*com

enRise 업무처리 API 가이드

Version 1.0

2017.01.18

SN Technolodgy

개정 이력

버전	제/개정 일자	제/개정 페이지 및 수정 내용	제/개정자	승인자
1.0	2017.01.18	최초 작성	최승환	

목 차

1 업무처리 API

enRiseFramework 의 업무처리에 주로 사용되는 공통기반의 API 를 설명한다

1.1 Validator

1) 명명규칙

- 패키지 명 : com + enrise + (제품구분) + (기능구분) + vo + validator

  ex) com.enrise.share.orgn.vo.validator (그룹웨어시스템의 조직관리업무)

- 클래스명 : (VO 명) + Validator

  ex) UserInfoValidator (UserInfo VO 클래스의 Validator 클래스)

2) Validator 클래스

- Validator 클래스는 springframework 의 validation 패키지의 Validator 인터페이스를 Implements 받아서 클래스를 생성하고 유효성을 체크할 VO 클래스에 맞게 메서드를 오버라이딩한다
- springframework 의 Singleton registry 에 객체를 바인드 하기 위해 @Component 어노테이션을 사용하고 id 는 클래스명과 일치시킨다
- 유효성 검증은 Validate Utility 클래스의 메소드를 활용한다

ex)

```
package com.enrise.share.orgn.vo.validator;


import org.springframework.stereotype.Component;
import org.springframework.validation.Errors;
import org.springframework.validation.Validator;


import com.enrise.share.util.Validate;



@Component("UserInfoValidator")  <---- id 명은 클래스명과 반드시 일치
public class UserInfoValidator implements Validator {  <---- Validator 인터페이스 implements


        @Override
```

```
        public boolean supports(Class<?> clazz) {
                return (UserInfoVO.class.isAssignableFrom(clazz));
                        <---- 오버라이딩한 메소드의 유효성 체크할 클래스로 변경
        }
       @Override
        public void validate(Object target, Errors errors) {
                UserInfoVO userInfo = (UserInfoVO)target;
                   <---- 오버라이딩한 메소드의 유효성 체크할 클래스로 명시적 형변환
                if(!Validate.required(userInfo.getUserNm())){
                        <---- Validate Utility 클래스를 통한 데이터 유효성 검증
                          errors.rejectValue("userNm", "MSG_INFO_INPUT_USER_NAME");
                }
}
```

3) Validate Utility 클래스

- com.enrise.share.util 패키지에 위치한 Validate 클래스
- 유효성 검증 메소드는 아래와 같다

메소드명	설명
required	null 이거나 “” 빈문자열인지 여부를 체크하며 값이 존재하는 경우 true 리턴한다
email	문자열 데이터가 이메일 형식인지를 체크한다
url	문자열 데이터가 URL 형식인지를 체크한다
password	문자열 데이터가 6 자 이상 20 자 이하이면서 숫자와 문자로 되어있는지를 체크한다
tel	문자열 데이터가 전화번호 형식인지를 체크한다
ipAddress	문자열 데이터가 IP 주소 형식인지를 체크한다
minByte	문자열 데이터의 바이트수가 지정한 바이트수보다 크거나 같은지를 체크한다
maxByte	문자열 데이터의 바이트수가 지정한 바이트수보다 작거나 같은지를 체크한다
rangeByte	문자열 데이터의 바이트수가 지정된 범위 내의 바이트수인지를 체크한다
equalLength	문자열의 길이가 지정한 길이와 같은지를 체크한다 (바이트와 무관)
minLength	문자열의 길이가 지정한 길이보다 크거나 같은지를 체크한다 (바이트와 무관)
maxLength	문자열의 길이가 지정한 길이보다 작거나 같은지를 체크한다 (바이트와 무관)

rangeLength	문자열의 길이가 지정된 범위 내의 길이인지를 체크한다 (바이트와 무관)
min	정수 또는 실수가 지정한 수보다 크거나 같은지를 체크한다
max	정수 또는 실수가 지정한 수보다 작거나 같은지를 체크한다
range	정수 또는 실수가 지정된 범위내의 수인지를 체크한다

4) Controller 에서 유효성 체크

- Controller 클래스의 부모클래스인 AbstractController 클래스의 bindRequestValidateErrorHandler 메서드를 통해 유효성을 검증하고 성공여부에 따라 분기처리한다

ex)

```
@Controller
public class OrgnUserInfoController extends AbstractController {
        @Resource(name="UserInfoValidator")
        private UserInfoValidator userInfoValidator;

       @RequestMapping("/ajax/orgn/admin/setPersonInfo.do")
        @ResponseBody
        public ResultVO setPersonInfo(HttpServletRequest request,
                        @RequestBody UserInfoVO userInfo, Locale locale){
              ResultVO ret = new ResultVO();
             userInfo.setActionType(Globals.ACTION_TYPE.UPDATE);
              ret = super.bindRequestValidateErrorHandler(request,
                                        userInfoValidator, userInfo, "UserInfo");
                    <---- bindRequestValidateErrorHandler 를 통한 유효성 검증
              if(!ret.isSuccess()){
                    return ret;
              }
             …
       }
}
```

1.2 Message 처리

1) Message Properties 파일

- main/resources/com/enrise/config/message 폴더의

  message_en_US.properties 와 message_ko_KR.properties 파일에 사용할 메시지를 작성한다

  (한국어, 영어 이외의 다른 언어를 사용할 경우 message_{locale}.properties 파일을 생성한 후 모든 메시지를 동일하게 작성하여 사용한다)

- 메시지에 파라미터를 사용하는 경우는 {index}를 사용한다 (index 는 0 부터 사용함)

  ex)

MSG_WARN_OVER_LENGTH={0}의 길이가 {1}를 초과하였습니다

- 메시지명 명명규칙

  Label , 버튼, 메뉴등 UI 텍스트는 UI_(기능구분)_(명칭) 형식으로 작성한다 (공통사항인 경우 기능구분은 생략할 수 있다) ex) UI_APVL_DOCINFO=결재문서정보

  경고, 정보, 에러등 메시지 문장은 MSG_(기능구분)_(메시지구분)_(명칭) 형식으로 작성한다 (공통사항인 경우 기능구분은 생략할 수 있다) ex)
  MSG_APVL_ERR_SET_DOC_INFO=결재문서정보를 설정시 문제가 발생하였습니다

- 메시지 구분

  ERR : 에러 메시지

  WARN : 경고 메시지

  INFO : 확인 및 정보 메시지

  DEBUG : 디버그용 메시지

2) EnriseMessageSource 클래스

- 메시지는 Springframework 의 ReloadableResourceBundleMessageSource 클래스를 활용한다. Locale 은 LocaleChangeInterceptor 를 통해서 변경 가능하며 세션에 지정된 locale 정보를 통해 메시지소스 클래스로 바인딩 된다

- 디버그용 메시지를 제외하가 Client 에 전달할 메시지는 Controller 클래스에서 생성하는 것을 원칙으로 하며, Service 및 DAO 에서는 BizException 을 통해 메시지코드 및 파라미터를 전달하고 Controller 에서 이를 통해 메시지를 생성하여 Client 에 전달한다

- 메시지를 Client 에 전달할 경우 반드시 Controller mapping 메소드에 Locale locale 를 지정하고 이를 통해 다국어 처리가 가능하도록 구현해야 한다

  ex) 서비스 메소드에서 메시지 코드 전달

```
throw processException("MSG_ERR_FAIL_INSERT_USER_INFO");
 <---- processException 메소드는 AbstractServiceImpl 에 존재하며 오버로딩 되어 있음. 해당 클래스를 참고하면 메시지 파라미터를 전달 할 수 있으며, 결과는 BizException 으로 throw 됨
```

ex) Controller 클래스 메시지 생성

```
@Resource(name="enriseMessageSource")
private EnriseMessageSource messageSource;


@RequestMapping("/ajax/orgn/admin/insertUserInfo.do")
@ResponseBody
public ResultVO insertUserInfo(@RequestBody OrgnUserInfo param,
                HttpServletRequest request, Locale locale) {  <---- Locale 파라미터 지정
        try{
            …
        }catch(Exception e) {
                String messageKey = "MSG_ERR_FAIL_INSERT_USER_INFO";
                if(e instanceof BizException) {
                    messageKey = ((BizException)e).getMessageKey();
                     <---- BizException 을 통해 메시지 코드 설정
                }
                ret.addError("exception", messageSource.getMessage(messageKey));
                     <---- EnriseMessageSource 를 통해 메시지 생성
        }
}
```

3) JSP 페이지 메시지 생성

- spring 의 message 태그라이브러리를 사용한다
- message 태그라이브러리 사용시 반드시 text 속성을 작성한다

ex)

```
<spring:message code=" UI_APVL_COMPLETE_DIST_DOC_BOX" text="배부완료함" />
   <---- spring:message 태그라이브러리를 사용함 text 속성은 반드시 작성하여 유지보수가 용이하도록 함
```

1.3 Globals 설정

1) Globals Properties 파일

- 시스템 공통 사항 및 옵션은 globals.properties 파일에 작성한다
- maven 설정으로 인해 main/resources/com/enrise/config/props/ 폴더에 globals.properties 와 local, dev 폴더 하위의 globals.properties 총 3 개의 파일을 수정해야 한다

2) Globals 클래스

- Globals 클래스는 com.enrise.share.util 패키지에 위치한다
- Globals 의 설정정보 추가는 아래와 같으며 반드시 static final 예약어를 지정하도록 한다

ex) Globals 클래스에 프로퍼티 추가

```
package com.enrise.share.util;


public class Globals {


	/**
	 * 시스템 유형
	 */
	public static final String SYSTEM_TYPE = EnriseProperties.getProperty("Globals.SYSTEM_TYPE");
		<---- EnriseProperties.getProperty(“globals.properties 파일에 지정한 프로퍼티명”);


	/**
	 * 시스템 명
	 */
	public static final String SYSTEM_NAME = EnriseProperties.getProperty("Globals.SYSTEM_NAME");
	…
```

ex) Globals 프로퍼티 사용

```
If(“EKP”.equals(Globals.SYSTEM_TYPE)) {
    …
}
```

1.4 사용자 인증 및 UserDetailHelper 설정

1.4.1 사용자 인증

- enRiseFramework 현 버전은 Session 을 통한 사용자 인증방식을 사용한다
- context-common.xml 에 설정된 authCheckInterceptor, ajaxAuthCheckInterceptor Interceptor 를 참고한다 (Interceptor 를 활용하기 때문에 요청 mapping 메소드에는 인증로직을 별도로 구현하지 않는다)
- authCheckInterceptor 는 세션을 통해 인증되지 않은 경우 로그인 페이지로 이동되며 ajaxAuthCheckInterceptor 는 세션을 통해 인증되지 않은 경우 JSON 값을 responce 한다
- 인증이 체크를 하지 않는 URL 인 경우는 authExceptHandlerMapping 에 해당 URL 을 설정한다

1) 로그인 처리

- 로그인 처리는 OrgnLoginController 클래스에서 처리한다

```
@RequestMapping("/orgn/simpleLogin.do")
public String simpleLogin(HttpServletRequest request, HttpServletResponse response,
        ModelMap model, @RequestParam("userId")String userId,
        @RequestParam("userPw")String userPw,
        @RequestParam(value="isSaveId", required=false, defaultValue="N")String isSaveId) {
    …
    try{
        …
        loginVO = loginService.login(loginVO);
            <---- 로그인 서비스를 통해 실사용자인지 비밀번호가 일치하는지 체크
        String targetURL = null;
        //인증에 성공한 사용자인 경우
        if(loginVO.isCertification()) {
            HttpSession session = request.getSession();
            …
            session = request.getSession(true);
            session.setAttribute("LoginVO", loginVO);
            <---- 인증성공시 loginVO 정보를 반드시 “LoginVO” 속성에 추가
                세션에 추가정보를 저장할 때는 LoginVO 에 필드를 생성하여 추가함
```

```
                세션정보는 LoginVO 이외에는 추가로 설정하지 않는 것을 원칙으로함
session.setAttribute("binding.listener", OrgnSessionManager.getInstance());
<---- 세션 중복 체크를 위해 binding.listener 추가
…
```

2) 로그인 사용자 정보 얻기

- OrgnUserDetailsHelper 클래스를 사용한다

ex)

```
@RequestMapping("/ajax/orgn/admin/getUserInfo.do")
@ResponseBody
public ResultVO getUserInfo(){
        LoginVO loginInfo = OrgnUserDetailsHelper.getAuthenticatedUser();
          <---- getAuthenticatedUser 메소드를 통해 세션에 저장된 사용자정보를 가져온다
        List<String> roles = OrgnUserDetailsHelper. getAuthorities ();
          <---- getAuthorities 메소드를 통해 사용자 권한정보를 가져온다
```

1.4.2 사용자 권한 체크

1) RoleMenuService 클래스

- 시스템 메뉴에 대한 사용자 권한을 체크한다

ex)

```
@Resource(name=" RoleMenuService")
private RoleMenuService menuRole;


@RequestMapping("/apvl/some/index.do");
public String someMethod() {
    …
    List<String> roles = OrgnUserDetailsHelper. getAuthorities ();
    If(menuRole.isRole(roles, RoleCoGlobals. MENU_TYPE.APVL,
                                RoleCoGlobals.APVL_MENU_DRAFT)) {
        … <---- isRole 메소드를 통해 특정 메뉴의 권한여부를 체크
    }
```

```
}


@RequestMapping("/apvl/index.do")
public String goIndex(ModelMap model) throws Exception{
        List<String> userRole = OrgnUserDetailsHelper.getAuthorities();
        model.addAttribute("roles",
                menuService.inRole(userRole, menuService.getRoles(RoleCoGlobals.MENU_TYPE.APVL)));
            <---- inRole 메소드를 통해 특정 메뉴타입의 모든 메뉴의 권한을 map 형식으로 리턴
        return "apvl/index.layout";
}
```

```
<%-- jsp 페이지 --%>
<c:if test="${roles.distribute == true}">
    <li><spring:message code=" UI_APVL_COMPLETE_DIST_DOC_BOX" text="배부완료함" /></li>
</c:if>
```

2) RoleAuthInfoService 클래스

- 사용자의 특정 권한 여부를 체크할 때 사용한다

ex)

```
@Resource(name=" RoleAuthInfoService")
private RoleAuthInfoService authRole;


@RequestMapping("/apvl/some/index.do");
public String someMethod() {
    …
    List<String> roles = OrgnUserDetailsHelper. getAuthorities ();
    If(authRole.inRole(roles, RoleCoGlobals.AUTH. SYS_ADMIN, "부서 KID", "직위 KID")) {
        … <---- inRole 메소드를 통해 사용자의 권한을 체크
    }
}
```

1.5 JMS 및 비동기 처리

- JMS 는 ActiveMQ Open Source 를 사용한다 (상용 WAS 인 경우는 WAS 자체 JMS 를 사용하기를 권장하며 JMS 는 JNDI 를 통해 접근하여 사용하도록 설정한다)
- 대용량 처리, 메일발송, 모바일푸쉬알림 등 처리시간이 오래걸리는 업무는 JMS 를 활용하여 비동기 처리한다. 비동기 처리이기 때문에 트랜젝션이 별도로 분리되어 처리의 성공여부가 반드시 상태 관리되도록 구현되어져야 한다
- 특히 모바일 푸쉬알림 구현시는 PushAlarmListener 를 활용하여 처리되도록 구현해야 한다

  구체적인 사항은 아래의 예제소스를 확인하도록 한다

1) Spring JMS 설정

- context-asynctask.xml 에 설정하여 사용한다

```
<!—ActiveMQ ConnectionFactory 생성. 상용 WAS 의 JMS 인 경우는 JNDI 로 변경 -->
<bean id="amqConnectionFactory" class="org.apache.activemq.ActiveMQConnectionFactory">
        <property name="brokerURL">
                <value>tcp://localhost:61616</value>
        </property>
</bean>
<!—Connection 캐쉬를 통한 성능 고려 -->
<bean id="cachedConnectionFactory"
    class="org.springframework.jms.connection.CachingConnectionFactory">
        <property name="targetConnectionFactory" ref="amqConnectionFactory" />
        <property name="sessionCacheSize" value="10" />
</bean>

<!— Queue 설정 예시 start ------------------------------------------------------------------------------>
<!— 1. destination 설정 -->
<bean id="destination" class="org.apache.activemq.command.ActiveMQQueue">
        <constructor-arg value="messageQueue" />
</bean>
<!— 2. jmsTemplate 설정-->
<bean id="jmsExcelTemplate" class="org.springframework.jms.core.JmsTemplate">
```

```
        <property name="connectionFactory" ref="cachedConnectionFactory" />
        <property name="defaultDestination" ref="destination" />
</bean>
<!— 3. 메시지 자동처리를 위한 listenr 설정 (수동처리시는 생략)-->
<bean id="makeAarsExcelListener" class="com.enrise.office.aars.listener.AarsMakeExcelListener" />
<jms:listener-container connection-factory="cachedConnectionFactory">
        <jms:listener destination="messageQueue" ref="makeAarsExcelListener" />
</jms:listener-container>
<!— Queue 설정 예시 end ------------------------------------------------------------------------------>

<!— Topic 설정 예시 start ---------------------------------------------------------------------------->
<!— 1. destination 설정 -->
<bean id="pushDestination" class="org.apache.activemq.command.ActiveMQTopic">
        <constructor-arg value="pushMessageTopic" />
 </bean>
<!— 2. jmsTemplate 설정-->
<bean id="jmsPushTemplate" class="org.springframework.jms.core.JmsTemplate">
       <property name="connectionFactory" ref="cachedConnectionFactory" />
       <property name="defaultDestination" ref="pushDestination" />
 </bean>
<!— 3. 메시지 자동처리를 위한 listenr 설정 (수동처리시는 생략)
        Topic 인 경우는 1 대 N 이기 때문에 다중 listenter 설정 가능 -->
<bean id="pushAlarmListener" class="com.enrise.share.alarm.listener.PushAlarmListener" />
<bean id="mesAlarmListener" class="com.enrise.share.alarm.listener.MesAlarmListener" />
<jms:listener-container connection-factory="cachedConnectionFactory" destination-type="topic">
       <jms:listener destination="pushMessageTopic" ref="pushAlarmListener" />
       <jms:listener destination="pushMessageTopic" ref="mesAlarmListener" />
 </jms:listener-container>
<!— Topic 설정 예시 end ------------------------------------------------------------------------------>
```

2) JMS Template

- Spring JMS Template 를 통해 쉽게 메시지를 전달 할 수 있음

ex) Message Send

```
@Resource(name="jmsPushTemplate")
private JmsTemplate jmsPushTemplate;


public void send() {
    …
    jmsPushTemplate.send(new MessageCreator() {
            @Override
            public Message createMessage(Session session) throws JMSException {
                    ObjectMessage message = session.createObjectMessage();
                    String alarmMessage = messageSource.getMessage(massegeCode);
                    message.setObjectProperty("userIds", userIds);
                    message.setStringProperty("menu", PushAlarmService.APPROVAL_MENU);
                    message.setStringProperty("massegeCode", senderNm+alarmMessage);
                    return message;
            }
    }); <---- Spring JMS Template 를 통한 메시지 전송
}
```

ex) Message Receive (Listener)

```
import javax.jms.MessageListener;


public class PushAlarmListener implements MessageListener{
     <---- 반드시 jms 패키지의 MessageListenr 를 implements 해야함
      @Override
      public void onMessage(Message message) {
           <---- onMessage 메소드를 오버라이딩 하여 전달 받은 메시지를 처리한다
          ObjectMessage msg = (ObjectMessage)message;
          List<String> userIds = (List<String>) msg.getObjectProperty("userIds");
```

```
        String massegeCode = (String) msg.getObjectProperty("massegeCode");
        String menu = (String) msg.getObjectProperty("menu");
        …
    }
}
```

1.6 스케줄러

- 일정 기간 또는 시간동안 반복하여 처리하거나 특정일에 처리되도록 설정해야하는 스케줄 업무 처리에 사용한다
- context-scheduler.xml 에 설정된 Bean 을 사용한다
- 업무처리를 위한 Job 클래스만 생성하고 환경설정 -> 스케줄관리 메뉴에 등록하여 사용한다
- 기본적으로 Quartz Open Source 를 사용하며 해당 가이드에서는 Job 클래스를 생성하는 방식을 설명한다

1) 명명규칙

- 패키지 명 : com + enrise + (제품구분) + (기능구분) + schedule

  ex) com.enrise.share.orgn.schedule (그룹웨어시스템의 조직관리업무)

- 클래스명 : (기능구분) + (세부업무) + Job

  ex) OrgnSyncUserInfoJob (조직관리 연동 스케줄러 Job 클래스)

2) Job 클래스

- com.enrise.framework.schedule.AbstractJobBean 을 반드시 상속받아 클래스를 생성한다

- Job 클래스는 applicationJobScheduler 를 통해서 Bean 이 생성되어 지기 때문 @Resource 와 @Autowise 어노테이션을 통해 서비스 및 DAO 등 Singleton 객체를 생성할 수 없다

- Singleton 객체를 생성시는 AbstractJobBean 의 getBean 메소드를 통해 생성한다

ex) ApvlRelayService relayService = super.getBean(ctx, "ApvlRelayService", ApvlRelayServiceImpl.class);

ex) Job 클래스 예제

```
public class ApvlReceiveExternalJob extends AbstractJobBean {
        @Override
        protected void executeInternal(JobExecutionContext ctx)
                        throws JobExecutionException {
            <---- executeInternal 메소드를 오버라이딩 하여 처리할 업무를 구현한다
```

```
                ApvlRelayService relayService = super.getBean(ctx, "ApvlRelayService",
                        ApvlRelayServiceImpl.class);
                …  <---- Spring Single 객체를 생성시는 super.getBean 메소드를 사용한다
        }
}
```

1.7 메일 발송

자바 메일을 통해 SMTP 프로토콜을 통해 메일을 발송한다

관련 클래스는 com.enrise.framework.mail 패키지에 저장되어 있다

1) 메일서버 설정

- content-mail.xml 의 mailSender 객체의 host, port, username, password 를 설정한다
- host, port 는 일반적으로 globals.properties 에 설정되어 있다

2) 관련 클래스

클래스명	설명
MailContent	메일본문 정보
MailAttach	메일첨부파일 정보
Member	메일 발송자 또는 수신자 정보
SendMailService	메일 발송 인터페이스
SimpleSendMailServiceImpl	간단한 텍스트를 본문으로 발송하는 메일 발송 구현 클래스 (첨부파일 지원하지 않음)
MimeSendMAilServiceImpl	HTML 형태의 본문내용을 발송하는 메일 발송 구현 클래스

3) 예제 코드

```
@Resource(name="mimeMailSender")
private SendMailService mailService;  //사용할 메일 Service Bean 을 가져옴


@Test
public void sendMail() throws Exception {
    Member to = new Member();
    to.setId("f00013");
```

```
    to.setName ("홍길동");
    to.setEmail("f00013@ibk.co.kr");
    MailContent content = new MailContent();
    content.setSubject("메일 제목");
    content.setContent("<b>본문내용</b>");
    mailService.sendMail(mailContent, to);
}
```

1.8 엑셀 파일 생성

리스트 자료를 엑셀 파일로 서버에 생성하여 업로드/다운로드하는 기능으로 대용량을 고려하여 오픈소스인 OpenXML 의 형식으로 엑셀을 파일을 생성하는 기능도 지원한다

관련 클래스는 com.enrise.framework.excel 패키지에 저장되어 있다

1) 명명규칙

- 패키지 명 : com + enrise + (제품구분) + (기능구분) + service + handler

  ex) com.enrise.share.orgn.service.handler (그룹웨어시스템의 조직관리업무)

- 클래스명 : (기능구분) + (세부업무) + ExcelRowHandler

ex) ApvlDocListExcelRowHandler (전자결재 엑셀처리 Handler)

2) 관련 클래스

관련클래스	설명
SpreadSheetWriter	OpenXML 파일을 생성하고 OpenXML Schema 형태로 엑셀 Row 및 셀을 생성한다
ExcelMergeParamVO	셀 병합 정보
ExcelDownloadService	엑셀을 POI 라이브러리 또는 OpenXML 방식으로 생성하기 위한 Service 인터페이스
ExcelDownloadServiceImpl	엑셀을 POI 라이브러리 또는 OpenXML 방식으로 생성하기 위한 Service 인터페이스를 구현한 클래스
ExcelDownloadDAO	엑셀을 저장하는 그리드의 수가 많은 경우는 서버에 부하가 발생되고 Out Of Memory 가 발생될 수 있으며, RowHandler 를 통해 리스트를 가져온다
AbstractExcelRowHandler	엑셀로 생성하는 그리드의 종류는 개발자가 선택해야 하며, 엑셀 다운로드시는 반드시 그리드에 해당하는 RowHandler 를 개발해야 하며 AbstractExcelRowHandler 를 상속받아야 한다

3) 예제 코드

```
package com.enrise.office.apvl.service.handler;


import java.util.Date;


import com.enrise.framework.excel.service.AbstractExcelRowHandler;
import com.enrise.share.util.CommonUtil;
import com.enrise.share.util.DateUtil;


public class ApvlFileExcelRowHandler extends AbstractExcelRowHandler{
	@Override
	public String convert(String fieldName, Object field) {
		 <---- 각 필드명에 해당하는 데이터를 변환하여 리턴한다
		if("draftDt".equals(fieldName) || "compDt".equals(fieldName)){
			return DateUtil.getDateString((Date)field);
		}
		return CommonUtil.checkNull(field);
	}
}
```

4) Excel 다운로드 Client

snt.excelSavePopup.js Plugin 을 사용하여 엑셀 다운로드를 구현한다

framework.excel.web.ExcelDownloadController 를 통해 엑셀이 다운로드되면 JMS 를 통해 비동기 처리되도록 구현되어 있다

JMS 는 com.enrise.office.aars.listner.AarsMakeExcelListener 클래스를 참고한다

ex)

```
@Resource(name="ExcelDownloadService")
private ExcelDownloadService excelDownloadService;


public File downloadAarsDocument(ExcelDownload param) throws Exception {
	AarsExcelRowHandler handler = new AarsExcelRowHandler();
	return  excelDownloadService.download(param.getFileName(),param.getSqlId(),
```

```
                param.getListParam(), handler, AarsExcelListVO.class, param.getExcelParams());
}
```

1.9 파일 업로드/다운로드

파일 업로드/다운로드는 com.enrise.share.file 패키지에 구현되어 있으며, 업로드 Plugin 은 PLUploader 를 사용한다

(참고로 기존에는 이노릭스 업로드를 사용했으나 크로스브라우징 문제로 현재는 사용하지 않으며 일부 사이트에서 IE 만 사용하는 경우 이노릭스 업로더를 사용하기도 한다)

파일 업로드/다운로드는 FileCmmnController 클래스를 통해서 구현되어 있으며 업로드, 다운로드는 아래와 같다

1) 파일 업로드

- PLUploader : /ajax/plupload/fileUpload.do
- 이노릭스 : /innorixsv/fileUpload.do
- 한글 또는 HTML 본문 : /ajax/file/uploadHwpContents.do
- 이미지 Canvas(사진 수정 후 업로드) : /file/uploadCanvas.do

2) 파일 다운로드

- 일반 다운로드 : /file/fileDownload.do
- HTML MimeType 다운로드 : /file/fileDownloadHTML.do
- PDF MimeType 다운로드 : /file/fileDownloadPDF.do

1.10 Logging

로그는 Log4j Open 소스 프레임워크를 사용한다

로그 객체는 반드시 static final 키워드를 사용한다

LogFactory 로 로그 객체 생성시 반드시 현재 클래스를 통해 생성하도록 한다

Debug 로그시는 반드시 isDebugEnabled 메서드를 통해 조건을 부여하고 사용하는 것을 원칙으로 한다

ex)

```
@Service("OrgnUserInfoService")
public class OrgnUserInfoServiceImpl extends AbstractServiceImpl
                implements OrgnUserInfoService{
    private static final Log LOG = LogFactory.getLog(OrgnUserInfoServiceImpl.class);
```

```
public void someMethod() {
        LOG.info("some Method start :: ");
        try {
              ...
        }catch(Exception e) {

            if(LOG.isDebugEnabled()) {
                LOG.debug("someMethod error", e);
            }
        }
}
```

1.11 기타 Utilities

개발자가 업무관련 처리 구현시 공통적으로 사용되는 Utility 클래스로 com.enrise.share.util 패키지에 저장되어 있다

1) 관련 클래스

클래스명	설명
Base64Encoder	Basce64 로 인코딩하기 위한 클래스
ByteUtils	Byte 형의 자료를 연산할 때 사용되는 클래스
CommonUtil	공통적으로 사용되는 기능으로 구성된 클래스
DateUtil	날짜 형의 자료를 연산할 때 사용되는 클래스
DES	DES 암호화 알고리즘 클래스
EnriseProperties	properties 파일의 정보를 메모리에 읽어오기 위한 클래스
FileUtil, FileUtility	파일 처리를 할 때 사용되는 클래스
Globals	공통으로 사용되는 옵션이 저장된 클래스로 globals.properties 값이 저장되어 있다
KID	일련번호가 아닌 Unique 한 Key 를 생성하기 위한 클래스
ListUtil	리스트 구현시 사용되는 클래스
Locale	서버 언어에 따라 한글 인코딩을 처리하기 위한 클래스

NumberUtil	숫자 형의 자료를 연산할 때 사용되는 클래스
SecurityUtil	보안관련 처리를 할 때 사용되는 클래스
StringArray	JDK1.5 이전 버전에서 StringArray 를 사용
StringUtil	문자열 형의 자료를 연산할 때 사용되는 클래스
Validate	유효성 검증 클래스
ValidChecker	XML 의 유효성을 체크하기 위한 클래스
WebUtil	웹 관련 기능을 제공하는 클래스

1.12 web.xml

web.xml 에 주요 설정 사항을 설명한다

구분	설정	설명
Filter	encodingFilter	org.springframework.web.filter.CharacterEncodingFilter 를 사용하며 .do 요청에 대해 모든 요청데이터의 encoding 을 utf-8 로 변경한다
Filter	HTMLTagFilter	com.enrise.framework.filter.HTMLTagFilter 를 사용하며, 요청데이터의 cross site script 공격을 막기위해 HTML 요소문자열을 특수문자로 치환한다
Filter	httpMethodFilter	org.springframework.web.filter.HiddenHttpMethodFilter 를 사용하며 .do 요청에 대해서 spring 3.05 에서 지원하지 않는 RestFul Request 방식을 지원한다 (ex POST, GET, PUT, DELETE 등)
Listener	ContextLoaderListener	org.springframework.web.context.ContextLoaderListener 리스너를 통해 Spring Framework 를 사용한다 스프링 설정 파일은 contextConfigLocation 파라미터 경로를 사용하며 Spring Framework 설정 및 SingleTon Registry Bean 을 생성한다
Listener	Log4j	Log4j 설정 리스너 (classpath:log4j.xml 설정파일을 사용한다
servlet	DispatcherServlet	스프링 MVC 처리를 위한 DispatcherServlet 을 설정하면 요청 URI 패턴은 .do 를 사용한다
servlet	MesSvAssistant	구 EJB framework 메신저 연계를 위한 servlet
servlet	OrgnSvResponseXML	구 EJB framework 메신저 연계를 위한 servlet
etc	Session timeout	1440 분 (24 시간)으로 설정
etc	Welcome File	index.jsp

etc	Tag 라이브러리	Uri : http://www.esnt.co.kr/enrise-common 설정파일위치 : /WEB-INF/tlds/enrise-common.tld
etc	Error Page	404 : /error404.do 500 : /error500.do CmmnController 에 구현되어 있음