSize="5" maxPoolSize="20" maxQueueSize="250" />

        <route>
          <from uri="file://d:/camel/input/src?filter=#myFilter&amp;delay=2000" />
          <threads executorServiceRef="EsbPool">
            <to uri="file://d:/camel/output" />
          </threads>
        </route>
    </camelContext>

 </beans>
```

<threadPool>, <threadPoolProfile> 이용하여 custom thread pool 정의

executorServiceRef 이용하여 어떤 thread pool을 사용할 것인지 설정

→ 만약 executorServiceRef에 설정한 thread가 없다면 fall-back 한 뒤, 그 이름으로 정의된 thread profile이 있는지 찾아본다. 있다면, 해당 profile을 기반으로 thread pool을 만든 뒤 사용

(주의)

<threadPool> 사용할 때는 "threadName"을 꼭 명시해줘야 한다.

<threadPoolProfile> 사용할 때는 <threadPool>과 다르게, "threadName" option을 사용할 수 없다.

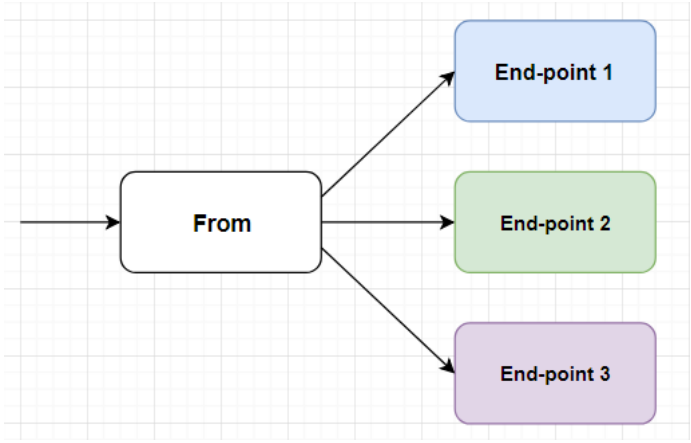
4.3 Using concurrency with EIPs

1. Threads EIP

threadPool 이용하여 concurrency 구현 (<threadPool>, <threadPoolProfile>)

Threads EIP를 가장 많이 사용하는 component : **File** component

2. Multicast EIP



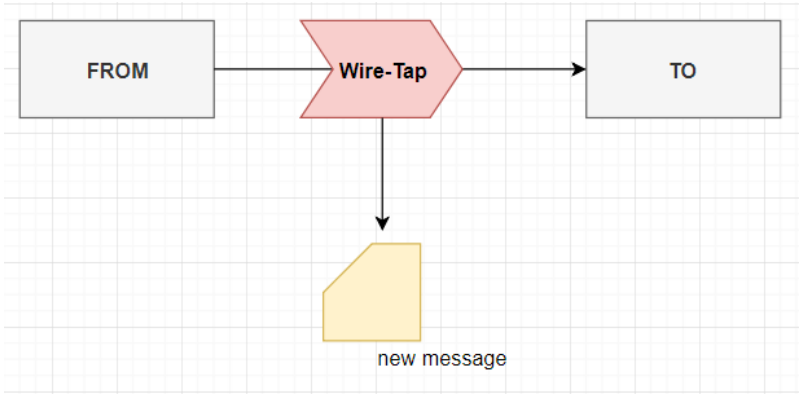
Multicast EIP에서 concurrency를 사용하지 않는다면, camel route는 정의한 순서대로 수행하기 때문에 오랜 시간 소요

→ 하나의 message를 서로 다른 대상자에게 동시에 보냄으로써 효율성 ↑

"parallelProcessing=true" 이용

```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
    <route>
      <from uri="file:d:/camel/input/src?delay=2000"/>
      <multicast parallelProcessing="true">
        <to uri="file://d:/camel/output" />
        <to uri="direct:mid" />
        <to uri="direct:end" />
      </multicast>
    </route>
</camelContext>
```

3. Wire Tap EIP



Processing 과정이 수행 중, 종료되기 전에 새로운 message를 생성하여 보내고 싶을 때 사용

새로 만든 message는 새롭게 생성한 thread를 이용하여 정해진 endpoint로 보냄과 동시에, 원래 thread는 그대로 진행

```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-2.5.xsd
        http://camel.apache.org/schema/spring http://camel.apache.org/schema/spring/camel-spring.xsd">

    <bean id="jetty" class="org.apache.camel.component.jetty9.JettyHttpComponent9">
    </bean>

    <camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
        <threadPool id="EsbPool" threadName="ESB" poolSize="5" maxPoolSize="20" maxQueueSize="250" />

        <route>
          <from uri="file://d:/camel/input/src?move=done&amp;moveFailed=fail&amp;delay=2000" />
          <wireTap uri="direct:tap" executorServiceRef="EsbPool" />
          <to uri="mock:result" />
        </route>

        <route>
          <from uri="direct:tap" />
          <to uri="mock:tap" />
        </route>
    </camelContext>

 </beans>
```

4. Splitter EIP

Message를 분할하여 각각 processing을 진행할 때 concurrency 필요

threadPool 이용하여 concurrency 구현

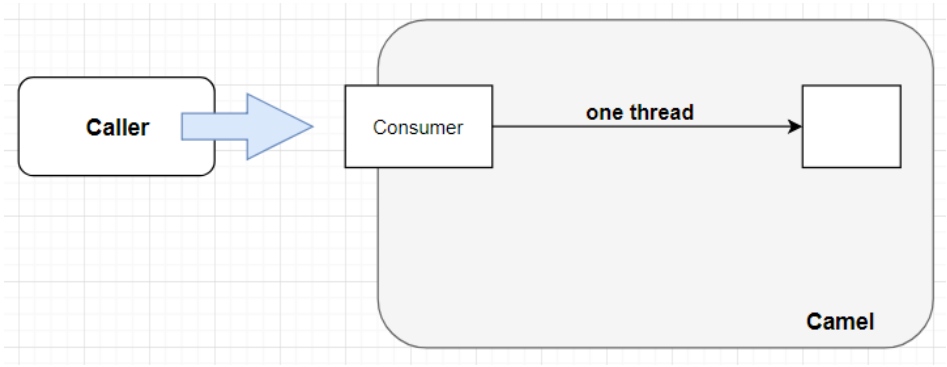
split option 중 executorServiceRef 통해 custom thread pool 정의, 사용

→ 이 option을 사용하게 되면, parallel processing은 자동으로 들어가게 된다.

```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-2.5.xsd
        http://camel.apache.org/schema/spring http://camel.apache.org/schema/spring/camel-spring.xsd">

    <bean id="jetty" class="org.apache.camel.component.jetty9.JettyHttpComponent9">
    </bean>

    <camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
        <threadPool id="EsbPool" threadName="ESB" poolSize="5" maxPoolSize="20" maxQueueSize="250" />

        <route>
          <from uri="file://d:/camel/input/src?move=done&amp;moveFailed=fail&amp;delay=2000" />
          <split executorServiceRef="EsbPool">
                <tokenize token=";"/>
                <to uri="mock:result"/>
            </split>
        </route>
    </camelContext>

 </beans>
```

4.4 Synchronicity and threading

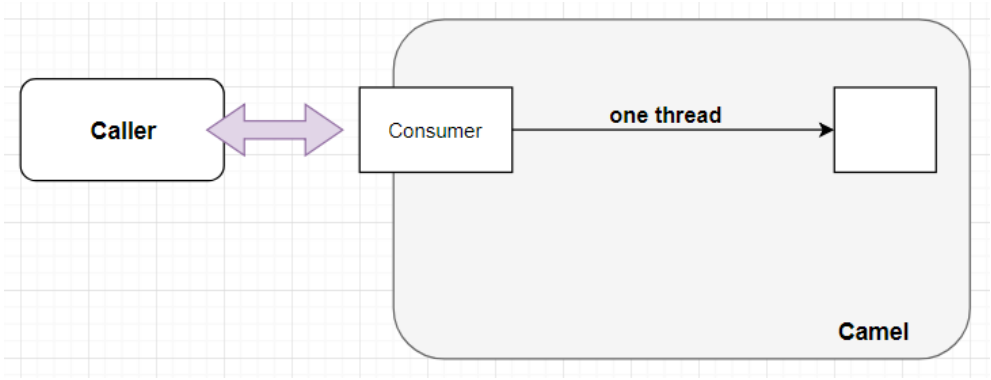
1. Asynchronous caller (단일 thread)



InOnly message 전송

※ InOnly message: 한번 전송한 후 response를 기다리지 않고 끝난다.
1개의 thread만 가지고 process 진행

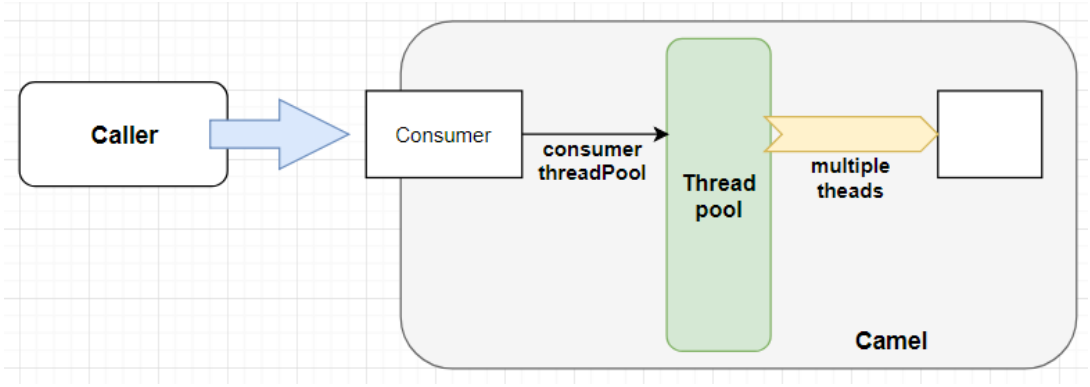
2. Synchronous caller (단일 thread)



InOut message 전송

※ InOut message: message를 전송한 후 response를 기다린다.
Caller는 message를 보내며 request한 뒤 response가 올 때까지 기다린다.
1개의 thread만 가지고 process 진행

3. Asynchronous caller (다중 threads)

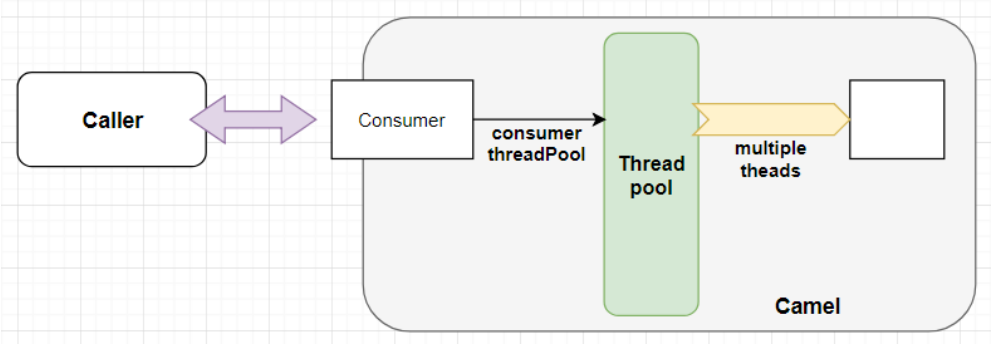


InOnly message 전송

Caller는 response를 기다리지 않고 message를 전송

Camel 안에서 processing 할 때 1개의 thread가 아닌 multiple thread 사용하여 진행

4. Synchronous caller (다중 threads)



InOut message 전송

Camel 안에서 processing 할 때 1개의 thread가 아닌 multiple thread 사용하여 진행

Consumer thread pool에서는 request를 보낸 뒤 response를 받기 전까지는 반드시 block 상태를 하고 있어야 한다.

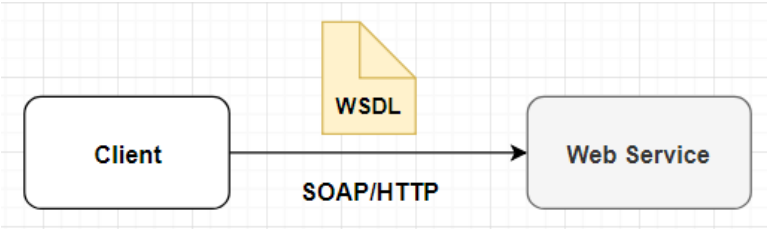
5 CXF Component

5.1 CXF

Web service 이용하기 위해서 사용

WSDL 이용하여 정의

Apache CXF 이용하여 web service publishing (JAX-WS 사용)



Message는 XML로 구성, SOAP/HTTP로 전송된다.

1. Option

Option	Default	Description
wsdlURL		WSDL 위치
serviceClass		SEI class
serviceName		WSDL에 1개 이상의 serviceName이 존재했을 때 설정 필요
portName		wsdl:port@name과 매핑, Spring과 함께 쓰임
dataFormat	POJO	Data type message (option : POJO/PAYLOAD/MESSAGE)

※ serviceClass는 필수로 작성

2. URI

CXF endpoint bean으로 사용

cxf:bean:[CXF 정의한 bean 이름]

```
<?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:cxf="http://camel.apache.org/schema/cxf"
    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
        http://camel.apache.org/schema/cxf http://camel.apache.org/schema/cxf/camel-cxf.xsd">

    <bean id="jetty" class="org.apache.camel.component.jetty9.JettyHttpComponent9">
    </bean>

    <cxf:cxfEndpoint id="HelloServiceEndpoint"
        serviceClass="kr.co.bizframe.camel.cxf.client.HelloPortType"
        address="http://localhost:10040/helloService">
    </cxf:cxfEndpoint>

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

        <route>
          <from uri="file://d:/camel/input/src?move=done&amp;moveFailed=fail&amp;delay=2000" />
          <to uri="cxf:bean:HelloServiceEndpoint" />
        </route>

    </camelContext>

 </beans>
```

[route 설명]

CXF 이용하기 위해 정의할 bean 이름, serviceClass, 주소 정의

→ kr.co.bizframe.camel.cxf.client.HelloPortType class를 serviceClass로 지정하고, HelloServiceEndpoint라는 이름을 가진 cxf bean 정의

route에서 cxf bean을 사용하기 위해 cxf:bean:HelloServiceEndpoint로 uri 지정

5.2 WSDL(Web Service Description Language)

네트워크로 연결된 XML 기반 서비스를 기술하는 새로운 스펙

웹서비스에서 사용하기 위해서는 지켜야 하는 구조(operations, input type, output type) 정의

1. 구조

ⅰ. Type

웹서비스에서 사용할 data 타입

XML schema 이용

name과 각 parameter type 명시

ⅱ. Message

웹서비스에서 사용할 message

iii. portType

웹서비스에서 드러날 interface name, operation

iv. Binding

Transport 타입, message encoding

v. Service

웹서비스에 대해서 정의

웹서비스를 드러낼 port binding

2. Format

```
<wsdl:definitions xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:tns="http://server.hello.cxf.camel.bizframe.co.kr/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:nsl="http://schemas.xmlsoap.org/soap/http" name="Service" targetNamespace="http://server.hello.cxf.camel.bizframe.co.kr/">
  <wsdl:types>
    <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:tns="http://server.hello.cxf.camel.bizframe.co.kr/" attributeFormDefault="unqualified" elementFormDefault="qualified"
     targetNamespace="http://server.hello.cxf.camel.bizframe.co.kr/">
      <xsd:element name="sayMoa" type="tns:sayMoa"/>
      <xsd:complexType name="sayMoa">
        <xsd:sequence>
          <xsd:element minOccurs="0" name="arg0" type="xsd:string"/>
        </xsd:sequence>
      </xsd:complexType>
      <xsd:element name="sayMoaResponse" type="tns:sayMoaResponse"/>
      <xsd:complexType name="sayMoaResponse">
        <xsd:sequence>
          <xsd:element minOccurs="0" name="return" type="xsd:string"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:schema>
  </wsdl:types>
  <wsdl:message name="sayMoa">
    <wsdl:part element="tns:sayMoa" name="parameters"> </wsdl:part>
  </wsdl:message>
  <wsdl:message name="sayMoaResponse">
    <wsdl:part element="tns:sayMoaResponse" name="parameters"> </wsdl:part>
  </wsdl:message>
  <wsdl:portType name="ServicePortType">
    <wsdl:operation name="sayMoa">
      <wsdl:input message="tns:sayMoa" name="sayMoa"> </wsdl:input>
      <wsdl:output message="tns:sayMoaResponse" name="sayMoaResponse"> </wsdl:output>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="ServiceSoapBinding" type="tns:ServicePortType">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="sayMoa">
      <soap:operation soapAction="" style="document"/>
      <wsdl:input name="sayMoa">
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="sayMoaResponse">
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="Service">
    <wsdl:port binding="tns:ServiceSoapBinding" name="ServicePort">
      <soap:address location="http://localhost:10080/helloService"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

```
<wsdl:definitions xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
 xmlns:nsl="http://schemas.xmlsoap.org/soap/http" name="Service" targetNamespace="http://server.hello.cxf.came
```

WSDL은 <wsdl:definitions>으로 시작해, <wsdl:definitions>으로 끝난다.

WSDL의 모든 문서는 <wsdl:definitions> 구문으로 묶어준다.

```
<wsdl:types>
  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:tns="http://server.hello.cxf.camel.bizframe.co.kr/" attributeFormDefault="unqualified" elementFormDefault="qualified"
   targetNamespace="http://server.hello.cxf.camel.bizframe.co.kr/">
    <xsd:element name="sayMoa" type="tns:sayMoa"/>
    <xsd:complexType name="sayMoa">
      <xsd:sequence>
        <xsd:element minOccurs="0" name="arg0" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="sayMoaResponse" type="tns:sayMoaResponse"/>
    <xsd:complexType name="sayMoaResponse">
      <xsd:sequence>
        <xsd:element minOccurs="0" name="return" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:schema>
</wsdl:types>
```

<wsdl:types>

사용할 메시지/데이터 형식 정의

```
<wsdl:message name="sayMoa">
  <wsdl:part element="tns:sayMoa" name="parameters"> </wsdl:part>
 </wsdl:message>
<wsdl:message name="sayMoaResponse">
  <wsdl:part element="tns:sayMoaResponse" name="parameters"> </wsdl:part>
 </wsdl:message>
```

<wsdl:message>

각 개별로 전송하는 message 데이터 포맷 정의

전송을 메시지 부분들로 나누는 것은 데이터 논리적 뷰에 의존

```
<wsdl:portType name="ServicePortType">
 ▼<wsdl:operation name="sayMoa">
    <wsdl:input message="tns:sayMoa" name="sayMoa"> </wsdl:input>
    <wsdl:output message="tns:sayMoaResponse" name="sayMoaResponse"> </wsdl:output>
  </wsdl:operation>
</wsdl:portType>
```

<wsdl:portType>

하나의 논리적 연산을 형성하는 메시지들을 grouping

ex) 하나의 리소스에 필요한 input, output, fault를 처리하는 각 메시지를 하나의 연산으로 grouping하여 이용

```
<wsdl:binding name="ServiceSoapBinding" type="tns:ServicePortType">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
 ▼<wsdl:operation name="sayMoa">
    <soap:operation soapAction="" style="document"/>
   ▼<wsdl:input name="sayMoa">
      <soap:body use="literal"/>
    </wsdl:input>
   ▼<wsdl:output name="sayMoaResponse">
      <soap:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
 </wsdl:binding>
```

<wsdl:bindings>

논리적 모델과 물리적 모델 사이에 연결 제공

추상적 포트 유형을 통해 정의했던 연산을 사용하여 구체적으로 연결

```
<wsdl:service name="Service">
 ▼<wsdl:port binding="tns:ServiceSoapBinding" name="ServicePort">
    <soap:address location="http://localhost:10030/helloService"/>
  </wsdl:port>
 </wsdl:service>
```

<wsdl:service>

End point 물리적 위치 지정

Port 유형과 binding을 상용하고 특정 공급자용 웹 주소/URI 부여

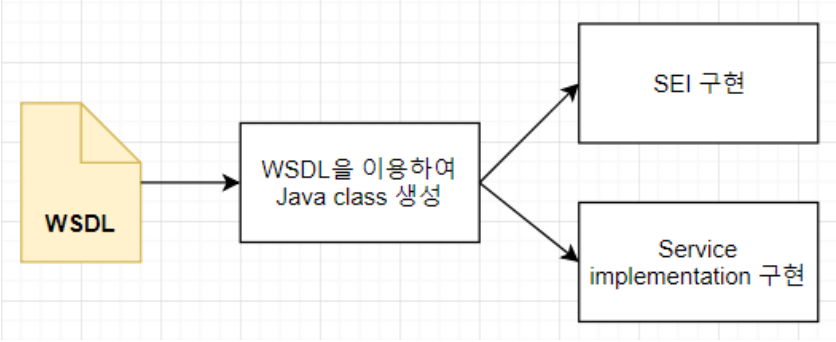
5.3 Contract-first development

CXF component 정의하는 방법

1. Contract-first development
2. Code-first development

· Contract-first development

WSDL file을 먼저 생성 → CXF 통하여 WSDL에서 Java class 생성하여 이용



CXF endpoint 정의하여 Camel에서 CXF component 사용

```
<?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:cxf="http://camel.apache.org/schema/cxf"
   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
       http://camel.apache.org/schema/cxf http://camel.apache.org/schema/cxf/camel-cxf.xsd">

    <bean id="jetty" class="org.apache.camel.component.jetty9.JettyHttpComponent9">
    </bean>

    <cxf:cxfEndpoint id="HelloServiceEndpoint"
        serviceClass="kr.co.bizframe.camel.cxf.client.HelloPortType"
        address="http://localhost:10040/helloService">
    </cxf:cxfEndpoint>
```

camel-cxf 사용하기 위한 cxf/camel-cxf 추가

endpoint 정의

· 단 1개의 <cxf:cxfEndpoint> 정의

→ CXF가 자동으로 선택

· 2개 이상의 <cxf:cxfEndpoint> 정의

→ serviceName(WSDL service element name)과 endpointName(port element name)를 이용하려는 endpoint bean에 정의해야 한다.

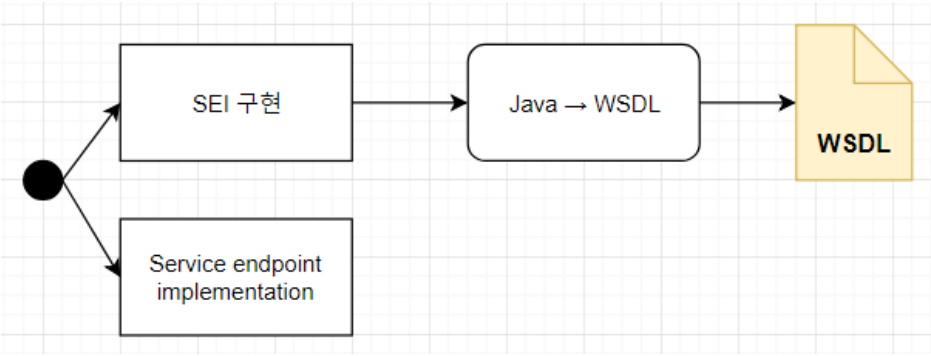
```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">

    <route>
      <from uri="file://d:/camel/input/src?move=done&amp;moveFailed=fail&amp;delay=2000" />
      <to uri="cxf:bean:HelloServiceEndpoint" />
    </route>

</camelContext>
```

Route에 cxf endpoint를 정의하여 camel route에서 web service 이용

5.4 Code-first development

Java Interface class 구현 → CXF 이용하여 WSDL 생성 후 사용

Java class 구현할 때는 JAX-WS annotation 이용



1. 순서

ⅰ. 사용할 method, type, parameter 정의

ⅱ. 웹 서비스로 이용할 Interface 생성

ⅲ. 웹 서비스 class에 request/response 구현

```
@WebService(targetNamespace = "http://server.hello.cxf.camel.bizframe.co.kr/", name = "MoaPortType")
@XmlSeeAlso({ObjectFactory.class})
public interface MoaPortType {

    @WebMethod
    @RequestWrapper(localName = "sayMoa", targetNamespace = "http://server.hello.cxf.camel.bizframe.co.kr/", className = "kr
    @ResponseWrapper(localName = "sayMoaResponse", targetNamespace = "http://server.hello.cxf.camel.bizframe.co.kr/", classN
    @WebResult(name = "return", targetNamespace = "http://server.hello.cxf.camel.bizframe.co.kr/")
    public String sayMoa(
        @WebParam(name = "str", targetNamespace = "http://server.hello.cxf.camel.bizframe.co.kr/") String str);
}
```

JAX-WS annotation 이용하여 CXF에게 어떤 class가 웹 서비스 Interface인지 알려준다.

ⅳ. Camel-route에 cxf component 정의 후, 사용

6.REST Component

6.1 REST Component

1. 정의

REST DSL 이용하여 REST endpoint 정의

Camel component에 plug-in하여 REST transport 사용

2. URI format

rest://method:path[:uriTemplate]?[options]

Option	Description
Method	HTTP method (get/post/put/delete)
Path	Base path
uriTemplate	URI template
Consumes	REST service에서 받아들일 media type (text/xml, application/json …)
produces	REST service에서 return할 media type

```
<rest path="/say"> path option
    <get uri="/hello">
        <route>
            <transform>
                <constant>Hello World</constant>
            </transform>
        </route>
    </get>
    <get uri="/bye"> uriTemplate option
        <route>
            <transform>
                <constant>Bye World</constant>
            </transform>
        </route>
    </get>
</rest>
```

[route]

[address]/say/hello 주소로 접속하면 "Hello World" 출력

[address]/say/bye 주소로 접속하면 "Bye World" 출력

→ 2개 모두 같은 path 사용, 다른 uriTemplate 사용

3. uriTemplate

HTTP URI 정의

경로와 쿼리(query), 두 부분으로 구성

ex) /user/Request/{what}?name={header.id}

/user/Request/{what} : 경로

/user/Request : 리터럴

/{what} : 변수

?name={header.id} : 쿼리(query)

uriTemplate 문자열 내의 모든 변수 이름은 고유성을 유지해야 한다. (대/소문자 구분 X)

i. 경로

'/'로 구분된 세그먼트로 구성

각 세그먼트는 리터럴(고정 데이터), 변수(중괄호({})로 묶음) 또는 와일드 카드로 구성

ii. 쿼리(query)

생략 가능

'&'로 구분된 이름/값 쌍을 지정, 짝이 이루어져 있어야만 사용 가능
Query의 오른쪽에만 변수를 포함 가능

4. REST Component 구현 방법

REST service를 만든 후, REST DSL 이용해 camel route에 REST service 이용

5. REST service 정의하는 방법

ⅰ. JAX-RS REST

JAX-RS 이용하여 REST service 정의, camel 이용

@Path, @GET, @POST, @PUT, @DELETE, @PathParam annotation 등으로 구현

ⅱ. Camel REST component

REST service를 정의할 수 있는 camel component 이용

ex) jetty, spark-rest, restlet....

6.2 REST DSL

1. REST DSL

End users에게 REST service를 정의하고 REST verb(get, post, put, delete)을 사용하게 해줄 수 있는 DSL

REST DSL 지원하는 camel component

camel-netty-http, camel-netty4-http, camel-jetty, camel-restlet, camel-servlet, camel-spark-rest

```
<?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">

    <bean id="jetty" class="org.apache.camel.component.jetty9.JettyHttpComponent9">
    </bean>

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

        <restConfiguration component="jetty" scheme="http" host="localhost" port="10050" bindingMode="auto">
        </restConfiguration>

        <rest path="/say">
            <get uri="/hello">
                <to uri="direct:hello" />  to() 이용하여 endpoint 설정
            </get>
        </rest>

        <route>
            <from uri="direct:hello" />
                <transform>
                    <constant>Hello World</constant>
                </transform>
            <to uri="direct:end" />
        </route>

    </camelContext>

 </beans>
```

```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-2.5.xsd
       http://camel.apache.org/schema/spring http://camel.apache.org/schema/spring/camel-spring.xsd">

   <bean id="jetty" class="org.apache.camel.component.jetty9.JettyHttpComponent9">
   </bean>

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

       <restConfiguration component="jetty" scheme="http" host="localhost" port="10050" bindingMode="auto">
       </restConfiguration>

       <rest path="/say">
           <get uri="/hello">
               <route>
                   <transform>
                       <constant>Hello World</constant>
                   </transform>
               </route>
           </get>
           <get uri="/bye">
               <route>
                   <transform>
                       <constant>Bye World</constant>
                   </transform>
               </route>
           </get>
       </rest>

   </camelContext>

</beans>
```

<rest> 안에 <route>를 직접 넣어 endpoint 설정

2가지 방법으로 모두 REST DSL 정의 가능

2. REST DSL의 path, uriTemplate

```xml
<rest path="/user">
    <get uri="/{id}">
        <to uri="bean:userBean" />
    </get>
</rest>
```

base path와 uriTemplate 모두 슬래쉬(/)로 시작
path로는 /user 사용(리터럴값)
uriTemplate는 /{id} 사용 (변수({ }))

```xml
<rest path="/rest">
    <get uri="/get">
        <to uri="bean:helloBean" />
    </get>
    <post uri="/post">
        <to uri="bean:postBean" />
    </post>
</rest>
```

uriTemplate 없이 path option만 단독으로 사용 가능
path와 uriTemplate 모두 설정하여 사용 가능

```xml
<rest>
    <get uri="/rest/get">
        <to uri="bean:helloBean" />
    </get>
    <post uri="/rest/post">
        <to uri="bean:postBean" />
    </post>
</rest>
```

<rest>의 path option은 설정하지 않고, uriTemplate에 base path까지 모두 설정하여 사용 가능

3. Option

```
<restConfiguration component="jetty" scheme="http" host="localhost" port="10050" bindingMode="auto">
</restConfiguration>
```

ⅰ. component

REST transport로 사용할 Camel REST component

ex) jetty, restlet, spark-rest, servlet ...

ⅱ. scheme

(default) http

REST service에서 사용할 scheme (http/https)

ⅲ. port

REST service에 접근할 포트 번호

ⅳ. host

REST service에서 expose할 hostname

ⅴ. bindingMode

REST service에서의 request, response에 대한 binding mode 설정

Incoming, outgoing message 모두 적용

① off (default)

Incoming, outgoing message에 대해 자동으로 binding X

② auto

binding 활성화 + JSON, XML 중 classpath에 library가 포함되어 있는 format만 지원

③ json

JSON binding만 활성화 (library에 camel-jackson을 포함해야 사용 가능)

④ xml

XML binding만 활성화 (library에 camel-jaxb을 포함해야 사용 가능)

⑤ json_xml

XML, JSON binding 모두 활성화

"auto"와 다른 점

- auto는 JSON, XML 중 단 하나의 library라도 존재한다면 오류 없이 수행 가능
- json_xml은 JSON, XML binding에 필요한 camel-jackson와 camel-jaxb, 2개의 library 중 하나라도 존재하지 않는다면 사용할 수 X

4. consumes, produces

<rest>와 REST method(<get>, <post> ...) 모두사용 가능

<rest>: 정의한 REST service 모든 범위에 적용

REST method: 정의한 REST service의 해당 method에만 적용

content-type 지정할 때는 "produces" option 사용

```
<rest path="/rest" consumes="application/json" produces="application/json">
    <get uri="/entire" type="kr.co.bizframe.camel.rest.User">
        <to uri="bean:userService" />
    </get>
</rest>
```

[route 설명]

<rest>에 consumes, produces 설정

→ "/rest"로 시작하는 모든 rest service에 적용

```
<rest path="/rest">
    <get uri="/individual" consumes="application/json" produces="application/json">
        <to uri="bean:userService" />
    </get>
</rest>
```

[route 설명]

<rest>안의 <get> method에 consumes, produces 설정

→ "/rest"로 시작하는 rest service 중에서도 "/rest/individual" uri 사용하는 rest service에 적용

· 종류

type

-text/plain

-text/html

- application/json

- application/xml

5. type, outType

i. type

Incoming(input) type 명시

ii. outType

HTTP response message(outgoing)으로 나타낼 Java type 명시

(주의) bindingMode가 활성화 되어있어야만 type, outType 사용 가능

```
<restConfiguration component="jetty" scheme="http" host="localhost" port="10050" bindingMode="auto">
</restConfiguration>
```

binding 활성화를 해주어야 사용 가능

```
<rest path="/user" consumes="application/json" produces="application/json">
    <get uri="/{id}" outType="kr.co.bizframe.camel.rest.User">
        <to uri="bean:userService?method=getUser(${header.id})" />
    </get>
    <put uri="/update" type="kr.co.bizframe.camel.rest.User">
        <to uri="bean:userService?method=updateUser" />
    </put>
    <get uri="/list" outType="kr.co.bizfrmae.camel.rest.User[]">
        <to uri="bean:userService?method=listUsers" />
    </get>
</rest>
```

[설명]

Incoming type을 kr.co.bizframe.camel.rest.User class로 지정

지정한 class format대로 input이 되어 있지 않는다면 오류 발생

```
<rest path="/user" consumes="application/json" produces="application/json">
    <get uri="/{id}" outType="kr.co.bizframe.camel.rest.User">
        <to uri="bean:userService?method=getUser(${header.id})" />
    </get>
    <put uri="/update" type="kr.co.bizframe.camel.rest.User">
        <to uri="bean:userService?method=updateUser" />
    </put>
    <get uri="/list" outType="kr.co.bizfrmae.camel.rest.User[]">
        <to uri="bean:userService?method=listUsers" />
    </get>
</rest>
```

[설명]

output type을 지정함으로써 service 결과를 지정한 Java type에 맞게 출력

[POJO]

```
package kr.co.bizframe.camel.rest;

public class User {

    private int id;
    private String name;

    public User() {}

    public User(int id, String name) {

        this.id = id;
        this.name = name;
    }

    public int getId() {

        return id;
    }

    public void setId(int id) {

        this.id = id;
    }

    public String getName() {

        return name;
    }

    public void setName(String name) {

        this.name = name;
    }
}
```

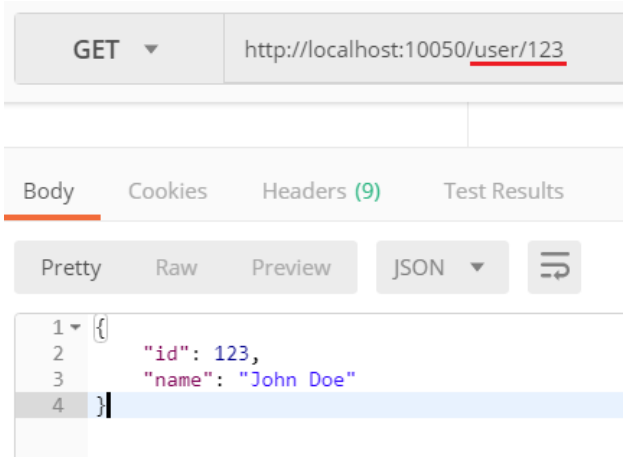
```
package kr.co.bizframe.camel.rest;

import java.util.Collection;
import java.util.Map;
import java.util.TreeMap;

public class UserService {

    private final Map<String, User> users = new TreeMap<>();

    public UserService() {
        users.put("123", new User(123, "John Doe"));
        users.put("456", new User(456, "Donald Duck"));
    }

    public User getUser(String id) {

        return users.get(id);
    }

    public Collection<User> listUsers() {
        return users.values();
    }

    public void updateUser(User user) {
        users.put("" + user.getId(), user);
    }
}
```

UserService의 list에 임의로 2개의 정보({123, John Doe}, {456, Donald Duck}) insert

[route]

i .

```
<?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">

    <bean id="jetty" class="org.apache.camel.component.jetty9.JettyHttpComponent9">
    </bean>

    <bean id="userService" class="kr.co.bizframe.camel.rest.UserService" />

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

        <restConfiguration component="jetty" scheme="http" host="localhost" port="10050" bindingMode="auto">
        </restConfiguration>

        <rest path="/user" consumes="application/json" produces="application/json">
            <get uri="/{id}" outType="kr.co.bizframe.camel.rest.User">
                <to uri="bean:userService:method=getUser(${header.id})" />
            </get>
        </rest>

    </camelContext>

 </beans>
```

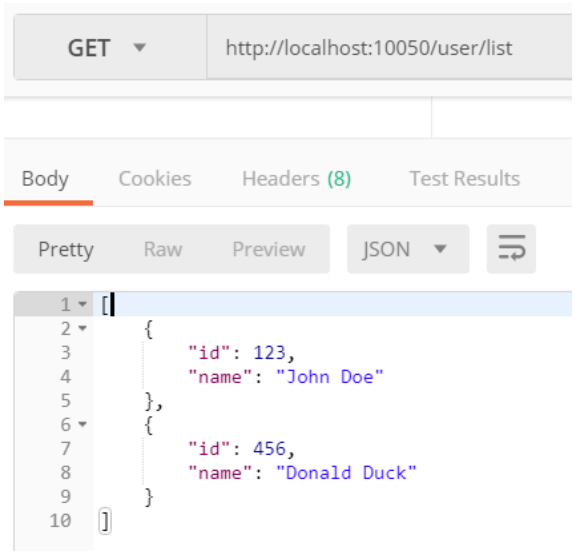


"/user/{id}" uri로 접속한다면, {id}가 일치하는 정보(id, name) 확인

id가 123인 사람 정보{John Doe, 123} 출력

ii.

```
<?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">

    <bean id="jetty" class="org.apache.camel.component.jetty9.JettyHttpComponent9">
    </bean>

    <bean id="userService" class="kr.co.bizframe.camel.rest.UserService" />

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

        <restConfiguration component="jetty" scheme="http" host="localhost" port="10050" bindingMode="auto">
        </restConfiguration>

        <rest path="/user" consumes="application/json" produces="application/json">
            <get uri="/list" outType="kr.co.bizframe.camel.rest.User[]">
                <to uri="bean:userService:method=listUsers" />
            </get>
        </rest>

    </camelContext>

 </beans>
```

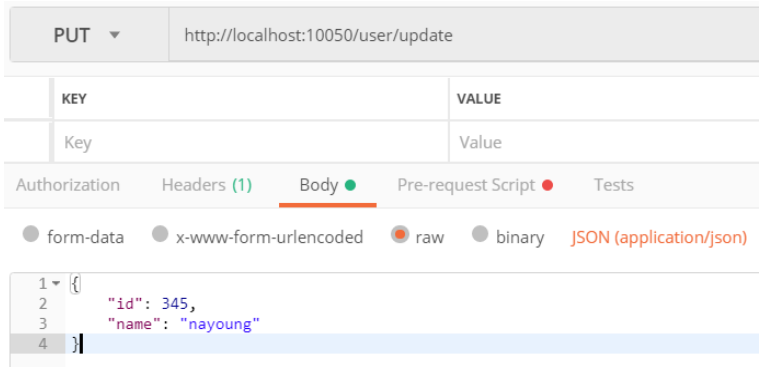


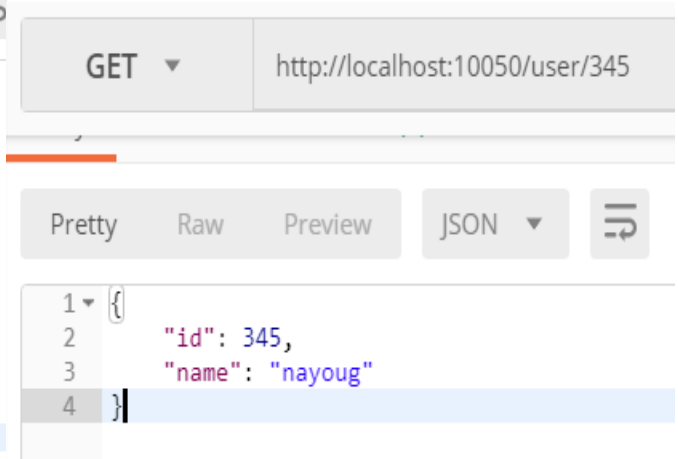
"/user/list"는 등록된 모든 사람들의 정보를 list([]) 형태로 보여준다.

현재 저장되어 있는 list(users[])에 존재하는 2개의 정보를 {id, name} 형태(JSON)로 출력

iii.

```
<?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">

    <bean id="jetty" class="org.apache.camel.component.jetty9.JettyHttpComponent9">
    </bean>

    <bean id="userService" class="kr.co.bizframe.camel.rest.UserService" />

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

        <restConfiguration component="jetty" scheme="http" host="localhost" port="10050" bindingMode="auto">
        </restConfiguration>

        <rest path="/user" consumes="application/json" produces="application/json">
            <put uri="/update" type="kr.co.bizframe.camel.rest.User">
                <to uri="bean:userService?method=updateUser" />
            </put>
        </rest>

    </camelContext>

 </beans>
```





"/user/update" uri로 들어가면 PUT method 실행 가능

PUT method 통해 {id=345, name=nayoung}이라는 데이터를 JSON 형태로 insert

Update 이후, "/user/345", "/user/list" 통해 제대로 PUT method가 실행되었다는 것 확인

6.3 Spark-rest Component

Spark REST Java library 이용하여 REST 정의

Java 8 이상 환경에서만 사용 가능

(주의) Apache Spark와는 연관 X

→ Apache spark를 camel component로 사용하고 싶다면 camel-spark 사용

URI format

spark-rest://verb:path?[options]

- Verb : GET, POST, PUT, DELETE
- path : 경로

```
<restConfiguration component="spark-rest" host="localhost" port="10050" bindingMode="auto">
</restConfiguration>

<rest path="/rest">
    <get uri="/get">
        <to uri="bean:helloBean" />
    </get>
    <post uri="/post">
        <to uri="bean:postBean" />
    </post>
</rest>
```

6.4 Restlet Component

Restlet 이용하여 REST service 정의

```
<restConfiguration component="restlet" host="localhost" port="10050" bindingMode="auto">
</restConfiguration>

<rest path="/rest">
    <get uri="/get">
        <to uri="bean:helloBean" />
    </get>
    <post uri="/post">
        <to uri="bean:postBean" />
    </post>
</rest>
```