

표준프레임웍3.2 기반 nexacro platform UI Adaptor 연동모듈

The Best UI·UX platform for Business User eXperience



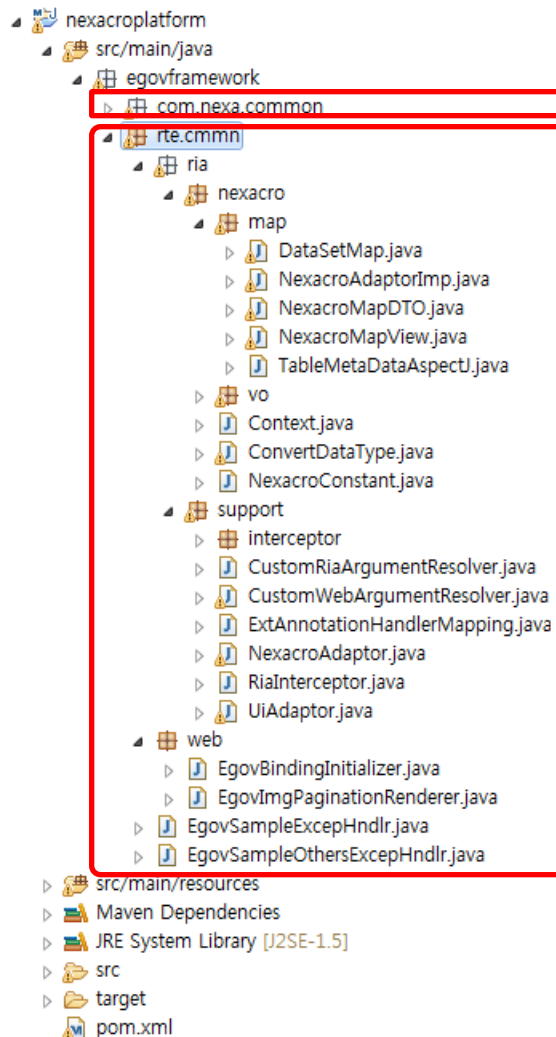
What's exchanged? UX Changed Business!

Agenda

1. 표준프레임워크3.2 연동 UI Adapter 패키지구조
2. 표준프레임워크3.2 개발 아키텍처 View
3. 표준프레임워크3.2 개발 아키텍처 적용규칙
4. 표준프레임워크3.2 개발 데이터처리 적용규칙
5. 표준프레임워크3.2 연동 UI Adapter 적용가이드
6. 컨트롤러 메소드 구현 예
7. UI Adaptor 역할
8. UI Adaptor 구성
9. UI Adaptor 기능구현
10. UI Adaptor 자료구조

What's exchanged? UX Changed Business!

1. 표준프레임워크3.2 연동 UI Adaptor 패키지구조

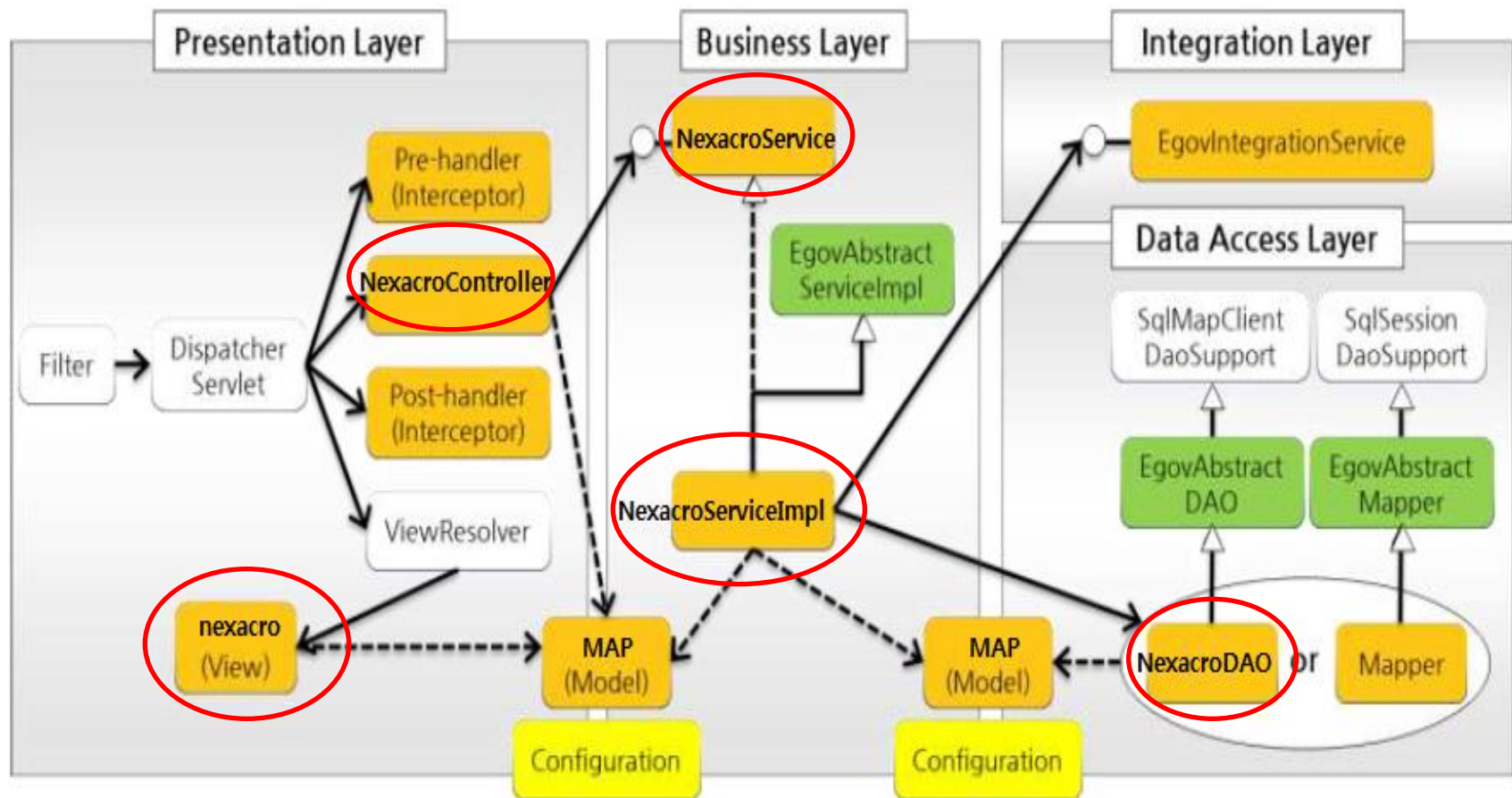


1 업무서비스 Package

2 UI Adapter Package

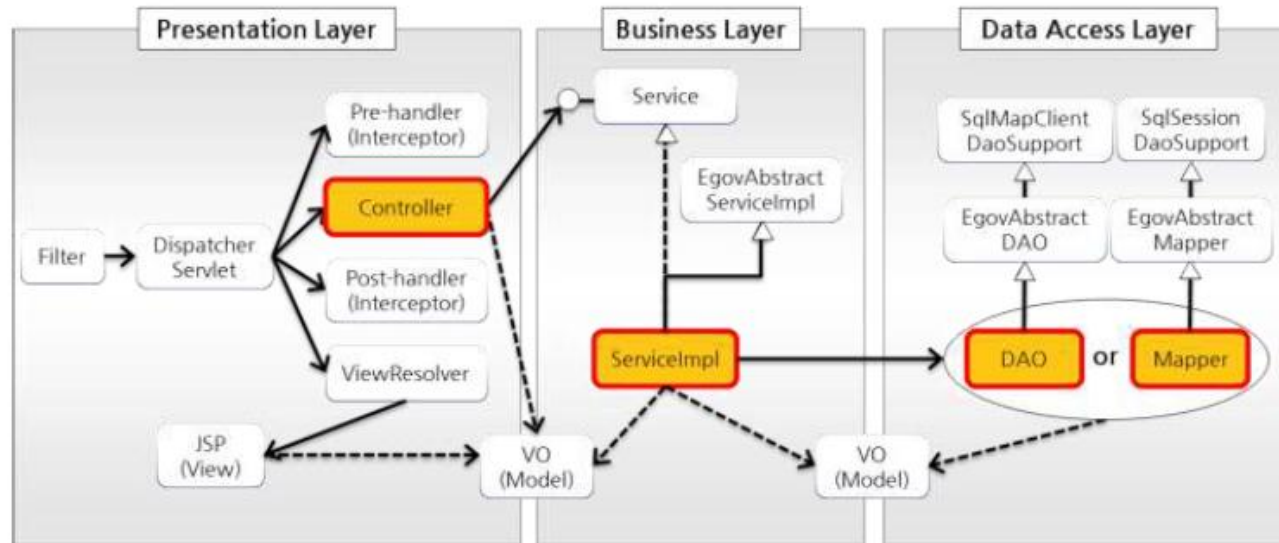
2. 표준프레임워크3.2 개발 아키텍처 View

- egovframework.com.nexa.common (업무서비스 Package)



3. 표준프레임워크3.2 개발 아키텍처 적용규칙(1/2)

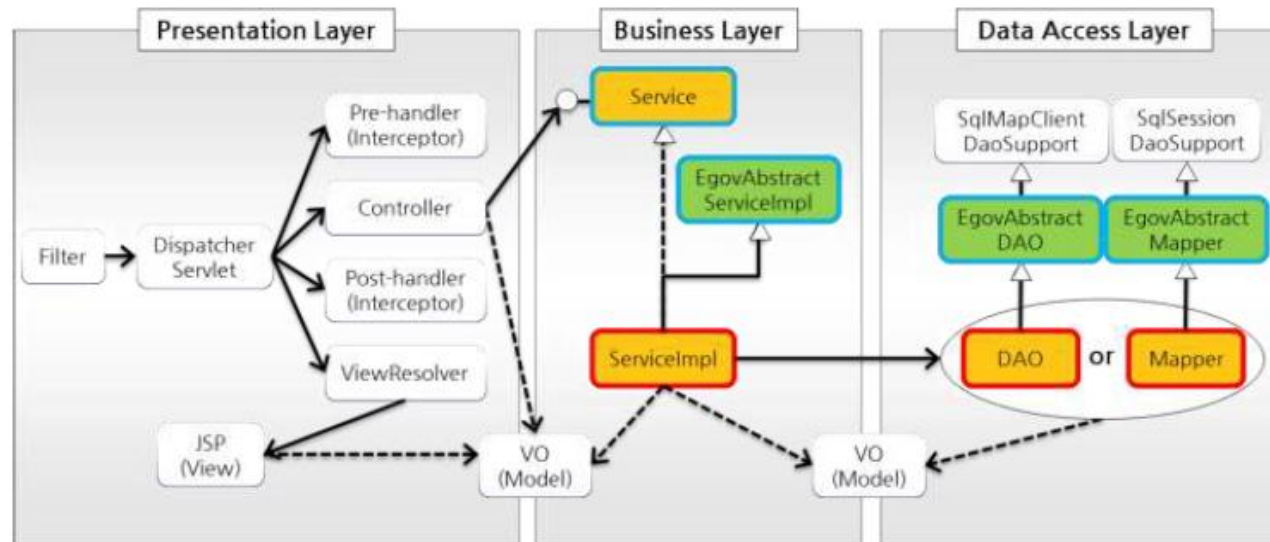
□ Annotation 기반 Spring MVC 및 Layered architecture 준수



대상 클래스	적용규칙
Controller 클래스	<ul style="list-style-type: none"> 클래스 상단에 <code>@Controller</code> 선언 URL Mapping 시 <code>@RequestMapping</code> 선언 View 부분과 Model(business logic 및 data) 부분을 Controller를 통해 분리
ServiceImpl 클래스	<ul style="list-style-type: none"> 클래스 상단에 <code>@Service</code> 선언
DAO/Mapper 클래스	<ul style="list-style-type: none"> 클래스 상단에 <code>@Repository</code> 선언 단, <i>MyBatis Mapper Interface</i> 방식을 활용하는 경우 에는 <code>@Mapper</code> 선언

3. 표준프레임워크3.2 개발 아키텍처 적용규칙(2/2)

□ 클래스 상속 및 인터페이스 구현 규칙



대상 클래스	적용규칙
ServiceImpl 클래스	<ul style="list-style-type: none"> AbstractServiceImpl 또는 EgovAbstractServiceImpl(3.0버전 이상)을 상속/확장하고, 업무에 대한 특정 Service 인터페이스를 구현하여야 함
DAO/Mapper 클래스	<ul style="list-style-type: none"> EgovAbstractDAO(iBatis) 또는 EgovAbstractMapper(MyBatis)를 상속/확장하여야 함 MyBatis Mapper Interface 방식의 경우, @Mapper 선언 Hibernate/JPA 혹은 Spring Data JPA 방식의 경우, 정해진 규칙 없음 ‘데이터처리 적용규칙’ 참조

4. 표준프레임워크3.2 개발 데이터처리 적용규칙(1/2)

□ iBatis 적용 시 데이터 처리 방법 예시

- EgovAbstractDAO 클래스 상속/확장

```
@Repository("employeeDao")  
public class EmployeeDao extends EgovAbstractDAO { ... }
```

```
public class EgovAbstractDAO extends SqlMapClientDaoSupport { ... }
```

- iBatis 사용을 위한 XML 설정파일 3가지
 - iBatis 공통설정파일 (<sqlMapConfig> ~ </sqlMapConfig>)
 - SQL 매핑파일 (<sqlMap> ~ </sqlMap>)
 - SqlMapClientFactoryBean 빈설정

❑ MyBatis 적용 시 데이터 처리 방법 예시 (1/2)

1) EgovAbstractMapper 클래스 상속/확장

```
@Repository("employeeMapper")  
public class EmployeeMapper extends EgovAbstractMapper { ... }
```

```
public abstract class EgovAbstractMapper extends SqlSessionDaoSupport {
```

- MyBatis 사용을 위한 XML 설정파일 3가지

- MyBatis 공통설정파일 (<configuration> ~ </configuration>)
- SQL 매핑파일 (<mapper> ~ </mapper>)
- SqlSessionFactoryBean 빈설정

2) Mapper Interface 방식

```
@Mapper("employeeMapper")  
public interface EmployeeMapper { ... }
```

- MyBatis Mapper Interface 사용을 위한 XML 설정파일 4가지

- MyBatis 공통설정파일 (<configuration> ~ </configuration>)
- SQL 매핑파일 (<mapper> ~ </mapper>)
- SqlSessionFactoryBean 빈설정
- MapperConfigurer 빈설정

```
<!-- MapperConfigurer setup for MyBatis Database Layer -->  
<bean class="egovframework.rte.psl.dataaccess.mapper.MapperConfigurer">  
<property name="basePackage" value="스캔할 Mapper Interface가 속한 패키지명" />  
</bean>
```

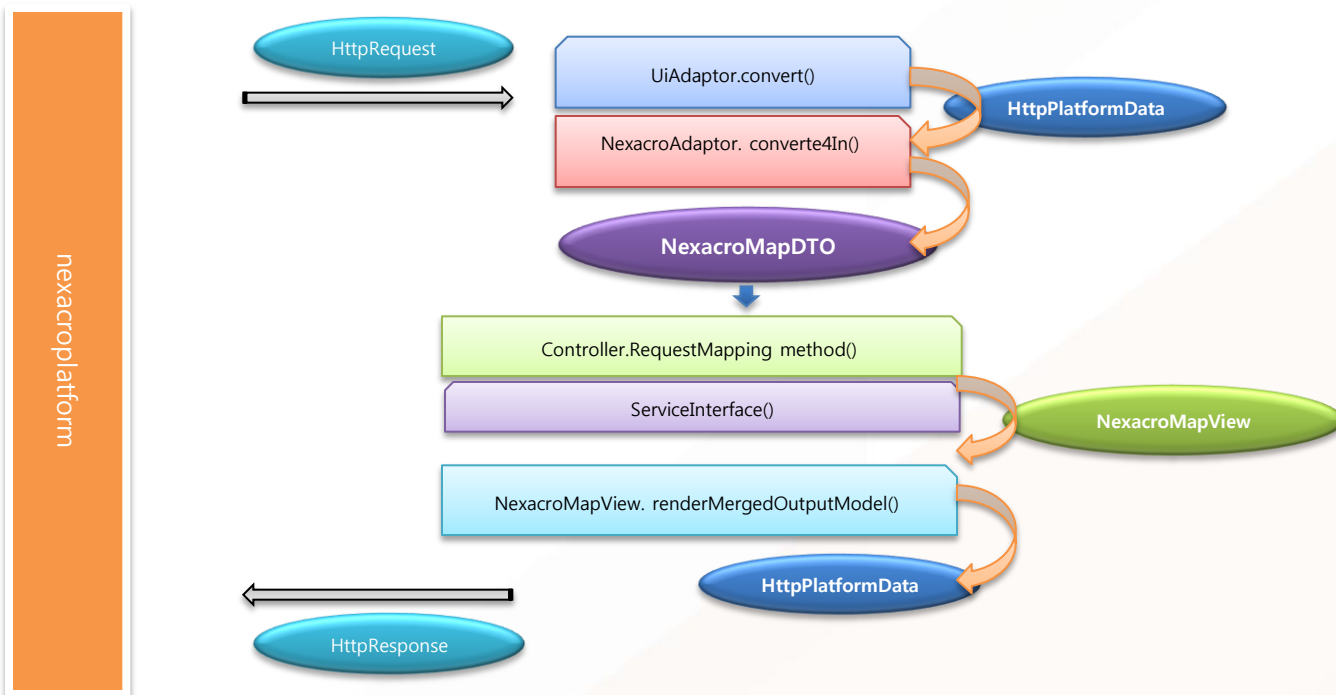

5. 표준프레임워크3.2 연동 UI Adapter 적용 가이드(1/4)

전자정부 표준 프레임워크와 UI솔루션(넥사크로플랫폼, XPLATFORM, 마이플랫폼)을 연동하기 위한 아키텍처 패턴.

전자정부 표준프레임워크에서는 **Spring MVC Annotation** 기반으로 개발시 요청되는 **URI** 와 **Controller** 클래스내의 메소드를 매핑하고 있으므로 메소드의 파라미터로 넘어오는 객체를 **request** 객체가 아닌 업무용 **DTO** 클래스로 전달할 수 있게 가이드 하는 방식

전자정부 표준 프레임 워크 UI Adaptor 설정순서

1. **CustomRiaArgumentResolver**
Adaptor 구분하여 해당 UiAdaptor를 호출한다.
HttpRequest를 nexacroplatform의 PlatformData로 변경한다.
2. **NexacroAdaptorImp**
XPlatformDTO를 생성한다.
DTO class로 값을 변경/설정 한다.
3. **NexacroMapDTO**
Xplatform과 전자정부프레임워크 사이의 Data 전달 클래스
Request의 요청Data를 nexacroplatform의 Data형태의 Variable 및 DataSet으로 변환한다.
4. **NexacroMapView**
서비스 처리후의 결과 정보를 nexacroplatform의 Data형태의 Variable 및 DataSet으로 변환한다.



[설정 정보] : CustomRiaArgumentResolver

```
<bean class="org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdaptor">
  <property name="webBindingInitializer">
    <bean class="egovframework.rte.fdl.web.common.EgovBindingInitializer" />
  </property>
  <property name="customArgumentResolvers">
    <list>
      <bean class="egovframework.rte.fdl.sale.web.CustomRiaArgumentResolver">
        <property name="uiAdaptor">
          <ref bean="riaAdaptor" />
        </property>
      </bean>
    </list>
  </property>
</bean>
```

[CustomRiaArgumentResolver.java] : 클라이언트 식별

```
Public class CustomRiaArgumentResolver implements WebArgumentResolver {
  private UiAdaptor uiA;
  public void setUiAdaptor ( UiAdaptor uiA ) {          this.uiA = uiA ;          }
  public Object resolveArgument(MethodParameter methodParameter, NativeWebRequest webRequest) throws Exception {
    Class<?> type = methodParameter.getParameterType();
    Object uiObject = null ;
    if ( uiA == null ) return UNRESOLVED ;
    //Controller의 실행되는 메소드의 파라미터타입 정보가 MethodParameter를 통해 넘어온다
    //설정한 UiAdaptor 구현체에 등록되어 있는 UTO와 비교한다.
    if ( type.equals( uiA.getModelName() ) ) {
      HttpServletRequest request = (HttpServletRequest) webRequest.getNativeRequest() ;
      // 여기서 데이터 만들어 넘긴다.
      uiObject = (UdDTO) uiA.convert( request ) ;
      return uiObject ;
    }
    return UNRESOLVED ;
  }
}
```

[RiaAdaptorImpl.java] : nexacro platform ⇒ UdDTO

```
Public class RiaAdaptorImpl implements UiAdaptor {

    protected Log log = LogFactory.getLog( this.getClass() ) ;

    // resolveArgument 메소드에서 호출하는 메소드
    public Object conver( HttpServletRequest request) throws Exception {

        PlatformRequest platformRequest = null ;
        try {
            platformRequest = new PlatformRequest( request, PlatformRequest.CHARSET_UTF8) ;
            platformRequest.receiveData();
        } catch ( IOException ex) {
            ex.printStackTrace() ;
        }
        // UI 솔루션 데이터에서 DTO 객체로 변환
        UdDTO dto= convert4Inf(platformRequest) ;
        return dto;
    }

    private UdDTO convert4In(PlatformRequest platformRequest) {
        UdDTO dto = new UdDTO();

        // DTO 또는 VO 값 채우기
        ...
        return dto ;
    }

    public Class getModelName() {
        return UdDTO.class;
    }
}
```

[RiaView.java] : DTO ⇒ nexacro platform

```
public class RiaView extends AbstractView {
    protected Log log = LogFactory.getLog( this.getClass() ) ;
    @SuppressWarnings("unchecked")
    @Override
    protected void renderMergedOutputModel(Map model, HttpServletRequest request, HttpServletResponse response) throws Exception {
        VariableList miVariableList = new VariableList();
        DataSetList miDataSetList = new DataSetList();
        platformData platformData = new PlatformData( miVariableList, miDataSetList ) ;
        List list = (List) model.get("MiDTO");
        Iterator<Map> iterator = list.iterator();
        Iterator<Map> dataIterator = list.iterator();
        Dataset dataset = new Dataset("egovDs");
        while( iterator.hasNext() ) {
            Map<String, Object> record = iterator.next();
            Iterator<String> si = record.keySet().iterator();
            while( si.hasNext() ) {
                dataset.addColumn(key, ColumnInfo.COLUMN_TYPE_STRING, (short) 255 ) ;
            }
        }
        while( dataIterator.hasNext() ) {
            Map<String, Object> record = dataIterator.next();
            Iterator<String> si = record.keySet().iterator();
            while( si.hasNext() ) {
                String key = si.next();
                dataset.addColumn(key, ColumnInfo.COLUMN_TYPE_STRING, (short) 255) ;
            }
            int row = dataset.appendRow();
            Iterator<String> si2 = record.keySet().iterator();
            while( si2.hasNext() ) {
                String key = si2.next();
                String value = (String) record.get(key);
                dataset.setColumn(row, key, value);
            }
            miDataSetList.add(dataset);
        }
        try { new PlatformResponse( response, PlatformConstants.CHARSET_UTF8).sendData(platformData); }
        catch( IOException ex) { if ( log.isDebugEnabled() ) { log.error(" Exception occurred while writing xml to MIPlatform Stream.", ex); }
            throw new Exception(); }
    }
}
```

- UdDTO 클래스는 CustomRiaArgumentResolver 에서 만들어져 Controller 의 메소드의 parameter 형태로 전달.
아래 예는 Controller 단의 메소드 구현 예.

[XXCategoryController.java] : nexacro platform 연계 컨트롤로 적용 예

```
@RequestMapping("/sample/miplatform.do")
Public ModelAndView selectCategoryList4Mi(UdDTO miDto, Model model) throws Exception {

    ModelAndView mav = new ModelAndView("riaView") ;

    Map<String, String> smp = new HashMap<String, String>();      // 조회조건이 있을 경우 사용될 맵
    try {

        List resultList = CategoryService.selectCategoryList(smp); // Biz Layer를 호출한다.
        mav.addObject("MiDTO", resultList);                       // 결과값을 모델에 저장

    } catch( Exception ex )
        log.info( ex.getStackTrace(), ex ) ;
    }
    return mav ;
}
```

7. UI Adaptor 역할(1/2)

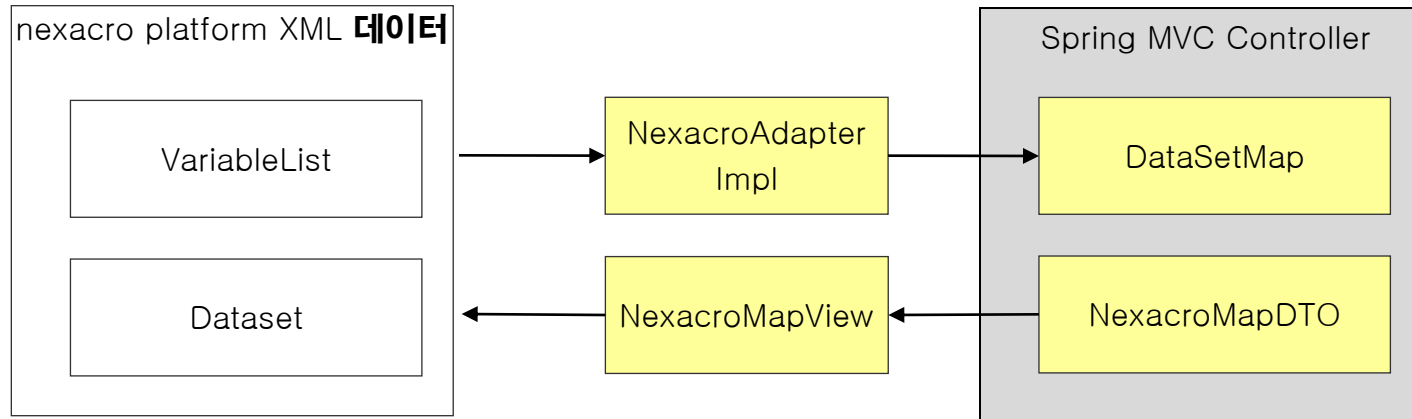
RIA, HTML5 솔루션 연동 필요
nexacro platform 연동 코딩 최소화

UI Adaptor(nexacro platform Converter) 제공

화면데이터를 Map Object로
자동 변환

멀티데이터셋 지원

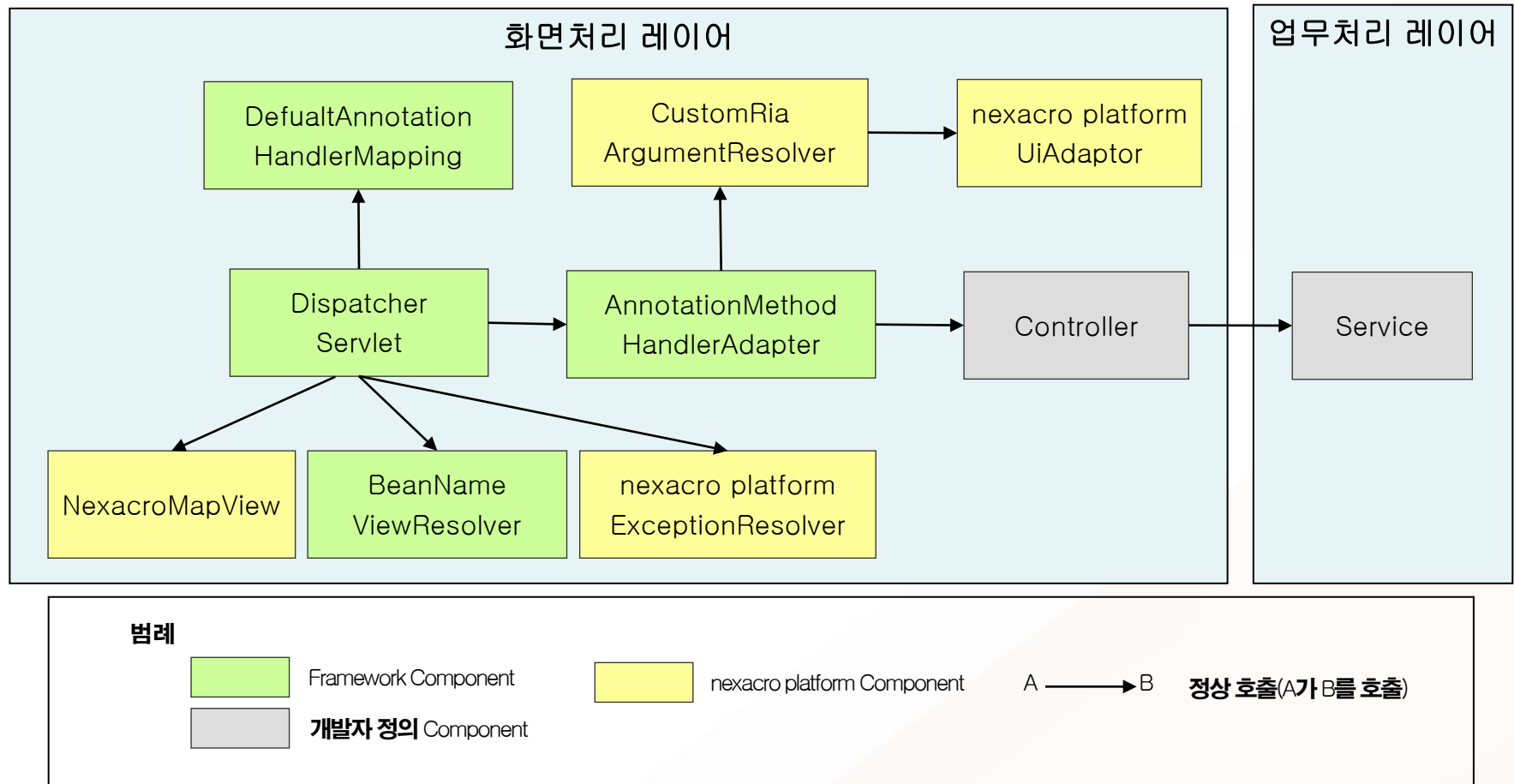
예외발생시 처리



- ✓ nexacro platform과 Spring MVC 연계 컴포넌트는 nexacro platform 과 송수신 데이터 변환을 다룸
- ✓ nexacro platform 은 VariableList와 Dataset으로 구성되는 데이터 구조를 가짐
- ✓ 수신한 데이터는 nexacro platform UiAdaptor를 통해 NexacroMapDTO 로 변환되어 Controller에 전달함
- ✓ Controller는 데이터 처리 후 결과를 NexacroMapDTO 데이터에 담아 NexacroMapView 를 통해 nexacro platform에 전달함

7. UI Adaptor 역할(2/2)

Controller의 메소드 호출 시 넘겨지는 request객체를 Argument Resolver를 이용하여 nexacro platform DTO로 변환하는 방식을 적용함



8. UI Adaptor 구성(1/2)

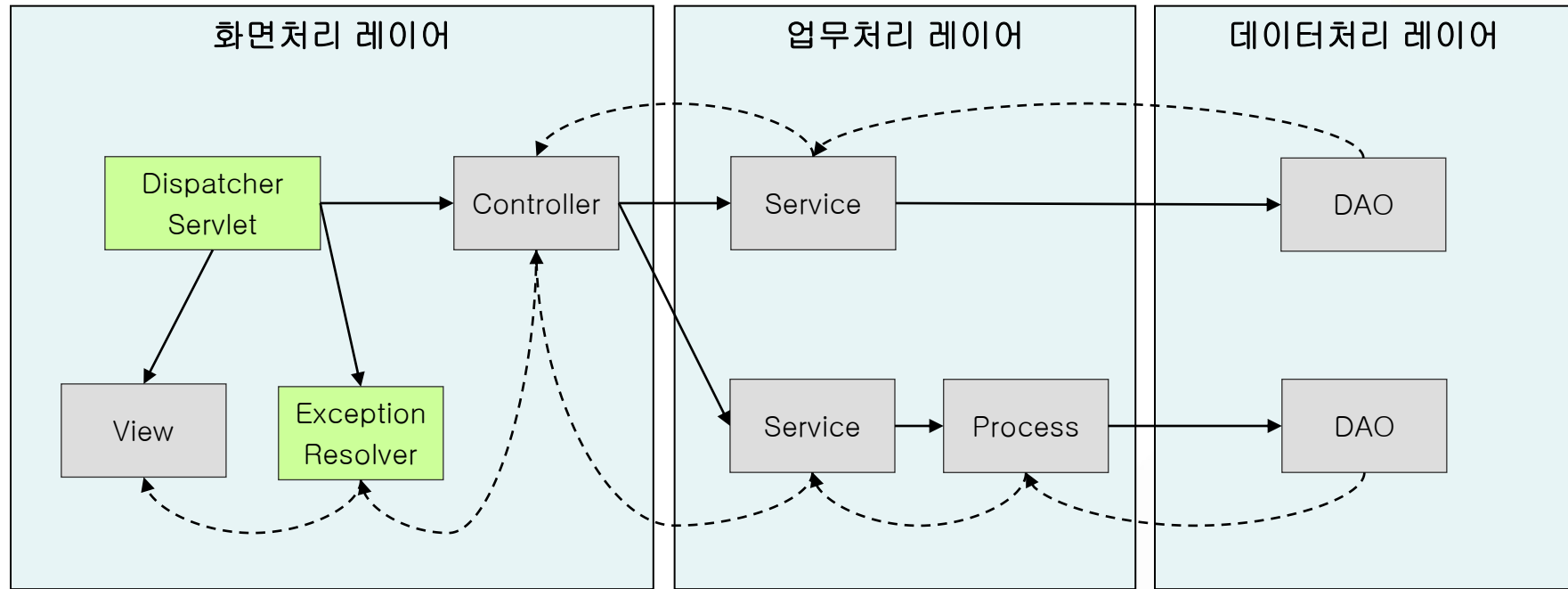
컴포넌트	설명
DispatcherServlet	<ul style="list-style-type: none"> ● 스프링 MVC의 Front Controller로서, 웹요청과 응답의 생명주기를 주관함 <ul style="list-style-type: none"> ✓ 사용자의 브라우저에서 웹서버를 통해 보내는 http/https 요청을 접수하는 단일화된 통로임 ✓ 브라우저와 Controller 사이의 연결 역할을 수행함 ✓ 사용자의 요청을 토대로 controller 정보를 호출하고 결과를 view로 dispatch 함 ✓ Exception 종류에 따른 처리를 위임함
DefaultAnnotationHandlerMapping	<ul style="list-style-type: none"> ● 웹 요청 시 해당 URL을 Annotation 매핑에 기반하여 처리할 Controller를 검색함
AnnotationMethodHandlerAdaptor	<ul style="list-style-type: none"> ● 웹 요청 시 HTTP path, method, 요청 파라미터에 기반하여 Controller에서 처리 method를 검색함
CustomRiaArgumentResolver	<ul style="list-style-type: none"> ● HTTP Request를 Controller에서 사용할수 있는 형태로 변환하여 Controller 에 전달함
NexacroAdaptor	<ul style="list-style-type: none"> ● HTTP Request를 MiPlatform 개체로 변환하는 기능을 제공함
Controller	<ul style="list-style-type: none"> ● 사용자 요청 데이터를 이용하여 비즈니스 로직을 수행하고 결과 데이터와 화면(ModelAndView)을 전달함 ● 시스템 접근 권한 및 서비스 사용 권한을 점검함 ● 업무처리, 데이터처리에서 발생한 예외를 처리하여 사용자에게 전달함 ● UI Validation 수행함 ● 비즈니스 처리는 서비스(Service) 컴포넌트에 위임함

8. UI Adaptor 구성(2/2)

컴포넌트	설명
Nexacro platform ExceptionResolver	<ul style="list-style-type: none">● nexacro platform Client에게 전송할 예외 처리를 수행함
BeanName ViewResolver	<ul style="list-style-type: none">● Controller 수행 후 사용자에게 전송할 View를 BeanName으로 결정함
NexacroMapView	<ul style="list-style-type: none">● 사용자에게 처리된 결과 데이터를 MiPlatform Client에 전송함● 사용자 입력한 값에 대한 검증을 수행함.● 사용자 요청을 입력 받아 서버로 요청 처리함
Service	<ul style="list-style-type: none">● 단일화된 비즈니스 서비스를 접속 지점을 클라이언트에게 제공함<ul style="list-style-type: none">✓ 업무 로직이 단순한 경우 직접 엔티티(DAO) 컴포넌트를 호출하여 서비스를 제공함✓ 업무 로직이 복잡한 경우 업무 처리 흐름에 따라 Process 컴포넌트를 호출하여 서비스를 제공함● 비즈니스 트랜잭션을 단일화하여 처리함● 보안 관리(인증, 권한 관리)를 담당함

9. UI Adaptor 기능구현(1/6) - Exception 처리

개발자가 개발하는 대부분의 method는 Exception을 catch하여 사용하지 않고 단지 throw 하는 형태로 작성된다



범례



Framework Component



개발자 정의 Component

A → B 정상 호출(A가 B를 호출)

A ← B 예외 호출(B가 A를 호출)

개발자에 의한 exception 처리 시에는 try~catch하지 않는다.
만약 try~catch가 필요한 경우는 exception chaining을 위해 catch부에서 반드시 원본 exception을 포함하여 throw 하여야 한다

[BizException 사용 예]

```
//특정한 Biz rule 위배시
if ( ! userMgtService.dupCheckA1User(paramMap).equals("0") ) {
    throw new BizException("CM_ERR_034|A1 사용자"); //A1 사용자가 아닙니다.
}
```

//try~catch가 예외적으로 필요한 경우
.....

```
try {} catch(Exception e) {
    throw e;
}
try {} catch(Exception e) {
    throw new BizException("MD_ERR_001", e);
}
}
```

[RiaArgumentResolver] : nexacro platform 에서 전달되는 data를 처리하여 controller의 argument로 넘기기 위한 클래스

```
public class RiaArgumentResolver implements WebArgumentResolver {

    private static final Logger log = Logger.getLogger(RiaArgumentResolver.class);

    /**
     * request의 내용을 변경해 주는 uiAdaptor
     */
    private UiAdaptor uiAs;

    public void setUiAdaptors(UiAdaptor uiAs) {
        this.uiAs = uiAs;
    }

    public Object resolveArgument(MethodParameter methodParameter, NativeWebRequest webRequest) throws Exception {

        Class<?> type = methodParameter.getParameterType();
        Object uiObject = null;

        if (uiAs == null) {
            return UNRESOLVED;
        }
        if (type.equals(uiAs.getModelName())) {
            log.debug("[method]" + methodParameter.getMethod().getName() + " [type]" + type);
            HttpServletRequest request = (HttpServletRequest) webRequest.getNativeRequest();
            uiObject = (Object) uiAs.convert(request);
            return uiObject;
        }
        return UNRESOLVED;
    }
}
```

9. UI Adaptor 기능구현(4/6) - NexacroMapDTO

【NexacroAdaptorImpl】 : nexacro platform에서 전송된 request의 내용을 NexacroMapDTO Object에 담아 return하여 주는 클래스

```
public class NexacroAdaptorImpl extends NexacroAdaptor {

    @Override
    public Object converte4In(HttpPlatformRequest httpPlatformRequest, HttpServletRequest request) {
        log.debug("1.XPlatformAdaptorImpl.convert4In() start");
        NexacroMapDTO dto = new NexacroMapDTO();
        PlatformData pfd = httpPlatformRequest.getData();
        log.debug("    Request Variable & DataSet debug");
        //log.debug(pfd.saveXml());

        dto.setVariableMap(pfd.getVariableList(), pfd.getDataSetList());
        dto.setTranInfoMap(pfd.getDataSetList());
        dto.setInDataSetMap(pfd.getDataSetList());
        return dto;
    }

    @Override
    public Class getModelName() {
        return (new NexacroMapDTO().getClass()); // 단순히 nexacro platform Model을 생성하고 전달함
    }
}
```

9. UI Adaptor 기능구현(5/6) - NexacroMapView

[NexacroMapView] :nexacro platform으로 Response를 전달하는 View 클래스

```
public class NexacroMapView extends AbstractView {

    @Override
    protected void renderMergedOutputModel(Map model, HttpServletRequest request, HttpServletResponse response) throws Exception {
        logger.debug("XPlatformView.renderMergedOutPutModel() start%%");
        PlatformData platformData = new PlatformData();
        VariableList outVariableList = new VariableList();
        DataSetList outDatasetList = new DataSetList();

        setResult(model, outVariableList);
        setOutVariableMap(model, outVariableList);
        setOutDataSetMap(model, outDatasetList);

        platformData.setVariableList(outVariableList);
        platformData.setDataSetList(outDatasetList);

        logger.debug(platformData.saveXml());

        /** XML output 객체(PlatformResponse) 만들기 **/
        HttpPlatformResponse hPlatformRsp = new HttpPlatformResponse(response);
        hPlatformRsp.setCharset(PlatformType.DEFAULT_CHAR_SET);
        hPlatformRsp.setData(platformData);
        hPlatformRsp.sendData();
    }

    ...

}
```


[Argument Resolver 설정 예]

```
<!-- nexacro platform handler adapter -->
<bean class="org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter">
  <property name="webBindingInitializer">
    <bean class="egovframework.rte.cmmn.web.EgovBindingInitializer"/>
  </property>
  <property name="customArgumentResolvers">
    <list>
      <bean class="egovframework.rte.cmmn.ria.support.CustomRiaArgumentResolver">
        <property name="uiAdaptor">
          <ref bean="xpMapAdaptor" />
        </property>
      </bean>
    </list>
  </property>
</bean>

<bean name="xpMapAdaptor" class="egovframework.rte.cmmn.ria.nexacro.map.NexacroAdaptorImp" />
```

[Controller 코딩 예]

```
@RequestMapping("/nexacroController.do")
public ModelAndView operate(NexacroMapDTO xpDto, Model model) throws Exception {
    ModelAndView mav = new ModelAndView("nexacroMapView")

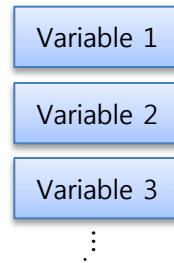
    ...

    return mav;
}
```

1. nexacroplatform에서 입력 Variables과 DataSets 값을 세팅한다.
2. nexacroplatform에서 설정된 값으로 eGov서버로 Http Request한다.
3. NexacroAdaptor에서 PlatformData로 변경한다.
4. NexacroAdaptorImpl에서 NexacroMapDTO를 생성한다.
5. NexacroAdaptor에서 PlatformData(Variable,DataSet)를 NexacroMapDTO에 자동 변환된다.
6. 변환될 data는 Variable은 inVariable, DataSet은 inDataset 멤버로 설정된다.
7. eGov서버는 호출된 URL에 맞는 Controller에 전달한다.
Controller의 @RequestMapping 형식으로 url 호출로 진행 시킨다.
8. 추상서비스단을 호출한다.
serviceMethod(Map tranInfoDataset, Map inVariables, Map inDatasets, Map outVariables, outDatasets);
9. Service에 들어온 variables 과 DataSets를 이용하여 실제서비스를 요청한다.
10. Service Implement 의 결과값을 DTO의 variable과 DataSet에 저장한다.
10. ModelAndView에 결과값인 outDataSets와 outVariables를 추가한다.
11. ModelAndView에 XPlatform의 ErrorCode와 ErrorMessage값을 추가한다.
12. NexacroMapView에서 결과값들을 Variables과 DataSet으로 변환한다.
13. nexacroplatform에 response 한다.

- nexacroplatform의 transaction을 이용 시 사용되는 데이터 통신의 기본 단위이다. PlatformData는 n개의 Variable (VariableList)와 n개의 DataSet (DataSetList) 로 구성되어있다.

VariableList



DataSetList

dataset1

Column 1	Column 2	Column 3	Column 4
1	a1	b1	c1
2	a2	b2	c2
3	a3	b3	c3

...

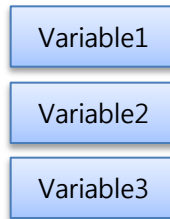
dataset2

Column 1	Column 2	Column 3	Column 4
1	a1	b1	c1
2	a2	b2	c2
3	a3	b3	c3

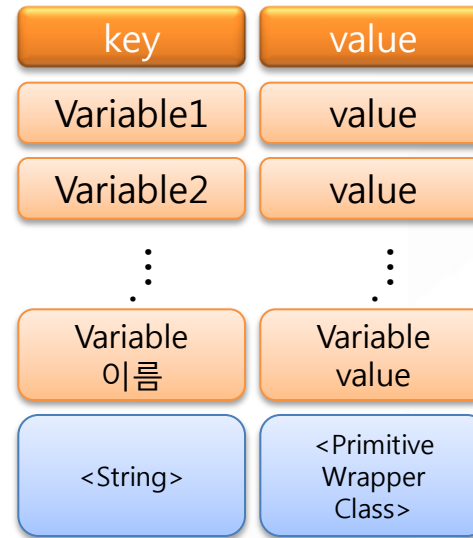
● 변환 작업

● VariableList

<List>



<Map>



List 형태의 Variable 들을 Map 형태로 변환한다.
Variable의 이름을 key로 사용하고 Primitive Wrapper class로 value를 저장한다.

...

● 변환 작업

● DataSetList

<List>

dataset1	Colu mn1	Colu mn2	Colu mn3	Colu mn4
	1	a1	b1	c1
	2	a2	b2	c2
	3	a3	b3	c3

dataset2	Colu mn1	Colu mn2	Colu mn3	Colu mn4
	1	a1	b1	c1
	2	a2	b2	c2
	3	a3	b3	c3



<Map>

key	value
dataset1	value
dataset2	value
⋮	⋮
DataSet 이름	DataSet value
<String>	<List>

⋮

List 형태의 DataSet 들을 Map 형태로 변환한다.
DataSet이름을 key로 사용하고 실제Data는 List형태로 저장한다.

...

● Variable

- 데이터를 저장하는 변수를 의미한다
식별자(name)와 값(value)로 구성된다.
데이터의 형식(type)도 소유하고 있다

● Variable

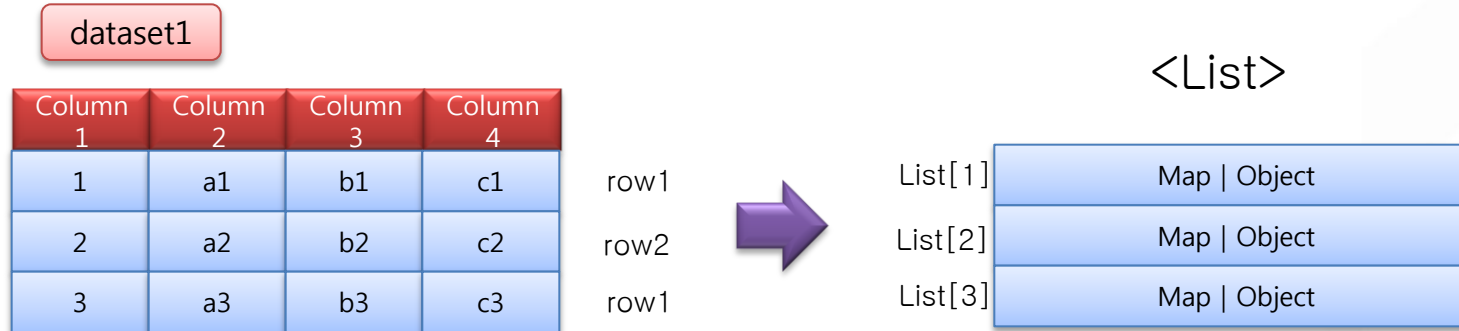


...

DataSet

- 열(column)과 행(row)으로 구성되는 2차원 데이터를 저장한다.

List형태로 변환



DataSet values의 변환은 List 타입 변경하고
하나의 row 정보는 Map이나 Object의 정보로 담는다.

주의) 현재 개발된 소스는 Map 정보로만 개발 되어 있음

...

1) Map으로 변환.

Column 1	Column 2	Column 3	Column 4
1	a1	b1	c1

row1

List[1] Map

key	value
column1	1
column2	a1
⋮	⋮
rowType	Inserted Updated Deleted
Column 이름	row value
<String>	<Primitive Wrapper Class>



2	a2	b2	c2
3	a3	b3	c3

row2
row1

List[2] Map
List[3] Map

Key는 column이름을 사용하며, value는 해당 column의 row에 위치하는 값이다.

하나의 row를 저장하므로 DataSet의 row Type의 정보를 추가로 저장될수있다.

Row type은 Inserted(삽입), Updated(수정), Deleted(삭제)이며 해당 정보를 이용하여 서비스에 반영한다

1) Map으로 변환.

- UI에서 올려주는 Transaction 정보가 필요

Dataset이름

__DS_TRANS_INFO__

Column정보

Request DataSet 명

Response DataSet 명

Ex)

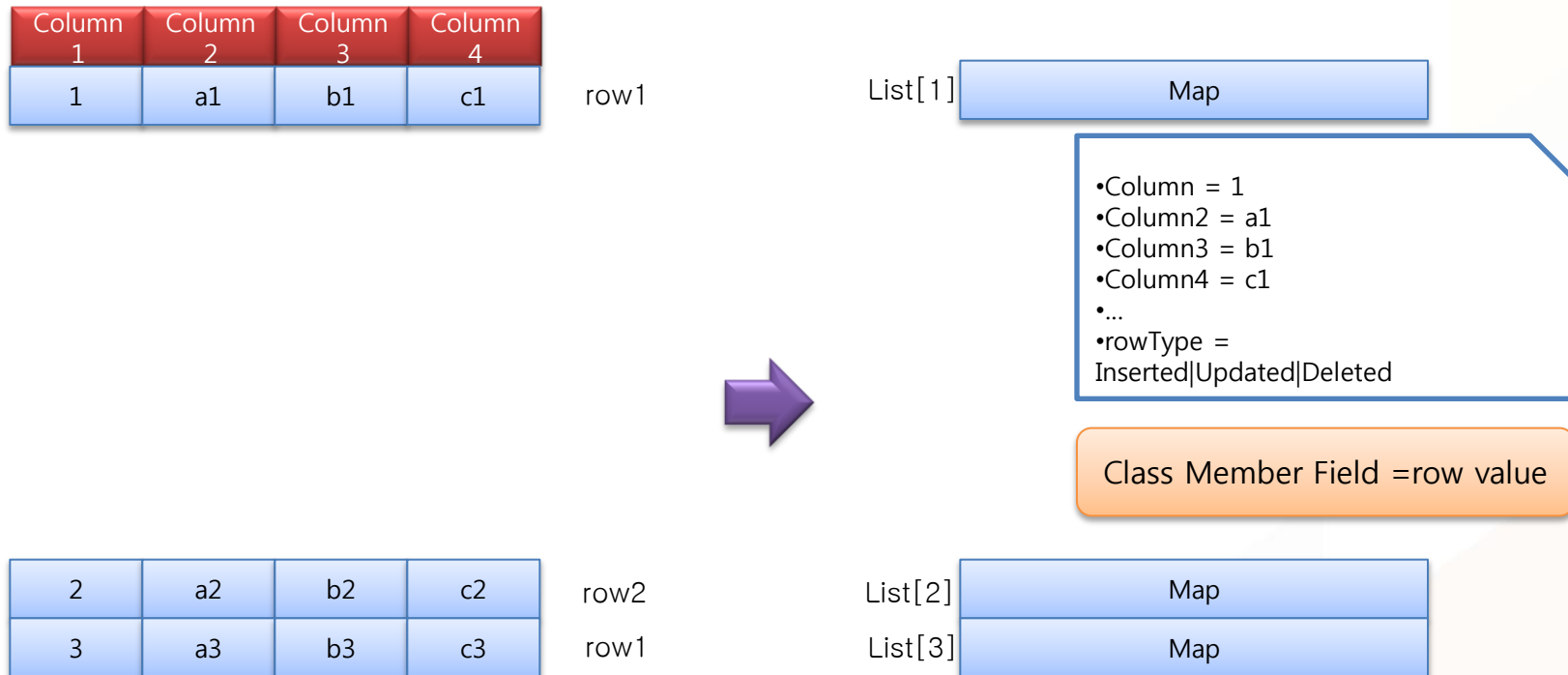
__DS_TRANS_INFO__

Request DataSet 명	Response DataSet 명
inDataSet1	outDataSet1
inDataSet2	outDataSet2

서비스 개발자는 Request DataSet 중 “__DS_TRANS_INFO__”의 명을 가진 DataSet을 이용한다.
해당 DataSet의 Request의 DataSet 명과 Response DataSet 명을 이용하여 서비스를 개발한다.



2) Object 로 변환.



DataSet의 column명이 Value Object(VO)의 멤버필드로 매핑되어 변환되는 것이다.
rowType 정보를 이용하기 위해서는 AbstractXPlatformVO를 상속받아 사용한다.

2) Object 로 변환.

- UI에서 올려주는 Transaction 정보가 필요

Dataset이름 __DS_TRANS_INFO__

Column정보	Request DataSet 명	Request DataSet 매핑 VO클래스명	Response DataSet 명	Response DataSet 매핑 VO 클래스명
----------	-------------------	---------------------------	--------------------	-----------------------------

Ex)

	__DS_TRANS_INFO__			
	Request DataSet 명	Request DataSet 매핑 VO클래스명	Response DataSet 명	Response DataSet 매핑 VO 클래스명
	inDataSet1	com.tobesoft.sample.service.SampleVO1	outDataSet1	com.tobesoft.sample.service.SampleOut1
	inDataSet2	com.tobesoft.sample.service.SampleVO2	outDataSet2	com.tobesoft.sample.service.SampleOut2

서비스 개발자는 Request DataSet 중 “__DS_TRANS_INFO__”의 명을 가진 DataSet을 이용한다.
 해당 DataSet의 정보를 이용하여 DataSet과 VO클래스명을 이용하여 변환작업을 한다.
 Request DataSet정보는 매핑VO클래스 형태로 변환된다.
 서비스 호출 후 결과값인 VO 클래스를 Response DataSet명으로 세팅한다.

감사합니다.