

본 문서의 저작권은 (주)대우정보시스템에 있으며 당사의 허락없이 변경, 복제, 배포가 불가합니다. 본 문서에 오류가 있다고 판단될 경우
이슈로 등록해주시면 적절한 조치를 취하도록 하겠습니다. 본 문서는 저작권 조약에 의해 보호받고 있습니다.

1. 3. 개발메뉴얼	2
1.1 00. JCF3.6 기반 개발표준 가이드	2
1.2 01. 실행서비스	16
1.2.1 01.표현레이어	17
1.2.1.1 01.SpringMVC+Flex연계	17
1.2.1.1.1 1.Controller 작성하기(RestFul+Flex연계)	17
1.2.1.1.2 2.Controller 작성하기(AMF방식Flex연계)	24
1.2.1.2 02.SpringMVC+Miplatform 연계	27
1.2.1.2.1 01.Controller 작성하기	27
1.2.1.2.2 02.데이터 유형별 처리하기	30
1.2.1.3 03.SpringMVC+표준웹연계	33
1.2.1.3.1 01. Controller작성하기(일반jsp기반)	33
1.2.2 02.업무처리레이어	37
1.2.2.1 1.서비스클래스작성법	38
1.2.3 03.데이터처리레이어	39
1.2.3.1 1.ibatis사용가이드	40
1.2.3.2 2. iBatis 다이내믹쿼리가이드	42
1.2.3.3 3.util사용하기	43
1.2.3.3.1 1.CamelCaseMap사용하기	43
1.2.3.4 4.StoredProcedure처리하기	44
1.2.3.4.1 1.SP 호출하기(oracle)	45
1.2.3.4.2 2. SP로 리스트조회(oracle)	47
1.2.3.4.3 3. SP로 리스트 조회 (MSSQL)	49
1.2.3.4.4 4. SP로 리스트조회(sysbase)	50
1.2.3.5 7. sequence호출(oracle)	52
1.2.4 05.연계처리레이어	53
1.2.4.1 01.CXF 기반 웹서비스	53
1.3 02. 공통서비스	55
1.3.1 01. 파일처리	55
1.3.1.1 1. 파일업로드	55
1.3.1.1.1 1. SWFUpload 이용 파일업로드	55
1.3.1.2 2. 파일다운로드	62
1.3.1.2.1 1. 파일 다운로드 서블릿	62
1.3.2 02. 로그처리	63
1.3.2.1 1. 로그처리	63
1.3.2.2 2.SQL로깅	68
1.3.2.3 3. iBatis Batch를 사용한 이벤트 로그 처리	71
1.3.2.4 이벤트로깅	73
1.3.3 03. 다국어처리	76
1.3.3.1 1. 다국어처리 - SpringMVC 사용시	77
1.3.3.2 2. 다국어처리 - Struts2 사용시	80
1.3.4 11.세션객체사용하기	80
1.3.5 04. 예외처리(메시지처리)	81
1.3.5.1 1.예외처리(springMVC+Miplatform)	81
1.3.6 06. 트랜잭션처리	83
1.3.7 07. 캐쉬처리	85
1.3.7.1 1. 캐쉬처리 - Java Caching System (JCS) 적용하기	85
1.3.8 08. 보안처리	91
1.3.8.1 1.SpringSecurity 가이드	91
1.3.8.2 2. 세션 중복(중복 로그인) 방지 처리	94
1.3.9 09. 엑셀처리	95
1.3.9.1 01. 엑셀파일생성(jxl 사용)	95
1.3.10 10. property 서비스	98

3. 개발메뉴얼

JCF 개발에 대한 메뉴얼이 있는 공간입니다.

- 스타트 개발가이드는 JCF 초급개발자를 위한 스타트 가이드이며,
- 실행서비스는 JCF 어플리케이션 프레임워크의 Core 를 구성하는 각 Tier 별 컴포넌트 모듈을 구성하고 비즈니스 어플리케이션 개발을 위한 개발패턴을 정의한 가이드이며,
- 공통서비스는 어플리케이션 각 Tier에서 공통으로 사용되는 보안, 트랜잭션, 에러처리와 같은 공통 기능에 대한 가이드입니다.

스타트 개발가이드

00. JCF3.6 기반 개발표준 가이드

실행서비스

- 01.표현레이어
 - 01.SpringMVC+Flex연계
 - 02.SpringMVC+Miplatform 연계
 - 03.SpringMVC+표준웹연계
- 02.업무처리레이어
 - 1.서비스클래스작성법
- 03.데이터처리레이어
 - 1.ibatis사용가이드
 - 2. iBatis 다이내믹쿼리가이드
 - 3.util사용하기
 - 4.StoredProcedure처리하기
 - 7. sequence호출(oracle)
- 04. 배치 시스템
 - 1. 배치관리시스템 적용가이드
 - 2. 배치관리시스템 사용가이드
 - 3. 배치 개발 가이드
 - 98. Shell or Dos 명령어 처리
 - 99. temp
- 05.연계처리레이어
 - 01.CXF 기반 웹서비스

공통서비스

- 01. 파일처리
- 02. 로그처리
- 03. 다국어처리
- 04. 예외처리(메시지처리)
- 06. 트랜잭션처리
- 07. 캐쉬처리
- 08. 보안처리
- 09. 엑셀처리
- 10. property 서비스
- 11.세션객체사용하기

00. JCF3.6 기반 개발표준 가이드



목차

- 개요
- Step 1. 개발환경 설치
 - 설치하기
 - 환경셋팅하기
 - 개인개발환경구성
- Step 2. 프로젝트 생성 및 실행
 - 소제목
- Step 3. 구현하기
 - Model 작성
 - SQL 문 작성(Sql Map XML 파일)
 - Service 작성
 - Controller 작성
 - 화면(JSP) 작성
- Step 4. 테스트
- 첨부파일

문서정보

- 제목: JCF3.6 기반 개발표준 가이드
- 최초작성자: 고경철
- 최초작성일:
- 이문서는 사용자에게 **72**번 보여졌습니다.

개요

본 가이드는 JCF(버전 3.6)기반으로, 단순한 CRUD(입력,수정,삭제,저장) 애플리케이션을 작성해 봄으로써, JCF의 기본 흐름을 이해하기 위하여 제공한다.

본 가이드의 Spring Framework, ibatis등의 기본지식은 다른 페이지에 따로 가이드하며, 편의상 자세한 내용은 이 문서에서 생략하기로 한다.

아래의 4Step에 따라 순서대로 따라하기 방식으로 진행된다.

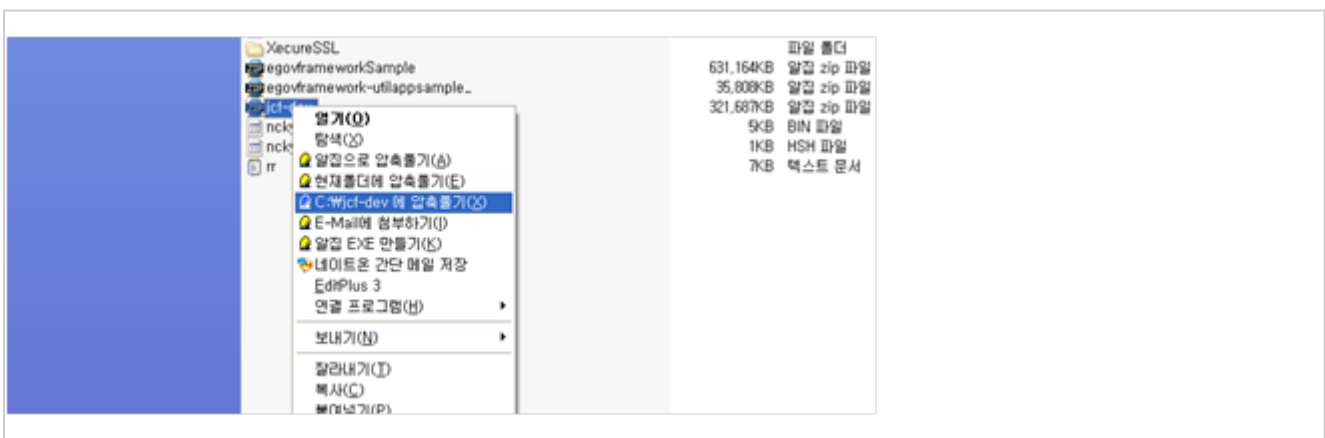
1. 개발환경 설치
2. 프로젝트 생성
3. 구현하기
4. 테스트하기

Step 1. 개발환경 설치

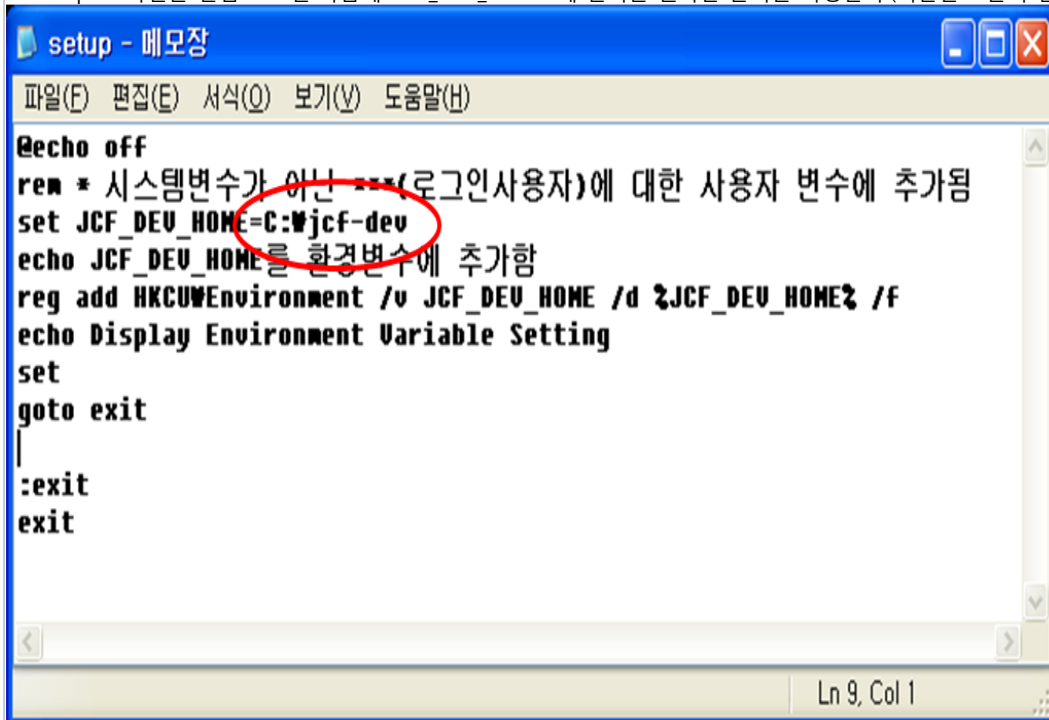
- 개인개발환경은 이클립스의 플러그인 구성에 따라 standard, professional 용으로 나뉜다.
- standard 용은 springIDE와 메이븐 등 JCF 기반 하에서 개발에 필요한 최소한의 필요한 플러그인만으로 구성하였고, professional 용은 그 외에도 EMMA 나 PMD등 품질관리 플러그인, 이슈관리, HTTP 뷰어 플러그인 등으로 구성하였다.

설치하기

1. 제공된 혹은 다운받은 jcf-dev.zip 파일을 설치를 원하는 폴더에 압축을 풀어 놓는다.



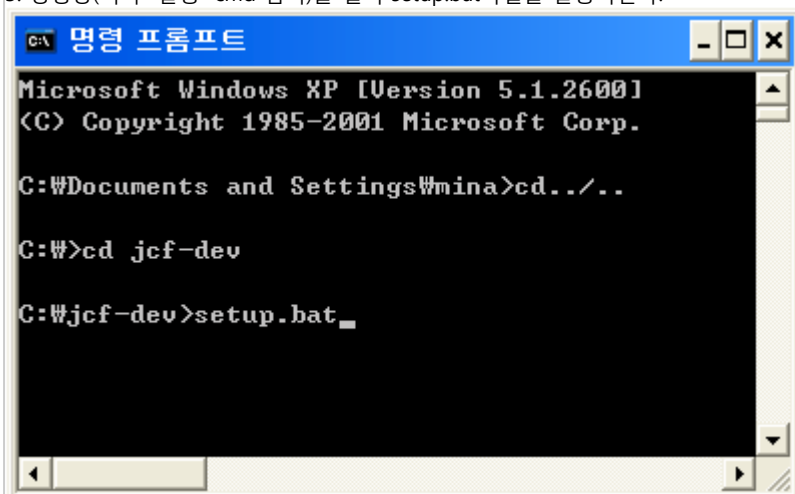
2. setup.bat파일을 편집으로 연 다음에 JCF_DEV_HOME 에 설치를 원하는 폴더를 지정한다.(기본은 C:폴더 밑이다)



```

@echo off
rem * 시스템변수가 아닌 ***(로그인사용자)에 대한 사용자 변수에 추가됨
set JCF_DEV_HOME=C:\jcf-dev
echo JCF_DEV_HOME를 환경변수에 추가함
reg add HKCU\Environment /v JCF_DEV_HOME /d %JCF_DEV_HOME% /f
echo Display Environment Variable Setting
set
goto exit
:exit
exit
  
```

3. 명령창(시작>실행>cmd 입력)을 열어 setup.bat파일을 실행시킨다.



```

C:\> 명령 프롬프트

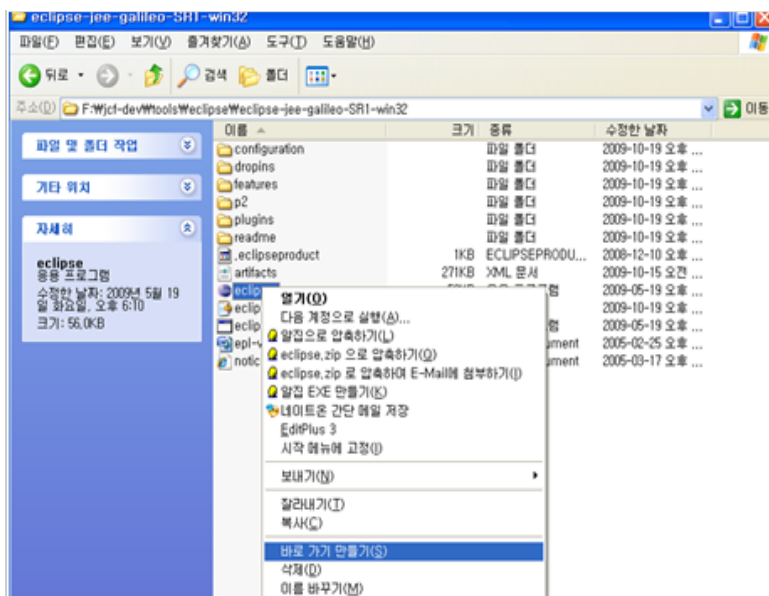
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Wmina>cd ../..

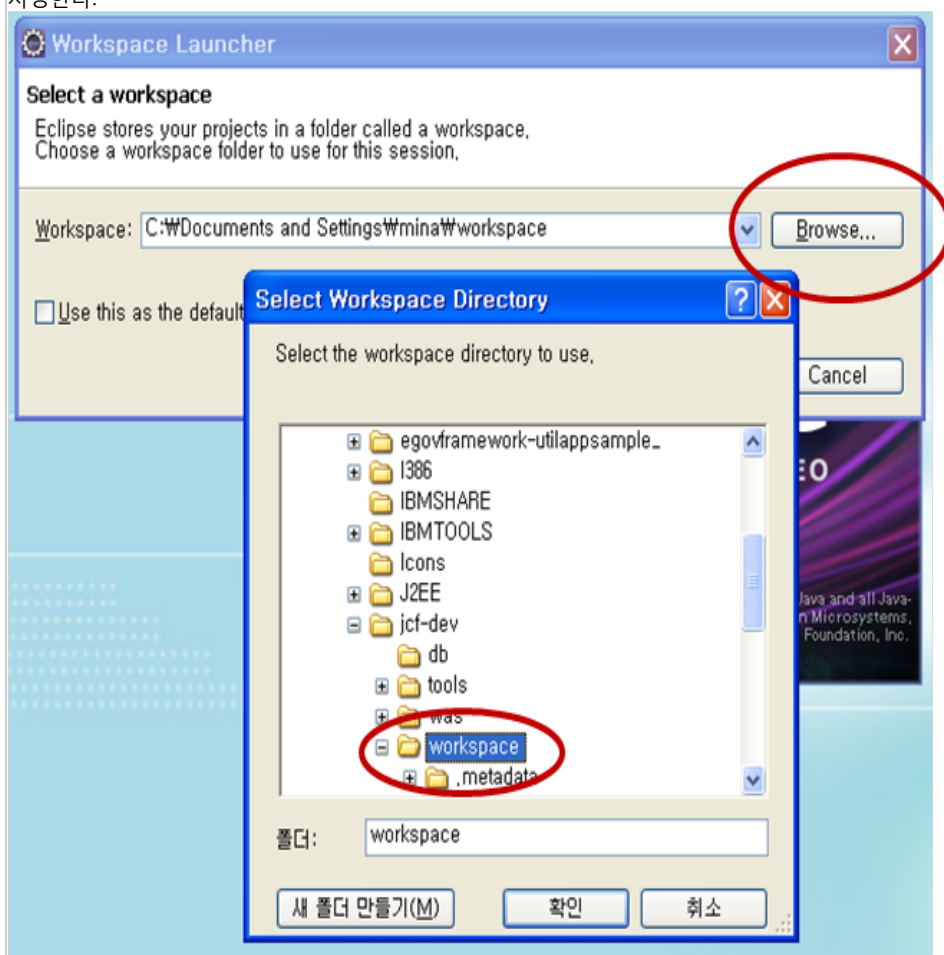
C:\>cd jcf-dev

C:\jcf-dev>setup.bat
  
```

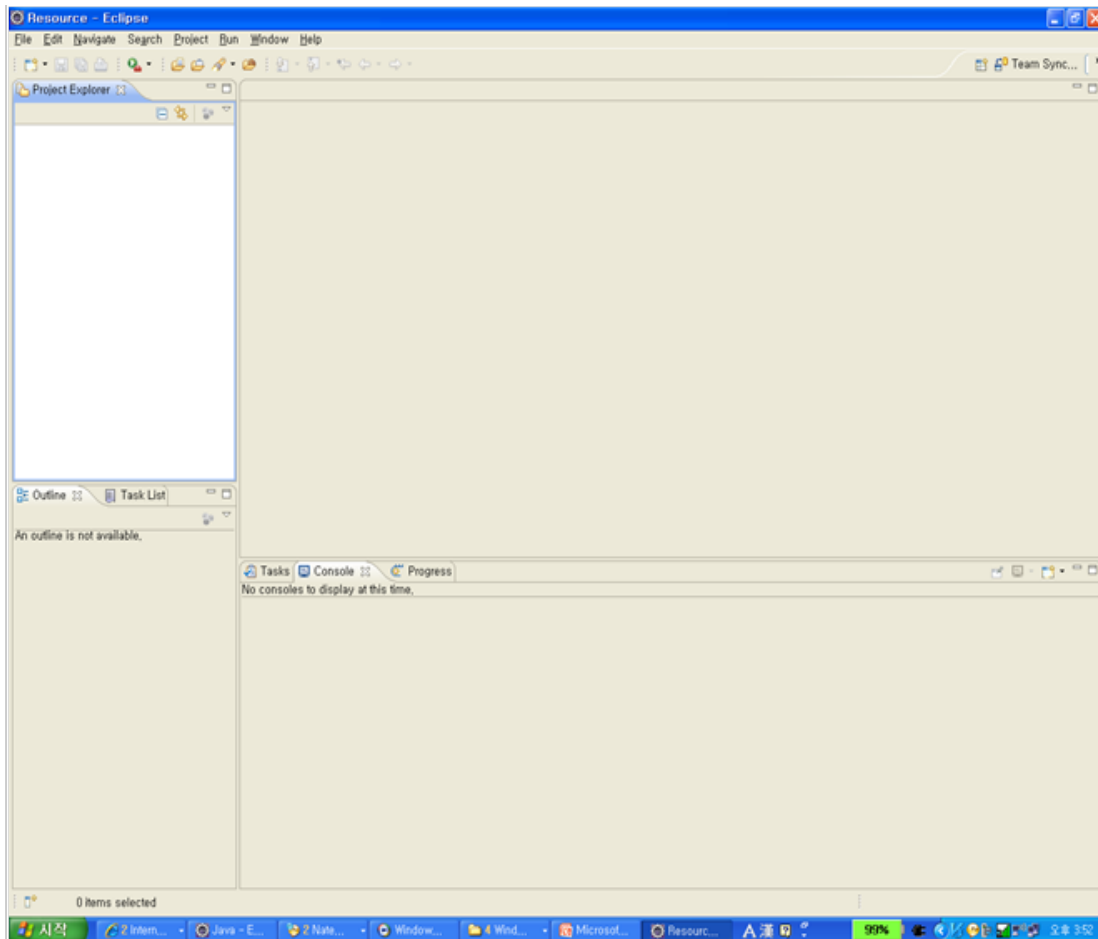
4. jcf-dev\tools\weclipse\weclipse-jee-galileo-SR1-win32 폴더 안에 있는 eclipse.exe파일에 대해 바로가기기를 만든 다음 바탕화면으로 이동시킨다.



5. 5에서 바로가기로 만든 eclipse바로가기를 실행시킨 후 밑에와 같이 Workspace 를 jcf-dev 폴더 바로 밑에있는 workspace 로 지정한다.



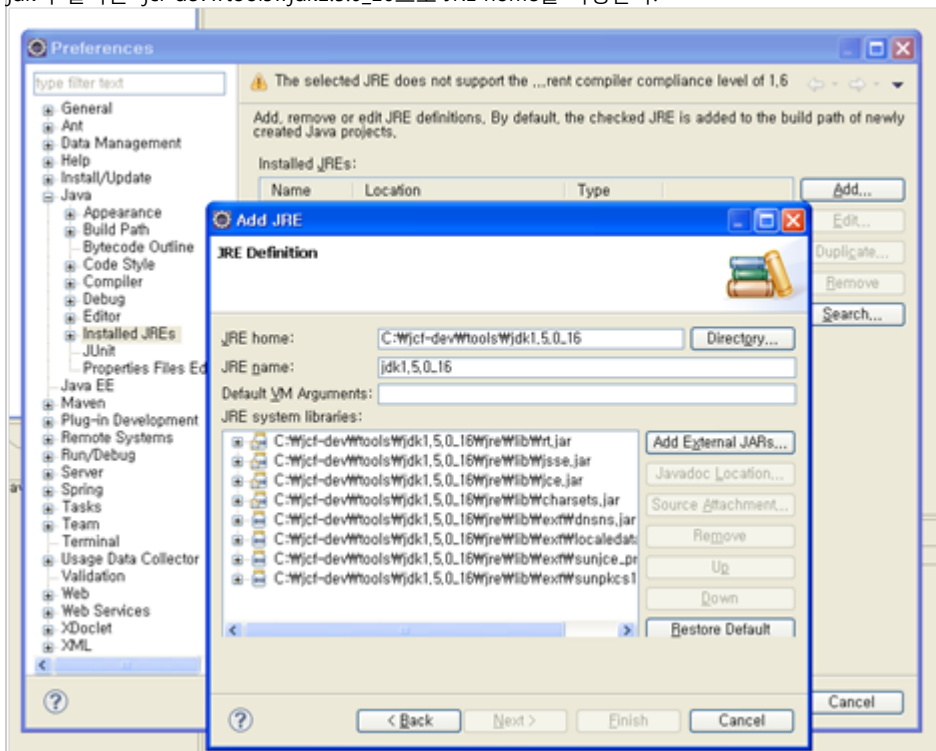
6. 밑에와 같은 화면에 나왔다면 성공한 것이다.



환경설정하기

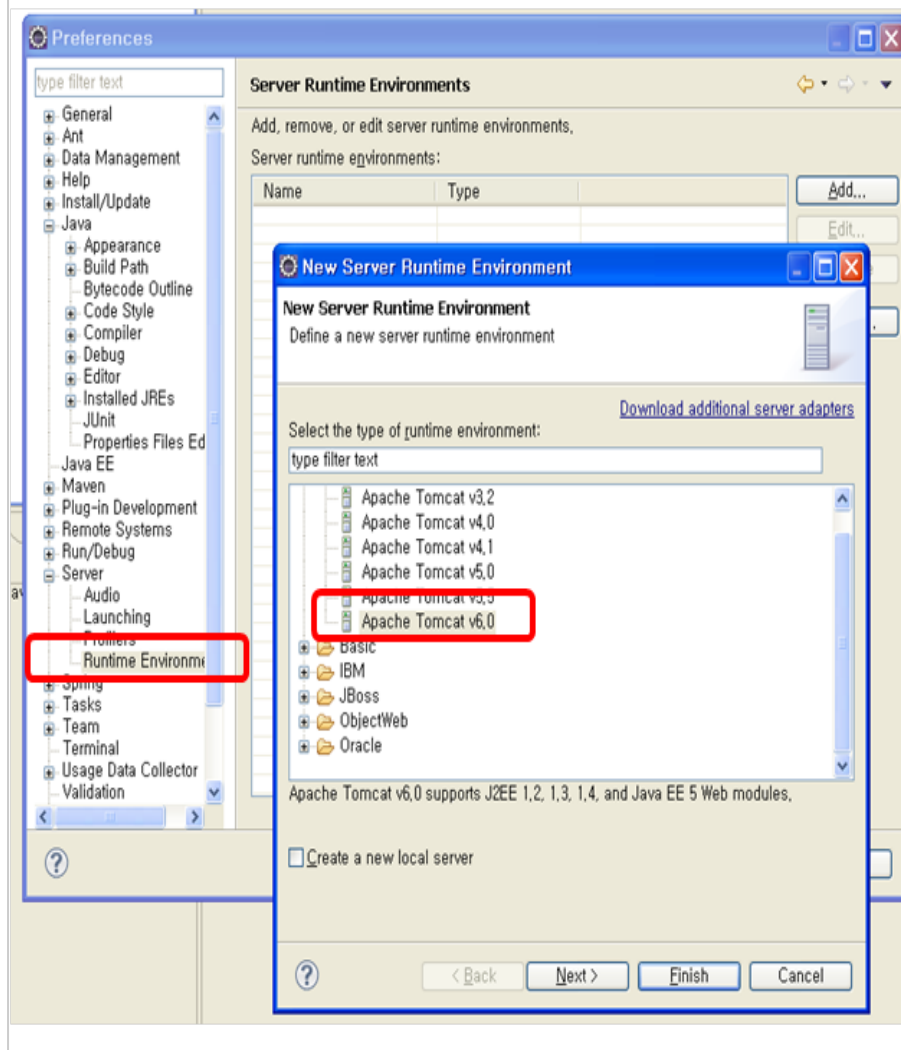
1. JAVA 셋팅하기

이클립스 메뉴 상단에 Window>Preferences 에서 Java>installed JREs에 JREs가 설정되어있지않다면, Add버튼을 눌러 Standard VM 선택 jdk가 설치된 jcf-devWoolsWjdk1.5.0_16으로 JRE home을 지정한다.



2. Server 셋팅하기

이클립스 메뉴 상단에 Window>Preferences 에서 Server>Runtime Environment 에 서버가 등록되어있지않다면 Add 후 Apache의 Apache Tomcat v6.0을 선택 후 next를 클릭 Tomcat installation direction 을 jcf-dev\was\apache-tomcat-6.0.20 으로 지정한다.



개인개발환경구성

- 폴더구조

개인 개발환경을 설치 시 폴더 구조는 밑에와 같다.

jcf-dev	db	hsqldb (테스트용 db)	
	docs (가이드문서)		
	tools	eclipse (IDE)	eclipse-jee-galileo-SR1-win32 (버전별 이클립스)
			template (이클립스 템플릿)
		jdk1.5.0_20 (버전별 JDK)	
		maven	setting.xml
			respository
		visualvm (모니터링툴)	
	was	apache-tomcat-6.0.20	
	workspace		
	setup.bat		

Step 2. 프로젝트 생성 및 실행

소제목

Step 3. 구현하기

Model 작성

- 모델 클래스는 첫글자가 대문자로 시작하도록 한다.
- DB 컬럼과 맵핑할 변수를 private 으로 선언한 후
- 이클립스 상위 메뉴의 Source>Generate Getters and Setters 를 통해 get/set메소드를 만든다.

```
01.package business.demo.user;
02.public class User {
03.    private String userId;
04.    private String userName;
05.    private String password;
06.    private String deptName;
07.    private String email;
08.    private String regDate;
09.    private String logDate;
10.    public String getUserId() {
11.        return userId;
12.    }
13.    public void setUserId(String userId) {
14.        this.userId = userId;
15.    }
16.    public String getUserName() {
17.        return userName;
18.    }
19.    public void setUserName(String userName) {
20.        this.userName = userName;
21.    }
22.    public String getPassword() {
23.        return password;
24.    }
25.    public void setPassword(String password) {
26.        this.password = password;
27.    }
28.    public String getDeptName() {
29.        return deptName;
30.    }
31.    public void setDeptName(String deptName) {
32.        this.deptName = deptName;
33.    }
34.    public String getEmail() {
35.        return email;
36.    }
37.    public void setEmail(String email) {
38.        this.email = email;
39.    }
40.    public String getRegDate() {
41.        return regDate;
42.    }
43.    public void setRegDate(String regDate) {
44.        this.regDate = regDate;
45.    }
46.    public String getLogDate() {
47.        return logDate;
48.    }
49.    public void setLogDate(String logDate) {
50.        this.logDate = logDate;
51.    }
52. }
53.
```

SQL 문 작성(Sql Map XML 파일)

sqlmap 폴더에 business 폴더와 같은 폴더 구조로 sqlmap 파일을 만든다. sqlmap 파일의 이름은 sqlmap - 소문자로 시작되도록 한다. 예) sqlmap-userInfo

sqlMap namespace 와 resultMap 선언


```

01.<?xml version="1.0" encoding="UTF-8" standalone="no"?>
02.<!DOCTYPE sqlMap PUBLIC "-//ibatis.apache.org/DTD SQL Map 2.0//EN" "
http://ibatis.apache.org/dtd/sql-map-2.dtd">
03.<sqlMap namespace="userInfo">
04. <typeAlias alias="user" type="business.demo.user.User"/>
05. <resultMap id="userMap" class="user" >
06.     <result property="userId" column="USER_ID"/>
07.     <result property="password" column="PASSWORD"/>
08.     <result property="userName" column="NAME"/>
09.     <result property="deptName" column="DEPT_NAME"/>
10.     <result property="email" column="EMAIL"/>
11.     <result property="regDate" column="REG_DATE"/>
12.     <result property="logDate" column="LOG_DATE"/>
13. </resultMap>
14.
15.....
16.
17.</sqlMap>

```

sqlMap namespace

- 위에 선언한 sqlMap namespace은 Service에서 부르는 sqlmap 파일의 이름이다
호출 예) dao.select("userInfo.getUserInfo", null)
- namespace명은 파일명에서 sqlmap- 을 뺀 명과 같게 지정한다.

typeAlias

- typeAlias 는 DB와 매핑을 할 모델(User)로
- alias 는 앞글자를 소문자로 바꾼 클래스의 이름, 예) User
- type 에는 사용할 모델의 클래스의 경로로 선언한다. 예) business.demo.user.User

resultMap

- resultMap 의 id 는 매핑하는 모델클래스의 앞글자를 소문자로 바꾼 이름 + Map 으로 지정하고 Class는 그 클래스에 대한 typeAlias 의 alias 명으로 지정한다.
- resultMap 는 DB 컬럼명과 그에 해당하는 모델의 프로퍼티를 선언한다.
- property 에는 모델의 프로퍼티, column 에는 DB의 컬럼명을 선언한다. . (예) DB 에서 USER_ID 라는 컬럼은 userId 라는 이름으로 가져온다.
-

Statement 작성

- 밑에는 입력 삭제, 수정, 조회 에 대한 Statement를 작성한 예이다,
- statement 란 Service 메소드에서 호출하는 이름이다.
예)서비스의 dao.select("userInfo.getUserInfo", null) 라면 userInfo 라는 namespace를 가진 sqlmap 파일의 getUserInfo 라는 statement 를 호출하는 것이다.
- parameterClass/parameterMap 는 DB 에 파라미터로 넘어가는 데이터의 유형, resultMap/resultClass 는 결과 값을 받을 데이터의 유형이다.

단 건 데이터 조회 시

```

1.<select id="selectUser" parameterClass="String" resultMap="userMap">
2.    select * from SDDUSER_INFO_00
3.    where
4.        USER_ID = #id#
5. </select>

```

- parameterClass 에는 Service에서 넘겨준 데이터의 유형을 선언한다. 여기서는 String
예) public SystemUser selectUser(String id) { return (SystemUser) dao.select("userInfo.getUserInfo", id); }
- , resultMap 으로는 위에서 선언한 모델(User)에 대한 resultMap 인 userMap 을 지정한다.

리스트 (여러 건)데이터 조회 시

```

01.<statement id="selectUserList" resultMap="userMap">
02. SELECT  USER_ID,
03.         PASSWORD,
04.         NAME,
05.         DEPT_NAME,
06.         EMAIL,
07.         REG_DATE,
08.         TO_CHAR(LOG_DATE, 'YYYY/MM/DD HH:MI:DD') as LOG_DATE
09. FROM    SDDUSER_INFO_00
10. ORDER BY USER_ID
11.</statement>

```

- 위 조회 statement 경우에는 파라미터로 들어가는 값이 없으므로 parameterClass 는 선언하지 않는다.
- , resultMap 으로는 위에서 선언한 모델(User)에 대한 resultMap 인 userMap 을 지정한다..

입력/삭제/수정 시

```

01.
02.<statement id="insertUserInfo" parameterClass="user">
03.INSERT INTO SDDUSER_INFO_00 (
04.    USER_ID,
05.    PASSWORD,
06.    NAME,
07.    DEPT_NAME,
08.    EMAIL,
09.    REG_DATE,
10.    LOG_DATE
11. )
12. VALUES
13. (
14.    #userId#,
15.    #password#,
16.    #userName#,
17.    #deptName#,
18.    #email#,
19.    SYSTIMESTAMP,
20.    SYSTIMESTAMP
21. )
22.</statement>
23.
24.<statement id="updateUserInfo" parameterClass="user">
25. UPDATE SDDUSER_INFO_00
26. SET
27.     PASSWORD = #password#,
28.     NAME = #userName#,
29.     DEPT_NAME = #deptName#,
30.     EMAIL = #email#,
31.     REG_DATE = SYSTIMESTAMP,
32.     LOG_DATE = SYSTIMESTAMP
33. WHERE USER_ID = #userId#
34.</statement>
35.
36.<statement id="deleteUserInfo" parameterClass="String" >
37. DELETE FROM
38.     SDDUSER_INFO_00
39. WHERE USER_ID = #userId#
40. </statement>

```

- 입력, 수정의 경우 받아올 결과 값이 없으므로 resultMap 는 선언하지않고, parameterClass 만 위에서 선언한 모델에 대한 typeAlias 인 user 으로 지정한다.
- 삭제일 경우에는 삭제할 userId 값만 받아오므로 parameterClass 는 String 으로 선언한다.
- ## 안에 바인딩 할 모델의 프러퍼티들을 넣는다.

Service 작성

클래스 작성규칙)

- Service 클래스 이름은 첫글자는 대문자 시작하고 Service 로 끝나도록 한다.
- Service 클래스 위에는 @Service 어노테이션을 선언한다.
- 클래스 선언 밑에는 공통 다오에 대한 인터페이스인 StatementMappingDataAccessOperations 를 dao 라는 이름으로 private 변수를 선언하고 @Autowired 어노테이션을 선언한다.

```

01.package business.demo.user;
02.import java.util.List;
03.@Service
04.public class SystemUserService {
05.
06.    @Autowired
07.    private StatementMappingDataAccessOperations dao;
08.
09.
10.....
11.

```

메소드 작성 규칙)

리스트 조회 시

```
1. public List<User> searchUser() {
2.     return dao.selectList("userInfo.selectUserInfo", null);
3. }
```

- 메소드 유형은 List<리스트안에 넣을 객체의 유형>으로 하고
- 공통다오의 selectList 메소드를 호출하여 첫번째 인자에는 호출할 sqlmap의 namespace, statement, 두번째 인자에는 넘길 파라미터(현재는 파라미터가 없으므로 null 값이 들어감)를 선언하여 리턴한다.

단 건 조회 시

```
1. public User selectUser(String id) {
2.     return (User) dao.select("userInfo.getUserInfo", id);
3. }
```

- 메소드 유형은 단 건으로 조회할 객체의 유형, 인자는 String 으로 지정하여 조회 시 넘어갈 조건 아이디를 Controller로 부터 받는다.
- dao의 select 메소드를 호출하여 첫번째 인자에는 호출할 sqlmap의 namespace. statement ("userInfo.getUserInfo") 두번째 인자에는 Controller 로 부터 받아온 값을 선언하여 조회한 값을 User 유형으로 Casting 하여 리턴한다.
- 넘길 파라미터의 유형은 sqlmap 의 statement에서 지정한 parameterClass 와 일치하여야 한다.
- 예) 현재 넘기는 파라미터 (id) 는 String 유형이므로 getUserInfo statement 의 parameterClass 는 "String" 이여야 한다.

삭제 시

```
1. public int deleteUser(String id) {
2.     return (Integer) dao.delete("userInfo.deleteUserInfo", id);
3. }
```

- 삭제 시 메소드 유형은 int, 인자는 String 으로 지정하여 삭제할 데이터를 (id) Controller 로 부터 받아와 dao의 delete 메소드에 넘겨 결과 여부를 Integer 로 변환하여 return 한다.

저장 시

```
1. public int saveUser(User user) {
2.     return (Integer) dao.insert("userInfo.insertUserInfo", user);
3. }
```

- 저장 시 메소드 유형은 int, 저장 할 데이터(user)를 Controller 로 부터 받아와 dao 의 insert 메소드에 넘겨 결과 여부를 Integer 로 변환하여 return 한다.

Controller 작성

클래스 작성 규칙)

```
1. package business.demo.user;
2.
3. @Controller
4. public class SystemUserController {
5.     @Autowired
6.     private SystemUserService systemUserService;
7.     ....
8. }
```

- 클래스 이름은 앞글자는 대문자 그리고 Controller 로 끝나도록 한다.
- 클래스 선언 위에는 @Controller 어노테이션을 선언한다.
- 사용하는 서비스는 private으로 선언하고 위에 @Autowired 어노테이션을 선언한다

메소드 작성 규칙)

- Controller의 메소드는 public 으로 선언하고, 리턴타입은 ModelAndView 로 선언한다.

*메소드 위에는 http 요청으로 호출할 수 있도록 RequestMapping 어노테이션과 호출될 주소를 지정한다.

예) <http://localhost:8080/demo/users/listUser.action> 의 경우 @RequestMapping("/users/listUser") 어노테이션을 선언한 메소드가 호출됨

리스트 조회 시

```
1. public ModelAndView list() {
2.     return new ModelAndView("user/list", "userList", systemUserService.searchUser()); }
```

- 메소드 유형은 ModelAndView
- 리턴값으로 new ModelAndView 안에 3개의 인자를 넘기는데
 **첫번째 인자는 조회 한 데이터를 넘길 페이지를 (여기서는 JCF.Sample.CRUD\$src\$main\$webapp\$WEB-INF\$jsp\$User\$list.jsp를 가르킴)
 - 두번째 인자는 첫번째 인자인 list.jsp 페이지에서 조회한 데이터를 받을 이름 예) list.jsp 에서 <c:forEach items="{userList}" var="user"> 로 받음
 - 세번째 인자에는 서비스의 조회 메소드를 호출하여 조회한 데이터를 지정한다.

단 건 조회 시

```
1.@RequestMapping("/users/view")
2. public ModelAndView view(@RequestParam("id")String id) {
3.     return new ModelAndView("user/view", "systemUser", userService.selectUser(id));
4. }
5.
```

*메소드 유형은 ModelAndView

*메소드 인수값으로 @RequestParam("id")String id 으로 지정하여 화면에서 id 로 넘긴 값을 받아온다.

- 리턴값으로 리스트 조회 시와 마찬가지로 new ModelAndView 안에 3개의 인자를 넘기는데
 - 첫번째 인자는 단 건 조회 한 데이터를 넘길 페이지를 (여기서는 JCF.Sample.CRUD\$src\$main\$webapp\$WEB-INF\$jsp\$User\$view.jsp를 가르킴)
 - 두번째 인자는 첫번째 인자인 view.jsp 페이지에서 조회한 데이터를 받을 이름
 - 세번째 인자에는 서비스의 조회 메소드에 id를 넘겨 조회한 데이터를 지정한다.

삭제 시

```
1.@RequestMapping("/users/delete")
2.public ModelAndView delete(HttpServletRequest request,
3.HttpServletResponse response) throws Exception {
4.
5.systemUserService.deleteUser(request.getParameter("id"));
6.return new ModelAndView(new RedirectView("../listUser.action"));
7.}
```

*메소드 유형은 ModelAndView

*메소드 인수값으로 HttpServletRequest request , HttpServletResponse response 을 지정한다.

- request 로 화면에서 id 로 넘긴 값을 받아온다.(단 건 조회 시와 기능은 같음)
- 리턴값으로 new ModelAndView 안에 데이터 삭제 후 호출할 메소드를 지정한다. (여기서는 같은 클래스의 listUser 메소드 호출함.)

입력 시

```
1.@RequestMapping("/users/insert")
2.public ModelAndView insert(User user) { userService.saveUser(user); return new ModelAndView(
new RedirectView("../listUser.action")); }
```

*메소드 유형은 ModelAndView

*메소드 인수값으로 User user 을 지정한다.(화면에서 입력하는 데이터를 User 라는 클래스에 담아서 옴)

- 서비스의 저장메소드에 user 데이터를 넘겨 데이터를 저장한다.
- 리턴값으로 데이터 삭제 때와 같이 new ModelAndView 안에 저장 후 호출할 메소드를 지정한다.(여기서는 같은 클래스의 listUser 메소드 호출함.)

입력 페이지 호출 시

```
1.@RequestMapping("/users/create")
2.public ModelAndView create() { return new ModelAndView("user/create"); }
```

- 위 메소드는 데이터의 저장이나 조회없이 단순히 입력한 페이지를 호출해 주는 메소드이다.
- 메소드 유형은 ModelAndView
- 리턴값으로 입력 페이지를 지정한다. (여기서는 JCF.Sample.CRUD\$src\$main\$webapp\$WEB-INF\$jsp\$User\$create.jsp를 가르킴)

화면(JSP) 작성

create.jsp

- create.jsp 는 사용자가 <http://localhost:8080/demo/create.action>을 호출했을 때 Controller 의 create 메소드 호출 후 리턴하는 페이지이다.
- form 태그의 action 에 데이터 입력 후 요청 할 URL(컨트롤러의 메소드)을 지정한다.
 예) action="insert.action" 인 경우 <http://localhost:8080/demo/user/insert.action> 로 호출되고 SystemUserController 의 insert 메소드를 호출한다.
- input 태그의 name 은 Controller 에서 받는 모델명의 프로퍼티로 선언한다.
 예)컨트롤러에서 받는 메소드가 public ModelAndView insert(User user) 의 경우 input 태그의 name은 User 라는 모델에 있는 프로퍼티여야 한다.

```

01.<%@page contentType="text/html; charset=UTF-8"%><%@
02.taglib
03.uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
04.
05.<html>
06.<head><title> </title>
07.<hr>
08.<SCRIPT language='JavaScript' type='text/javascript'>
09.function submit(){ document.systemUserForm.submit(); return; }
10.</SCRIPT>
11.
12.<link rel="stylesheet" href="../css/style.css" type="text/css"/>
13.</head>
14.
15.<form name="systemUserForm" action="insert.action" method="post" >
16.<table border=1 width="700" cellspacing=0>
17.<tr>
18.<td width="150"> ID</td>
19.<td colspan="3">
20.<input type="text" name="userId" />
21.</td>
22.</tr>
23.
24.<tr>
25.<td></td>
26.<td width="150">
27.<input type="text" name="userName" />
28.</td>
29.
30.</tr>
31.<tr>
32.<td>password</td>
33.<td colspan=3>
34.<input type="text" name="password" />
35.</td>
36.</tr>
37.<tr>
38.<td>deptName</td>
39.<td colspan=3>
40.<input type="text" name="deptName" />
41.</td>
42.</tr>
43.<tr>
44.<td>email</td>
45.<td colspan=3>
46.<input type="text" name="email" />
47.</td>
48.</tr>
49.
50.</table>
51.<input type="submit" value="" />
52.</form>
53.
54.<hr>

```

list.jsp

- list.jsp는 사용자가 <http://localhost:8080/demo/user/list.action>을 호출했을 때 SystemController 의 list 메소드 호출 후 리스트 결과값을 보여주는 페이지이다.
- C foreach 태그의 items 에 Controller 의 메소드의 리턴값인 ModelAndView 의 두번째 인자로 선언한 변수를 지정한다.
예) new ModelAndView("user/list", "userList", userService.searchUser());
- C foreach 태그의 var로 userList 안에 담긴 모델을 지정하고 리스트안에 있는 모델의 값(user.userName)들을 <td> 태그에 넣는다.

```

01.<%@page contentType="text/html; charset=UTF-8"%><%@
02.taglib
03.uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
04.
05.<html>
06.<head><title> </title>
07.<link rel="stylesheet" href="../../css/style.css" type="text/css"/>
08.</head>
09.hello, this is a "user/list" view.
10.<hr>
11.<table>
12.
13.<tr>
14.<th width="10">UserID</th>
15.<th width="90">UserName</th>
16.<th width="70">password</th>
17.<th width="90">deptName</th>
18.<th width="90">email</th>
19.<th width="90"></th>
20.<th width="90">logDate</th>
21.</tr>
22.<c:forEach items="${userList}" var="user">
23.<tr>
24.<td><a href="view.action?id=${user.userId}">${user.userId}</a></td>
25.<td>${user.userName}</td>
26.<td>${user.password}</td>
27.<td>${user.deptName}</td>
28.<td>${user.email}</td>
29.<td>${user.regDate}</td>
30.<td>${user.logDate}</td>
31.</tr>
32.</c:forEach>
33.</table>
34.<hr>
35.<a href = "../../users/create.action"></a><br/>
36.</html>

```

view.jsp

- view.jsp는 리스트 페이지에서 id를 클릭하면 SystemUserController의 view 메소드에 id를 넘겨 해당 User에 대한 정보를 조회하는 페이지이다.
- c:out 태그의 value로 Controller의 view 메소드 리턴값인 ModelAndView의 두번째 인자로 지정한 값과 그 안에 담김 프라퍼티를 지정한다.

예) return new ModelAndView("user/view", "user", userService.selectUser(id));

```

01.<%@ page contentType="text/html; charset=UTF-8" %>
02.<%@ page import="business.demo.user.User" %>
03.<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
04.<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
05.
06.<html>
07.<head><title> </title>
08.<SCRIPT language='JavaScript' type='text/javascript'>
09.function submit()
10.
11.{ document.userForm.submit(); return; }
12.</SCRIPT>
13.
14.
15.</head>
16.<link rel="stylesheet" href="../css/style.css" type="text/css"/>
17.hello, this is a "user/view" view.
18.<hr>
19.<body>
20.<form method="post" action="" >
21.<table border=1 width="700" cellspacing=0>
22.<tr>
23.<td width="150"> </td>
24.<td colspan="3">
25.<input type="text" name="userId" value='<c:out value="$
26.
27.{systemUser.userId}
28."/>' />
29.</td>
30.</tr>
31.
32.<tr>
33.<td> (/)</td>
34.<td width="150">
35.<input type="text" name="userName" value='<c:out value="$ {user.userName}" />' />
36.</td>
37.
38.</tr>
39.<tr>
40.<td>password</td>
41.<td colspan=3>
42.<input type="text" name="password" value='<c:out value="$ {user.password}" />' />
43.</td>
44.</tr>
45.<tr>
46.<td>deptName</td>
47.<td colspan=3>
48.<input type="text" name="deptName" value='<c:out value="$ {user.deptName}" />' />
49.</td>
50.</tr>
51.<tr>
52.<td>email</td>
53.<td colspan=3>
54.<input type="text" name="email" value='<c:out value="$ {user.email}" />' />
55.</td>
56.</tr>
57.<tr>
58.<td>regDate</td>
59.<td colspan=3>
60.<input type="text" name="regDate" readonly="true" value='<c:out value="$ {user.regDate}" />' />
61.</td>
62.</tr>
63.<tr>
64.<td>logDate</td>
65.<td colspan=3>
66.<input type="text" name="logDate" readonly="true" value='<c:out value="$ {systemUser.logDate}" />' />
67.</td>
68.</tr>
69.</table>
70.</form>
71.<br>
72.<a href="delete.action?id=${systemUser.userId}"></a>
73.</body>
74.</html>

```

Step 4. 테스트

첨부파일

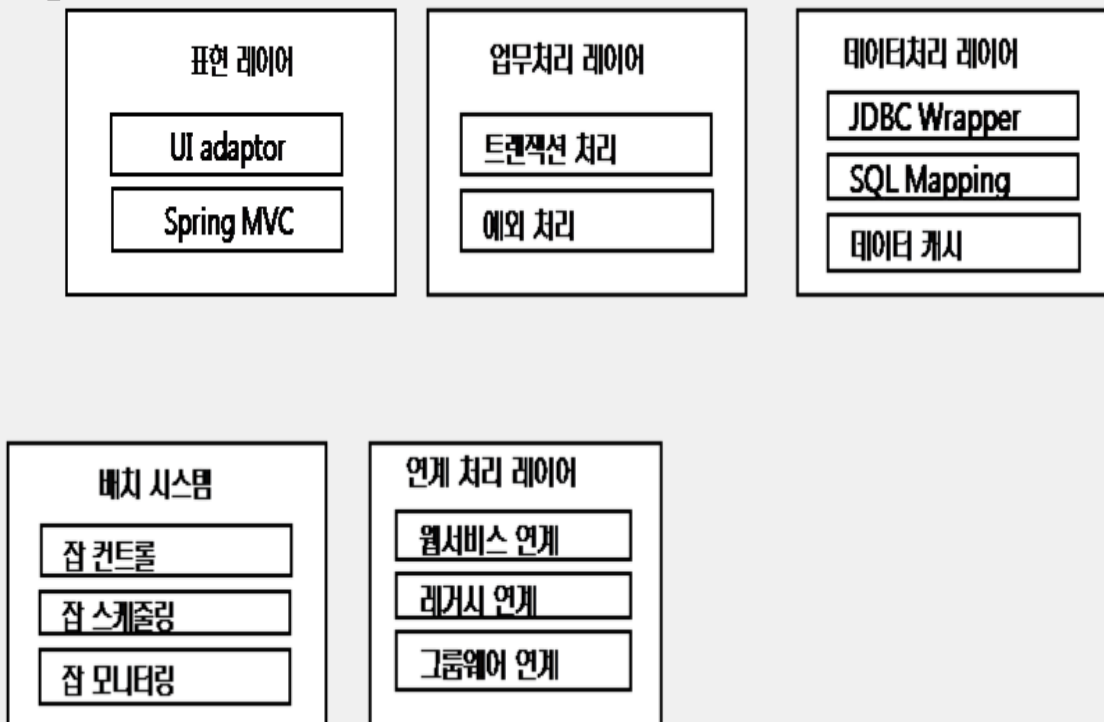
There are currently no attachments on this page.

Copyright © 2009 Daewoo Information Systems Co., Ltd.

01. 실행서비스

JCF 어플리케이션 프레임워크의 Core를 구성하는 각 Tier별 Component 모듈을 구성하고 비즈니스 어플리케이션 개발을 위한 개발패턴을 정의하는 부분이다.
실행서비스는 크게 화면(UI)으로 부터 받은 요청을 처리하는 표현 레이어, 업무로직을 처리하는 업무처리 레이어, 데이터를 DBMS에서 반영하는 데이터처리 레이어, 배치처리를 하는 배치 시스템, 웹서비스팀 타 시스템과 연계처리를 하는 연계 처리레이어로 나눌 수 있다.

실행 서비스



표현레이어

- 01.SpringMVC+Flex연계
 - 1.Controller 작성하기(RestFul+Flex연계)
 - 2.Controller 작성하기(AMF방식Flex연계)
- 02.SpringMVC+Miplatform 연계
 - 01.Controller 작성하기
 - 02.데이터 유형별 처리하기
- 03.SpringMVC+표준웹연계
 - 01. Controller작성하기(일반jsp기반)

업무처리레이어

- 1.서비스클래스작성법

데이터처리레이어

- 1.ibatis사용가이드
- 2. iBatis 다이내믹쿼리가이드
- 3.util사용하기
 - 1.CamelCaseMap사용하기
- 4.StoredProcedure처리하기
 - 1.SP 호출하기(oracle)
 - 2. SP로 리스트조회(oracle)
 - 3. SP로 리스트 조회 (MSSQL)
 - 4. SP로 리스트조회(sysbase)
- 7. sequence호출(oracle)

배치 시스템

- 1. 배치관리시스템 적용가이드
 - 1. 배치시스템 구성
- 2. 배치관리시스템 사용가이드
 - 1. 배치 등록 및 스케줄링 적용
- 3. 배치 개발 가이드
 - 1. 배치개발환경 설정
- 98. Shell or Dos 명령어 처리
- 99. temp

연계처리레이어

- 01.CXF 기반 웹서비스

01.표현레이어

- 표현 레이어는 클라이언트(UI)와 업무처리레이어와의 연결을 담당하는 레이어로 클라이언트(UI)에서 요청받은 사용자 요청을 검증하고, 클라이언트에서 받은 데이터를 업무처리레이어로 전달하거나 업무처리레이어에서 처리한 데이터를 클라이언트(UI)에 전달하는 역할을 한다.
- 현재 JCF의 표현레이어에서는 Spring MVC패턴 기반구조를 표준으로 채택하고 있으며, 마이플랫폼 연동과 같은 다양한 UI기술과의 연동 그리고 유효성검사 등의 기능을 지원하고 있다.

- 01.SpringMVC+Flex연계
 - 1.Controller 작성하기(RestFul+Flex연계)
 - 2.Controller 작성하기(AMF방식Flex연계)
- 02.SpringMVC+Miplatform 연계
 - 01.Controller 작성하기
 - 02.데이터 유형별 처리하기
- 03.SpringMVC+표준웹연계
 - 01. Controller작성하기(일반jsp기반)

01.SpringMVC+Flex연계

- 1.Controller 작성하기(RestFul+Flex연계)
- 2.Controller 작성하기(AMF방식Flex연계)

1.Controller 작성하기(RestFul+Flex연계)



목차

- 개요
- 설명
 - Controller 클래스 작성
 - 메소드 작성
 - 리스트조회메소드 작성
 - 한건 조회메소드 작성
 - 입력 메소드 작성
 - 수정 메소드 작성
 - 삭제 메소드 작성
- 참고자료
- 첨부파일

문서정보

- 제목: REST+FLEX+SPRING
- 최초작성자: 고재도SW
- 최초작성일: 2010년 9월 10일
- 이문서는 사용자에게 0번 보여졌습니다.

개요

본문서는 Flex를 UI로 사용하여 Spring MVC 기반의 Rest서비스 이용하는 방법을 기술한다.

설명

Controller 클래스 작성

- 클래스 이름은 화면의 이름과 같게한다.
- 맨 상단 줄에는 패키지 경로를 선언한다.(자동선언됨)
- 두번째 줄부터는 import 되는 클래스를 선언한다.(사용시 자동선언됨)
- import 가 끝난 다음에 클래스를 선언한다.(자동선언됨)
- 클래스를 선언한 뒤 줄에는 @Controller 어노테이션을 선언한다.
- 클래스를 선언한 뒤 줄에는 @RequestMapping 어노테이션을 선언한다.(URL 맵핑할 이름)
- 클래스 선언 한 밑에 줄부터는 Tab 키를 이용해서 한칸 띄고 본 클래스에서 사용하는 Service 들에 대해 선언한다.
- 서비스를 선언한 뒤 줄에는 @Autowired 어노테이션을 선언한다

MainController.java

```
1.@Controller
2.@RequestMapping("/main")
3.public class MainController {
4.
5.
6.         @Autowired
7.         private EmployeeService employeeService;
```

위의 소스를 보면 `http://<domain name>:<port>/<context path>/rest/main` 로 요청이 오게되면 MainController 라는 클래스의 자원을 main 라는 이름으로 사용할 수 있도록 하겠다는 의미이다. (@RequestMapping("/main") 선언)

메소드 작성

```
1.@RequestMapping(value =("/{id}", method = RequestMethod.GET)
2.public List listEmployees() {
3.    ...
4.}
```

- 모든 메소드에는 @RequestMapping 어노테이션을 선언한다.
- value 는 이 클래스의 자원을 접근하는 기본 URL (`http://<domain name>:<port>/<context path>/rest/main`) 다음에 붙여서 클래스의 메소드를 호출할 때 사용하는 이름이다.
- method 는 Client에서 받은 헤더정보의 메소드의 타입에 따라 메소드를 호출할 이름이다.
 - GET(조회)/POST(입력)/PUT(수정)/DELETE(삭제) 의 4가지 방식이 있음
 - 예)위 코드는 `http://localhost:8080/ems/employee/3` 이라는 요청을 헤더정보에 메소드 타입을 GET 으로 하여 보내면 호출되는 메소드이다.

리스트조회메소드 작성

Flex 스크립트 작성 (리스트조회)

main.xml(리스트조회)

```

01.[Bindable]
02. private var empList:Array;
03. private var uriBase: String = "http://localhost:8081/rest/main";
04.
05.     private function findEmpList() : void {
06.                                     var client:HttpClient = new HttpClient();
07.                                     var uri:URI = new URI(uriBase);
08.                                     var request:HttpRequest = new Get();
09.                                     request.addHeader("Accept", "application/json");
10.
11.                                     client.listener.onData = function(event:HttpDataEvent):void {
12.
13.                                         var userList: Object = JSON.decode( event.readUTFBytes() );
14.                                         users = userList.employeeList as Array ;
15.                                     };
16.
17.                                     client.request(uri, request);
18.     }
19.
20.     <mx:Button x="487" y="61" label="" click="findEmpList()"/>
21.     ]>

```

- http 통신을 위해 HttpClient 를 생성하고
- 조회시는 Get 객체를 선언하여 request 에 담는다.
- json형태로 조회한 데이터를 가져올 수 있도록 request 의 헤더 정보의 Accept 에 application/json 를 선언한다. (json형태로 받을 때)
- 조회한 데이터를 json 형식으로 받아 위에서 선언한 Array 형식의 empList 담을 수 있도록 client.listener.onData에 선언한다. (json 데이터에서 employeeList 객체를 추출하여 담음)
- client 의 request 에 url 과 선언한 request 를 보낸다.

Controller메소드 작성 (리스트조회)

리스트조회 메소드 (GET)

```

1.@RequestMapping(value = "", method = RequestMethod.GET)
2.public List listEmployees() {
3.
4.     List employees = employeeService.selectEmployeeList();
5.     return employees;
6.}

```

- RequestMapping 어노테이션을 선언하고,
- value 는 "" 으로 선언한다. (<http://localhost:8080/rest/employee> 로 호출함)
- 조회일 경우 method 는 RequestMethod.GET 으로 선언한다. (Flex 의 var request:HttpRequest = new Get(); 와 맵핑)
- 메소드 타입은 List 로 선언하고 서비스의 조회메소드를 호출하여 조회된 List 데이터를 리턴한다.

한건 조회메소드 작성

Flex 스크립트 작성 (한건조회)

main.xml(한건 조회)

```

01.      [Bindable]
02.          private var emp:Object = new Object();
03.
04.      private function findEmp() : void {
05.
06.          var client:HttpClient = new HttpClient();
07.          var uri:URI = new URI(uriBase+"/"+dg_empList.selectedItem.id);
08.          var request:HttpRequest = new Get();
09.          request.addHeader("Accept", "application/json");
10.          client.listener.onData = function(event:HttpDataEvent):void {
11.
12.              var jsonEmp: Object= JSON.decode( event.readUTFBytes() );
13.              emp = jsonEmp.employee as Object ;
14.
15.          };
16.          client.request(uri, request);
17.      }
18....
19.
20.      <mx:DataGrid x="10" y="35" id="dg_empList" dataProvider="{empList}"
21.          width="469" height="159" itemClick="findEmp()">
22.          .. </mx:DataGrid>
23.      <mx:Form x="10" y="202" width="469">
24.          <mx:FormItem label="User ID">
25.              <mx:TextInput id="user_id" text="{emp.id}" width="353"/>
26.          </mx:FormItem>
27.          <mx:FormItem label="User Name">
28.              <mx:TextInput id="user_name" text="{emp.name}" width="353"/>
29.          </mx:FormItem>
30.          <mx:FormItem label="User Email">
31.              <mx:TextInput id="user_email" text="{emp.email}" width="353"/>
32.          </mx:FormItem>

```

- http 통신을 위해 HttpClient 를 생성하고
- 리스트 조회 시와 같이 Get 객체를 선언하여 request 에 담고,
- json형태로 조회한 데이터를 가져올 수 있도록 request 의 헤더 정보의 Accept 에 application/json 를 선언한다. (json형태로 받을 때)
- 조회한 데이터를 json 형식으로 받아 위에서 선언한 emp 담을 수 있도록 client.listener.onData에 선언한다. (가져온 json 중에 employee 추출하여 담음)
- client 의 request 에 url 과 선언한 request 를 보낸다. (uriBase+"/"+dg_empList.selectedItem.id)

Controller메소드 작성 (단건 조회)

단건 조회 메소드 (GET)

```

1.      @RequestMapping(value =("/{id}", method = RequestMethod.GET)
2.      public Employee getEmployee(@PathVariable String id) {
3.
4.          Employee employee = employeeService.selectEmployeeById(Integer.parseInt(id));
5.          return employee;
6.      }

```

- RequestMapping 어노테이션을 선언하고,
- value 는

```
1.("/{id}",
```

으로 선언한다. (<http://localhost:8080/rest/employee/id> 로 호출함)

- 조회일 경우 method 는 RequestMethod.GET 으로 선언한다. (Flex 의 var request:HttpRequest = new Get(); 와 맵핑)
- 메소드 타입은 Employee 로 선언하고 서비스의 조회메소드를 호출하여 조회된 Employee 데이터를 리턴한다.

입력 메소드 작성

Flex 스크립트 작성 (입력)

main.xml(입력)

```

01.private function insertEmp() : void {
02.    var client:HttpClient = new HttpClient();
03.    var uri:URI = new URI(uriBase);
04.    var request:HttpRequest = new Post();
05.
06.    request.contentType = "application/json";
07.
08.    client.listener.onStatus =function(event:HttpStatusEvent):void {
09.        /**@ResponseStatus **/
10.        if(event.response.message.toString()=="OK"){
11.            findEmpList();
12.        };
13.    }
14.    client.listener.onError = function(event:ErrorEvent):void {
15.        var errorMessage:String = event.text;
16.        Alert.show(errorMessage);
17.    };
18.
19.    if( this.user_id.text.length > 0 ){
20.        emp.id = this.user_id.text;
21.        emp.name = this.user_name.text;
22.        emp.email = this.user_email.text;
23.        var json:String = JSON.encode( emp );
24.        var jsonData:ByteArray = new ByteArray();
25.        jsonData.writeUTFBytes(json);
26.        jsonData.position = 0;
27.        request.body= jsonData;
28.        client.request(uri, request);
29.    }
30.}

```

- http 통신을 위해 HttpClient 를 생성하고
- 입력일 경우 Post객체를 선언하여 request 에 담고,
- 입력, 수정, 삭제의 경우 request 의 contentType 을 "application/json" 라고 선언한다. (json형태로 넘길 때)
- 입력 후에 받은 ResponseStatus 값을 비교하여 "OK" 라면 재조회 메소드(findEmpList)를 호출하고 에러나면 에러메시지 출력하도록 선언한다.
- form의 user_id.text, user_name.text 의 데이터를 emp 에 담아 json encode 을 하여
- client 의 request 에 url 과 선언한 request 를 보낸다.

Controller메소드 작성 (입력)

입력 메소드 (POST)

```

1.    @RequestMapping(value = "", method = RequestMethod.POST)
2.@ResponseStatus(value = HttpStatus.OK)
3.public void saveEmployee( @RequestBody Employee employee) {
4.    employeeService.insertEmployee(employee);
5.
6.}

```

- RequestMapping 어노테이션을 선언하고,
- value 는 "" 으로 선언한다.
- 입력의 경우 method 는 RequestMethod.POST 으로 선언한다. (Flex 의 var request:HttpRequest = new Get(); 와 맵핑)
- 입력, 삭제, 수정의 경우 @ResponseStatus 어노테이션을 선언하여 value에 HttpStatus.OK 값을 선언한다. (Flex 의 event.response.message.toString()와 맵핑됨)
- 메소드 파라미터에 @RequestBody 어노테이션을 선언하여 화면에서 넘기는 데이터를 Employee 객체 유형으로 받아온다.
- 메소드 타입은 void 로 선언하고 서비스의 저장메소드를 호출하여 화면에서 받은 Employee 데이터를 넘긴다.

수정 메소드 작성

Flex 스크립트 작성 (수정)

main.xml(수정)

```

01.private function updateEmp() : void {
02.    if(dg_empList.selectedItem == null){
03.        return;
04.    }
05.
06.    var client:HttpClient = new HttpClient();
07.    var uri:URI = new URI(uriBase);
08.    var request:HttpRequest = new Put();
09.    request.contentType = "application/json";
10.
11.
12.        client.listener.onStatus =function(event:HttpStatusEvent):void {
13.            /**@ResponseStatus */
14.            if(event.response.message.toString()=="OK"){
15.                findEmpList();
16.            };
17.        }
18.    client.listener.onError = function(event:ErrorEvent):void {
19.        var errorMessage:String = event.text;
20.        Alert.show(errorMessage);
21.    };
22.    if( this.user_id.text.length > 0 ){
23.        emp.id = this.user_id.text;
24.        emp.name = this.user_name.text;
25.        emp.email = this.user_email.text;
26.        var json:String = JSON.encode( emp );
27.        var jsonData:ByteArray = new ByteArray();
28.        jsonData.writeUTFBytes(json);
29.        jsonData.position = 0;
30.        request.body= jsonData;
31.        client.request(uri, request);
32.    }
33.}

```

- http 통신을 위해 HttpClient 를 생성하고
- 입력, 수정, 삭제의 경우 request 의 contentType 을 "application/json" 라고 선언한다. (json형태로 넘길 때)
- 입력 후에 받은 ResponseStatus 값을 비교하여 "OK" 라면 재조회 메소드(findEmpList) 를 호출하고, 에러나면 에러메시지를 출력하도록 선언한다.
- form의 user_id.text, user_name.text 의 데이터를 emp 에 담아 json encode 을 하여
- client 의 request 에 접근할 자원에 대한 url 과 선언한 request 를 보낸다.

Controller 메소드 작성 (수정)

수정 메소드 (PUT)

```

1.    @RequestMapping(value = "", method = RequestMethod.PUT)
2.    @ResponseStatus(value = HttpStatus.OK)
3.    public void updateEmployee(@RequestBody Employee employee) {
4.        employeeService.updateEmployee(employee);
5.    }

```

- RequestMapping 어노테이션을 선언하고,
- value 는 "" 으로 선언한다.
- 수정의 경우 method 는 RequestMethod.PUT 으로 선언한다. (Flex 의 var request:HttpRequest = new Put(); 와 맵핑)
- 입력, 삭제, 수정의 경우 @ResponseStatus 어노테이션을 선언하여 value에 HttpStatus.OK 값을 선언한다. (Flex 의 event.response.message.toString()와 맵핑됨)
- 메소드 파라미터에 @RequestBody 어노테이션을 선언하여 화면에서 넘긴 데이터를 Employee 객체 유형으로 받아온다.
- 메소드 타입은 void 로 선언하고 서비스의 수정메소드를 호출하여 화면에서 받은 Employee 데이터를 넘긴다.

삭제 메소드 작성

Flex 스크립트 작성 (삭제)

main.xml(삭제)

```

01.private function deleteEmp() : void {
02.    if( dg_empList.selectedItem == null ){
03.        Alert.show(" ");
04.        return;
05.    }
06.    var client:HttpClient = new HttpClient();
07.    var listener:HttpDataListener = new HttpDataListener();
08.    client.listener.onStatus =function(event:HttpStatusEvent):void {
09.        /**@ResponseStatus */
10.        if(event.response.message.toString()=="OK"){
11.            findEmpList();
12.        };
13.    }
14.    client.listener.onError = function(event:ErrorEvent):void {
15.        Alert.show("onError");
16.        var errorMessage:String = event.text;
17.        Alert.show(errorMessage);
18.    };
19.
20.var uri:URI = new URI(uriBase + "/" + dg_empList.selectedItem.id);
21.    client.del(uri);
22.    }

```

- http 통신을 위해 HttpClient 를 생성하고.
- 삭제 후에 받은 ResponseStatus 값을 비교하여 "OK" 라면 재조회 메소드(findEmpList)를 호출하고, 에러나면 에러메시지를 출력하도록 선언한다.
- 삭제 할 데이터의 id인 dg_empList.selectedItem.id 을 uriBase 에 붙혀 client 의 del 메소드에 보낸다.

Controller 메소드 작성 (삭제)

삭제 메소드 (Delete)

```

1.@RequestMapping(value =("/{id}", method = RequestMethod.DELETE)
2.    @ResponseStatus(value = HttpStatus.OK)
3.    public void deleteEmployee(@PathVariable String id) {
4.        employeeService.deleteEmployee(Integer.parseInt(id));
5.    }

```

- RequestMapping 어노테이션을 선언하고,
- value 는

```
1."/{id}"
```

으로 선언한다. (<http://localhost:8080/rest/employee/id>(삭제할데이터의 id) 로 호출함)

- 삭제의 경우 method 는 RequestMethod.DELETE으로 선언한다. (Flex 의 client.del 와 맵핑)
- 입력, 삭제, 수정의 경우 @ResponseStatus 어노테이션을 선언하여 value에 HttpStatus.OK 값을 선언한다. (Flex 의 event.response.message.toString()와 맵핑됨)
- 메소드 파라미터에 @PathVariable 어노테이션을 선언하여 @RequestMapping value 에 선언한 id 값을 받아온다.
- 메소드 타입은 void 로 선언하고 서비스의 삭제 메소드를 호출하여 화면에서 받은 id 값을 넘긴다.

₩

참고자료



참고문헌

스프링 3 MVC with REST 관련

- <http://blog.springsource.com/2009/03/08/rest-in-spring-3-mvc/>
- <http://blog.smart-java.nl/blog/index.php/2010/01/16/spring-3-0-rest-services-with-spring-mvc/>
- <http://stsmidia.net/spring-finance-part-7-adding-support-for-json-and-xml-views/>

REST에 대한 Roy Fielding의 박사 논문

- <http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>




as3httpclientlib 라이브러리 사용법

- <http://code.google.com/p/as3httpclientlib/wiki/Examples>

REST를 이해하는데 좋은 자료


- <http://bradley-holt.com/2008/04/its-a-restful-world/>

첨부파일

이름	크기	생성자	Creation Date	댓글
 RESTful_Web_Services.pdf	2.60 MB	김민아	9월 29, 2010 18:13	
 01.jpg	18 kB	김민아	9월 29, 2010 18:13	
 RESTful_Java.ppt	1.43 MB	김민아	9월 29, 2010 18:13	

Copyright © 2009 Daewoo Information Systems Co., Ltd.

2.Controller 작성하기(AMF방식Flex연계)

 목차

- 개요
- 설명
 - Controller 클래스 작성
 - 메소드 작성
 - 리스트조회메소드 작성
 - 한건조회 메소드 작성
 - 저장 메소드 작성
- 참고자료
- 첨부파일

문서정보

- 제목: AMF방식통신방식사용하기
- 최초작성자: 김민아
- 최초작성일: 2010/09/12
- 이문서는 사용자에게 0 번 보여졌습니다.

개요

설명

Controller 클래스 작성

- 클래스 이름은 화면의 이름과 같게한다.
- 맨 상단 줄에는 패키지 경로를 선언한다.(자동선언됨)
- 두번째 줄부터는 import 되는 클래스를 선언한다.(사용시 자동선언됨)
- import 가 끝난 다음에 클래스를 선언한다.(자동선언됨)
- 클래스를 선언한 뒤 줄에는 @RemotingDestination 어노테이션을 선언한다.(RMI 통신)
- 클래스 선언 한 밑에 줄부터는 Tab 키를 이용해서 한칸 띄고 본 클래스에서 사용하는 Service 들에 대해 선언한다.
- 서비스를 선언한 뒤 줄에는 @Autowired 어노테이션을 선언한다

```

UserMgrController.java

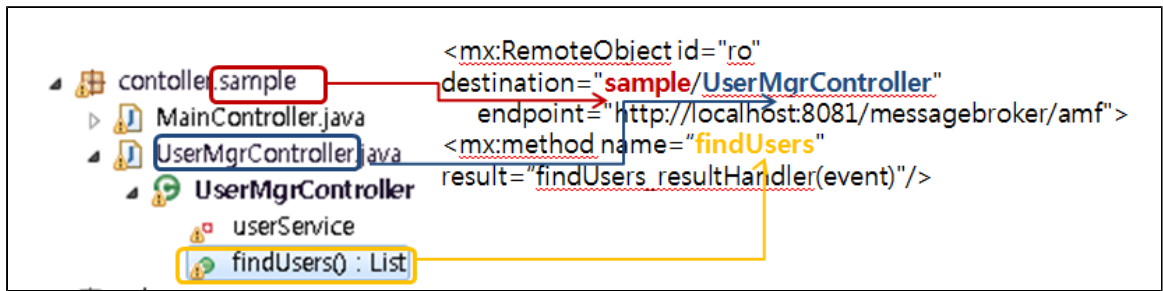
1.@RemotingDestination
2.public class UserMgrController {
3.
4.@Autowired
5. private EmployeeService employeeService;

```

메소드 작성

Flex 에서 호출시 매핑 경로

- Client에서 Controller를 호출 시 다음과 같이 맵핑된다.
 - destination= Controller 하위 패키지/해당 Controller 클래스 이름
 - method : 해당 Controller 의 메소드
 예)



리스트조회메소드 작성

Flex 작성(리스트조회)

```

usermgr.mxml(리스트조회)

01.[Bindable]
02.  private var empList:ArrayCollection=new ArrayCollection();
03.
04...
05.private function findAll():void
06.  {
07.    ro.findAllEmp();
08.  }
09. private function findAll_resultHandler(event:ResultEvent):void
10.  {    empList=ArrayCollection(event.result);
11.}
12.
13...
14.
15.
16.<mx:RemoteObject id="ro"
17.    destination="sample/UserMgrController"
18.    endpoint="http://localhost:8081/messagebroker/amf">
19.  ...
20.  <mx:method name="findAllEmp"
21.    result="findAllEmp_resultHandler(event)"/>
22.</mx:RemoteObject>
23.
24. <mx:Button label="" click="findAll()" x="0" y="155"/>

```

조회 버튼을 클릭하면 findAll 함수가 실행되어 id가 ro인 SCRemoteObject의 findAll 라는 메소드가 실행된다. result 에는 findAll호출 후 실행될 함수에 대한 선언으로 현재는 empList 에 Controller의 findAllEmp 메소드에서 조회해온 데이터를 담는다.

Controller 메소드 (리스트조회)

```

findAllEmp.java(리스트 조회)

1.public List  findAllEmp(){
2. List  employeeList = employeeService.selectEmployeeList();
3. return employeeList;
4.}

```

메소드 유형은 List 로 선언하여 서비스의 조회 메소드를 호출하여 데이터를 조회한 후 결과값을 화면에 리턴한다.

한건조회 메소드 작성

Flex 작성(한건조회)

usermgr.mxml(한건조회)

```

01.      [Bindable] private var emp:Object=new Object();
02.
03.private function find():void
04.    {
05.      ro.findEmp(this.dg_empList.selectedItem.id);
06.    }
07.
08...
09.
10.private function findEmp_resultHandler(event:ResultEvent):void
11.    {
12.      emp=event.result as Object;
13.    }
14.
15...
16.<mx:RemoteObject id="ro"
17.      destination="sample/UserMgrController"
18.      endpoint="[http://localhost:8081/messagebroker/amf|http://localhost
:8081/messagebroker/amf]">
19...
20.  <mx:method name="findEmp"
21.      result="findEmp_resultHandler(event)"/>
22.  .. </mx:RemoteObject>
23.
24.

```

- 리스트에서 한셀을 클릭하면 find 함수가 실행되어 id가 ro인 SCRemoteObject의 findEmp 라는 메소드가 실행되는데 클릭한 셀의 id를 파라미터로 보낸다.
- result 에는 findEmp호출 후 실행될 함수에 대한 선언으로 현재는 emp에 Controller 의 findEmp 메소드에서 조회해온 데이터를 담는다.

Controller의 find 메소드(한건 조회)

findEmp.java(한건조회)

```

1.public Employee findEmp(String id){
2.  Employee employee = employeeService.selectEmployeeById(Integer.parseInt(id));
3.  return employee;
4.}

```

- 메소드 유형은사용할 모델 (Emplyee)로 선언하고
- 화면에서 받은 arguments (id)를 서비스의 조회 메소드에 넘긴 후 받은 결과값을 화면에 리턴한다.

저장 메소드 작성

Flex 작성(저장)

usermgr.java(저장)

```

01.      private function save():void
02.    {
03.      emp.id=id2.text;
04.      emp.name=name2.text;
05.      emp.email=email2.text;
06.      ro.saveEmp(emp);
07.    }
08.....
09.private function saveEmp_resultHandler(event:ResultEvent):void
10.{
11.Alert.show(" ");
12.findAll();
13.}
14....
15.<mx:RemoteObject id="ro"
16.  destination="sample/UserMgrController"
17.  endpoint="[http://localhost:8081/messagebroker/amf|http://localhost:8081/messagebroker/amf]">
18.<mx:method name="saveEmp"
19.  result="saveEmp_resultHandler(event)"/>
20....
21.</mx:RemoteObject>

```

- 상세폼에서 수정 후 저장버튼을 클릭하면 save 가 함수가 실행되어 폼에 있던 데이터를 emp에 담은 후
- id 가 ro인 SCRemoteObject의 saveEmp라는 메소드에 emp를 보낸다.
- result 에는 saveEmp호출 후 실행될 함수에 대한 선언으로 현재는 저장되었다는 메시지와 재조회 메소드를 호출한다.

Controller의 save 메소드 작성

saveEmp(저장)

```

01. public void saveEmp(Employee employee){
02. Employee checkedEmployee = employeeService.selectEmployeeById((int) employee.getId());
03. if (checkedEmployee == null) {
04. employeeService.insertEmployee(employee);
05. }
06. else {
07. employeeService.updateEmployee(employee);
08. }
09. }

```

메소드 유형은 void 로 선언하고 받은 데이터의 id로 데이터의 존재여부를 조회하여 없으면 서비스의 입력메소드, 있으면 수정메소드를 호출한다..

참고자료



참고문헌

*

첨부파일

이름	크기	생성자	Creation Date	댓글
cm.png	43 kB	김민아	9월 29, 2010 18:20	

Copyright © 2009 Daewoo Information Systems Co., Ltd.

02.SpringMVC+Miplatform 연계

- 01.Controller 작성하기
- 02.데이터 유형별 처리하기

01.Controller 작성하기



목차

- 개요
- 구조 및 각 클래스의 역할
- 프로젝트 패키지 구조
 - 주요 클래스 및 명명 규칙
- Controller 클래스 작성
 - 클래스 작성지침
 - 메소드 작성지침
 - 조회메소드 작성(여러건 조회 시)
 - 저장메소드 작성지침(멀티데이터)
- 첨부파일

문서정보

- 제목: Controller 작성하기 (Miplatform+ SpringMVC)
- 최초작성자: 김민아
- 최초작성일: 2009/10/16
- 이문서는 사용자에게 **1**번 보여졌습니다.

개요

본 문서는 기본 어플리케이션 구조와 작성방법에 대해 설명한 문서이다.

구조및 각 클래스의 역할

1. Controller 작성

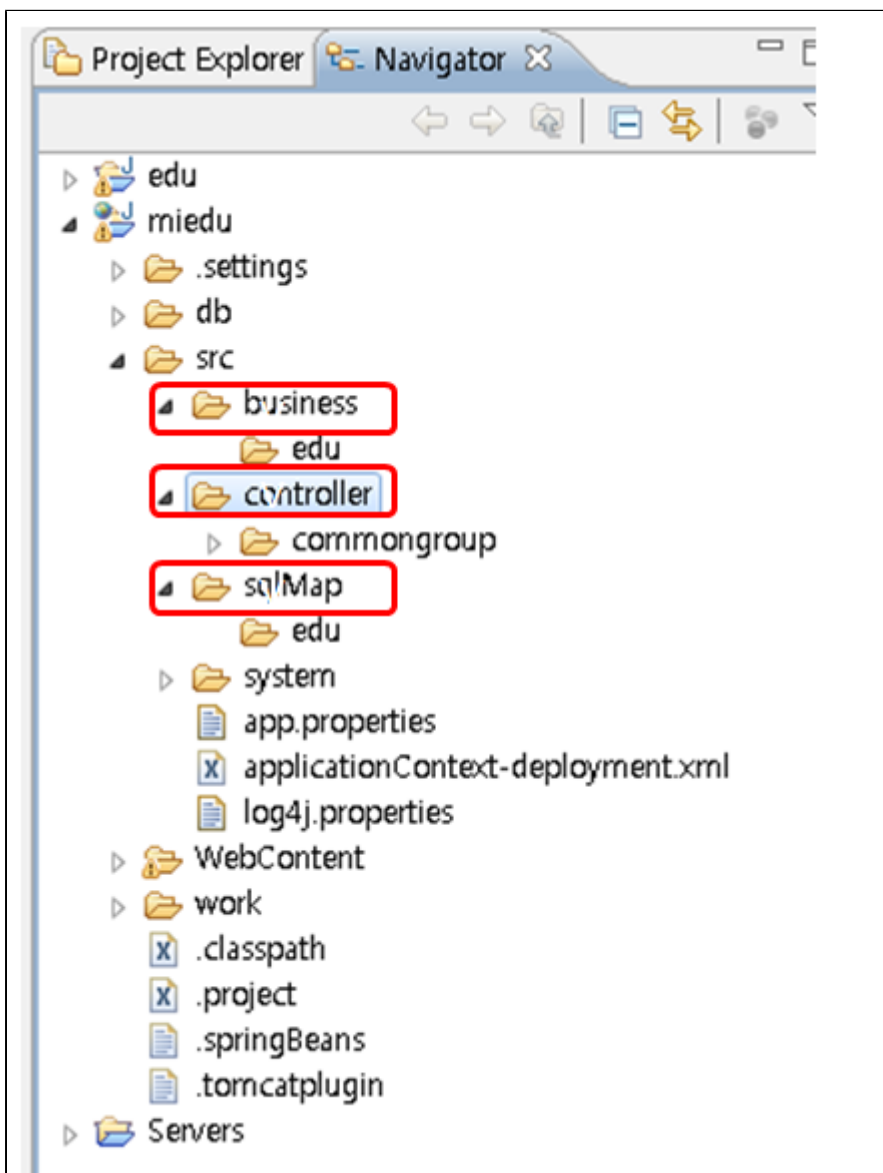
가. Controller는 화면과의 접점으로

나. HTTP 요청을 받아 해당 비즈니스 로직이 있는 Service 를 호출하고 Service 에서 처리 결과를 받아 화면으로 전달하는 역할을 한다.

프로젝트 패키지 구조

주요 클래스 및 명명 규칙

- controller 프리젠테이션 티어로 화면(마이플랫폼)과 같은 폴더 구조로 만든다.
- 모든 패키지는 소문자이며, 클래스는 첫글자만 대문자이다.



Controller	URL 자동 매핑을 통해서 화면의 요청 데이터를 받고 처리 결과를 화면에 전달한다. <u>명명규칙) 첫글자만 대문자이며 뒤에 Controller이란 단어가 들어간다.</u>	CrudController.java
------------	---	---------------------

Control클래스 작성

클래스 작성지침

```

01.1. package business.sample;
02.
03.@Controller
04.3. public class CrudController {
05.
06.     @Autowired
07.     4. private CrudService crudService;
08.
09.....
10. }

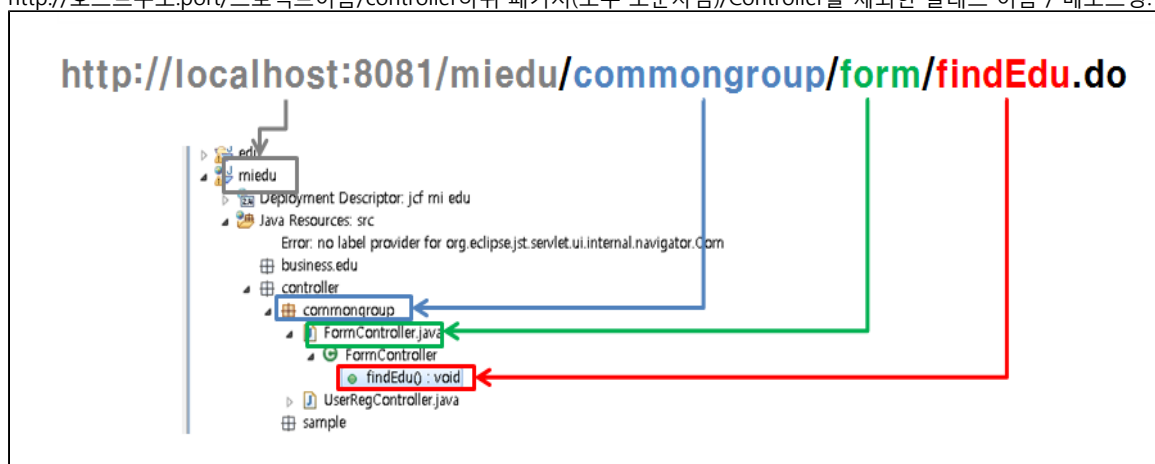
```

1. 맨 상단 줄에는 패키지 경로를 선언한다. (자동선언됨)
2. 두번째 줄부터는 import 되는 클래스를 선언한다. (사용시 자동선언됨)
3. import 가 끝난 다음에 클래스를 선언하고 위에 @Controller 어노테이션을 선언한다.
4. 클래스 선언 밑에 줄부터는 본 클래스에서 사용하는 Service 들에 대해 선언하고 위에 @Autowired 어노테이션을 선언한다.
- 5.

메소드 작성지침

화면에서 호출시 매핑 경로

- 화면에서 Controller 를 호출 시 다음과 같이 매핑된다.
- `http://호스트주소:port/프로젝트이름/controller하위 패키지(모두 소문자임)/Controller을 제외한 클래스 이름 / 메소드명.do`



* Controller의 모든 메소드 위에는 @RequestMapping라는 어노테이션을 선언한다.

조회메소드 작성(여러건 조회 시)

```

01./**      * UI GRID      *
02.** @param DatasetList
03.* @return DatasetList */
04.
05. @RequestMapping
06. public List<User> findUser(Map searchMap){
07.     List userList = crudService.findUser(searchMap);
08.     return userList;
09.}

```

1. 메소드 위에 @RequestMapping 어노테이션을 선언한다.
2. 여러건 조회 시 메소드명은 find+Entity명 + s으로 선언하며 메소드 유형(타입)은 List<리스트안에 담을 객체 유형>이며 파라미터 타입 searchMap이다.
3. List 유형의 Entity명 +List 를 선언하여 서비스의 조회 메소드(findUsers)로 조건값(searchMap)을 넘겨 데이터를 받아온다. -
4. 5에서 받은 데이터(userList)를 리턴한다.

저장메소드 작성지침(멀티데이터)




```

1.1. @RequestMapping
2.2. public void saveUsers(List<User> list) {
3.3.     crudService.saveUsers(list)
4.4. };

```

1. 메소드 위에 @RequestMapping 어노테이션을 선언한다.
2. 메소드명은 save+Entity명+ s으로 선언하며 메소드 유형(타입)은 void 이며 파라미터 타입은 List<리스트안에 담긴 객체 유형> 이다.
3. 서비스의 저장메소드(saveUsers)를 호출하여 데이터를 저장한다.

첨부파일

이름	크기	생성자	Creation Date	댓글
 패키지~2.png	70 kB	김민아	9월 30, 2010 08:58	
 fq.png	66 kB	김민아	9월 30, 2010 08:58	
 method.png	38 kB	김민아	9월 30, 2010 08:58	

Copyright © 2010 Daewoo Information Systems Co., Ltd.

02.데이터 유형별 처리하기



목차

- 개요
- 설명
 - 클래스 작성법
 - 메소드 작성법
- 참고자료
- 첨부파일

문서정보

- 제목:데이터 유형별 처리하기
- 최초작성자: 김민아
- 최초작성일: 02/06
- 이문서는 사용자에게 **0**번 보여졌습니다.

개요

본 문서는 JCF3.6기반에서 Miplatform 과의 연동 시 Controller 작성법을 설명한다.

설명

클래스 작성법

```

1.package business.jcfcom.user; ...
2.@Controller
3.public class UserController {
4.
5.@Autowired
6.     private UserService userService;
7.}

```

모든 클래스는 위와 같이 business 패키지 밑에 들어가며 클래스 위에는 @Controller 어노테이션을 설정한다.

사용할 서비스를 private으로 선언하고 @Autowired 어노테이션을 설정한다

메소드 작성법

URL 매핑을 하는 모든 Controller의 메소드에는 @RequestMapping 어노테이션을 설정한다.

마이플랫폼에서 데이터 넘기기

String 일 경우

Miplatform
<pre>1.srv_fn_AsyncCall(2."findRole", 3."Common::jcfcom/user/user/findUserRole.do?userName="+cb_userName.Value", 4."", 5."", 6."tranCallback"); 7.}</pre>
Controller.java
<pre>1.@RequestMapping 2.public findUserRole(String userName){ 3.List userRoleList = (List) userService.findUserRole(userName); 4... }</pre>

위와 같이 마이플랫폼에서 userName를 파라미터를 넘기면 컨트롤러 메소드 인자의 이름을 userName으로 넣는다..

Model 일 경우

Miplatform
<pre>01.transaction(02."saveUserPw", 03."Common::jcfcom/user/user/saveUserPw.do", 04."user=ds_user:u", 05."", 06."", 07."fn_LoginCallBack" 08. 09.)</pre>

위에서 4번째 줄인 "user=ds_user:u" 에서 user 가 Controller 에서 받을 인자값이다.
즉 컨트롤러의 user라는 변수에 ds_user 를 넘기겠다는 의미이다.

Controller.java
<pre>1.@RequestMapping 2.public saveUserPw(User user) throws Exception { 3. userService.saveUserPw(user); 4. }</pre>

Map 일 경우

Miplatform
<pre>1.srv_fn_AsyncCall(2."findUserRole", 3."Common::jcfcom/user/user/viewUserRoleByRoleId.do", 4."", 5."", 6."sysCd="+quote(systemCd) + " roleId="+quote(roleId), 7."tranCallback"</pre>
Controller.java
<pre>1.@RequestMapping 2.public List<UserRole> viewUserRoleByRoleId(Map params) { 3. String systemCd = (String) params.get("sysCd"); 4. String roleId = (String) params.get("roleId"); 5.}</pre>

메소드 인자로 Map 이름은 params 으로 설정한다.

List 일 경우

Miplatform
<pre> 1.srv_fn_AsyncCall(2. "rolesSave", 3. "Common::jcfcom/user/user/saveRole.do", 4. "roleList=ds_role:u", 5. "", 6. "", 7."tranCallback"); </pre>

위에서 4번째 줄인 roleList=ds_role 에서 roleList 가 컨트롤에서 인자로 받을 이름이다.

Controller.java
<pre> 1.@RequestMapping 2.public void saveRole(List<Role> roleList) { 3.userService.saveRole(roleList); 4.} </pre>

메소드 인자로 List<마이플랫폼과 매핑되는 모델유형 또는 맵> 받을 이름(List<Role> roleList)를 지정한다.

컨트롤러에서 마이플랫폼으로 데이터 넘기기

List 일 경우

Controller.java
<pre> 1.@RequestMapping 2. public List<RoleMenu> findRoleMenuByRoleId(Map params) { 3. return authService.findRoleMenuByRoleId(params); 4. } </pre>

메소드 유형으로 List<마이플랫폼이랑 매핑할 모델 유형> (List<RoleMenu>) 을 지정한다.

Miplatform
<pre> 1.srv_fn_AsyncCall(2."FINDROLE_MENU", //fn_TranCallBack Service ID 3."Common::jcfcom/authority/auth/findRoleMenuByRoleId.do", 4. "", // dataset 5."DS_ROLE_MENU=roleMenuList", // dataset 6. "", 7."tranCallback"); </pre>

위의 5 째 줄에서 roleMenuList 가 Controller에서 받는 데이터셋의 이름이다.

이 이름은 컨트롤러의 메소드 유형에서 List 가 붙은 이름이 된다. (앞글자 소문자로 변경)

메소드 유형 List<RoleMenu> ---> 마이플랫폼에서 받는 데이터셋 roleMenuList 이 된다.

model 일 경우

Controller.java
<pre> 1.@RequestMapping 2. public User findUser() { 3. return (User) userService.findUser(currentUser.getName()); 4. } </pre>

메소드 유형이 데이터셋이름이 된다.(User)

Miplatform
<pre> 01.srv_fn_AsyncCall(02. "findGlobal", 03. "Common::jcfcom/user/user/findUser.do", 04. "", 05. "ds_global=user", 06. "", 07. "tranCallBack", 08. true 09.); //() </pre>

Map 일 경우


```

Controller.java

01. @RequestMapping
02. public Map findAllCodes() {
03.
04. List systemCodes = new ArrayList();
05.     Map dataSetMap = new HashMap();
06.     dataSetMap.put("ds_systemcode", systemCodes);
07. ...
08.
09.     return dataSetMap;
10. }

```

위에서 Map 의 id 가 마이플랫폼에서 받는 데이터 셋명이된다.


```

Miplatform

1.srv_fn_AsyncCall(
2."find",
3."Common::jcfcom/code/code/findAllCodes.do",
4."",
5."DS_SYSTEMCD=ds_systemcode",
6."",
7."tranCallBack",
8.true);

```

참고자료

 참고문헌
*

첨부파일


이름	크기	생성자	Creation Date	댓글
 마이플랫폼과컨트롤러처리)0824.ppt	419 kB	김민아	9월 30, 2010 09:04	

Copyright © 2009 Daewoo Information Systems Co., Ltd.

03.SpringMVC+ 표준웹연계

- 01. Controller작성하기(일반jsp기반)

01. Controller작성하기(일반jsp기반)

 목차

- 개요
- Controller 작성
- 화면(JSP) 작성

문서정보

- 제목: SpringMVC+ 표준웹연계(일반jsp)
- 최초작성자: 고경철
- 최초작성일:
- 이문서는 사용자에게 0번 보여졌습니다.

개요

본 가이드는 JCF(버전 3.6)기반으로, 표준웹과 연동하는 가이드로 특정 UI제품을 사용하지않을 경우 기본 가이드이다.

Controller 작성

클래스 작성 규칙)

```

1.package business.demo.user;
2.
3.@Controller
4.public class SystemUserController {
5.    @Autowired
6.    private SystemUserService systemUserService;
7. ....
8.}

```

- 클래스 이름은 앞글자는 대문자 그리고 Controller 로 끝나도록 한다.
- 클래스 선언 위에는 @Controller 어노테이션을 선언한다.
- 사용하는 서비스는 private으로 선언하고 위에 @Autowired 어노테이션을 선언한다

메소드 작성 규칙)

- Controller의 메소드는 public 으로 선언하고, 리턴타입은 ModelAndView 로 선언한다.

*메소드 위에는 http 요청으로 호출할 수 있도록 RequestMapping 어노테이션과 호출될 주소를 지정한다.

예) <http://localhost:8080/demo/users/listUser.action> 의 경우 @RequestMapping("/users/listUser") 어노테이션을 선언한 메소드가 호출됨

리스트 조회 시

```

1.public ModelAndView list() {
2.    return new ModelAndView("user/list", "userList", systemUserService.searchUser()); }

```

- 메소드 유형은 ModelAndView
- 리턴값으로 new ModelAndView 안에 3개의 인자를 넘기는데
 - **첫번째 인자는 조회 한 데이터를 넘길 페이지를 (여기서는 JCF.Sample.CRUD\$src\$main\$webapp\$WEB-INF\$jsp\$User\$list.jsp를 가르킴)
 - 두번째 인자는 첫번째 인자인 list.jsp 페이지에서 조회한 데이터를 받을 이름 예)list.jsp 에서 <c:forEach items="\${userList}" var="user"> 로 받음
 - 세번째 인자에는 서비스의 조회 메소드를 호출하여 조회한 데이터를 지정한다.

단 건 조회 시

```

1.@RequestMapping("/users/view")
2. public ModelAndView view(@RequestParam("id")String id) {
3.    return new ModelAndView("user/view", "systemUser", systemUserService.selectUser(id));
4. }
5.

```

*메소드 유형은 ModelAndView

*메소드 인수값으로 @RequestParam("id")String id 으로 지정하여 화면에서 id 로 넘긴 값을 받아온다.

- 리턴값으로 리스트 조회 시와 마찬가지로 new ModelAndView 안에 3개의 인자를 넘기는데
 - 첫번째 인자는 단 건 조회 한 데이터를 넘길 페이지를 (여기서는 JCF.Sample.CRUD\$src\$main\$webapp\$WEB-INF\$jsp\$User\$view.jsp를 가르킴)
 - 두번째 인자는 첫번째 인자인 view.jsp 페이지에서 조회한 데이터를 받을 이름
 - 세번째 인자에는 서비스의 조회 메소드에 id를 넘겨 조회한 데이터를 지정한다.

삭제 시

```

1.@RequestMapping("/users/delete")
2.public ModelAndView delete(HttpServletRequest request,
3.HttpServletResponse response) throws Exception {
4.
5.systemUserService.deleteUser(request.getParameter("id"));
6.return new ModelAndView(new RedirectView("../listUser.action"));
7.}

```

*메소드 유형은 ModelAndView

*메소드 인수값으로 HttpServletRequest request , HttpServletResponse response 을 지정한다.

- request 로 화면에서 id 로 넘긴 값을 받아온다.(단 건 조회 시와 기능은 같음)
- 리턴값으로 new ModelAndView 안에 데이터 삭제 후 호출할 메소드를 지정한다. (여기서는 같은 클래스의 listUser 메소드 호출함.)

입력 시

```

1.@RequestMapping("/users/insert")
2.public ModelAndView insert(User user) { systemUserService.saveUser(user); return new ModelAndView(
new RedirectView("../listUser.action")); }

```

*메소드 유형은 ModelAndView

*메소드 인수값으로 User user 을 지정한다.(화면에서 입력하는 데이터를 User 라는 클래스에 담아서 옴)

- 서비스의 저장메소드에 user 데이터를 넘겨 데이터를 저장한다.
- 리턴값으로 데이터 삭제 때와 같이 new ModelAndView 안에 저장 후 호출할 메소드를 지정한다.(여기서는 같은 클래스의 listUser

메소드 호출함.)

입력 페이지 호출 시

```
1.@RequestMapping("/users/create")
2.public ModelAndView create() { return new ModelAndView("user/create"); }
```

- 위 메소드는 데이터의 저장이나 조회없이 단순히 입력한 페이지를 호출해 주는 메소드이다.
- 메소드 유형은 ModelAndView
- 리턴값으로 입력 페이지를 지정한다. (여기서는 JCF.Sample.CRUDWsrcWmainWwebappWWEB-INFWjspWuserWcreate.jsp를 가르킴)

화면(JSP) 작성

create.jsp

- create.jsp 는 사용자가 <http://localhost:8080/demo/create.action>을 호출했을 때 Controller 의 create 메소드 호출 후 리턴하는 페이지이다.
- form 태그의 action 에 데이터 입력 후 요청 할 URL(컨트롤러의 메소드)을 지정한다.
예) action="insert.action" 인 경우 <http://localhost:8080/demo/user/insert.action> 로 호출되고 SystemUserController 의 insert 메소드를 호출한다.
- input 태그의 name 은 Controller 에서 받는 모델명의 프로퍼티로 선언한다.
예)컨트롤러에서 받는 메소드가 public ModelAndView insert(User user) 의 경우 input 태그의 name은 User 라는 모델에 있는 프로퍼티여야 한다.

```
01.<%@page contentType="text/html; charset=UTF-8"%><%@
02.taglib
03.uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
04.
05.<html>
06.<head><title> </title>
07.<hr>
08.<SCRIPT language='JavaScript' type='text/javascript'>
09.function submit(){ document.systemUserForm.submit(); return; }
10.</SCRIPT>
11.
12.<link rel="stylesheet" href="../css/style.css" type="text/css"/>
13.</head>
14.
15.<form name="systemUserForm" action="insert.action" method="post" >
16.<table border=1 width="700" cellspacing=0>
17.<tr>
18.<td width="150"> ID</td>
19.<td colspan="3">
20.<input type="text" name="userId" />
21.</td>
22.</tr>
23.
24.<tr>
25.<td></td>
26.<td width="150">
27.<input type="text" name="userName" />
28.</td>
29.
30.</tr>
31.<tr>
32.<td>password</td>
33.<td colspan="3">
34.<input type="text" name="password" />
35.</td>
36.</tr>
37.<tr>
38.<td>deptName</td>
39.<td colspan="3">
40.<input type="text" name="deptName" />
41.</td>
42.</tr>
43.<tr>
44.<td>email</td>
45.<td colspan="3">
46.<input type="text" name="email" />
47.</td>
48.</tr>
49.
50.</table>
51.<input type="submit" value="" />
52.</form>
53.
54.<hr>
```

list.jsp

- list.jsp는 사용자가 <http://localhost:8080/demo/user/list.action>을 호출했을 때 SystemController 의 list 메소드 호출 후 리스트 결과값을 보여주는 페이지이다.
- C foreach 태그의 items 에 Controller 의 메소드의 리턴값인 ModelAndView 의 두번째 인자로 선언한 변수를 지정한다.
예) new ModelAndView("user/list", "userList", userService.searchUser());
- C foreach 태그의 var로 userList 안에 담긴 모델을 지정하고 리스트안에 있는 모델의 값(user.userName)들을 <td> 태그에 넣는다.

```

01.<%@page contentType="text/html; charset=UTF-8"%><%@
02.taglib
03.uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
04.
05.<html>
06.<head><title> </title>
07.<link rel="stylesheet" href="../css/style.css" type="text/css"/>
08.</head>
09.hello, this is a "user/list" view.
10.<hr>
11.<table>
12.
13.<tr>
14.<th width="10">UserID</th>
15.<th width="90">UserName</th>
16.<th width="70">password</th>
17.<th width="90">deptName</th>
18.<th width="90">email</th>
19.<th width="90"></th>
20.<th width="90">logDate</th>
21.</tr>
22.<c:forEach items="${userList}" var="user">
23.<tr>
24.<td><a href="view.action?id=${user.userId}">${user.userId}</a></td>
25.<td>${user.userName}</td>
26.<td>${user.password}</td>
27.<td>${user.deptName}</td>
28.<td>${user.email}</td>
29.<td>${user.regDate}</td>
30.<td>${user.logDate}</td>
31.</tr>
32.</c:forEach>
33.</table>
34.<hr>
35.<a href = "../users/create.action"></a><br/>
36.</html>

```

view.jsp

- view.jsp는 리스트 페이지에서id를 클릭하면 SystemUserController 의 view 메소드에 id를 넘겨 해당 User 에 대한 정보를 조회하는 페이지이다.
- c : out 태그의 value 로 Controller 의 view 메소드 리턴값인 ModelAndView 의 두번째 인자로 지정한 값과 그 안에 담긴 프라퍼티를 지정한다.

예) return new ModelAndView("user/view", "user", userService.selectUser(id));

```

01.<%@ page contentType="text/html; charset=UTF-8" %>
02.<%@ page import="business.demo.user.User" %>
03.<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
04.<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
05.
06.<html>
07.<head><title> </title>
08.<SCRIPT language='JavaScript' type='text/javascript'>
09.function submit()
10.
11.{ document.userForm.submit(); return; }
12.</SCRIPT>
13.
14.
15.</head>
16.<link rel="stylesheet" href="../css/style.css" type="text/css"/>
17.hello, this is a "user/view" view.
18.<hr>
19.<body>
20.<form method="post" action="" >
21.<table border=1 width="700" cellspacing=0>
22.<tr>
23.<td width="150"> </td>
24.<td colspan="3">
25.<input type="text" name="userId" value='<c:out value="$
26.
27.{systemUser.userId}
28."/>' />
29.</td>
30.</tr>
31.
32.<tr>
33.<td> (/)</td>
34.<td width="150">
35.<input type="text" name="userName" value='<c:out value="$ {user.userName}" />' />
36.</td>
37.
38.</tr>
39.<tr>
40.<td>password</td>
41.<td colspan=3>
42.<input type="text" name="password" value='<c:out value="$ {user.password}" />' />
43.</td>
44.</tr>
45.<tr>
46.<td>deptName</td>
47.<td colspan=3>
48.<input type="text" name="deptName" value='<c:out value="$ {user.deptName}" />' />
49.</td>
50.</tr>
51.<tr>
52.<td>email</td>
53.<td colspan=3>
54.<input type="text" name="email" value='<c:out value="$ {user.email}" />' />
55.</td>
56.</tr>
57.<tr>
58.<td>regDate</td>
59.<td colspan=3>
60.<input type="text" name="regDate" readonly="true" value='<c:out value="$ {user.regDate}" />' />
61.</td>
62.</tr>
63.<tr>
64.<td>logDate</td>
65.<td colspan=3>
66.<input type="text" name="logDate" readonly="true" value='<c:out value="$ {systemUser.logDate}" />' />
67.</td>
68.</tr>
69.</table>
70.</form>
71.<br>
72.<a href="delete.action?id=${systemUser.userId}"></a>
73.</body>
74.</html>

```

02.업무처리레이어

- 업무처리레이어는 표현레이어에서 전달받은 요청이나 데이터에 대해 업무 규칙에 따라 로직을 처리하는 레이어로 로직 처리 후 데이터를 데이터처리레이어에 전달하거나 데이터처리 레이어로부터 데이터를 전달받아 다시 표현 레이어로 전달하는 역할을 한다.
- 업무처리레이어에서는 업무처리 외에도 트랜잭션처리와 예외처리를 담당한다.

- 1.서비스클래스작성법

1.서비스클래스작성법



목차

- 개요
- 설명
 - Service 클래스 작성
 - Service 메소드 작성
 - 전체 조회 메소드
 - 한건 조회 메소드
 - 입력, 삭제, 수정 메소드
- 참고자료
- 첨부파일

문서정보

- 제목: 서비스클래스 작성하기
- 최초작성자: 김민아
- 최초작성일: 2010/09/12
- 이문서는 사용자에게 **13**번 보여졌습니다.

개요

본 문서는 service 클래스 작성법에 대해 설명한다.

설명

Service 클래스 작성

1. 맨 상단 줄에는 패키지 경로를 선언한다. (자동선언됨)
2. 두번째 줄부터는 import되는 클래스를 선언한다. (자동선언됨)
3. Service 클래스가 선언된 위에 @Service 어노테이션을 선언한다.
4. CommonDao 를 dao로 선언하고 그 바로 밑줄 @Autowired 어노테이션을 선언한다.

UserServiceImpl.java

```
01.package business.sample;
02.
03.import java.util.List;
04.
05.import org.springframework.beans.factory.annotation.Autowired;
06.import org.springframework.stereotype.Service;
07.
08.import system.dao.CommonDao;
09.
10.
11.@Service
12.public class EmployeeService {
13.
14.    @Autowired
15.    private CommonDao dao;
16.
17...}
```

Service 메소드 작성

전체 조회 메소드

전체 조회 메소드

```

1. public List selectEmployeeList() {
2.
3.         return dao.selectList("employee.selectEmployeeList", null);
4.     }
5.
6. }
```

- 메소드 유형은 List 선언하고 파라미터는 선언하지 않는다.
- dao의 selectList 를 호출하여 실행 할 sqlmap 의 namespace (첫글자만 대문자). statement id 와 null 을 넘겨 조회해온 데이터를 return 한다.

한건 조회 메소드

```
public Employee selectEmployeeById(int id)
```

```
Unknown macro: { return (Employee) dao.select("employee.selectEmployee", id); }
```

- 메소드 유형은 model 타입(Employee) 선언하고 파라미터로 조회할 조건값(id)을 받아온다.
- dao의 select를 호출하여 실행 할 sqlmap 의 namespace (첫글자만 대문자). statement id 와 조건값을 넘겨 조회해온 데이터를 model 유형으로 변환하여 return 한다.

입력, 삭제, 수정 메소드

전체 조회 메소드

```

01.         public void insertEmployee(Employee e) {
02.             dao.insert("employee.insert", e);
03.         }
04.
05.
06. public void deleteEmployee(int id) {
07.         dao.delete("employee.delete", id);
08.     }
09.
10.         public void updateEmployee(Employee e) {
11.             dao.update("employee.update", e);
12.     }
```

- 메소드 유형은 void , 아규먼트로 입력, 삭제, 수정할 model(Employee) 을 선언한다.
- 입력일 경우에는 dao 의 insert, 삭제일 경우에는 delete, 수정일 경우에는 update 을 호출하여 첫번째 파라메타에는 실행 할 sqlmap 의 namespace (첫글자만 대문자). statement id 와 두번째 파라메타에는 조건값을 넘긴다.

참고자료



참고문헌

*

첨부파일

There are currently no attachments on this page.

Copyright © 2009 Daewoo Information Systems Co., Ltd.

03.데이터처리레이어

- 실제 데이터를 DB에 반영하는 티어로 업무처리레이어에도 받은 데이터를 DB에서 반영하거나, DB로부터 데이터를 조회해서 업무처리 레이어에 전달하는 역할을 하는 레이어이다.
- 현재 JCF 데이터처리레이어에서는 ibatis 오픈소스를 표준으로 채택하고 적용하였다.

- 1. ibatis사용가이드
- 2. iBatis 다이내믹쿼리가이드
- 3. util사용하기
 - 1. CamelCaseMap사용하기
- 4. StoredProcedure처리하기
 - 1. SP 호출하기(oracle)
 - 2. SP로 리스트조회(oracle)
 - 3. SP로 리스트 조회 (MSSQL)
 - 4. SP로 리스트조회(sysbase)

- 7. sequence호출(oracle)

1.ibatis사용가이드

✓ 목차

- 개요
- iBatis 실행 구조
- iBatis 실행 절차
- SQL-Mapping 파일작성 지침
 - 파일 작성
 - Statement 작성
- 참고자료
- 첨부파일

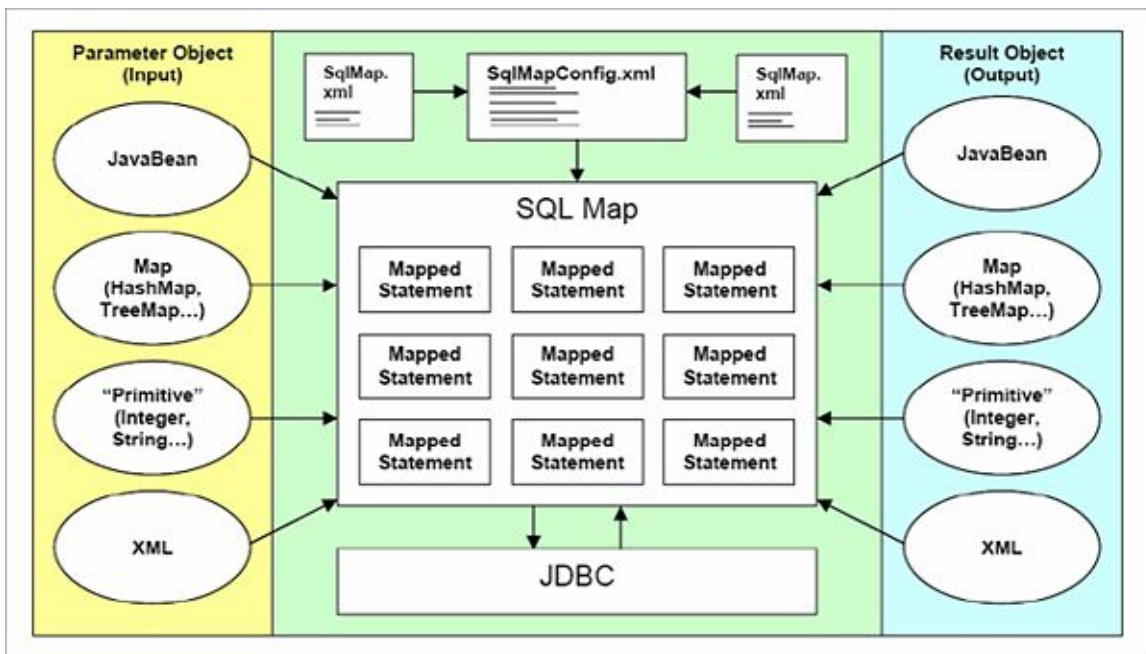
문서정보

- 제목: iBatis 기본원리
- 최초작성자: 김민아
- 최초작성일:2009/10/27
- 이문서는 사용자에게 **27**번 보여졌습니다.

개요

본 문서는 iBatis 기본 원리와 작성법에 대해서 다룬다.

iBatis 실행 구조



- iBatis는 JDBC 로 부터 쿼리를 호출하여 자원을 가져오거나 입력하는 부분을 XML 파일에 쿼리로 작성해서 실행할 수 있도록 만든 오픈소스이다.
- 개발자에게 자바 오브젝트를 PreparedStatement parameters와 ResultMaps로 쉽게 매핑을 할 수 있도록 지원한다..
- 이를 통하여 DB커넥션 생성, SQL문 전송, 결과처리, DB 커넥션 끊기, 트랜잭션 관리 등 database에 접근하기 위한 자바 코드의 양을 줄일 수 있다.

iBatis 실행 절차

1. 객체를 파라미터로 전달
 - JavaBeans, Map or primitive wrapper
2. 매핑되는 SQL문장을 수행
 - SQL Map 프레임워크는 PreparedStatement 인스턴스 생성
 - 객체로부터 제공되는 값들을 파라미터로 설정

3. SQL 문장을 수행하고 ResultSet으로부터 결과 객체를 생성.
 - resultMap 또는 resultClass 형태로 결과값을 받음

SQL-Mapping 파일작성 지침

파일 작성

예)

```

sqlmap-User.xml
1.2. <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2.<!DOCTYPE sqlMap PUBLIC "-//ibatis.apache.org/DTD SQL Map 2.0//EN" "
http://ibatis.apache.org/dtd/sql-map-2.dtd">
3.3. <sqlMap namespace="User">
4.
5.....
6.4. </sqlmap>

```

1. 파일이름은 sqlmap-Entity명(앞글자만 대문자)로 작성한다.
2. iBatis에 필요한 네임스페이스 넣는다.
3. sqlMap namespace는 sqlmap- 을 제외한 파일 이름으로 지정한다.
 - dao에서 호출할 때의 sqlMap namespace 명과 같아야 한다. 예) dao.queryForList("User.findUser", searchMap)
4. 파일 끝에는 sqlmap 태그를 닫아준다.

Statement 작성

조회 statement 예)

```

sqlmap-User.xml
1.1. <statement id="findUsers" 2. parameterClass="java.util.Map" 3. resultClass="orderedMap" >
2.4. select * from user_info
3.     where USER_ID=#USER_ID#
4.5. </statement>

```

1. 모든 쿼리는 statement안에 들어야하고 id를 지정한다.
 - dao에서 호출할 때의 statement id 명과 같아야 한다. 예) dao.queryForList("User.findUser", searchMap)
2. parameterClass 는 쿼리가 호출될 때의 input 값으로 dao에서 파라미터로 넘겨주는 값의 유형과 같아야 한다.

예) dao.queryForList("User.findUser", searchMap) 와 같이 dao에서 넘기는 값이 searchMap 즉 Map이라면 parameterClass도 java.util.Map으로 지정한다.
3. resultClass 는 쿼리가 실행되고 난 후 결과값이 대해서 돌아오는 때의 오브젝트 유형이다.

resultClass 는 개발자가 작성한 Model (예:User.java)또는 java.util.HashMap, String 등 결과 값에 따라 모든 자바 유형으로 지정 가능하다.

여기서는 orderedMap 는 sqlmap-config.xml 등 에서 typeAlias 로 지정한 alias로 실제로는 java.util.LinkedHashMap 이다.

```

1.<typeAlias alias="orderedMap" type="java.util.LinkedHashMap" />

```

4. statement 안에는 실행할 쿼리를 작성하고 input 으로 들어가는 값은 ## 안에 넣는다.
5. 쿼리 작성이 끝났으면 statement 태그를 닫는다.

참고자료



참고문헌

* ibatis 구위키가이드 http://wiki.dev.daewoobrenic.co.kr/mediawiki/index.php/IBatis_Framework

첨부파일

이름

크기

생성자

Creation Date

댓글

2. iBatis 다이내믹쿼리가이드



목차

- 개요
- 비교체크
 - 사용가능한 Dynamic Query
 - 사용 예)
- 단일조건 체크
 - 사용가능한 Dynamic Query
 - 사용 예
- 그 외
 - Iterate
- 첨부파일

문서정보

- 제목:iBatis Dynamic Query사용하기
- 최초작성자: 김민아
- 최초작성일: 2009/10/27
- 이문서는 사용자에게 0번 보여졌습니다.

개요

iBatis를 DBMS를 호출하는 부분이 XML파일 형태로 되어있기 때문에 JAVA 코딩 처럼 유연하기 사용하기 어렵다. 그래서 Dynamic Query를 사용하여 값을 비교하거나, null값 체크 등 조건값 등을 체크할 수 있다.

비교체크

. 비교 체크란 2개의 값을 비교해서 같은 값인지, 크거나 작은지 체크하는 것을 말한다.

사용가능한 Dynamic Query

- <isEqual> : 프로퍼티가 값 또는 다른 프로퍼티가 같은지 체크
- <isNotEqual> : 프로퍼티가 값 또는 다른 프로퍼티가 같지 않은지 체크
- <isGreaterThan> : 프로퍼티가 값 또는 다른 프로퍼티 보다 큰지 체크
- <isGreaterEqual> : 프로퍼티가 값 또는 다른 프로퍼티 보다 크거나 같은지 체크
- <isLessThan> : 프로퍼티가 값 또는 다른 프로퍼티 보다 작은지 체크
- <isLessEqual> : 프로퍼티가 값 또는 다른 프로퍼티 보다 작거나 같은지 체크

사용 예)

비교체크 Dynamic Query 예

```
1.1. <isLessEqual 2. prepend="AND" 3. property="age" 4. compareValue="18">
2. ADOLESCENT = 'TRUE'
3.5. </isLessEqual>
```

1. age가 18보다 작으면 ADOLESCENT 을 'TRUE' 로 지정하는 Dynamic Query 예이다.

2. prepend : the statement에 붙을 오버라이딩 가능한 SQL부분 (옵션)
3. property : 비교되는 property 값이다 (필수)
4. compareProperty : 비교되는 다른 property (필수 또는 compareValue으로 사용)
5. compareValue : 비교되는 값(필수 또는 compareProperty으로 사용)

단일조건 체크

단일 조건 체크는 프로퍼티가 널 또는 유효하지않은지 등 특수한 조건을 위해 프로퍼티의 상태를 체크하는 말한다.

사용가능한 Dynamic Query

- <isPropertyAvailable> : 프로퍼티가 유효한지 체크

- <isNotPropertyAvailable> : 프로퍼티가 유효하지 않은지 체크
- <isNull> : 프로퍼티가 null인지 체크
- <isNotNull> : 프로퍼티가 null이 아닌지 체크
- <isEmpty> : Collection, 문자열 또는 String.valueOf() 프로퍼티가 null이거나 empty(" or size() < 1)인지 체크
- <isNotEmpty> : Collection, 문자열 또는 String.valueOf() 프로퍼티가 null 이아니거나 empty(" or size() < 1)가 아닌지 체크
- <isParameterPresent> : 파라미터 객체가 존재(not null)하는지 체크
- <isNotParameterPresent> : 파라미터 객체가 존재하지(null) 않는지 체크

사용 예

비교체크 Dynamic Query 예

```
01.<statement id="findUser" parameterClass="java.util.Map" resultClass="orderedMap">
02. <![CDATA[
03. select * from USER_INFO
04. ]]>
05.1. <dynamic 2. prepend = "WHERE">
06.     <isNotEmpty prepend="AND" 3. property="USER_ID">
07.         USER_ID = #USER_ID#
08.     </isNotEmpty>
09.     <isNotEmpty prepend="AND" property="NAME">
10.         NAME like '%$NAME$%'
11.     </isNotEmpty>
12. </dynamic>
13.</statement>
```

1. USER_ID 또는 NAME으로 사용자를 조회하는 Dynamic Query 예이다.
2. prepend : 조건을 탄다면 statement에 붙을 SQL 구문 이다. (옵션)
3. property : 체크하기 위한 프로퍼티(필수)값이다.

그 외

Iterate

- 이 태그는 Collection을 반복하거나 리스트내 각각을 위해 몸체 부분을 반복하는 태그이다.

```
1.1. <iterate 2 prepend="AND" 3 open="(" 4. close=")" 5. conjunction="OR">
2.username = #[]#
3.</iterate>
```

1. dao에서 파라미터로 넘기는 값들을 반복해서 username 의 조건값으로 넣는 쿼리문이다
2. prepend : the statement에 붙을 오버라이딩 가능한 SQL부분 (옵션)
3. open : 반복의 전체를 열기 위한 문자열, 괄호를 위해 유용하다. (옵션)
4. close : 반복의 전체를 닫기 위한 문자열, 괄호를 위해 유용하다. (옵션)
5. conjunction : 각각의 반복 사이에 적용되기 위한 문자열, AND 그리고 OR을 위해 유용하다. (옵션)
6. dao에서 넘긴 리스트 유형의 파라미터가 []안에 들어간다. (위에서 parameterClass로 지정하지 않아도 된다.)



iterator요소를 사용할 때 리스트 프로퍼티의 끝에 중괄호[]를 포함하는 것은 중요하다.
중괄호는 문자열처럼 리스트를 간단하게 출력함으로부터 파서를 유지하기 위해 리스트처럼 객체를 구별한다

첨부파일

There are currently no attachments on this page.

Copyright © 2009 Daewoo Information Systems Co., Ltd.

3.util사용하기

* 데이터처리 레이어에서 사용 할 수 있는 유틸에 대해 설명한다.

- 1.CamelCaseMap사용하기

1.CamelCaseMap사용하기



목차

- 개요
- 적용방법
 - sqlmap-config.xml CamelCaseMap 등록하기
 - SQL-Map 에서 CamelCaseMap 으로 가져오기
- 참고자료
- 첨부파일

문서정보

- 제목: CamelCaseMap 사용하기
- 최초작성자: 김민아
- 최초작성일: 02/09
- 이문서는 사용자에게 4 번 보여졌습니다.

개요

본 문서는 DB에서 데이터를 조회 할 때 alias를 주지않고 컬럼명을 CamelCase으로 변환하여 가져오는 방법에 대해 소개한다.

예) USER_ID --> userId

적용방법

sqlmap-config.xml CamelCaseMap 등록하기

```
1.<?xml version="1.0" encoding="EUC-KR" standalone="no"?>
2.<!DOCTYPE sqlMapConfig PUBLIC "-//ibatis.apache.org//DTD SQL Map Config 2.0//EN" "
http://ibatis.apache.org/dtd/sql-map-config-2.dtd">
3.<sqlMapConfig>
4.<typeAlias alias="CamelCaseMap" type="jcf.dao.ibatis.util.ColumnNameCamelCaseMap" />
5....
6.</sqlMapConfig>
```

ColumnNameCamelCaseMap 경로가 너무 길으므로 sqlmap-config.xml 파일에 위와 같이 typeAlias를 등록한다.

SQL-Map 에서 CamelCaseMap 으로 가져오기

```
1.<statement id="selectRoleForMenu" parameterClass="string"
2.  resultClass="camelCaseMap">
3.SELECT sys_cd, role_id, role_name,
4.         description FROM ROLE
5.         WHERE role_id IN (SELECT role_id FROM ROLE_MENU
6.         WHERE menu_id = #id#)
7.  </statement>
```

위와 같이 조회 할 때 alias를 주지 않고 컬럼네임 그대로 조회 한 후에 resultClass 로 "camelCaseMap"을 선언한다.

참고자료



참고문헌

*

첨부파일

There are currently no attachments on this page.

Copyright © 2009 Daewoo Information Systems Co., Ltd.

4.StoredProcedure처리하기

- 1.SP 호출하기(oracle)
- 2. SP로 리스트조회(oracle)

- 3. SP로 리스트 조회 (MSSQL)
- 4. SP로 리스트조회(sysbase)

1.SP 호출하기(oracle)

목차

- 개요
- SP로 값을 INPUT만하는 경우
 - 프로시저 만들기
 - iBatis를 통한 프로시저 호출하기
- SP로 값을 OUTPUT 하는 경우
 - 프로시저 만들기
 - iBatis를 통한 프로시저 호출하기
- SP로 값을 INPUT/OUTPUT하는 경우
 - 프로시저 만들기
 - iBatis를 통한 프로시저 호출하기
- 첨부파일

문서정보

- 제목:SP 호출하기
- 최초작성자: 김민아
- 최초작성일: 2009/10/27
- 이문서는 사용자에게 0번 보여졌습니다.

개요

본 문서는 iBatis에서 stored procedure 를 호출하여 데이터를 입출력하는 방법에 대해 다룬다.

SP로 값을 INPUT만하는 경우

프로시저 만들기

procedurein procedure

```
1.create or replace PROCEDURE procedurein (p_param IN NUMBER)
2.IS BEGIN DBMS_OUTPUT.put_line (p_param); I
3.NSERT INTO tablein VALUES (p_param); commit; END;
4.
5.
```

#테이블 만들기

```
1.create table tablein(val number);
```

위 코드를 복사하여 SP와 테이블을 생성한다.

위코드는 파라미터를 받아 tablein 테이블에 입력하는 예제이다.

iBatis를 통한 프로시저 호출하기

inProcedure

```
1.1. <procedure id="inProcedure" 2, parameterClass="java.lang.Integer">
2.3. { call procedurein(#{val#}) }
3. </procedure>
```

1. 프로시저는 <procedure> 태그로 지정하여 statement와 같이 id를 준다..
2. input 데이터의 유형으로 parameterClass 로 지정한다.
3. 프로시저를 호출하며 input 데이터는 # # 안에 넣는다.
4. 호출이 끝났으면 <procedure> 태그를 닫는다.

테스트 케이스 실행

```

1. public void inProcedureTestDAO() {
2.     Integer val = 3;
3.1. dao.queryForObject("SpTest.inProcedure", val);
4.
5. }

```

1. dao의 queryForObject를 호출하여 실행한 프로시저(SpTest.inProcedure)와 parameterClass와 같은 유형의 값(val)을 파라미터로 보낸다.

SP로 값을 OUTPUT 하는 경우

프로시저 만들기

procedure_out PROCEDURE

```

1. create or replace PROCEDURE procedure_out(p_inval in integer, p_outval in out integer) IS
2. BEGIN
3.     p_outval := p_inval+3;
4. END;
5. /

```

위 코드를 복사하여 SP를 생성한다.

위 코드는 파라미터 1개를 받아 p_inval 에 3을 더해서 출력하는 예제이다.

iBatis를 통한 프로시저 호출하기

parameterMap 만들기

프로시저를 통해 OUT 데이터를 받아와야 할 경우 밑에와 같이 parameterMap과 procedure 를 선언한다.

parameterMap 과 procedure 호출

```

01.1. <parameterMap id="outProcedureMap" 2.class="java.util.Map">
02.3.     <parameter property="p_inval" 4.javaType="java.lang.Integer" 5.jdbcType="INTEGER" 6.mode=
    "IN" />
03.     <parameter property="p_outval" javaType="java.lang.Integer" jdbcType="INTEGER" mode="OUT" />
04.7. </parameterMap>
05. <procedure id="outProcedure" 8. parameterMap="outProcedureMap" >
06.9.     { call procedure_out (?, ?) }
07. </procedure>
08.
09.

```

1. parameterMap 은 procedure에 parameterMap 으로 들어가여 id를 지정한다.
2. class는 dao에서 파라미터로 넘기는 변수의 유형과 일치하며 보통 java.util.Map으로 지정한다.
3. parameter 태그로 IN /OUT 데이터를 지정하며
4. property는 dao에서 넘긴 Map의 key와 일치하여야 하며
5. javaType은 dao에서 넘긴 Map에 담긴 value의 유형이고
6. jdbcType은 SP에 받는 데이터의 유형이다.
7. mode는 이것이 IN 데이터인지 OUT인지 선언한 것이다.
8. parameterMap은 1에서 지정한 parameterMap 이다.
9. 변수는 ? 으로 설정하면 parameterMap 이 지정된 parameter 순서대로 바인딩된다.

테스트 케이스 실행

```

1. @Test
2. @Transactional
3. public void outProceduretestDAO() {
4.1. Map<String, Integer> m = new HashMap<String, Integer>;
5. m.put("p_inval", 7);
6.2. dao.queryForObject("SpTest.outProcedure", m);
7. System.out.println("p_outval : " + Integer.parseInt(m.get("p_outval").toString()));
8. }

```

1. Map을 생성하여 p_inval라는 key 로 Integer 7을 넣는다.
2. dao의 queryForObject를 호출하여 실행한 프로시저(SpTest.outProcedure)와 생성한 Map을 파라미터로 보낸다.

실행결과

```
1.DEBUG [main] sqlonly (Slf4jSpyLogDelegator.java:225)
2.2. { call procedure_out (7, '<OUT>') }
3.p_outval : 10
```

위와 같이 p_outval :10 이 콘솔에 떨어지면 성공한것이다. (p_inval+3)

SP로 값을 INPUT/OUTPUT하는 경우

프로시저 만들기

procedure_inout PROCEDURE

```
01.create or replace PROCEDURE
02.procedure_inout(p_inout1 in out integer, p_inout2 in out integer)
03. IS
04.tmpVar NUMBER;
05.BEGIN
06.    tmpVar:=p_inout1;
07.    p_inout1:=p_inout2;
08.    p_inout2:=tmpVar;
09.END;
10./
```

위 코드를 복사하여 SP를 생성한다.

위코드는 파라미터 2개를 받아 서로 바꾸어 출력하는 예제이다.

iBatis를 통한 프로시저 호출하기

parameterMap 만들기

프로시저를 통해 OUT 데이터를 받아와야 할 경우 밑에와 같이 parameterMap 과 procedure 를 선언한다.

parameterMap 과 procedure 호출

```
1.<parameterMap id="inoutProcedureMap" class="java.util.Map">
2.    <parameter property="p_inout1" javaType="java.lang.Integer" jdbcType="INTEGER" 1. mode="INOUT"
/>
3.    <parameter property="p_inout2" javaType="java.lang.Integer" jdbcType="INTEGER" mode="INOUT" />
4. </parameterMap>
5.
6. <procedure id="inoutProcedure" resultClass="java.util.Map" parameterMap="inoutProcedureMap">
7.    { call procedure_inout(?, ?) }
8. </procedure>
```

1. input, output 모두 할 경우 mode는 INOUT 으로 설정한다.

테스트 케이스 실행

```
01.@Test
02.@Transactional
03.public void inoutProcedureTestDAO() {
04.    Map<String, Integer> m = new HashMap<String, Integer>(2);
05.    m.put("p_inout1", 7);
06.    m.put("p_inout2", 5);
07.    dao.queryForObject("SpTest.inoutProcedure", m);
08.    System.out.println("p_inout1 : " + Integer.parseInt(m.get("p_inout1").toString()));
09.    System.out.println("p_inout2 : " + Integer.parseInt(m.get("p_inout2").toString()));
10.}
```

첨부파일

There are currently no attachments on this page.

Copyright © 2009 Daewoo Information Systems Co., Ltd.

2. SP로 리스트조회(oracle)



목차

- 개요
- 설명
 - Stored procedure 작성
 - ibatis xml 파일 작성
 - testCase작성
- 참고자료
- 첨부파일

문서정보

- 제목:ibatis SP로 리스트조회가이드
- 최초작성자: 김민아
- 최초작성일: 2010/02/25
- 이문서는 사용자에게 9 번 보여졌습니다.

개요

본 문서는 ibatis 에서 SP (stored procedure)사용 시 리스트 형태로 조회 가이드이다.

설명

Stored procedure 작성

- 테이블생성

```
01.CREATE TABLE "CODE" (  
02.id VARCHAR2(20) PRIMARY KEY not null,  
03.name VARCHAR2(20),  
04.  
05.  
06.)  
07.  
08.INSERT INTO  
09.                CODE( id, name)  
10.                VALUES( 'ALSDKZZ' 'TEST1' )  
11.  
12.INSERT INTO  
13.                CODE( id, name)  
14.                VALUES( 'CALEY' 'TEST2' )
```

CODE 테이블을 만들고 2건의 데이터를 넣는다

- SP 생성

```
01.create or replace PROCEDURE procedure_out  
02.(  
03.OUT_CUR OUT SYS_REFCURSOR  
04.) IS  
05.    BEGIN  
06.        OPEN OUT_CUR FOR  
07.SELECT * FROM CODE ;  
08.END;  
09.  
10.
```

CODE 테이블의 모든 데이터를 조회하는 프로시저를 생성한다.

ibatis xml 파일 작성

오라클일 경우


```

01.<?xml version="1.0" encoding="UTF-8" standalone="no"?>
02.<!DOCTYPE sqlMap PUBLIC "-//iBATIS.com//DTD SQL Map 2.0//EN" "
http://www.ibatis.com/dtd/sql-map-2.dtd">
03.
04.<sqlMap namespace="spTest">
05.
06.    <typeAlias alias="code" type="business.jcfcom.code.Code" />
07.
08.    <resultMap class="code" id="resultMap-code">
09.        <result property="id" column="id" />
10.        <result property="name" column="name" />
11.    </resultMap>
12.
13.
14.    <parameterMap id="param" class="java.util.HashMap">
15.        <parameter property="OUT_CUR" jdbcType="ORACLECURSOR" mode="OUT"
16.            javaType="java.sql.ResultSet" />
17.    </parameterMap>
18.
19.    <procedure id="procedure_out" parameterMap="param" resultMap="resultMap-code">
20.        { call procedure_out( ? ) }
21.</procedure>
22.
23.</sqlMap>
24.

```

parameterMap 에 OUT_CUR 라는 오라클 커서에 대한 parameter 를 지정하여 procedure 를 호출할 때 parameterMap 으로 넣고, resultMap 으로 맵핑할 클래스에 대한 resultMap 을 지정한다.

testCase작성

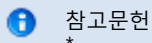
```

1.@Autowired
2.private StatementMappingDataAccessOperations dao;
3.
4.
5.    @Test
6.    public void testname() throws Exception {
7.        List<Code> list= dao.selectList("spTest.procedure_out", null);
8.        System.out.println(list.size());

```

StatementMappingDataAccessOperations (commonDao)를 private으로 선언하고 @Autowired 어노테이션을 넣은 다음 dao.selectList 하여 ibatis xml 에 있는 procedure를 호출하여 조회해온 데이터를 list에 담는다.

참고자료



첨부파일

There are currently no attachments on this page.

Copyright © 2009 Daewoo Information Systems Co., Ltd.

3. SP로 리스트 조회 (MSSQL)

목차

- 개요
- 리스트 조회
 - sqlmap 파일 예)
 - Service 호출 예)
- 참고자료
- 첨부파일

문서정보

- 제목:ibatis SP호출하기 (mssql 기반)
- 최초작성자: 김민아
- 최초작성일: 2010/05/20
- 이문서는 사용자에게 **12**번 보여졌습니다.

개요

DB가 mssql 일 때 ibatis sqlmap에서 Stored Procedure 호출에 대한 가이드이다.

리스트 조회

sqlmap 파일 예)

```
1.      <statement id="selectFic01TopList" parameterClass="map" resultClass="camelCaseMap" >
2.          dbo.usp_fic01_list #a_from_dt#, #a_to_dt#, #a_bsp#, #a_acot_fr#,
#a_acot_to#
3.</statement>
```

- 일반 sql호출과 같이 statement 로 선언한 후 id를 지정하고,
 - parameterClass 는 map
 - resultClass 는 camelCaseMap 으로 지정한다.

Service 호출 예)

```
01.public List  searchFic01TopList(Map model){
02.            /**
03.                Map model= new HashMap();
04.                model.put("a_from_dt", "200801");
05.                model.put("a_to_dt", "200812");
06.                model.put("a_bsp", "10");
07.                model.put("a_acot_fr", "1");
08.                model.put("a_acot_to", "9");
09.                */
10.                List list = dao.selectList("fic01.selectFic01TopList", model);
11.                return list;
12.}
```

- 메소드 유형은 List 아규먼트는 조회할 데이터를 Map 유형으로 받아온다.
- 받아온 데이터를 dao의 selectList 를 호출하여 첫번째 아규먼트는 실행할 sqlmap의 id, 두번째 아규먼트는 액션에서 받아온 map 유형의 데이터를 넘긴다.

참고자료



참고문헌
*

첨부파일

There are currently no attachments on this page.

Copyright © 2009 Daewoo Information Systems Co., Ltd.

4. SP로 리스트조회(sysbase)



목차

- 개요
- 설명
 - ibatis xml 파일 작성
 - testCase작성
- 참고자료
- 첨부파일

문서정보

- 제목:ibatis SP로 리스트조회가이드
- 최초작성자: 김민아
- 최초작성일: 2010/02/25
- 이문서는 사용자에게 0번 보여졌습니다.

개요

본 문서는 ibatis 에서 SP (stored procedure)사용 시 리스트 형태로 조회 가이드이다.

설명

ibatis xml 파일 작성

```

01.<?xml version="1.0" encoding="UTF-8" standalone="no"?>
02.<!DOCTYPE sqlMap PUBLIC "-//iBATIS.com//DTD SQL Map 2.0//EN" "
http://www.ibatis.com/dtd/sql-map-2.dtd">
03.
04.<sqlMap namespace="spTest">
05.
06.     <typeAlias alias="code" type="business.jcfcom.code.Code" />
07.
08.     <resultMap class="code" id="resultMap-code">
09.         <result property="id" column="id" />
10.         <result property="name" column="name" />
11.     </resultMap>
12.
13.
14.<parameterMap id="param" class="java.util.HashMap">
15.     <parameter property="p_bsp_cd" javaType="string" jdbcType="VARCHAR" mode=
"IN" />
16.     <parameter property="p_fr_dt" javaType="string" jdbcType="VARCHAR" mode="IN"
/>
17.     <parameter property="p_to_dt" javaType="string" jdbcType="VARCHAR" mode="IN"
/>
18.     <parameter property="p_drchr_ind" javaType="string" jdbcType="VARCHAR" mode=
"IN" />
19. </parameterMap>
20.
21.     <procedure id="procedure_out" parameterMap="param" resultMap=
"resultMap-code" >
22.         { call exec usp_fit02_03(?,?,?,?) }
23.</procedure>

```

- parameterMap 에 프로시저에 넘길 데이터를 IN 으로 지정한다,

testCase작성

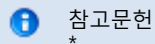
```

1.@Autowired
2.private StatementMappingDataAccessOperations dao;
3.
4.
5.     @Test
6.     public void testname() throws Exception {
7.         List<Code> list= dao.selectList("spTest.procedure_out", null);
8.         System.out.println(list.size());

```

StatementMappingDataAccessOperations (commonDao)를 private으로 선언하고 @Autowired 어노테이션을 넣은 다음 dao.selectList 하여 ibatis xml 에 있는 procedure를 호출하여 조회해온 데이터를 list에 담는다.

참고자료



첨부파일

There are currently no attachments on this page.

Copyright © 2009 Daewoo Information Systems Co., Ltd.

7. sequence호출(oracle)



목차

- 개요
- 사용방법
 - sequence 를 조회해올 경우
 - 일반 쿼리에 sequence 를 따와서 입력할 때
- 첨부파일

문서정보

- 제목: oracle sequence호출하기
- 최초작성자: 김민아
- 최초작성일: 2009/10/27
- 이문서는 사용자에게 **0** 번 보여졌습니다.

개요

본 문서는 iBatis에서 oracle sequence 를 따와서 id를 생성하는 방법에 대해 다룬다.

사용방법

sequence 를 조회해올 경우

```
1.<statement id="getSequence" parameterClass="string" resultClass="string">
2. SELECT SLTIS_SCRN_SEQ.nextval FROM DUAL
3.</statement>
```

일반 쿼리에 sequence 를 따와서 입력할 때

```
01.<insert id="insertUserBySeq" parameterClass="java.util.Map">
02.    <selectKey keyProperty="getSeq" resultClass="String">
03.        SELECT SLTIS_SCRN_SEQ.nextVal from dual
04.    </selectKey>
05.    INSERT INTO
06.        USER_INFO(
07.            USER_ID,
08.            NAME,
09.            PASSWORD,
10.            DEPT_NAME,
11.            EMAIL,
12.            REG_DATE,
13.            LOG_DATE
14.        )
15.        VALUES(
16.            #getSeq#,
17.            #NAME#,
18.            'default',
19.            #DEPT_NAME#,
20.            #EMAIL#,
21.            SYSDATE,
22.            SYSDATE)
23.</insert>
```

첨부파일

There are currently no attachments on this page.

Copyright © 2009 Daewoo Information Systems Co., Ltd.

05.연계처리레이어

* 웹서비스 등을 통해 그룹웨어 등 타시스템과 연계에 대한 가이드이다,

- 01.CXF 기반 웹서비스

01.CXF 기반 웹서비스



목차

- 개요
- 설명
 - 웹서비스로 올릴 클래스 생성
- 첨부파일

문서정보

- 제목: CXF 기반 웹서비스 구현하기
- 최초작성자: 김민아
- 최초작성일: 2010/07/18
- 이문서는 사용자에게 **17** 번 보여졌습니다.

개요

cxf 기반의 웹서비스 예제이다.

설명

웹서비스로 올릴 클래스 생성

- 웹서비스로 올린 서비스 클래스와 인터페이스를 생성 후 인터페이스를 웹서비스로 등록함

```

01.<simple:server id="hrWebServices" serviceClass="com.hhi.hiway.sample.ws.HumanResourceService"
02. address="/HumanResourceService">
03. <simple:dataBinding>
04.   <ref bean="aegisBean" />
05. </simple:dataBinding>
06. <simple:serviceBean>
07.   <ref bean="hrService" />
08. </simple:serviceBean>
09.</simple:server>

```

- 테스트 케이스 작성

```

01.package com.hhi.hiway.sample.ws;
02...
03.public class SimpleFrontEndTest {
04.
05. private HumanResourceService hrService; //
06. private HumanResourceService client; //Client
07. private Client clientProxy;
08.
09.
10. public void setUpService() throws Exception {
11.   // Create our service implementation
12.   hrService = new StubHumanResourceService();
13.   Map<String, Object> props = new HashMap<String, Object>();
14.   props.put("mtom-enabled", Boolean.TRUE); //
15.
16. //
17.   ServerFactoryBean svrFactory = new ServerFactoryBean();
18.   svrFactory.setServiceClass(HumanResourceService.class);
19.   svrFactory.setProperties(props);
20.   svrFactory.setAddress("http://localhost:8080/webservice/HumanResourceService");
21.   svrFactory.setServiceBean(hrService);
22.   TypeCreationOptions tOpts = new TypeCreationOptions();
23.   tOpts.setDefaultMinOccurs(1);
24.   tOpts.setDefaultNillable(false);
25.   AegisDataBinding db = new AegisDataBinding();
26.   svrFactory.getServiceFactory().setDataBinding(db);
27.   svrFactory.getInInterceptors().add(new LoggingInInterceptor());
28.   svrFactory.getOutInterceptors().add(new LoggingOutInterceptor());
29.   svrFactory.create();
30.
31.
32.//
33.   ClientProxyFactoryBean factory = new ClientProxyFactoryBean();
34.   factory.getInInterceptors().add(new LoggingInInterceptor());
35.   factory.getOutInterceptors().add(new LoggingOutInterceptor());
36.   factory.setServiceClass(HumanResourceService.class);
37.   factory.setProperties(props);
38.   factory.setAddress("http://localhost:8080/webservice/HumanResourceService");
39.   factory.getServiceFactory().setDataBinding(new AegisDataBinding());
40.   client = (HumanResourceService) factory.create();
41.   clientProxy = ClientProxy.getClient(client);
42. }
43.
44. @Test
45. public void testSetUpService() throws Exception {
46.   setUpService();
47.   assertNotNull(hrService);
48.   assertNotNull(client);
49.   assertNotNull(clientProxy);
50. }
51.
52.//
53. @Test
54. public void testCommunicateAndDataBinding() throws Exception {
55.   setUpService();
56.   String startDate = "2007-09-01";
57.   String endDate = "2008-09-01";
58.   String firstName = "JiHoon";
59.   String lastName = "Park";
60.   Employee employee = new Employee();
61.   Holiday holiday = new Holiday();
62.   employee.setNumber(1);
63.   employee.setFirstName(firstName);
64.   employee.setLastName(lastName);
65.   holiday.setHolidayStartDate(startDate);
66.   holiday.setHolidayEndDate(endDate);
67.   assertEquals(5, client.findHoliday(employee, holiday).size());
68. }
69.}

```

참고자료



참고문헌

*<http://blog.naver.com/dworyu?Redirect=Log&logNo=100033221834>

- * <http://dev.anyframejava.org/anyframe/doc/core/3.2.1/corefw/guide/jaxws.html>

첨부파일

이름	크기	생성자	Creation Date	댓글
웹서비스소개.ppt	957 kB	김민아	7월 22, 2010 12:03	
webservice-cxf-simple-jaxws.zip	15.18 MB	김민아	8월 20, 2010 09:16	라이브러리 포함

Copyright © 2009 Daewoo Information Systems Co., Ltd.

02. 공통서비스

JCF에 대한 공통서비스(System Common Service) 메뉴얼이 있는 공간입니다.

- 01. 파일처리
- 02. 로그처리
- 03. 다국어처리
- 11. 세션객체사용하기
- 04. 예외처리(메시지처리)
- 06. 트랜잭션처리
- 07. 캐쉬처리
- 08. 보안처리
- 09. 엑셀처리
- 10. property 서비스

01. 파일처리

- 1. 파일업로드
 - 1. SWFUpload 이용 파일업로드
- 2. 파일다운로드
 - 1. 파일 다운로드 서버릿

1. 파일업로드

- 1. SWFUpload 이용 파일업로드

1. SWFUpload 이용 파일업로드

문서정보

- 제목: SWFUpload 기반의 파일업로드 & 다운로드 가이드
- 최초작성자: 김민아
- 최초작성일: 2009-06-17
- 수정자:
- 수정일:
- 첨부파일:
Copyright © 2009 Daewoo Information Systems Co., Ltd.



목차

- SWFUpload를 이용한 파일 업로드 다운로드
 - SWFUpload란?
 - SWFUpload 구조
 - 파일 업로드 따라하기
 - 파일업로드 시퀀스 다이어그램
 - 1. Client모듈 구현
 - 1. server모듈 구현
 - 2 .파일업로드 후 업로드 정보저장
- # onLoadFile option
- 첨부파일

SWFUpload를 이용한 파일 업로드 다운로드

데모 보기

<http://wiki.dev.daewoobrenic.co.kr:8080/fileSample/simpliedemo/index.jsp>

소스 다운로드

샘플소스다운받기

실행방법)

1. 다운로드한 폴더에서 명령창으로 `mvn package --> mvn`
2. <http://localhost:8080/fileSample/simpliedemo/index.jsp> 으로 접속

SWFUpload란?

SWFUpload는 자바스크립트와 플래시를 이용하여 손쉽게 다중 파일 업로드, 프로그레스바, 파일사이즈 체크를 구현하는 오픈소스 라이브러리이다.

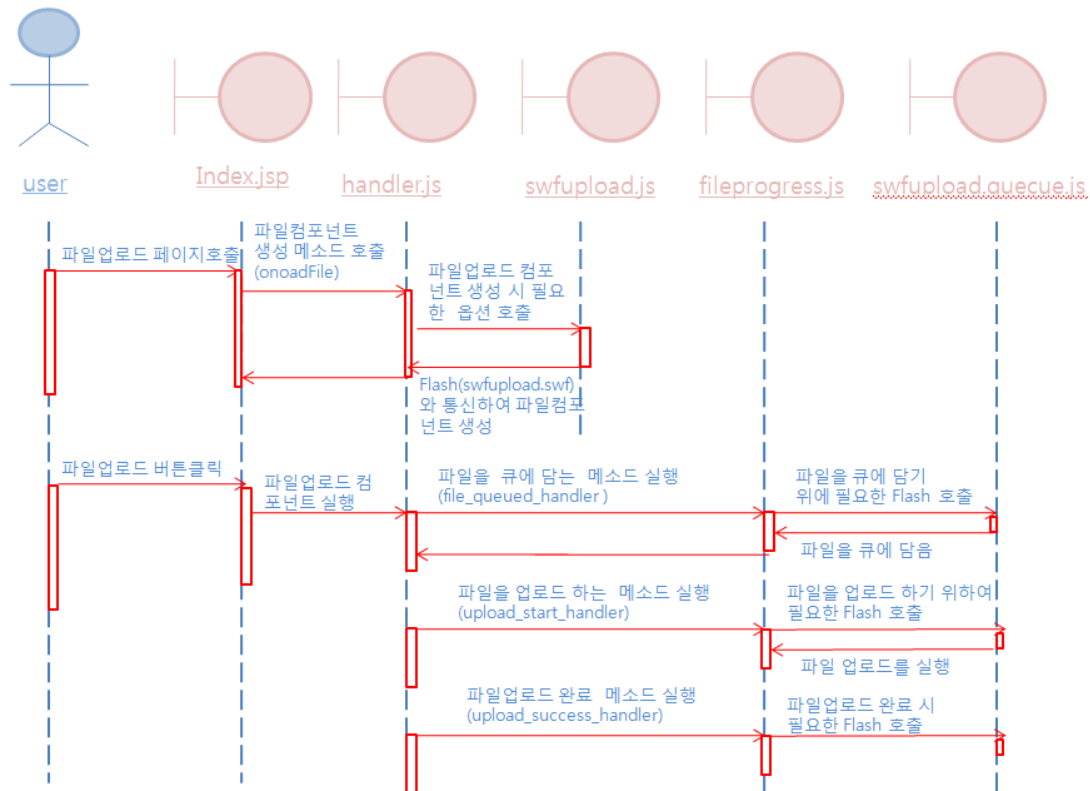
특징

- 멀티파일 선택가능
- 파일 업로딩 시 프로그레스바 제공
- 페이지 리로딩 없이 구현가능
- 파일 확장자 필터링 및 파일 사이즈 컨트롤가능
- 다양한 컨포넌트 제공

License

- MIT License

-
- SWFUpload 구조



- index.jsp파일이 로딩 시에 handler.js에 있는 파일업로드 컴포넌트를 생성하는 function을 호출(onloadFile) 한다.
- handler.js는 Fileprocess.js 와 swfupload.queue.js 에 있는 function들을 순차적으로 호출하면서 파일상태를 대기 및 Queue에 담기(pending)>

업로드(Uploading)>완료(Complete)으로 바꾸면서 파일을 queue에 담아 서버에 보낸다.

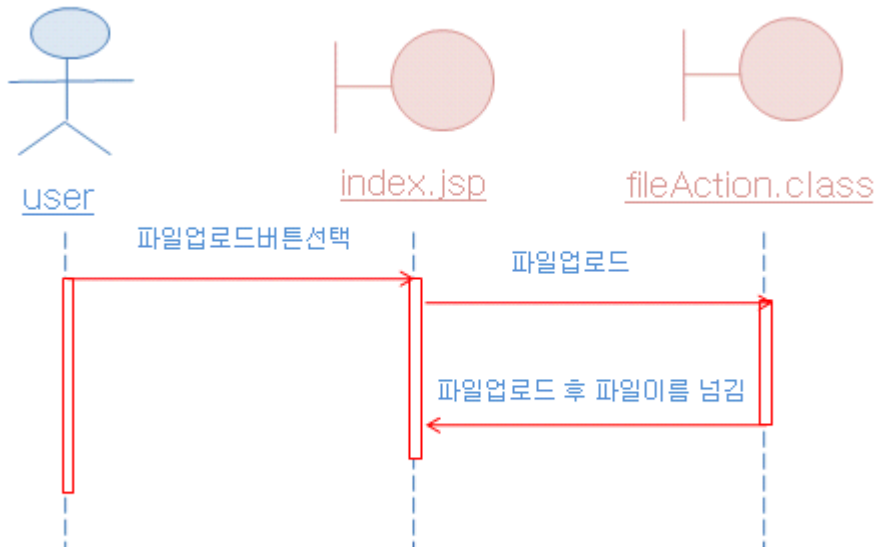
- swfUpload.js는 SWFUpload컴포넌트를 실행하기 위해 필요한 Flash파일을 호출하여 컨트롤한다.

참고 주소

- <http://www.swfupload.org/>
- <http://slog.aproxacs.com/pages/1306424>
- <http://demo.swfupload.org/Documentation/#flashReady>

파일 업로드 따라하기

파일업로드 시퀀스 다이어그램



1. Client모듈 구현

1.1필요한 js, swf 파일 복사

다음 zip파일을 다운받아 압축을 푼후 프로젝트의 web 디렉토리에 넣는다.

swfupload.zip

1.2. 필요한 js, css파일 import

1. 밑에와 같이 4개의 js파일(swfupload.js, swfupload.queue.js, fileprogress.js, handlers.js)과 1개의 CSS파일을 (default.css) import 한다.

```

1.<link href="../../swfupload/css/default.css" rel="stylesheet" type="text/css" />
2.<script type="text/javascript" src="../../swfupload/swfupload.js"></script>
3.<script type="text/javascript" src="../../swfupload/swfupload.queue.js"></script>
4.<script type="text/javascript" src="../../swfupload/fileprogress.js"></script>
5.<script type="text/javascript" src="../../swfupload/handlers.js"></script>
6.<script type="text/javascript">

```

1.3. onloadFilePath 스크립스 작성

밑에와 같이 페이지 로딩 실행되는 onloadFilePath function을 만들어 파일업로드 시 서버에 보낼 url (uploadUrl)과, 업로드 가능 파일사이즈 (file_size_limit), 파일 타입(file_types), 제한되는 파일 수(file_upload_limit)를 지정하여 onloadFile(handler.js 에 있음) 를 호출하여 parameter로 보낸다. (*참고 : file_upload_limit 파일 수에 제한이 없다는 의미임)

```

01.//
02.<script type="text/javascript">
03. window.onload = function onloadFilePath(uploadUrl) {
04.     var uploadUrl = "/fileSample/file/uploadFile.action";
05.     var file_size_limit = "100 MB";
06.     var file_types = "*.txt";
07.     file_upload_limit = 0;
08.     onloadFile(uploadUrl, file_size_limit, file_types, file_upload_limit);
09. }
10.</script>

```

1.4. 필요한 div 지정

밑에와 form을 만들어서 hidden 타입의 'fileStatus' 라는 id의 input, fsUploadProgress라는 id의 div, divStatus라는 id의 div, spanButtonPlaceholder라는 id의 span, btnCancel

라는 id의 input 들을 지정한다.

```

01. <form name="form1" action="/fileSample/file/saveFileInfo.action"
02. theme='simple' method="post" enctype="multipart/form-data">
03.   <input type="hidden" id='fileStatus' />
04.   <div class="fieldset flash" id="fsUploadProgress"></div>
05.   <div id="divStatus">0 Files Uploaded</div>
06.   <div>
07.     <span id="spanButtonPlaceHolder"></span>
08.     <input id="btnCancel" type="button" value=""
09.       onclick="swfu.cancelQueue();" disabled="disabled" />
10.   </div>
11.</form>

```

설명)

1. 'fileStatus' 은 파일이 대기,로딩 , 업로드, 완료 등 프로세스에 따라 상태값을 변화 시키는 태그이다. (import한 js파일 등에서 사용)
2. divStatus는 '1 Files Uploaded' 등 파일 첨부 후 첨부된 파일에 대해
3. fsUploadProgress은 파일업로드 구현 컴포넌트를 위치시킬 div이다, 라고 지정한다..
4. spanButtonPlaceHolder는 파일첨부 버튼을 위치시킬 span이다.
5. btnCancel 는 파일첨부 취소버튼을 위치시킬 input 태그이다.

1. server모듈 구현

1.1. Action.java 에 uploadFile 메소드구현

```

01.private String FILE_PATH="c:\\\\jcf"; //
02.     private FileInfo fileInfo; get/set
03.     public void uploadFile() throws Exception
04.     {   if ((fileInfo.getAttachFileName() != null) && (!fileInfo.getAttach().equals("")))
05.     {
06.         File tmpFile = File.createTempFile("jcf-", ".tmp", new File(FILE_PATH));
07.         FileUtils.copyFile(fileInfo.getAttach(), tmpFile);
08.         getResponse().resetBuffer(); getResponse().setCharacterEncoding("utf-8");
09.         PrintWriter writer = getResponse().getWriter(); writer.println("'" + tmpFile
10.         + "'");
11.         writer.close();   }
12.     }

```

- 위는 파일 처리를 할 upload URL(/fileSample/file/uploadFile.action)에 대한 action이다
- 여기서 fileInfo.getAttach 은 handler.js파일의 onloadFile function에서 file_post_name 으로 지정한 것이 고 (#onLoadFile option 참고)
- fileInfo.getAttachFileName 은 fileInfo.getAttach 으로 파일을 받아오면 자동으로 셋팅된다 (파일이 attach라면, 뒤에 FileName)
- FILE_PATH 에서 지정한 폴더로 파일이름을 만들어서 파일저장한다. (단, FILE_PATH 에서 지정한 폴더가 있어야함)
- 파일업로드가 끝난 후에는 writer.println를 통해 실제 업로드된 파일 이름을 Client에 보낸다.

1. 2 Model.java 구현

```

01.package fileSample.model;
02.     import java.io.File;
03.     public class FileInfo {
04.         private String saveFileName; get/set method
05.         private File attach; get/set method
06.         private String attachFileName; get/set method
07.         private String attachContentType; get/set method
08.     }
09.

```

- 위는 파일 처리를 할때 파일과 파일이름 등은 받아오거든 보내주는 모델이다.
- clientFileName은 유저가 업로드 하는 파일이름에 대한 property 이고
- attach은 파일을 받은 property 이고,
- attachFileName은 파일업로드 시 폴더에 실제로 저장되는 파일이름에 대한 property이거.
- attachContentType은 ContentType을 받아오는 property로 지정하지 않아도 자동으로 받아온다.

2.파일업로드 후 업로드 정보저장

2.1. index.jsp

```
1...
2.      <form name="form1" action="/fileSample/file/saveFileInfo.action"
3.          theme='simple' method="post" enctype="multipart/form-data">
4.      ..</form>
5.      <a href='javascript:submit()'> </a></div>
6.      ...
```

- form의 action에 파일 업로드 후 파일저장 정보를 저장 할 action url 을 지정해준다.

2.3. action.java(saveFileInfo)

```

01....
02.    public class FileAction extends BaseAction{
03.    ....
04.        private List    fileInfoList; get/set method
05.        private FileInfo fileInfo;  get/set method
06.        private String filename;    get/set method
07.
08.
09.        public String saveFileInfo() {
10.            String filename[] = getFilename().split(",");
11.            List  tempFileInfoList = new ArrayList();
12.            for(int i = 0;   i < filename.length/2;   i++ ) {
13.                String attachFileName =filename[i*2].trim();
14.                String realFileName =filename[i*2+1];
15.                FileInfo fileInfo= new FileInfo();
16.                fileInfo.setAttachFileName(attachFileName);
17.                fileInfo.setRealFileName(realFileName);
18.                tempFileInfoList.add(fileInfo);
19.            }
20.            this.fileInfoList= tempFileInfoList ;
21.            return SUCCESS;
22.        }
23.    }

```

- 화면에서 input데이터로 받은 파일이름(filename)에대한 정보를 잘라서 fileInfo객체의 attachFileName과 realFileName에 각각 담는다.
- 저장된 파일정보들을 fileInfoList에 담아 클라이언트로 보낸다.

2.4. 파일저장정보 result.jsp에 보여주기

```
01.<%@ page contentType="text/html; charset=UTF-8"%>  
02.    <%@ taglib prefix="s" uri="/struts-tags"%>  
03.        <html>  
04.            <head>  
05.                <s:head theme='simple' />  
06.            </head>  
07.            <body>  
08.                <hl> </hl>  
09.                <table cellpadding="3" cellspacing="0" border="1" width="50%">  
10.                    <tr>  
11.                        <td align="center"> </td>  
12.                        <td align="center"> </td>  
13.                        <td align="center"></td>  
14.                    </tr>  
15.                    <s:iterator value="fileInfoList">  
16.                        <tr>  
17.                            <td width='20%'><s:property value='clientFileName' /></td>  
18.                            <td width='50%'><s:property value='attachFileName' /></td>  
19.                            <td width='30%'>  
20.                                <a href="//fileSample/download/<s:property value='clientFileName'  
/?filename=<s:property value='attachFileName' />">  
21.                                    </a>  
22.                                </td>  
23.                            </tr>  
24.                        </s:iterator>  
25.                    </table>  
26.                    <br />  
27.                    <a href='%= request.getContextPath() %>/demo/simpledemo/index.jsp'%>  
28.                        ..</a>  
29.                    &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&~  
30.                </body>  
31.            </html>
```

- list로 받은 파일정보를 iterator로 화면에 뿌려준다.

```
# onLoadFile option
```

index.jsp에서 호출하는 handler.js의 onloadFile function 속성에서는 파일업로드와 관련되서 다양한 옵션을 줄 수가 있다.

속성명	설명	사용 예
file_post_name	파일저장 시 서버(action)에서 받을 파일에 대한 property	file_post_name: "fileInfo.attach"
post_params	파일저장 시 서버 보낼 parameter	<pre>1.post_params: 2. { "fileInfo.attachFileName" : attachFileName}</pre>
file_queue_limit	한번에 선택가능한 파일 수	file_queue_limit : 0 *0으로 지정하면 제한없다는 의미
debug	디버깅모드 설정 (true로 지정하면 화면에 로그가 보임)	debug: true
button_image_url	파일첨부 버튼에 적용할 image 경로	button_image_url: "images/TestImageNoText_65x29.png"
button_width	파일첨부 버튼 넓이 사이즈	button_width: "65"
button_height	파일첨부 높이 사이즈	button_height: "29"
button_text	파일첨부 버튼에 넣을 텍스트	button_text: '파일첨부'
button_text_style	파일첨부 버튼에 넣을 텍스트의 폰트	button_text: '파일첨부'
button_text_left_padding	파일첨부 버튼의 왼쪽 간격	button_text_left_padding: 12
button_text_top_padding	파일첨부 버튼의 위 간격	button_text_top_padding: 3
button_cursor	마우스 포인터가 파일첨부버튼에 갔을 때의 표시방식 (HAND or ARROW)	button_cursor: SWFUpload.CURSOR.HAND
file_queued_handler	파일이 큐에 담을 때 실행되는 function	file_queued_handler : fileQueued,
file_queue_error_handler	파일을 큐에 담는 도중 에러가 났을 때 실행되는 function	file_queue_error_handler : fileQueueError
file_dialog_complete_handler	파일을 큐에 담았을 때 실행되는 function	file_dialog_complete_handler : fileDialogComplete
upload_start_handler	큐에 담을 파일을 업로드를 시작할 때 실행되는 function	upload_start_handler : uploadStart
upload_progress_handler	파일을 업로드 시 보여지는 프로그래스바를 핸들링하는 function	upload_progress_handler : uploadProgress
upload_error_handler	파일업로드 시 에러가 났을 경우 실행되는 function	upload_error_handler : uploadError
upload_success_handler	파일업로드가 완료됐을 경우 실행되는 function	upload_success_handler : uploadSuccess
upload_complete_handler	파일업로드가 완료되어 다음큐에 있는 파일을 업로드를 준비 할 때 실행 되는 function	upload_complete_handler : uploadComplete
queue_complete_handler	큐에 담긴 파일이 모두 업로드가 완료되었을 때 실행되는 function	queue_complete_handler : queueComplete

좀더 자세한 api는 밑에 사이트를 참조
<http://demo.swfupload.org/Documentation/#flashReady>

첨부파일

이름	크기	생성자	Creation Date	댓글
 swfFile.bmp	615 kB	JCF관리자	6월 16, 2009 16:03	
 swfFile.PNG	11 kB	JCF관리자	6월 16, 2009 16:05	
 fileUploadSeq.bmp	543 kB	김민아	7월 06, 2009 11:34	

 swfupload.bmp	1.73 MB	김민아	7월 06, 2009 13:40
 fileSample.zip	55.21 MB	김민아	7월 06, 2009 13:53
 swfupload.zip	92 kB	김민아	7월 07, 2009 18:37

2. 파일다운로드

- 1. 파일 다운로드 서블릿

1. 파일 다운로드 서블릿

문서정보

- 제목: 파일 다운로드
- 최초작성자: 김민아
- 최초작성일:
- 수정자:
- 수정일:
- 첨부파일:

✔

목차

- 파일다운로드
 - fileprogress.js
 - DownloadServlet

파일다운로드

현재 샘플에서는 파일다운로드는 파일업로드 창(index.jsp)에서 바로 하거나, 파일업로드 후 파일업로드(result.jsp)에 대한 결과 창에서 하도록 되어있다.

파일업로드 창에서 바로 다운로드를 하는 url은 fileprogress.js의 function에서 지정한다.

fileprogress.js

fileprogress.js 파일의 fileProgress function 밑에 보면 다음과 같은 부분이있다.

파일업로드가 완료되면 위에서 지정한 innerHTML을 다음과 같이 파일 다운로드 하는 링크로 바꿔준다.

```

1.      status.innerHTML = null;
2.
3.      status.innerHTML = '<input id="filename" name="filename" type="checkbox"
checked="checked" value='+ fileInfo+' />
4.
5.<a href="/fileSample/download/' + file.name + '?filename='+ file.saveFileName+' "><span class=
"divFile"><u>'+ file.name+ ' /span></u></a>';

```

소스설명)

*위 링크는 파일 다운로드 서블릿에 url로 클라이언트에서 다운받을 파일이름(file.name)과 파라미터로 실제 파일이름(file.saveFileName)을 넘긴다.

*그러면 서버의 file.saveFileName 의 파일을 file.name 으로 다운로드 할 것이다.

DownloadServlet

다운로드에서 대한 서버 서블릿 소스는 다음과 같다. web.xml에 서블릿 필터를 등록한 후에 밑에와 같은 DownloadServlet 서블릿 클래스를 작성한다.

web.xml

```

1.<servlet>
2.  <servlet-name>download servlet</servlet-name>
3.  <servlet-class>download.DownloadServlet</servlet-class>
4.</servlet>
5.<servlet-mapping>
6.  <servlet-name>download servlet</servlet-name>
7.  <url-pattern>/download/*</url-pattern>
8.</servlet-mapping>

```

DownloadServlet .class

```

01.
02. public class DownloadServlet extends HttpServlet {
03.     private static final long serialVersionUID = 1L;
04.     private String FILE_PATH="c:\\jcf\\";
05.     /**
06.      * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
07.      */
08.     public void doGet(HttpServletRequest request,
09.         HttpServletResponse response) throws ServletException, IOException {
10.         String file_name = request.getParameter("filename");
11.         System.out.println(file_name);
12.         String file =file_name;
13.
14.         File f = new File(file);
15.         if(f.exists()){
16.             try {
17.                 response.setContentType("application/octet-stream");
18.                 response.setHeader("Content-Disposition","attachment");
19.                 byte[] buffer = new byte[1024];
20.                 BufferedInputStream ins = new BufferedInputStream(new FileInputStream(f));
21.                 BufferedOutputStream outs = new BufferedOutputStream(response.getOutputStream());
22.                 int read=0;
23.                 while ((read = ins.read(buffer)) != -1) {
24.                     outs.write(buffer, 0, read);
25.                 }
26.                 outs.close();
27.                 ins.close();
28.             } catch( IOException e){
29.                 PrintWriter out = response.getWriter();
30.                 out.print(e);
31.             }
32.         }
33.     }
34.
35. }

```

소스설명)

* doGet 메소드에서는 Client에서 받은 파일이름(file.saveFileName)을 가진 파일을 FILE_PATH에서 지정한 폴더에서 BufferedInputStream 으로 받을 후 OutputStream으로 Client에 보내준다.

* Content-Disposition 에서 다운로드 파일이름을 지정하지 않았으므로 Client에서 다운로드하는 파일이름은 URL에 붙여진 file.name 이 될것이다..

Copyright © 2009 Daewoo Information Systems Co., Ltd.

02. 로그처리

- 1. 로그처리
- 2.SQL로깅
- 3. iBatis Batch를 사용한 이벤트 로그 처리
- 이벤트로깅

-
- 로깅서비스 : 개발시 디버깅용 로그 메시지를 남기거나, 애플리케이션 개발 상에서 로그를 남기는 것과 관련된 표준 가이드
=> 1. 로그처리
 - SQL로깅: 어플리케이션에서 SQL 쿼리가 실행될 때 실행된 쿼리문(PreparedStatement일 경우 관련된 argument 값으로 대체된 쿼리문)에 대한 로깅
=> 2.SQL로깅

1. 로그처리



목차

- Common-logging과 Log4j 설정 및 사용법
 - 라이브러리
 - 설정 파일log4j.xml 설정
 - 실시간 설정변경 방법
- 애플리케이션 로그 남기는 방법
- 로그 수준의 구분
- 로그 메시지 포맷 설정
- 로그 어펜더 설정
 - 콘솔 로그 어펜더
 - 일일 파일 로그 어펜더
 - 롤링 파일 로그 어펜더
- Common Logging API
 - 애플리케이션에서 로그 객체 생성
 - 로그 객체의 메시지 API
 - 로그 객체의 수준 체크 API

문서정보

- 제목: 로그처리 가이드
- 최초작성자: 정광선
- 최초작성일:
- 수정자: 고경철
- 수정일:
- 첨부파일:
- 이문서는 사용자에게 145번 보여졌습니다.

로그 처리는 기본적으로 Common-logging과 Log4j를 사용한다.

많은 오픈 소스에서 사용하고 있는 apache-common-logging과 log4j를 활용하여 개발하고자 하는 애플리케이션 로그 뿐만 아니라, JCF에서 활용하는 다양한 오픈소스들의 로그를 다양하게 처리할 수 있다.

다음은 애플리케이션의 로그를 남기는 방법과 설정 방법에 대해 살펴 본다.

Common-logging과 Log4j 설정 및 사용법

Common-logging과 Log4j를 사용하기 위해서는 관련 라이브러리와 설정 파일을 작성하면 된다.

라이브러리

- commons-logging.jar
- log4j.jar

설정 파일log4j.xml 설정

log4j는 프라퍼티와 XML을 모두 설정 파일을 사용할 수 있다. 여기서는 XML을 사용하여 설정하는 방법을 살펴 본다.

설정 파일의 위치는 별도로 지정할 수도 있으나 기본적으로 자바 클래스 패스 루트(WTP의 src, 메이븐 프로젝트의 src/main/resoures)에 위치하게 된다.

src/main/resources/log4j.xml 파일 예

```

01.<?xml version="1.0" encoding="UTF-8" ?>
02.<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
03.
04.<log4j:configuration>
05.    <!-- Console Appender -->
06.    <appender name="CONSOLE_STDOUT"
07.        class="org.apache.log4j.ConsoleAppender">
08.        <layout class="org.apache.log4j.PatternLayout">
09.            <param name="ConversionPattern"
10.                value="%d{yyyy-MM-dd HH:mm:ss}] %5p (%c:%L) - %m%n" />
11.        </layout>
12.        <filter class="org.apache.log4j.varia.StringMatchFilter">
13.            <param name="StringToMatch" value="Result" />
14.            <param name="AcceptOnMatch" value="false" />
15.        </filter>
16.    </appender>
17.
18.    <logger name="org.springframework">
19.        <level value="INFO"/>
20.    </logger>
21.
22.    <logger name="java.sql">
23.        <level value="DEBUG"/>
24.    </logger>
25.
26.    <root>
27.        <level value="DEBUG"/>
28.        <appender-ref ref="CONSOLE_STDOUT" />
29.    </root>
30.
31.</log4j:configuration>

```

*<appender/>*는 로그를 출력할 매체와 그에 대한 상세 설정을 한다. 위에서는 콘솔 로그를 남기기 위한 어펜더 설정을 보여주고 있다.

각 <logger> 태그는 로그를 출력할 대상과 로그 레벨, 그리고 사용할 어펜더를 지정하게 된다.

<root> 태그는 별도로 설정되지 않은 모든 로그 대상들에 대해 일괄적으로 적용할 설정 내용을 두게 된다.

실시간 설정변경 방법

스프링 프레임워크를 사용하는 경우 다음과 같은 방법으로 web.xml에 추가적인 설정을 통하여 Log4j의 설정 내용들을 실시간으로 변경할 수 있다.

로그 설정 정보들을 애플리케이션의 리로딩 없이 반영하고 싶다면, 다음과 같은 설정을 web.xml에 추가하도록 한다.

```

01.<context-param>
02.    <param-name>webAppRootKey</param-name>
03.    <param-value>project.root</param-value>
04.</context-param>
05.
06.<context-param>
07.    <param-name>log4jConfigLocation</param-name>
08.    <param-value>
09.        classpath:/log4j.xml
10.    </param-value>
11.</context-param>
12.
13.<context-param>
14.    <param-name>log4jRefreshInterval</param-name>
15.    <param-value>1000</param-value>
16.</context-param>
17.
18.<listener>
19.    <listener-class>org.springframework.web.util.Log4jConfigListener</listener-
class>
20.</listener>

```

webAppRootKey, log4jConfigLocation, log4jRefreshInterval의 3가지 컨텍스트 파라미터와 하나의 리스너를 web.xml에 등록해 준다. 작성시 다음의 사항을 유의한다.

- webAppRootKey 컨텍스트 파라미터 작성시 그 값은 "프로젝트 명.root" 와 같이 명명한다.
- 주석에서 볼수 있는것과 같이, Log4jConfigListener를 스프링의 ContextLoaderListener 리스너 정의보다 상단에 정의하도록 한다.
- log4jRefreshInterval 컨텍스트 파라미터의 값은 리스닝 인터벌이 된다. 단위는 ms.

이와 같이 작성하면, log4jConfigLocation 컨텍스트 파라미터에 값으로 지정된 classpath:/log4j.xml 위치의 설정 파일이 변경될 때, 1초마다 변경 사항을 반영하게 된다.

애플리케이션 로그 남기는 방법

애플리케이션 상의 로그 정보들을 남기기 위해서는 로그 객체를 생성하고 해당 객체의 API를 통해서 로그를 남길 수 있다.

private 객체로 Log 객체를 생성한 후, 메시지의 종류에 따라 debug, info 등의 API를 사용하여 로그 메시지를 출력하도록 한다.

다오에서 로그를 남기는 예

```

01. public class PrgmMngtService {
02.     //
03.     private Log log = LogFactory.getLog(getClass());
04.
05.     public void insert(List list) {
06.
07.         //
08.         if ( log.isDebugEnabled() ) log.debug( "    : " + list.size() );      -----
09.
10.         //
11.         log.info("    ."); ----- 2
12.         //
13.
14.     }
15.
16. }

```

로그 정보가 커서 운용시간에는 제외 시켜야 하는 경우에는 앞선 예제의 1과 같이 로그 수준을 검사하여 쓸데 없는 데이터를 생성하지 않도록 한다. 일반적인 프로그램 수행 정보들을 정보(INFO) 수준의 로그로 남기면 된다.

위와 같이 로그를 남기면 일반적인 로그 설정에서는 다음과 같은 로그 정보들을 확인 할 수 있다.

- 로그 일시, 로그 레벨, 로그를 남긴 클래스와 라인수, 로그 내용

로그 수준의 구분

로그 메시지는 레벨을 갖는다.

로그 수준	설명	사용예
DEBUG	디버깅 용 상세 정보들	파라미터와 반환 데이터값 검증 특정 상황에 대한 시스템 상태값 검증
INFO	일반적인 애플리케이션 수행 정보	"프로그램 정보를 수정합니다." "프로그램 정보 수정을 완료했습니다."
WARN	주의해야 할 정보	"입력 파라미터가 소문자 입니다. 대문자로 변경하여 처리됩니다." "개발용으로 운영 중입니다. 운영 시에는 로그 정보를 확인하여 필요 없는 로그를 삭제하십시오."
ERROR	에러가 발생한 사항에 대한 정보	"입력값이 중복되었습니다."
FATAL	시스템 운영에 치명적인 이상 현상에 대한 정보	"Out of Memory 가 발생하였습니다."

로그 메시지 포맷 설정

Log4J 사용시 기존에 정의된 어팬더의 PatternLayout를 통해서 로그 메시지의 포맷을 정의할 수 있다.

```

01. //
02.
03.         class="org.apache.log4j.ConsoleAppender">
04.         <layout class="org.apache.log4j.PatternLayout">
05.             <param name="ConversionPattern"
06.                 value="[%d{yyyy-MM-dd HH:mm:ss}] %5p (%c:%L) - %m%n" />
07.         </layout>
08.
09.         //

```

위의 설정은 다음과 같은 로그 메시지를 출력하게 된다.

```

1. [2009-09-23 19:59:47] DEBUG (com.disc.sample.code.CodeAction:21) - sys .

```

ConversionPattern에 정의할 수 있는 항목들은 다음과 같다.

%d	포맷에 따른 날짜 정보. 포맷은 yyyy-MM-dd HH:mm:ss 등이 사용 될 수 있다.
%p	이벤트의 로그 수준
%c	이벤트의 카테고리를 출력. 클래스 패키지 명이 된다.
%L	라인 번호
%m	메시지
%n	플랫폼 독립적인 개행문자를 출력. 즉, 로그 메시지가 다음 라인으로 넘어감
%r	어플리케이션이 시작되어 로깅이벤트가 일어날때까지의 경과시간(밀리세컨드)
%t	스레드 번호 출력
%M	로그를 남기는 메소드 이름을 출력
%F	로그를 남기는 파일의 이름을 출력

로그 어펜더 설정

로그 프레임워크를 통해서 메시지를 남기게 되면, 어펜더의 설정에 따라 서버콘솔이나 파일, DB 등 다양한 로그 매체에 메시지를 남길 수 있다.

다음은 일반적으로 사용되는 로그 어펜더 설정 방법이다.

콘솔 로그 어펜더

콘솔로 로그 메시지를 출력해 주는 로그 어펜더는 다음과 같이 설정한다.

```
01. <appender name="CONSOLE_LOGGER" class="org.apache.log4j.ConsoleAppender">
02.     <layout class="org.apache.log4j.PatternLayout">
03.         <param name="ConversionPattern"
04.             value="[%d{yyyy-MM-dd HH:mm:ss}] %5p (%c:%L) - %m%n" />
05.     </layout>
06.     <filter class="org.apache.log4j.varia.StringMatchFilter">
07.         <param name="StringToMatch" value="Result" />
08.         <param name="AcceptOnMatch" value="false" />
09.     </filter>
10. </appender>
```

일일 파일 로그 어펜더

날짜별로 파일을 생성하여 로그를 남겨주는 파일 어펜더는 다음과 같이 설정한다.

```
01. <appender name="FILE_LOGGER"
02.     class="org.apache.log4j.DailyRollingFileAppender">
03.     <param name="File"
04.         value="c:/log/dlocalhost.log" />
05.     <param name="Append" value="true" />
06.     <param name="DatePattern" value="'.yyyy-MM-dd' />
07.     <layout class="org.apache.log4j.PatternLayout">
08.         <param name="ConversionPattern"
09.             value="[%d{yyyy-MM-dd HH:mm:ss}] %5p (%c:%L) - %m%n" />
10.     </layout>
11.     <filter class="org.apache.log4j.varia.StringMatchFilter">
12.         <param name="StringToMatch" value="Result" />
13.         <param name="AcceptOnMatch" value="false" />
14.     </filter>
15. </appender>
```

롤링 파일 로그 어펜더

지정된 사이즈의 로그 파일을 생성하면서 파일로 로그를 남기는 롤링 파일 로그 어펜더는 다음과 같이 설정한다.

```

1.<appender name="SyslogInfoLogFile" class="org.apache.log4j.RollingFileAppender">
2.    <param name="File" value="c:/log/rlocalhost.log" />
3.    <param name="MaxFileSize" value="10MB" />
4.    <param name="MaxBackupIndex" value="100" />
5.    <layout class="org.apache.log4j.PatternLayout">
6.        <param name="ConversionPattern" value="%d{yyyy-MM-dd HH:mm:ss}
%5p (%c:%L) - %m%n" />
7.    </layout>
8. </appender>

```

Common Logging API

애플리케이션에서 로그 객체 생성

로그를 남기기 위해 로그 객체를 생성해야 한다. 다음과 같은 Common Logging의 API를 사용한다.

- `LogFactory.getLog(Class clazz);`

```

1.protected final Log log = LogFactory.getLog(getClass()); //
2.protected final static Log log = LogFactory.getLog(.class); // static

```

로그 객체의 메시지 API

로그 객체를 통해서 로그 수준을 구별하여 메시지를 남길 때, 다음과 같은 API를 사용한다.

- `public void debug(Object message);`
- `public void info(Object message);`
- `public void warn(Object message);`
- `public void error(Object message);`
- `public void fatal(Object message);`

```

1.log.info(" ."); // INFO ,

```

로그 객체의 수준 체크 API

해당 클래스에 대해 설정된 로그 수준을 체크 하기 위한 API는 다음과 같다.

- `public boolean isDebugEnabled();`
- `public boolean isInfoEnabled();`
- `public boolean isWarnEnabled();`
- `public boolean isErrorEnabled();`
- `public boolean isFatalEnabled();`

```

1.if( log.isDebugEnabled() ) log.debug(" " + cate + " ."); // DEBUG .

```



참고문헌

•

2.SQL로깅

문서정보

- 제목: SQL로깅
- 최초작성자: 김민아
- 최초작성일: 2009/09/16
- 수정자:
- 수정일:
- 첨부파일:
- 이문서는 사용자에게 **84**번 보여졌습니다.



목차

- 필요한 라이브러리
- 사용방법
 - step1. datasource설정하기
 - Case1. JDBCDataSource를 사용할 경우
 - Case2. JNDIDataSource를 사용할 경우
 - step2. log4j.properties에 옵션추가하기

IBatis를 사용하는 경우 Log4j를 사용하면 적절한 SQL로그를 출력해 볼 수 있으나, 별도의 쿼리 프레임워크를 사용하고 있지 않는 경우, 다음에 소개되는 Log4jdbc를 사용하여 로그를 남길 수 있다.

IBatis를 사용하더라도 파라미터가 SQL 문장에 삽입된 결과를 로그로 출력하려는 경우에는 Log4jdbc를 사용하면 된다.

Log4jdbc를 사용하면 어플리케이션에서 SQL 쿼리가 실행될 때 실행된 쿼리문(PreparedStatement일 경우 관련된 argument 값으로 대체된 쿼리문)에 대해서 로그를 남기는 등의 유용한 로그를 출력할 수 있다.

Log4jdbc는 SQL로그를 남기기 위한 라이브러리로서 내부적으로 SLF 로그 라이브러리를 사용하고 있다. SLF는 Common-logging과 유사하게 로깅 구현체를 위한 인터페이스를 제공하는 로깅 어답터 라이브러리로서 여기서는 SLF와 Log4j를 함께 사용하는 방법을 설명한다.

필요한 라이브러리

Log4jdbc를 적용하기 위해서 필요 라이브러리는 다음과 같다.

- log4jdbc3-1.1.jar
- slf4j-api-1.5.8.jar
- slf4j-log4j12-1.5.8.jar
- log4j-1.2.15.jar

사용방법

step1. datasource설정하기

Case1. JDBCDataSource를 사용할 경우

```
1.driver=net.sf.log4jdbc.DriverSpy
2.url=jdbc:log4jdbc:oracle:thin:@localhost:1521:XE
3.username=jcfc.com
4.password=jcfc.com
```

- 밑의 DriverSpy에서 지원하는 JDBC 일 경우 driverClassName를 위와 같이 log4jdbc에 있는 net.sf.log4jdbc.DriverSpy로 설정한다
- 기존 JDBC URL의 jdbc 부분을 'jdbc:log4j'로 변경한다.

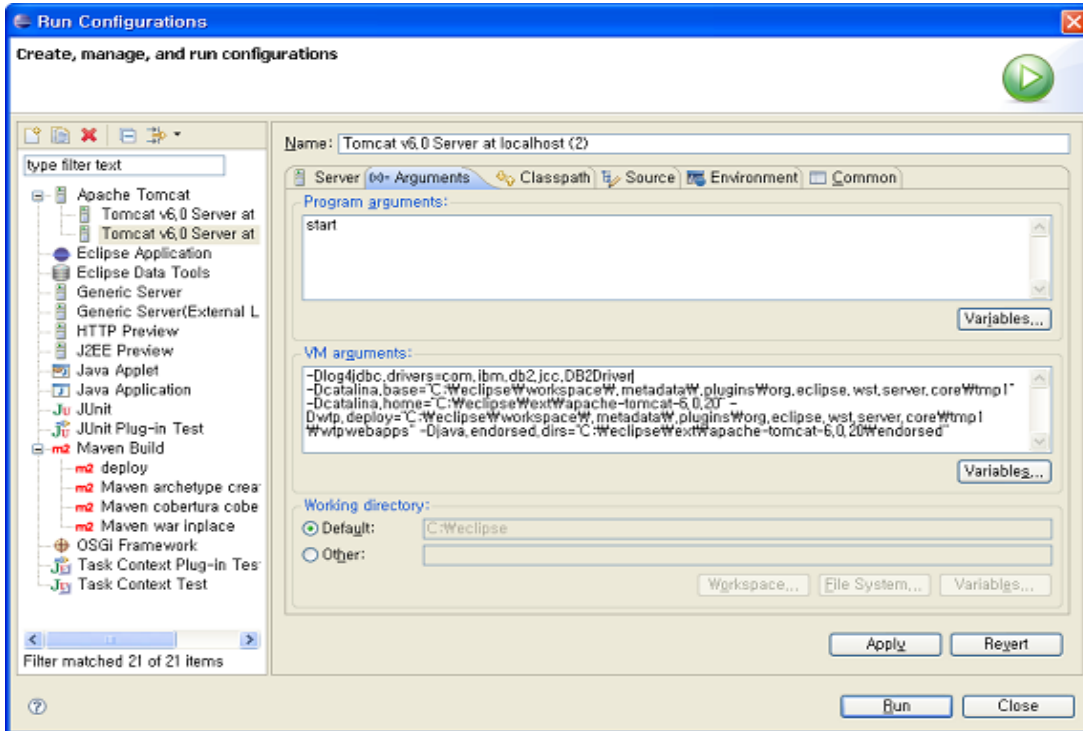
[참고] DriverSpy에서 지원하는 기본 JDBC Driver 목록

- oracle.jdbc.driver.OracleDriver
- com.sybase.jdbc2.jdbc.SybDriver
- net.sourceforge.jtds.jdbc.Driver
- com.microsoft.jdbc.sqlserver.SQLServerDriver
- weblogic.jdbc.sqlserver.SQLServerDriver
- com.informix.jdbc.IfxDriver
- org.apache.derby.jdbc.ClientDriver
- org.apache.derby.jdbc.EmbeddedDriver
- com.mysql.jdbc.Driver
- org.postgresql.Driver
- org.hsqldb.jdbcDriver
- org.h2.Driver

[위 목록에 없는 JDBC Driver일 경우]

net.sf.log4jdbc.DriverSpy에서 기본적으로 지원하는 JDBC Driver가 아닌 경우에는 위의 방식 대로 드라이버와 URL을 설정하되, 추가 셋팅이 필요하다. Eclipse를 통해 작업하는 경우 프로젝트를 선택해서 마우스 오른쪽버튼>Run AS> Run Configurations> > Arguments 탭 > VM arguments 에 log4jdbc.drivers 를 속성키로, 실제 DB의 Driver 클래스명을 속성값으로 정의해준다.

```
1.-Dlog4jdbc.drivers=com.ibm.db2.jcc.DB2Driver
```



Case2. JNDIDataSource를 사용할 경우

<가이드 추가예정>

참고 가이드 <http://dev.anyframejava.org/anyframe/doc/core/3.2.1/corefw/guide/query-sqllogging.html>WW

step2. log4j.properties에 옵션추가하기

```
01.# stdout      rootLogger
02.log4j.rootLogger=ERROR,stdout
03.
04.# the appender used for the JDBC API layer call logging above, sql only
05.log4j.appender.stdout=org.apache.log4j.ConsoleAppender
06.log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
07.
08.#
09.log4j.appender.stdout.layout.ConversionPattern=%5p [%t] %c{1} (%F:%L) %x - %m%n
10.
11.# Log4JDBC
12.log4j.logger.jdbc.sqlonly=INFO
13.log4j.logger.jdbc.sqltiming=WARN
14.log4j.logger.jdbc.audit =WARN
15.log4j.logger.jdbc.resultset=WARN
```

- 위와 같이 log4j.properties에 stdout 라는 consoleAppender를 선언하고 rootLogger 로 선언한 다음 로그 패턴(ConversionPattern)을 정의한다.(밑의 참고문헌의 로깅처리가이드참고)
 - jdbc, audit, sqlonly, resultset, sqltiming 등 옵션들 중에 남기고 싶은 로그레벨을 info 또는 debug로 선언한다.
 - jdbc.sqlonly - 실행된 SQL 쿼리문만 로그에 남긴다.
- 로그 예)

```
1.INFO [http-8081-2] sqlonly (Slf4jSpyLogDelegator.java:191) - select * from user_info where
name= 'ss'
```

- jdbc.sqltiming - SQL문과 해당 SQL을 실행시키는데 수행된 시간 정보(milliseconds)을 로그로 남긴다.

로그 예)

```
1.INFO \[http-8081-2\] sqltiming (Slf4jSpyLogDelegator.java:235)&nbsp;    ->  
2. select * from user_info where name= 'ss'{executed in 63 msec}
```

- jdbc.audit - ResultSet을 제외한 모든 JDBC 호출 정보를 로그로 남긴다. (많은양의 로그가 생성됨)

로그 예)

```

01. INFO [http-8081-1] audit (Slf4jSpyLogDelegator.java:156) - 1. Connection.setAutoCommit(true)
returned
02. INFO [http-8081-1] audit (Slf4jSpyLogDelegator.java:156) - 1. Connection.setReadOnly(false)
returned
03. INFO [http-8081-1] audit (Slf4jSpyLogDelegator.java:156) - 1. Connection.prepareStatement(
04. select * from user_info where name= ? ) returned net.sf.log4jdbc.PreparedStatementSpy
@a96eba
05. INFO [http-8081-1] audit (Slf4jSpyLogDelegator.java:156) - 1. PreparedStatement.setString(1,
"ss") returned
06. INFO [http-8081-1] audit (Slf4jSpyLogDelegator.java:156) - 1. PreparedStatement.execute()
returned true
07. INFO [http-8081-1] audit (Slf4jSpyLogDelegator.java:156) - 1.
PreparedStatement.getResultSet()
08. returned net.sf.log4jdbc.ResultSetSpy@8edb84
09. INFO [http-8081-1] audit (Slf4jSpyLogDelegator.java:156) - 1. Connection.getMetaData()
10. returned oracle.jdbc.driver.OracleDatabaseMetaData@13c765e
11. INFO [http-8081-1] audit (Slf4jSpyLogDelegator.java:156) - 1. PreparedStatement.close()
returned
12. INFO [http-8081-1] audit (Slf4jSpyLogDelegator.java:156) - 1. Connection.isClosed() returned
false
13. INFO [http-8081-1] audit (Slf4jSpyLogDelegator.java:156) - 1. Connection.getAutoCommit()
returned true
14. INFO [http-8081-1] audit (Slf4jSpyLogDelegator.java:156) - 1. Connection.clearWarnings()
returned

```

- jdbc.resultset: 모든 오픈된 커넥션 수와 커넥션이 열리고 닫힌 정보들을 로그로 남긴다. 커넥션 누수를 추적하는데 유용함

로그 예)

```

1. INFO [http-8081-1] resultset (Slf4jSpyLogDelegator.java:145) - 2. ResultSet.getType() returned
1003
2. INFO [http-8081-2] resultset (Slf4jSpyLogDelegator.java:145) - 1. ResultSet.getType()
returned 1003
3. INFO [http-8081-2] resultset (Slf4jSpyLogDelegator.java:145) - 1. ResultSet.next() returned
false
4. INFO [http-8081-2] resultset (Slf4jSpyLogDelegator.java:145) - 1. ResultSet.close() returned
5. INFO [http-8081-1] resultset (Slf4jSpyLogDelegator.java:145) - 2. ResultSet.next() returned
false
6. INFO [http-8081-1] resultset (Slf4jSpyLogDelegator.java:145) - 2. ResultSet.close() returned

```



참고문헌

- log4jdbc 가이드 <http://code.google.com/p/log4jdbc/>
- slf4j가이드+ <http://slf4j.org/>
- anyFramework log4jdbc가이드
<http://dev.anyframejava.org/anyframe/doc/core/3.2.1/corefw/guide/query-sqllogging.html>
- 로깅처리 가이드 <http://wiki.dev.daewoobrenic.co.kr/mediawiki/index.php/%EB%A1%9C%EA%B9%85>

3. IBatis Batch를 사용한 이벤트 로그 처리

문서정보

- 제목: IBatis Batch를 사용한 사용자 로그 처리
- 최초작성자:
- 최초작성일:
- 수정자:
- 수정일:
- 첨부파일:
- 이문서는 사용자에게 **22**번 보여졌습니다.



목차

- iBatis Batch 기본 개념
- iBatis Batch 적용 사례
 - iBatis Batch 적용 사례
 - ibatis batch 적용이 가능한 경우
 - ibatis batch 적용이 불가능한 경우
- iBatis Batch 를 사용한 이벤트 로그처리
 - 개념
 - 상세 코드
- 참고자료
- 첨부파일

iBatis Batch 기본 개념

한 번에 여러 개의 쿼리를 보내야할 때는 batch로 묶어서 실행시키는 것이 실행속도에 유리함. (얼마나 유리한지는 테스트 안해봐서 모르겠습니다.)

Spring에서는 이를 위한 템플릿 메소드를 만들어놓고 callback 메소드를 사용자 정의하도록 하여 처리하고 있음.

callback 메소드 내에서는 iBatis의 SqlMapExecutor가 노출이 되고, 이를 이용하여 쿼리를 수행

구현예제

```

01. public class SqlMapAccountDao extends SqlMapClientDaoSupport implements AccountDao {
02.     public void insertAccount(Account account) {
03.         getSqlMapClientTemplate().execute(new SqlMapClientCallback() {
04.             public Object doInSqlMapClient(SqlMapExecutor executor) throws
SQLException {
05.                 executor.startBatch();
06.                 executor.update("insertAccount", account);
07.                 executor.update("insertAddress",
account.getAddress());
08.                 executor.executeBatch();
09.
10.                 return null;
11.             }
12.         });
13.     }
14. }

```

iBatis Batch 적용 사례

iBatis Batch 적용 사례

실제 업무를 처리하는 웹 어플리케이션에서는 insert, update, delete와 같이 Transactional 데이터 처리(OLTP)가 빈번하게 발생한다. OLTP에 대한 대용량의 데이터 요청이 발생하면, 한 번의 데이터 Connection으로 다수의 쿼리(설정된 쿼리의 갯수)를 처리하여 Connection의 횟수를 줄이므로 성능을 향상시킬 수 있다. SqlMapClientCallback 클래스에 정의된 doInSqlMapClient 메소드에 노출된 SqlMapExecutor에 대한 startBatch()를 통해 단일 Connection에 대한 Transaction을 시작하고 다수의 쿼리를 startBatch() 이후의 로직에서 처리한다. 저장된 다수의 쿼리와 바인딩 데이터를 executeBatch()를 통해 일괄적으로 실행하고 commint하게 된다. ibatis batch를 적용하면, 대용량 데이터 처리시간을 상당히 많이 단축할 수 있다.

- ibatis batch 적용 Use Case
 - 다수의 insert, update, delete가 발생하는 배치작업
 - sequence를 보장하는 Transaction 관리가 필요한 작업 (commit, rollback, logging 처리 필요)
 - 일정한 데이터 범위(chunk) 단위로 Transaction 관리가 필요한 작업
 - DB Connection 횟수의 감소가 절대적으로 필요한 작업 (성능 향상의 목적)
- ibatis batch 적용 시 고려사항
 - 임시로 다수의 쿼리와 데이터를 저장할 수 있는 충분한 메인 메모리 확보
 - 실운영 시 다른 업무처리에 대한 영향도 분석
 - 일련의 동일한 작업을 처리하는지에 대한 업무적 특성 파악
 - 성능, 용량 등의 비기능적 요구사항 분석

ibatis batch 적용이 가능한 경우

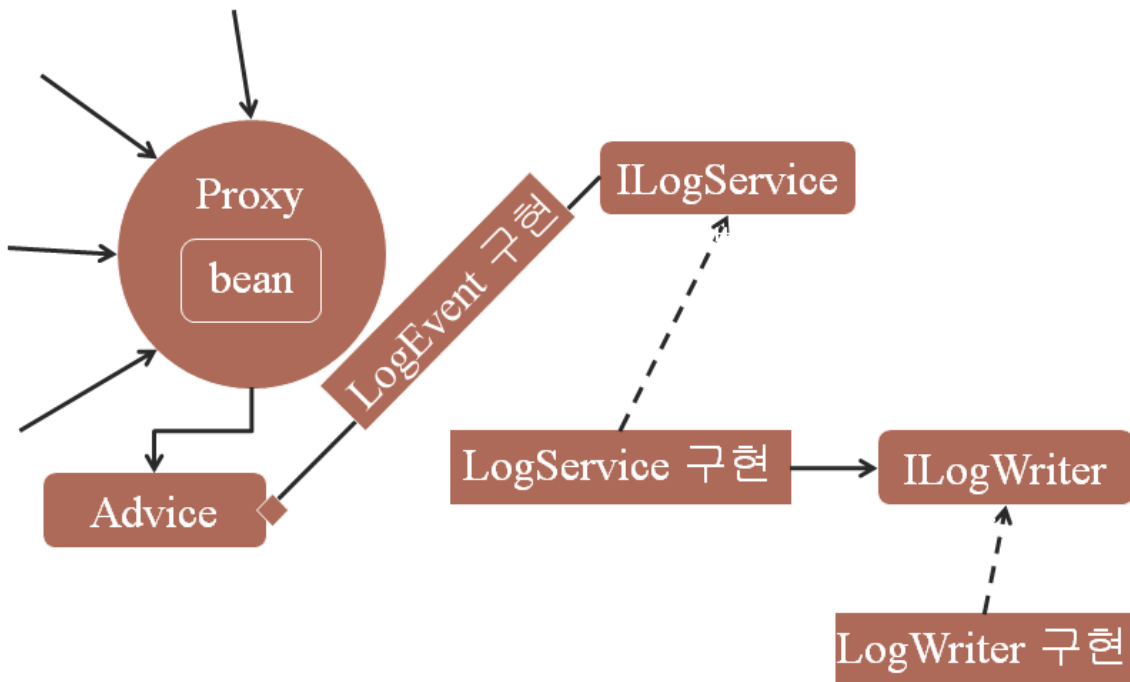
- 업무 로직이 insert, update, delete의 작업으로 구성됨
- 쿼리 실행 시 sqlmap에 작성된 쿼리의 변경이 없음
- 대상 쿼리에 정형화된 데이터가 파라미터로 전달됨 (model(Bean), list, map 등)
- 대용량의 OLTP성 데이터 처리가 필요함

ibatis batch 적용이 불가능한 경우

- 업무 로직이 select와 같이 OLAP성 데이터 처리로 구성됨
- sqlmap에 변수로 처리되어 쿼리 실행 시마다 쿼리의 변경이 발생함
- 쿼리에 변경이 발생하여 전달되는 파라미터가 비정형화된 형태로 제공됨

iBatis Batch 를 사용한 이벤트 로그처리

개념



1. Event Log 는 시스템에서 발생하는 모든 이벤트를 한곳으로 모아 미리 정의된 정책에 따라 지정된 장소(DB,File,또는 Console)에 기록하는 행위를 말한다.
2. 시스템에서 이벤트 발생시 마다 로그 기록시 시스템에서 I/O Traffic 발생하게 된다.
3. 프레임워크에서는 이벤트 로그를 발생 시점마다 즉시 기록하지 않고 관리자가 지정한 특정시간 또는 특정 이벤트 건수를 모아서 한번에 기록하여 I/O Traffic을 향상시킨다.

상세 코드

참고자료



참고문헌

- 구현 참조
 - <http://static.springframework.org/spring/docs/2.5.x/api/org/springframework/orm/ibatis/SqlMapClientTerr>
 - <http://static.springframework.org/spring/docs/2.5.x/api/org/springframework/orm/ibatis/SqlMapClientCall>
 - <http://ibatis.apache.org/docs/java/user/com/ibatis/sqlmap/client/SqlMapExecutor.html>
- 성능 비교
 - <http://www.oreilly.com/catalog/jorajdbc/chapter/ch19.html>

첨부파일

There are currently no attachments on this page.

Copyright © 2008 Daewoo Information Systems Co., Ltd.

이벤트로깅

**목차**

- 이벤트 로그
 - 용어정리
 - 개요
 - 사용법
 - 기록된 로그 예제
 - 이벤트 로그 구조
 - 참고소스

문서정보

- 제목: 이벤트 로깅
- 최초작성자: 김민아
- 최초작성일: 2009/10/22
- 이문서는 사용자에게 7번 보여졌습니다.

이벤트 로그**용어정리**

- 이벤트 로그란 사용자가 화면에서 호출한 URL, 그리고 호출된 클래스와 메소드에서 대해서 로그를 남기는 것을 말한다.

개요

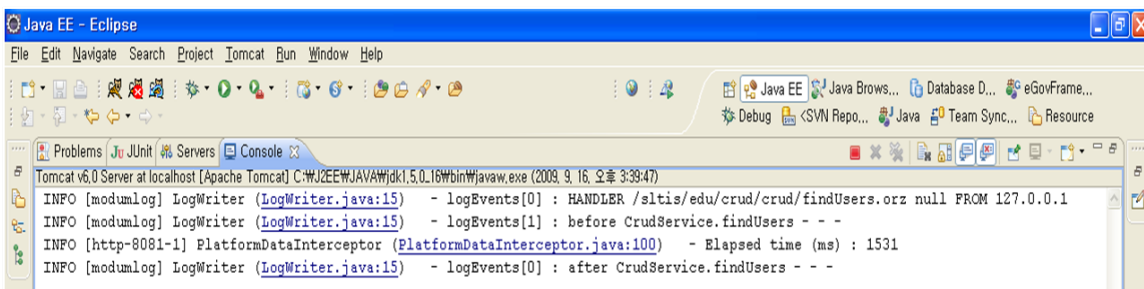
- . 본 프로젝트에서는 Event Log 라는 것을 사용하여 Client(MiplatForm)에서 요청한 URL 과 , Client IP 그리고 server호출 전 후에 호출된 메소드에 대해 로그를 남긴다.
- Event Log 는 시스템에서 발생하는 모든 이벤트를 한곳으로 모아 미리 정의된 정책에 따라 지정된 장소(DB,File,또는 Console)에 기록하는 행위를 말한다.
- 시스템에서 이벤트 발생시 마다 로그 기록시 시스템에서 I/O Traffic 발생하게 된다.
- 프레임워크에서는 이벤트 로그를 발생 시점마다 즉시 기록하지 않고 관리자가 지정한 특정시간 또는 특정 이벤트 건수를 모아서 한번에 기록하여 I/O Traffic을 향상시킨다.

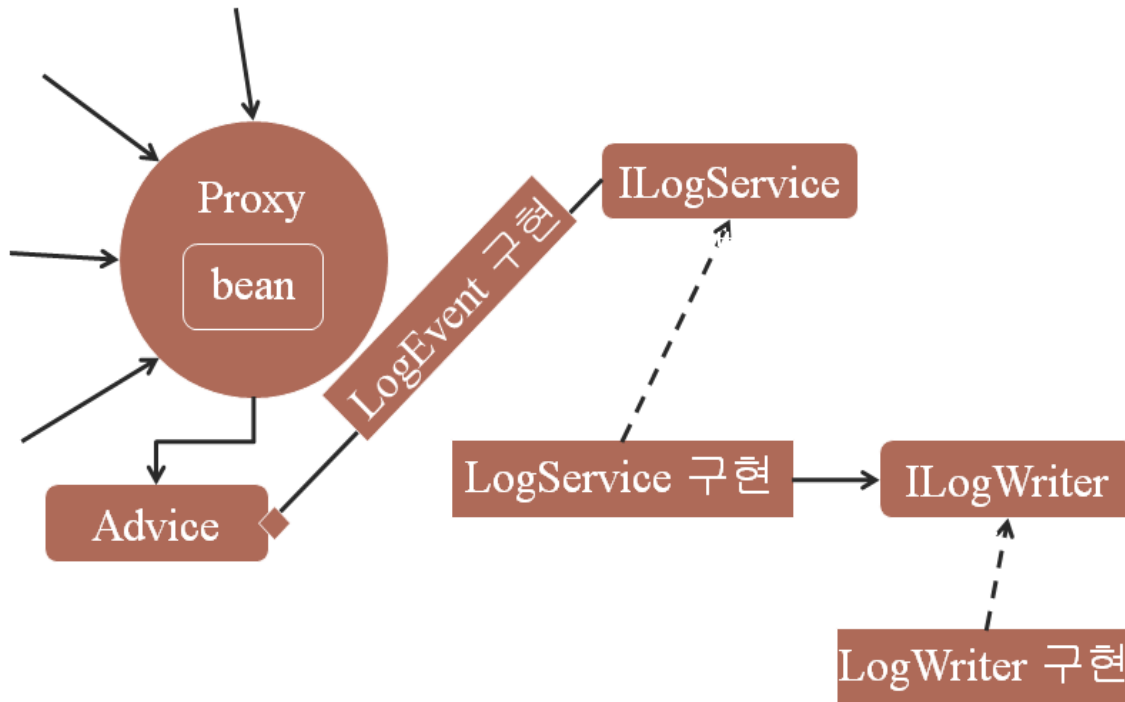
사용법

log4j.properties.파일에서 gov.seoul.asdp.sltis.system의 로그 레벨을 INFO이하로 설정한다.

```
#####
##### APPLICATION LOG CONFIGURATION #####
#log4j.logger.gov.seoul.asdp.sltis.business=DEBUG
log4j.logger.gov.seoul.asdp.sltis.system=INFO
```

참고 자료 : log4j를 이용한 로깅처리 <http://wiki.dev.daewoobrenic.co.kr/mediawiki/index.php/%EB%A1%9C%EA%B9%85>

기록된 로그 예제**이벤트 로그 구조**



참고소스

applicationContext.xml 파일

```

01.<!-- . 1/1000 -->
02. <bean class="gov.seoul.asdp.sltis.system.aspect.EventLogging">
03.   <property name="logService" ref="logService" />
04.   <property name="userSession" ref="userSession" />
05. </bean>
06. <bean id="logService" class="gov.seoul.asdp.sltis.system.log.ModumLogServiceBean">
07.   <property name="batchWriter">
08.     <bean class="gov.seoul.asdp.sltis.system.log.LogWriter" />
09.   </property>
10.   <property name="latency" value="1000" />
11.   <property name="threshold" value="100" />
12. </bean>

```

- 환경설정에서 latency(시간), threshold(건수)를 지정하여 로그 정책을 설정한다.

LogWriter.java

```

01.package gov.seoul.asdp.sltis.system.log;
02.
03.public class LogWriter implements IBatchWriter {
04.
05.   private static final Logger logger = LoggerFactory.getLogger(LogWriter.class);
06.
07.   public void write(Object[] logEvents) {
08.     for (int i = 0; i < logEvents.length; i++) {
09.       logger.info("logEvents[" + i + "] : " + logEvents[i]);
10.     }
11.   }
12.}

```

- 위 클래스의 write메소드에서는 모와진 이벤트에 대해 로그를 남긴다.

03. 다국어처리

✓ 목차

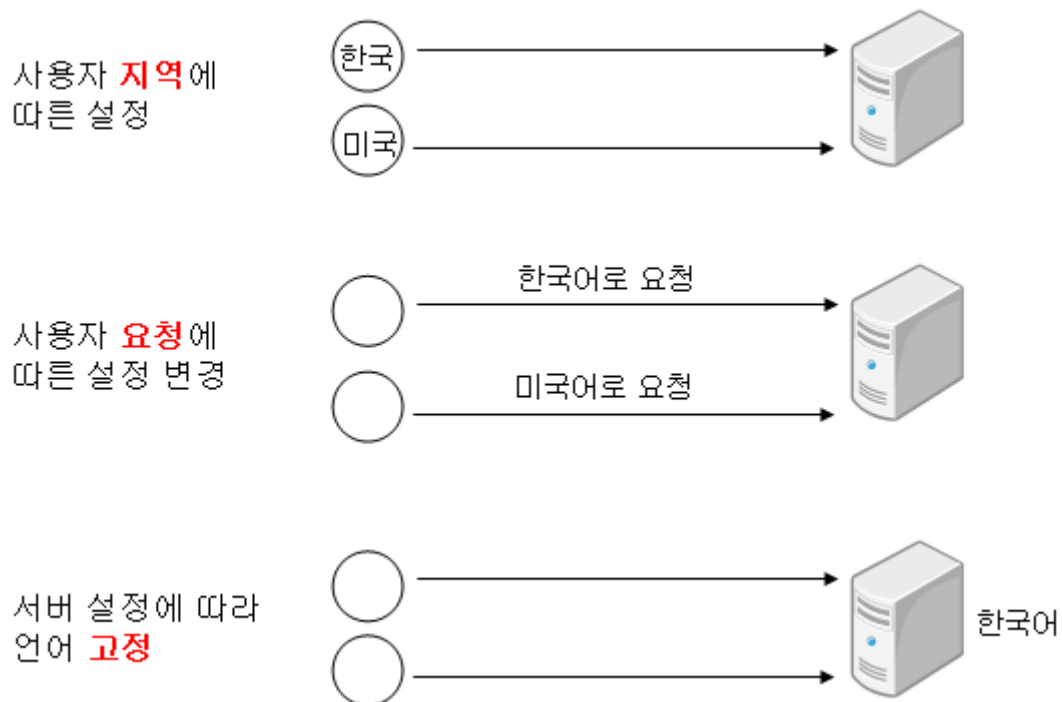
- 설정 타입
- 설정 타입별 적용 방법
 - 1. 사용자 지역에 맞는 언어 설정

문서정보

- 제목: 다국어처리
- *최초작성자: *
- 최초작성일:
- 수정자:
- 수정일:
- 첨부파일:
- 이문서는 사용자에게 **16**번 보여졌습니다.

설정 타입

기본적으로 다음의 3가지 타입의 다국어 설정을 고려할 수 있다.



설정 타입별 적용 방법

다국어를 처리하기 위해서는 우선 언어에 맞는 프라퍼티 파일 혹은 DB에 의한 메시지 정보가 있어야 한다. 여기서는 다음과 같은 메시지 프라퍼티 파일을 사용하는 것으로 한다.

- 한국어 프라퍼티 파일 : src/main/resources/message/message_ko.properties
- 영어 프라퍼티 파일 : src/main/resources/message/message_en.properties
- 중국어 프라퍼티 파일 : src/main/resources/message/message_cn.properties

1. 사용자 지역에 맞는 언어 설정

접근하는 사용자의 요청의 헤더에 담겨져 있는 사용자의 지역 정보를 기반으로 해당 지역의 언어에 맞는 메시지 프라퍼티 파일의 내용을 메시지로 사용한다. 프레임워크의 기본적인 설정타입이며, applicationContext.xml 파일에 다음과 같이 설정한다.

src/main/resources/config/applicationContext.xml 파일의 메시지 설정

```

01.<bean id="messageSourceAccessor"
02.    class="org.springframework.context.support.MessageSourceAccessor">
03.    <constructor-arg ref="messageSource" />
04.    </bean>
05.    <bean id="messageSource" class=
06.    "org.springframework.context.support.ReloadableResourceBundleMessageSource">
07.        <property name="basenames">
08.            <list>
09.                <value>WEB-INF/classes/message/messages</value>
10.            </list>
11.        </property>
12.        <property name="cacheSeconds" value="5" />
13.    </bean>
14.    <bean id="message" class="com.hhi.hiway.message.HHIMessage">
15.        <property name="messageSourceAccessor" ref="messageSourceAccessor" />
16.    </bean>

```

코드상에서 메시지를 참조하여 사용하는 예제

```

1.@Override
2.    public boolean select() {
3.        message.setErrorMessage(model, "COMN0018");
4.        return false;
5.    }

```

상단의 코드와 같이 messageSourceAccessor, messageSource, message를 사용하여 메시지를 프라퍼티 파일에서 가져오도록 설정이 되어 있다면 하단의 자바소스에서 사용하는 메시지(혹은 프레임워크내부에서 참조하는 메시지)들은 자동으로 요청자의 지역 정보에 맞는 언어로 메시지를 가져와 사용하게 된다.

- 1. 다국어처리 - SpringMVC 사용시
- 2. 다국어처리 - Struts2 사용시

1. 다국어처리 - SpringMVC 사용시



목차

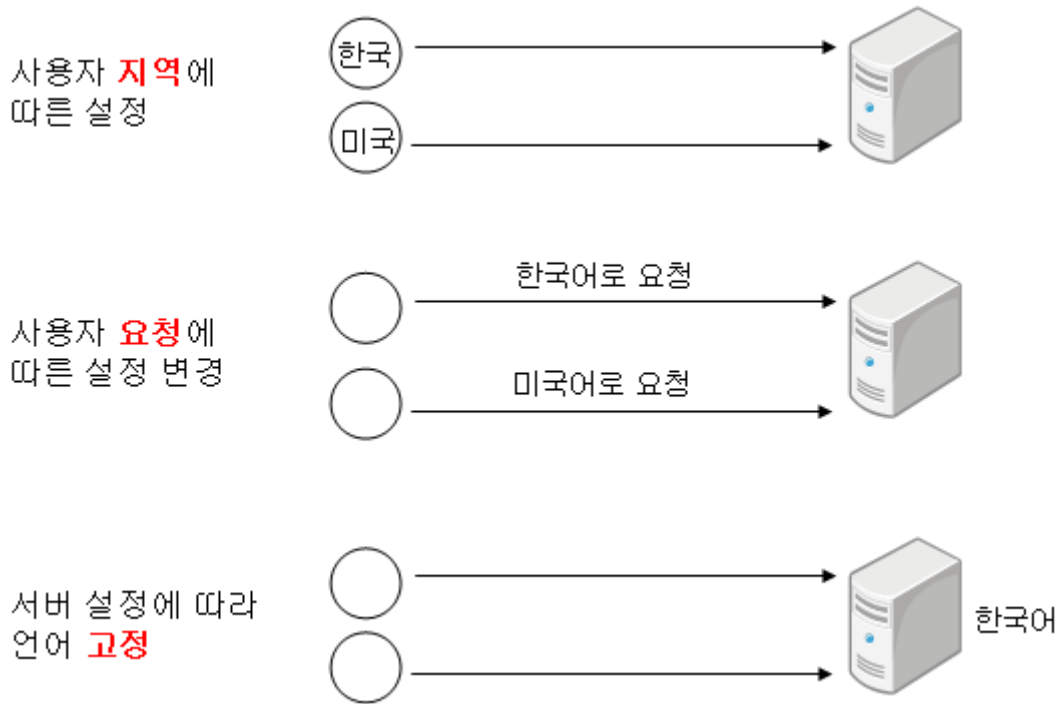
- 설정 타입
- 설정 타입별 적용 방법
 - 1. 사용자 지역에 맞는 언어 설정
 - 2. 사용자 요청에 의해 언어 변경
 - 3. 서버측에 설정한 언어 고정
- 각 언어별 코드값
- 참고자료
- 첨부파일

문서정보

- 제목: 다국어처리
- *최초작성자: *
- 최초작성일:
- 수정자:
- 수정일:
- 첨부파일:
- 이문서는 사용자에게 **26**번 보여졌습니다.

설정 타입

기본적으로 다음의 3가지 타입의 다국어 설정을 고려할 수 있다.



설정 타입별 적용 방법

다국어를 처리하기 위해서는 우선 언어에 맞는 프라퍼티 파일 혹은 DB에 의한 메시지 정보가 있어야 한다. 여기서는 다음과 같은 메시지 프라퍼티 파일을 사용하는 것으로 한다.

- 한국어 프라퍼티 파일 : src/main/resources/message/message_ko.properties
- 영어 프라퍼티 파일 : src/main/resources/message/message_en.properties
- 중국어 프라퍼티 파일 : src/main/resources/message/message_cn.properties

1. 사용자 지역에 맞는 언어 설정

접근하는 사용자의 요청의 헤더에 담겨져 있는 사용자의 지역 정보를 기반으로 해당 지역의 언어에 맞는 메시지 프라퍼티 파일의 내용을 메시지로 사용한다. 프레임워크의 기본적인 설정타입이며, applicationContext.xml 파일에 다음과 같이 설정한다.

src/main/resources/config/applicationContext.xml 파일의 메시지 설정

```

01.<bean id="messageSourceAccessor"
02.    class="org.springframework.context.support.MessageSourceAccessor">
03.    <constructor-arg ref="messageSource" />
04.</bean>
05.<bean id="messageSource" class=
06.    "org.springframework.context.support.ReloadableResourceBundleMessageSource">
07.    <property name="basenames">
08.    <list>
09.    <value>WEB-INF/classes/message/messages</value>
10.    </list>
11.    </property>
12.</bean>
13.<bean id="message" class="com.hhi.hiway.message.HHIMessage">
14.    <property name="messageSourceAccessor" ref="messageSourceAccessor" />
15.</bean>

```

코드상에서 메시지를 참조하여 사용하는 예제

```

1.@Override
2.public boolean select() {
3.    message.setErrorMessage(model, "COMN0018");
4.    return false;
5.}

```

상단의 코드와 같이 messageSourceAccessor, messageSource, message를 사용하여 메시지를 프라퍼티 파일에서 가져오도록 설정이 되어 있다면 하단의 자바소스에서 사용하는 메시지(혹은 프레임워크내부에서 참조하는 메시지)들은 자동으로 요청자의 지역 정보에 맞는 언어로 메시지를 가져와 사용하게 된다.

2. 사용자 요청에 의해 언어 변경

사용자 지역에 맞는 언어 설정과 동일하며 단지 다음과 같은 설정이 추가적으로 사용된다. 추가적인 설정은 다음과 같다.

```

01.<bean id="localeChangeInterceptor" class=
"org.springframework.web.servlet.i18n.LocaleChangeInterceptor">
02.    <property name="paramName" value="toLang" />
03.</bean>
04.
05.<bean id="localeResolver" class="org.springframework.web.servlet.i18n.SessionLocaleResolver" />
06.
07.<bean class="org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping">
08.    <property name="interceptors">
09.        <list>
10.            <ref bean="localeChangeInterceptor" /> // ----- 1
11.            <bean class=
"com.hhi.hiway.web.interceptor.DefaultMultiChannelEnableInterceptor" />
12.        </list>
13.    </property>
14.</bean>

```

localeChangeInterceptor와 localeResolver 빈들을 스프링 서블릿 XML에 작성한다. 이렇게 작성된 localeChangeInterceptor 인터셉터를 "1"과 같이 DefaultAnnotationHandlerMapping의 인터셉터로 등록해주면 컨트롤에서 요청을 처리할때, 요청 파라미터에 toLang라는 파라미터에 지정된 언어로 요청자의 메시지 언어로 바꾸어준다. 이러한 요청은 한번 지정되면 다음에 변경될 때까지 계속해서 적용된다(세션에 언어설정 저장함).

toLang 파라미터가 ko면 한국어, en이면 영어, cn이면 중국어를 의미하는 언어코드값이 된다.

만약 서버측의 언어만 변경하려고 한다면, LocaleController를 참조하여 언어만 변경할 수 있도록 하면 된다.

```

01.@Controller
02.public class LocaleController {
03.
04.    private Log log = LogFactory.getLog(getClass());
05.
06.    @RequestMapping("/locale/change.do")
07.    public void changeLocale(HttpServletRequest request, HttpServletResponse response){
08.
09.        Locale locale = RequestContextUtils.getLocale(request);
10.        log.info("Locale is changed to " + locale.toString());
11.
12.        OutChannel.success();
13.    }
14.}

```

3. 서버측에 설정한 언어 고정

마지막으로 서버측에 애플리케이션에서 사용할 언어를 한가지로 고정하는 경우에 사용하는 방법이다.

사용자 요청에 의해 언어 변경에서 사용한 설정에서 localeResolver 빈의 정의 부분만 달라진다.

```

01.<bean id="localeChangeInterceptor" class=
"org.springframework.web.servlet.i18n.LocaleChangeInterceptor">
02.    <property name="paramName" value="toLang" />
03.</bean>
04.
05.<bean id="localeResolver" class="org.springframework.web.servlet.i18n.FixedLocaleResolver">
06.    <property name="defaultLocale" value="en" />
07.</bean>
08.
09.<bean class="org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping">
10.    <property name="interceptors">
11.        <list>
12.            <ref bean="localeChangeInterceptor" />
13.
14.            <bean class=
"xxx.web.interceptor.DefaultMultiChannelEnableInterceptor" />
15.        </list>
16.    </property>
17.</bean>

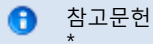
```

위와 같이 localeResolver를 정의할 때, FixedLocaleResolver를 사용하여 프라퍼티의 defaultLocale 정보로 특정 언어코드를 지정해 주면, 언제나 지정된 언어로만 메시지를 생성하게 된다. 위의 코드에서는 영어(en)을 언어코드를 사용했다.


각 언어별 코드값

- http://ko.wikipedia.org/wiki/ISO_639 페이지에서 자세히 참조할 수 있다.

참고자료



첨부파일

이름	크기	생성자	Creation Date	댓글
 i18n_config_type.PNG	19 kB	고경철	8월 24, 2009 11:08	

2. 다국어처리 - Struts2 사용시

국가세정 케이스

11.세션객체사용하기

목차

- 개요
- 설명
 - 설정파일에 userSession 등록하기
 - userSession 모델 만들기
 - UserSession 사용하기
- 참고자료
- 첨부파일

문서정보

- 제목: userSession 사용가이드
- 최초작성자: 김민아
- 최초작성일: 02/06
- 이문서는 사용자에게 0번 보여졌습니다.

개요

본 문서는 세션 사용법에 대해 다룬다.

설명

설정파일에 userSession 등록하기

system 폴더 밑에 applicationContext.xml 파일에 다음과 같이 등록한다. 밑에 와 같이 설정하면 UserSession 에 대한 클래스가 session scope형태로 생성될 것이다.

```
1.<!-- HTTP request -->
2.<bean autowire-candidate="true" id="userSession" class="business.jcfcom.user.UserSession"
3.    scope="session">
4.    <aop:scoped-proxy proxy-target-class="true" />
5.</bean>
```

userSession 모델 만들기


```
01.package business.jcfcom.user;
02.public class UserSession {
03.    private String userId;
04.    private String ip;
05.
06.    public String getUserId() {
07.        return userId;
08.    }
09.    public void setUserId(String userId) {
10.        this.userId = userId;
11.    }
12.    public void setIp(String ip) {
13.        this.ip = ip;
14.    }
15.    public String getIp() {
16.        return ip;
17.    }
18.
19.}
```

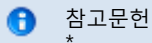
위 클래스에 세션객체에 담을 정보를 추가한다.

UserSession 사용하기

```
01.@Autowired
02.private UserSession userSession;
03.@RequestMapping
04.    public void saveUser(List<User> userList) throws Exception{
05.System.out.println(userSession.setUserId("test"));
06.
07.    System.out.println(userSession.getUserId());
08.    userService.saveUser(userList);
09.    }
```

- 사용자 세션정보가 필요한 클래스에서 위와 같이 private 변수로 선언 후 Autowired 어노테이션을 선언한다. .
- get/set 메소드를 통해 데이터를 담거나 꺼낸다.

참고자료



첨부파일

There are currently no attachments on this page.

Copyright © 2009 Daewoo Information Systems Co., Ltd.

04. 예외처리(메시지처리)

- 1.예외처리(springMVC+Miplatform)

1.예외처리(springMVC+Miplatform)

문서정보

- 제목00.예외처리 가이드
- 최초작성자: 김민아
- 최초작성일: 2009 년 09월 02일
- 수정자:
- 수정일:
- 첨부파일:



목차

- 예외처리개요
 - 개요
 - 특징
- 예외처리 예제
 - Service
 - Message Properties
 - 예외처리화면예제
 - 참고자료

예외처리개요

개요

본 프로젝트에서 예외 처리는 Spring MVC를 사용하여 처리하고 있다.

특징

시스템에서 발생하는 모든 예외는 프레임워크에서 자동으로 처리하고 UI로 전송한다

try catch 를 처리를 하지않고 예외가 발생하면 예외메시지를 화면으로 전송한다.

모든 예외에 대한 내용은 Message.properties에 정의된다.(정의된 메시지로 UI에 표시됨)

예외를 시스템예외, 비즈니스 예외로 나눈다.

- System 예외
 - 예상할 수 없거나, 검사할 수 없는 예외이며. 예)무결성예외, 커넥션 획득 실패 등
- 코딩 상으로 처리하지않아도 자동으로 처리된다,
- Biz 예외
 - 검사할 수 있거나 로직상의 예외이며 예) 계좌 이체 한도 초과 등
 - 비즈니스 적인 예외 이므로 코딩상으로 처리한다.

예외처리 예제

Service

```
01. public List<Map<String,?>> throwException(Map<String,?> userMap) {  
02.     List<Map<String,?>> list = null;  
03.                                     List list = dao.findUsers("User.findUsers",  
userList);  
04.     Map userMap= (HashMap)list.get(0);  
05.     String ssn = userMap.get("USER_SSN");  
06.     if (ssn == null) {  
07.         throw new BusinessException("exception.ssn");  
08.     } else return list;  
09. }
```

- 위 코드와 같이 로직 상으로 예외를 검사하여 예외가 났을 경우(ssn이 널일경우) BusinessException을 내보낸다.

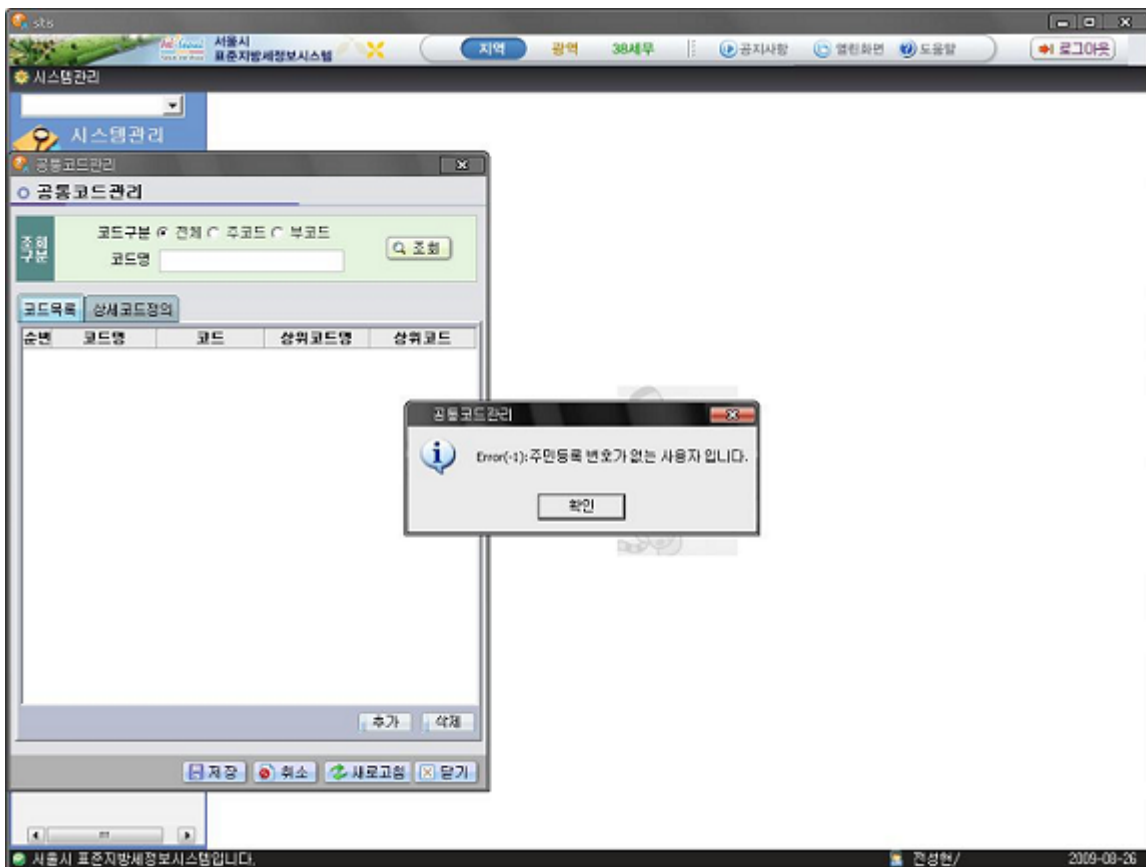
-----> throw new BusinessException("exception.ssn");

Message Properties

```
01.exception.DataIntegrityViolationException=
02.
03.exception.EmptyResultDataAccessException=
04.
05.exception.InvalidResultSetAccessException=
06.
07.exception.IncorrectResultSizeDataAccessException=
08.
09.exception.Throwable=
10.
11.exception.login=
12.
13.exception.validate =
14.
15.exception.ssn =
```

- 모든 예외에 대해(비즈니스적인 예외, 시스템적인 예외) 위와 같이 화면에 출력할 메시지를 지정한다.

예외처리화면예제



- 위 그림과 같이 Message Properties 에 지정한 예외 메시지가 화면에 출력된다.

참고자료



참고문헌

- spring 예외처리하기 http://wiki.dev.daewoobrenic.co.kr/mediawiki/index.php/SpringMVC_Exception_Handling

06. 트랜잭션처리

**목차**

- 개요
- 설명
 - 트랜잭션패턴
- 참고자료
- 첨부파일

문서정보

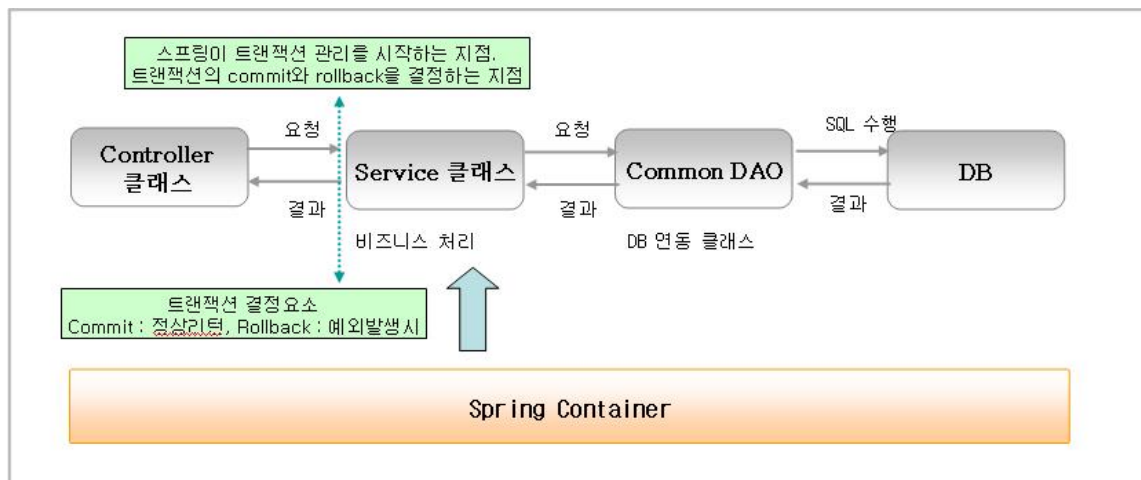
- 제목: 트랜잭션 처리
- 최초작성자: 김민아
- 최초작성일: 2010/05/24
- 이문서는 사용자에게 71번 보여졌습니다.

개요

트랜잭션은 데이터 베이스에 DML(INSERT, UPDATE, DELETE) 문장을 수행하는 경우 적합하게 Commit하고 Rollback하는 데이터 베이스 트랜잭션을 의미한다.

본 문서는 스프링 트랜잭션 관리 기능을 사용하면 데이터 베이스를 연동하는 다양한 기술들(JDBC, Stored Procedure, ibatis, hibernate, EJB, ..)에 대해서 일관된 방법으로 트랜잭션을 관리하는 방법에 대해 설명한다.

스프링 2.0이상의 버전에서 제공하는 AOP 기능을 사용하면 개발자가 로직에서 트랜잭션을 처리하지않고 자동으로 처리를 할수있다.

설명**처리흐름**

본 프로젝트에서는 AOP 기술을 사용하요 Service 클래스 수행 시점에서 트랜잭션을 거는 것으로한다.

service에서 dao를 여러번 호출하여 데이터를 처리하는 경우 에러가나면 이전에 처리했던 데이터들은 롤백이 되고 에러메시지를 화면에 뿌려줄것이다.

트랜잭션패턴

밑에와 같이 서비스 메소드 이름이 save, insert, update, delete 로 시작되는 메소드에 대해서 트랜잭션이 발생한다.

applcaionContext-tx.xml

```

01.<aop:aspectj-autoproxy proxy-target-class="true" />
02.<tx:advice id="txAdvice">
03. <tx:attributes>
04.   <tx:method name="tx*" propagation="REQUIRED" />
05. </tx:attributes>
06.</tx:advice>
07.<aop:config>
08. <aop:advisor advice-ref="txAdvice"
09.   pointcut="execution(* business..*Service.*(..))" />
10.</aop:config>
11.<!-- tx   Spring bean-->
12.<bean autowire-candidate="true" id="transactionManager"
13.  class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
14.  <property name="dataSource" ref="dataSource" />
15.</bean>

```

참고자료



참고문헌

*

첨부파일

이름	크기	생성자	Creation Date	댓글
 tx.bmp	854 kB	김민아	7월 19, 2010 15:50	

Copyright © 2009 Daewoo Information Systems Co., Ltd.

07. 캐시처리

- 1. 캐시처리 - Java Caching System (JCS) 적용하기

1. 캐시처리 - Java Caching System (JCS) 적용하기

문서정보

- 제목: 캐시처리 - Java Caching System (JCS) 적용하기
- 최초작성자: 서경진 DR
- 최초작성일: 2009년 9월 11일 금요일
- 수정자:
- 수정일:
- 첨부파일:
- 이문서는 사용자에게 **18**번 보여졌습니다.



목차

- 캐시처리 개요
 - 캐시적용목적
 - 캐시적용 요구사항
- JCS 개요
 - JCS란?
- JCS 적용하기 (JDK 1.4)
 - 필수라이브러리
 - 적용순서
- JCS 적용하기 (JDK 1.5 이상)
 - 설정방법
- 참고자료
- 첨부파일

캐시처리 개요

캐시적용목적

- 빈번한 DB IO로 인해 발생할 수 있는 성능저하를 제거하기 위해 캐시를 적용한다.
- 대부분이 조회이고 CUD가 거의 없는 정적인 데이터를 DB조회 없이 로딩하기 위해 캐시를 적용한다.

캐시적용 요구사항

- 애플리케이션 메소드의 동작에 따라 성능향상을 위한 적절한 캐시를 선별한다.
- 사용 목적을 명확히 정의하여 필요한 부분에만 캐시를 적용한다.
- 캐시의 잘못된 사용은 데이터의 동일성, 일치성에 악영향을 줄 수 있다.

JCS 개요

JCS란?

- 자바로 구현된 분산 캐시 시스템이다.
- 다양한 캐시 데이터 관리방법을 제공하여 자바 기반 애플리케이션의 속도(성능)를 향상시킨다.
- JCS는 데이터의 읽기(Read)에 최적화된 캐시관리방법을 제공한다.
- JCS에서 제공하는 효과적인 캐시관리를 통해 데이터베이스 처리로 발생하는 병목현상을 제거한다.

JCS의추가적인 기능

- Memory management
- Disk overflow (and defragmentation)
- Thread pool controls
- Element grouping
- Minimal dependencies
- Quick nested categorical removal
- Data expiration (idle time and max life)
- Extensible framework
- Fully configurable runtime parameters
- Region data separation and configuration
- Fine grained element configuration options
- Remote synchronization
- Remote store recovery
- Non-blocking "zombie" (balking facade) pattern
- Lateral distribution of elements via HTTP, TCP, or UDP
- UDP Discovery of other caches
- Element event handling
- Remote server chaining (or clustering) and failover
- Custom event logging hooks
- Custom event queue injection
- Custom object serializer injection
- Key pattern matching retrieval
- Network efficient multi-key retrieval

JCS 적용하기 (JDK 1.4)

필수라이브러리

- spring-modules-cache-0.8.jar
- jcs-1.2.6.5.jar
- concurrent-1.3.4.jar
- commons-logging-1.1.1.jar
- commons-lang-2.4.jar
- commons-collections-3.2.1.jar

적용순서

- 데이터를 캐시할 데이터 클래스를 java.io.Serializable 인터페이스를 implements하여 구현한다.
 - 데이터 클래스에 대한 데이터형에는 제약사항이 없다.
- JCS 기반으로 캐시관리를 수행할 빈을 applicationContext.xml에 등록한다.
 - 빈으로 등록해야 하는 대상은 다음과 같다.
 - cacheManager - org.springframework.cache.provider.jcs.JcsManagerFactoryBean
 - configLocation - classpath:jcs-config.properties 설정
 - cacheProviderFacade - org.springframework.cache.provider.jcs.JcsFacade
 - cachingInterceptor - org.springframework.cache.interceptor.caching.MethodMapCachingInterceptor
 - cachingModels - 캐시할 메소드 패턴을 등록하고 사용할 캐시명을 맵핑한다.
 - flushingInterceptor - org.springframework.cache.interceptor.flush.MethodMapFlushingInterceptor
 - flushingModels - 플러시할 메소드 패턴을 등록하고 대상 캐시명을 맵핑한다.

- ServiceAutoProxyCreator - org.springframework.aop.framework.autoproxy.BeanNameAutoProxyCreator
 - beanNames - 프록시에 적용할 클래스명 패턴을 등록한다.
 - interceptorNames - 프록시에서 사용할 인터셉터명을 등록한다.
- cacheManager 생성 시 사용할 jcs-config.properties를 설정한다.
- 웹 애플리케이션을 구동하면 JCS가 적용되어 설정에 따라 캐쉬관리하는 것을 확인할 수 있다.

1. java.io.Serializable 인터페이스를 implements한 SampleModel 개발하기

- 다음과 같이 SampleModel.java (클래스)를 구현한다.
- 예제1)

```
01. public class SampleModel implements Serializable {
02.
03.     private String id;
04.
05.     private String name;
06.
07.     public String getId() {
08.         return id;
09.     }
10.
11.     public void setRowStatus(String id) {
12.         this.id= id;
13.     }
14.
15.     public String getName() {
16.         return name;
17.     }
18.
19.     public void setName(String name) {
20.         this.name= name;
21.     }
```

- 예제2)

```
01. public class OZFxModel extends AbstractModel implements Serializable {
02.
03.     private String rowStatus;
04.
05.     private String masterset;
06.
07.     public String getRowStatus() {
08.         return rowStatus;
09.     }
10.
11.     public void setRowStatus(String rowStatus) {
12.         this.rowStatus = rowStatus;
13.     }
14.
15.     public String getMasterset() {
16.         return masterset;
17.     }
18.
19.     public void setMasterset(String masterset) {
20.         this.masterset = masterset;
21.     }
22.
23.     public String toString(String[] properties) throws IllegalAccessException,
24.         InvocationTargetException, NoSuchMethodException {
25.         return getProperty(properties);
26.     }
27. }
```

- 예제3)

```

01. public class OZFxmModelList extends AbstractModelList implements Serializable {
02.
03.     public void sort() {
04.         Comparator c = new Comparator() {
05.             public int compare(Object o1, Object o2) {
06.                 OZFxmModel a = (OZFxmModel) o1;
07.                 OZFxmModel b = (OZFxmModel) o2;
08.                 String sa = a.toString();
09.                 String sb = b.toString();
10.                 return sa.compareTo(sb);
11.             }
12.         };
13.         Collections.sort(getRows(), c);
14.     }
15.
16.     public void sort(final String[] properties) throws Throwable {
17.         Comparator c = new Comparator() {
18.             public int compare(Object o1, Object o2) {
19.                 OZFxmModel a = (OZFxmModel) o1;
20.                 OZFxmModel b = (OZFxmModel) o2;
21.                 String sa = null;
22.                 String sb = null;
23.                 try {
24.                     sa = a.getProperty(properties);
25.                     sb = b.getProperty(properties);
26.                 } catch (IllegalAccessException e) {
27.                     throw new InternalWrappedException(e);
28.                 } catch (InvocationTargetException e) {
29.                     throw new InternalWrappedException(e);
30.                 } catch (NoSuchMethodException e) {
31.                     throw new InternalWrappedException(e);
32.                 }
33.                 return sa.compareTo(sb);
34.             }
35.         };
36.         Collections.sort(getRows(), c);
37.     }
38. }

```

- 특별히 캐쉬할 데이터 모델의 데이터형에는 제약이 없다.
- 단지 Serializable 인터페이스를 구현한 클래스이면 JCS에서 캐쉬관리를 할 수 있다.

2. Spring Bean으로 applicationContext.xml에 등록하기

- applicationContext.xml에 다음과 같이 등록한다.

applicationContext.xml에 JCS 적용하기

```

01. <!--
02.             ===== Java Caching System CONFIGURATION
03.             =====
04.             -->
05.             <!--
06.             The created cache manager is a singleton instance of
07.             org.apache.jcs.engine.control.CompositeCacheManager
08.             -->
09.             <bean id="cacheManager"
10.                 class="org.springframework.cache.provider.jcs.JcsManagerFactoryBean">
11.                 <property name="configLocation" value="classpath:jcs-config.properties" />
12.             </bean>
13.
14.             <bean id="cacheProviderFacade" class="org.springframework.cache.provider.jcs.JcsFacade">
15.                 <property name="cacheManager" ref="cacheManager" />
16.             </bean>
17.
18.             <bean id="cachingInterceptor"
19.                 class=
20.                 "org.springframework.cache.interceptor.caching.MethodMapCachingInterceptor">
21.                 <property name="cacheProviderFacade" ref="cacheProviderFacade" />
22.                 <property name="cachingModels">
23.                     <props>
24.                         <prop key=
25.                         "com.woorifn.sample.biz.service.IProductService.select*"
26.                         >cacheName=woorifnCache</prop>
27.                     </props>
28.                 </property>
29.             </bean>
30.
31.             <bean id="flushingInterceptor"
32.                 class=
33.                 "org.springframework.cache.interceptor.flush.MethodMapFlushingInterceptor">
34.                 <property name="cacheProviderFacade" ref="cacheProviderFacade" />
35.                 <property name="flushingModels">
36.                     <props>
37.                         <prop key=
38.                         "com.woorifn.sample.biz.service.IProductService.save*"
39.                         >cacheName=woorifnCache</prop>
40.                         <prop key=
41.                         "com.woorifn.sample.biz.service.IProductService.update*"
42.                         >cacheName=woorifnCache</prop>
43.                         <prop key=
44.                         "com.woorifn.sample.biz.service.IProductService.insert*"
45.                         >cacheName=woorifnCache</prop>
46.                         <prop key=
47.                         "com.woorifn.sample.biz.service.IProductService.delete*"
48.                         >cacheName=woorifnCache</prop>
49.                     </props>
50.                 </property>
51.             </bean>
52.
53.             <bean id="ServiceAutoProxyCreator"
54.                 class="org.springframework.aop.framework.autoproxy.BeanNameAutoProxyCreator"
55.                 >
56.                 <property name="proxyTargetClass" value="false"></property>
57.                 <property name="beanNames">
58.                     <list>
59.                         <value>*Service</value>
60.                     </list>
61.                 </property>
62.                 <property name="interceptorNames">
63.                     <list>
64.                         <value>cachingInterceptor</value>
65.                         <value>flushingInterceptor</value>
66.                     </list>
67.                 </property>
68.             </bean>

```

3. jcs-config.properties에 JCS 기본 설정하기

- 위에서 설정한 cacheManager가 생성 시 jcs-config.properties를 로딩한다.
- 다음과 같이 jcs-config.properties 파일을 설정한다.

```

01. #####
02. # Java Cache System
03.
04. # DEFAULT CACHE REGION
05. jcs.default=DC
06. jcs.default.cacheattributes=org.apache.jcs.engine.CompositeCacheAttributes
07. jcs.default.cacheattributes.MaxObjects=1000
08. jcs.default.cacheattributes.MemoryCacheName=org.apache.jcs.engine.memory.lru.LRUMemoryCache
09. jcs.default.cacheattributes.UseMemoryShrinker=true
10. jcs.default.cacheattributes.MaxMemoryIdleTimeSeconds=3600
11. jcs.default.cacheattributes.ShrinkerIntervalSeconds=60
12. jcs.default.cacheattributes.MaxSpoolPerRun=500
13. jcs.default.elementattributes=org.apache.jcs.engine.ElementAttributes
14. jcs.default.elementattributes.IsEternal=false
15.
16. # PRE-DEFINED CACHE REGIONS
17. jcs.region.woorifnCache=DC
18. jcs.region.woorifnCache.cacheattributes=org.apache.jcs.engine.CompositeCacheAttributes
19. jcs.region.woorifnCache.cacheattributes.MaxObjects=1000
20.
jcs.region.woorifnCache.cacheattributes.MemoryCacheName=org.apache.jcs.engine.memory.lru.LRUMemoryCache
21. jcs.region.woorifnCache.cacheattributes.UseMemoryShrinker=true
22. jcs.region.woorifnCache.cacheattributes.MaxMemoryIdleTimeSeconds=3600
23. jcs.region.woorifnCache.cacheattributes.ShrinkerIntervalSeconds=60
24. jcs.region.woorifnCache.cacheattributes.MaxSpoolPerRun=500
25. jcs.region.woorifnCache.elementattributes=org.apache.jcs.engine.ElementAttributes
26. jcs.region.woorifnCache.elementattributes.IsEternal=false

```

JCS 적용하기 (JDK 1.5 이상)

설정방법

- 스프링빈 등록: applicationContext.xml

```

01....
02.<bean id="cachingAttributeSource" class=
"org.springframework.cache.annotations.AnnotationCachingAttributeSource" />
03.
04.<bean id="cachingInterceptor" class=
"org.springframework.cache.interceptor.caching.MetadataCachingInterceptor">
05.     <property name="cacheProviderFacade" ref="cacheProviderFacade" />
06.     <property name="cachingAttributeSource" ref="cachingAttributeSource" />
07.     <property name="cachingModels">
08.         <props>
09.             <prop key="dictCaching">cacheName=dictCache</prop>
10.         </props>
11.     </property>
12.</bean>
13.
14.<bean id="flushingAttributeSource" class=
"org.springframework.cache.annotations.AnnotationFlushingAttributeSource" />
15.
16.<bean id="flushingInterceptor" class=
"org.springframework.cache.interceptor.flush.MetadataCachingInterceptor">
17.     <property name="cacheProviderFacade" ref="cacheProviderFacade" />
18.     <property name="flushingAttributeSource" ref="flushingAttributeSource" />
19.     <property name="flushingModels">
20.         <props>
21.             <prop key="dictFlushing">cacheNames=dictCache</prop>
22.         </props>
23.     </property>
24.</bean>
25....

```

- 자바 클래스 구현방법 : Annotation 적용

```

1....
2.@Cacheable(modelId = "dictCaching")
3.public Dict load(Long id);
4.
5.@CacheFlush(modelId = "dictFlushing")
6.public void update(Dict dict);
7....

```

참고자료

참고문헌

- [캐쉬관리] Java Caching System : <http://java-source.net/open-source/cache-solutions/java-caching-system>
- [캐쉬관리] Open Source Cache 프레임워크 비교자료:
<http://javalandscape.blogspot.com/2009/03/intro-to-cachingcaching-algorithms-and.html>
- [캐쉬관리] cache4j 및 다른 cache 프레임워크 비교자료: <http://cache4j.sourceforge.net/ptest.htm>
- [캐쉬관리] JCS : <http://jakarta.apache.org/jcs/index.html>
- [캐쉬관리] JCS java-source.net: <http://java-source.net/open-source/cache-solutions/java-caching-system>
- [캐쉬관리] JCS와 Spring :
<http://gleichmann.wordpress.com/2008/04/29/pragmatic-caching-a-simple-cache-configuration-model-for-spring>
- [캐쉬관리] JCS와 Spring AOP : <http://forum.springsource.org/showthread.php?t=18665>
- [캐쉬관리] SpringModules와 JCS (AOP 적용) :
<https://springmodules.dev.java.net/docs/reference/0.9/html/cache.html>
- [캐쉬관리] SpringModules ehCache 적용사례 : <http://whiteship.me/1256>
- [캐쉬관리] JCS 적용 중국어사이트 : <http://littcai.javaeye.com/blog/309619>

첨부파일

There are currently no attachments on this page.

Copyright © 2008 Daewoo Information Systems Co., Ltd.

08. 보안처리

- 1.SpringSecurity 가이드
- 2. 세션 중복(중복 로그인) 방지 처리

1.SpringSecurity 가이드

목차

- 개요
 - acegi란?
 - Spring Security(acegi)의 기능
- 기능 상세
 - 웹어플리케이션이 실행될 때
 - 로그인 시
 - 세션이 끊기면 하는 일
- 참고자료
- 첨부파일

문서정보

- 제목: 대우버스 Acegi 가이드
- 최초작성자: 김민아
- 최초작성일: 2010/04/02
- 이문서는 사용자에게 **0**번 보여졌습니다.

개요

본 문서는 현재 대우버스 해외보증수리에 적용된 Acegi (시큐리티) 에 대한 가이드이다.

acegi란?

Ben Alex

알파벳 홀수 1, 3, 5, 7, 9를 따서 만든 이름

Acegi는 스프링 프레임워크의 공인 서브 프로젝트이며 <http://acegisecurity.org> 에서 운영되고 있는 오픈소스 프로젝트

Spring Security(acegi)의 기능

- 로그인 기능 제공
- password를 통한 user 인증
- 각 개인이 허락된 리소스에 대한 check과 리소스 redirect

- 로그인 실패 시 Denied page 보여줌 인증 성공한 유저에 대해 유저 클라이언트에 보한 쿠키를 세팅해줌으로서 다음 인증 때 로그인 없이 열람가능
- 로그아웃 시 유저 클라이언트의 보안세션을 삭제 유저들의 security 등급 및 정책을 DB에서 관리할 수 있음.

기능 상세

웹어플리케이션이 실행될 때

- 어플리케이션이 실행되면 web.xml 에서 filter 로 설정한 Acegi Filter Chain Proxy 가 읽히면서 ApplicationContext-Security 파일이 실행될 것이다.
- ApplicationContext-Security 파일이 실행되면서 secureUrlDao 빈에 작성한 쿼리가 실행이 되어 URL 과 해당 URL 이 접근 가능한 권한들을 가져와 어플리케이션이 실행될 때까지 그 정보를 가지고 있을 것이다.
- 현재는 ROLE 테이블 안에 있는 모든 ROLE_NAME 에 대해 http://localhost:8080/dows/**/!**/*.do 자원에 접근 할 수 있는 권한을 주는 것이다.
- 만약에 로그인 하지않았다면 ROLE_ANONYMOUS_USER 라는 권한(ANONYMOUS 권한)이 셋팅되며 해당 어플리케이션의 자원을 임의로 호출하면(예 컨트롤 임의 호출) 접근이 되지않으며 설정한 에러 페이지로 갈 것이다.

코드 예)

```
01.<!-- url , url -->
02.<bean id="secureUrlDao"
03.  class="business.jcfcom.acegisecurity.intercept.web.SecureUrlJdbcDao">
04.  <property name="dataSource">
05.    <ref bean="dataSource" />
06.  </property>
07.  <property name="secureUrlQuery">
08.    <!-- -->
09.    <value>
10.      SELECT '/*/*/*/*.*.do' url, ROLE_NAME authority FROM ROLE
11.    </value>
12.  </property>
13.</bean>
```

코드 예)

- ANONYMOUS 권한 지정

ANONYMOUS 권한 일경우 anonymousUser라는 유저이름, ROLE_ANONYMOUS_USER 라는 권한으로 세팅된다.

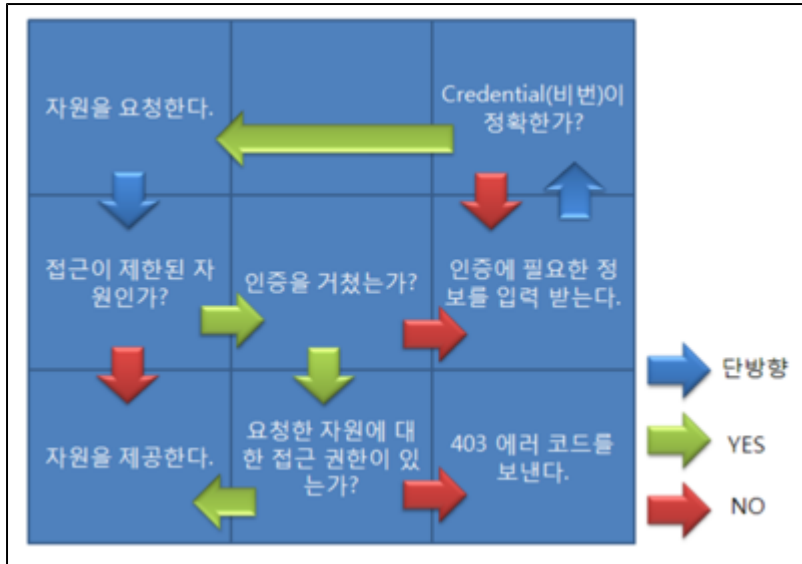
```
01.<bean id="anonymousProcessingFilter"
02.  class="org.acegisecurity.providers.anonymous.AnonymousProcessingFilter">
03.  <property name="key">
04.    <value>foobar</value>
05.  </property>
06.  <property name="userAttribute">
07.    <value>anonymousUser,ROLE_ANONYMOUS_USER</value>
08.  </property>
09.</bean>
```

- 권한이 없을 경우 가는 페이지 지정

```
01.      <!--exceptionTranslationFilter -->
02.<bean id="exceptionTranslationFilter">
03.  <property name="authenticationEntryPoint">
04.    <bean
05.      class="org.acegisecurity.ui.webapp.AuthenticationProcessingFilterEntryPoint">
06.        <property name="loginFormUrl" value="/commonJSP/loginError.jsp" /> //
07.        <property name="serverSideRedirect" value="true" />
08.      </bean>
09.    </property>
10.  ... </bean>
```

로그인 시

- 로그인 프로세스는 밑의 그림과 같다.



로그인 요청 url은 login.jsp의 form의 밑의 action 이다.

```
1.action="/j_acegi_security_check"
```

위와 같은 요청이 오면 밑의 코드에 따라 성공하면 defaultTargetUrl 인 /commonJSP/index_start.jsp, 실패하면 authenticationFailureUrl 인 /commonJSP/loginFailure.jsp 로 간다.

```

1.<bean id="authenticationProcessingFilter"
2.  class="org.acegisecurity.ui.webapp.AuthenticationProcessingFilter">
3.  <property name="authenticationManager" ref="authenticationManager" />
4.  <property name="authenticationFailureUrl" value="/commonJSP/loginFailure.jsp" />
5.  <property name="defaultTargetUrl" value="/commonJSP/index_start.jsp" />
6.  <property name="filterProcessesUrl" value="/j_acegi_security_check" />
7.</bean>

```

로그인 ip 와 pw를 확인하는 부분은 jdbcDaoImpl 이다. usersByUsernameQuery 는 사용자와 id 와 pw가 맞는지 , account_enabled 는 사용가능 여부이다.

authoritiesByUsernameQuery 는 사용자에게 해당되는 권한을 가져오는 부분이다.

```

01.<bean id="jdbcDaoImpl" class="org.acegisecurity.userdetails.jdbc.JdbcDaoImpl">
02.<property name="dataSource">
03.<ref bean="dataSource" />
04.</property>
05.<property name="usersByUsernameQuery">
06.<value>SELECT username, password, account_enabled FROM JCF_USER WHERE
07.username = ?</value>
08.</property>
09.<property name="authoritiesByUsernameQuery">
10.<value>SELECT user_role.username, role.role_name from user_role,role
11.where role.role_id=user_role.role_id and user_role.username = ? </value>
12.</property>
13.</bean>

```

세션이 끊기면 하는 일

세션이 끊기면 권한이 ROLE_ANONYMOUS_USER 로 바뀌며 위에서 ROLE_ANONYMOUS_USER 이 어플리케이션의 자원을 호출했을 경우 가는 페이지(권한이 없을 경우)인 loginError.jsp로 간다.

- 권한이 없을 경우 가는 페이지 지정

```

01.      <!--exceptionTranslationFilter -->
02.<bean id="exceptionTranslationFilter">
03.  <property name="authenticationEntryPoint">
04.    <bean
05.      class="org.acegisecurity.ui.webapp.AuthenticationProcessingFilterEntryPoint">
06.        <property name="loginFormUrl" value="/commonJSP/loginError.jsp" />  //
07.        <property name="serverSideRedirect" value="true" />
08.      </bean>
09.    </property>
10.    ... </bean>

```

참고자료



참고문헌

*http://wiki.dev.daewoobrenic.co.kr/mediawiki/index.php/Spring_security

첨부파일

이름	크기	생성자	Creation Date	댓글
Acegi.png	47 kB	김민아	9월 29, 2010 17:51	

Copyright © 2009 Daewoo Information Systems Co., Ltd.

2. 세션 중복(중복 로그인) 방지 처리

문서정보

- 제목: 세션 중복(중복 로그인) 방지 처리
- 최초작성자: 고경철
- 최초작성일:
- 수정자:
- 수정일:
- 첨부파일:
- 이문서는 사용자에게 **42**번 보여졌습니다.



목차

- 동시 세션 처리
 - 동시 세션 처리를 위한 준비
 - 상세 설명
- 참고자료
- 첨부파일

동시 세션 처리

동시 세션 처리를 위한 준비

- 먼저 web.xml에 아래의 리스너를 등록한다.

web.xml

```

1.<listener>
2.    <listener-class
>org.springframework.security.ui.session.HttpSessionEventPublisher</listener-class>
3.</listener>

```

- 스프링 시큐리티 사용시 동시 세션 즉 중복 로그인 방지를 위해 아래와 같이 concurrent-session-control 설정을 추가하면 간단히 동시 세션에 대한 컨트롤을 할 수 있다.

applicationContext-security.xml

```

01.....
02.<sec:http auto-config='true' access-denied-page="/404.jsp">
03.    <sec:form-login login-page="/login.jsp"
04.        authentication-failure-url="/404.jsp" default-target-url="/index.jsp"
05.        always-use-default-target='true' />
06.    <sec:logout logout-success-url="/login.jsp" />
07.{quote}
08.    <sec:concurrent-session-control max-sessions="1" exception-if-maximum-exceeded="false"
09.    expired-url="/sessionExpired.jsp" /> ---> (1)
10.</sec:http>
11.....

```

- (1)의 라인을 추가하면 동시 세션 컨트롤을 할 수 있다.

상세 설명

- max-sessions : 최대 허용 세션 수
- exception-if-maximum-exceeded : true ⇒ 최대 세션 수 초과할 경우 Exception 발생 , false ⇒ 최대 세션 수 초과할 경우 강제 로그아웃



- 기존 로그인한 사람을 로그아웃 시키려면 exception-if-maximum-exceeded 속성 값을 false로 처리
- 최대 허용 세션 수를 충족한 상태에서 기존 로그인한 사람(들)을 로그아웃 하시키지 않고 추가로 로그인 시도를 막으려면 exception-if-maximum-exceeded 속성 값을 true 로 처리

- expired-url : 최대 허용 세션 수를 초과한 상태에서 로그인 시도시 던져주는 url

expired-url에 설정된 jsp에서 client 단으로 메시지를 던져주는 코드(/sessionExpired.jsp)

```
1.....
2.<%
3.        try{
4.            OutChannel.error("-1", "    .");
5.        }
```



exception-if-maximum-exceeded = true 설정시

- 최대 세션 수 초과할 경우, 같은 id의 로그인은 불가능 하고 아래와 같은 Exception이 발생

```
1.DEBUG
  (org.springframework.security.ui.webapp.AuthenticationProcessingFilter:412) -
  Authentication request failed:
2.org.springframework.security.concurrent.ConcurrentLoginException: Maximum
  sessions of 1 for this principal exceeded
```

참고자료



참고문헌

- <http://whiteship.tistory.com/1631>
- http://www.egovframe.go.kr/wiki/doku.php?id=egovframework:rte:fdl:server_security:authentication
- <http://forum.kslug.org/viewtopic.php?f=6&t=402&p=1920#p1920>

첨부파일

There are currently no attachments on this page.

Copyright © 2008 Daewoo Information Systems Co., Ltd.

09. 엑셀처리

- 01. 엑셀파일생성(jxl 사용)

01. 엑셀파일생성(jxl 사용)



목차

- 개요
- 설명
- 소스설명
- 첨부파일
- 조회한 데이터를 엑셀파일로 생성하는 방법에 대한 임시가이드임

문서정보

- 제목: 엑셀파일 생성
- 최초작성자: 2010/03/02
- 최초작성일:
- 이문서는 사용자에게 **23**번 보여졌습니다.

개요

조회한 데이터를 엑셀파일로 생성하는 방법에 대한 임시가이드임

설명

소스설명

ExcelUtil.java

```

01.package com.system.util;
02.
03....
04.public class ExcelUtil {
05.
06.
07.     private static final Logger logger = LoggerFactory.getLogger(ExcelUtil.class);
08.
09.     //1.   ;   ,
10.
11.     public void makeExcelFileFromList(String filePath, List cellInfoList, List dataList )
12.     {
13.
14.         OutputStream outputStream = null;
15.
16.         try {
17.
18.             logger.info( filePath+ "make file" );
19.
20.             //2.
21.             outputStream = new FileOutputStream(filePath);
22.         } catch (FileNotFoundException e) {
23.             // TODO Auto-generated catch block
24.             e.printStackTrace();
25.         }
26.
27.         try {
28.
29.             logger.info("create "+filePath+ " + file" );
30.             WritableWorkbook workbook = Workbook.createWorkbook(outputStream);
31.
32.             //3.
33.             WritableSheet sheet = workbook.createSheet("Sheet1", 0);
34.
35.             WritableCellFormat titleFormat = new WritableCellFormat();
36.             WritableCellFormat dataFormat = new WritableCellFormat();
37.
38.             //4.
39.             titleFormat.setAlignment(Alignment.CENTRE);
40.             titleFormat.setVerticalAlignment(VerticalAlignment.CENTRE);
41.             titleFormat.setBorder(Border.ALL, BorderLineStyle.THIN);
42.             titleFormat.setBackground(Colour.GRAY_50);
43.
44.             FontRecord fr = new FontRecord("", 11, BoldStyle.BOLD.getValue(),
45.                 false, 0, Colour.WHITE.getValue(), 0) {
46.             };
47.             titleFormat.setFont(fr);
48.
49.             //5.
50.             dataFormat.setAlignment(Alignment.CENTRE);
51.             dataFormat.setVerticalAlignment(VerticalAlignment.CENTRE);
52.             dataFormat.setBorder(Border.ALL, BorderLineStyle.THIN);
53.
54.             Label labels = null;
55.
56.             //6.
57.             for (int i = 0; i < cellInfoList.size(); i++) {
58.                 String cellInfo = (String)cellInfoList.get(i);
59.                 logger.debug(i + "0 input"+ cellInfo );
60.                 labels = new Label(i, 0, cellInfo, titleFormat);
61.                 sheet.addCell(labels);
62.             }
63.         }
64.     }
65. }

```



```

57.                }
58.
59.
60.//7.
61.
62.                for (int i = 0; i < dataList.size(); i++) {
63.                    HashMap data = (HashMap) dataList.get(i);
64.                    Iterator dataSetNameIter = ((HashMap) dataList.get(0
65.                    int j = 0;
66.                    while(dataSetNameIter.hasNext()) {
67.                        String dataSetName = (String)
68.                        labels = new Label(j, i+1,
69.                        logger.debug(j + "i+1 input"+
70.                        sheet.addCell(labels);
71.                        j= j+1;
72.
73.                    }
74.
75.                }
76.
77.                workbook.write();
78.                workbook.close();
79.
80.            } catch (IOException e) {
81.                // TODO Auto-generated catch block
82.                e.printStackTrace();
83.            } catch (WriteException e) {
84.                // TODO Auto-generated catch block
85.                e.printStackTrace();
86.            }
87.
88.
89.
90.        }
91.    }

```

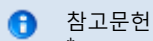
테스트케이스

```

01. @Test
02. public void testname() throws Exception {
03.//1.
04.    String fileName = "test.xls";
05.    String filePath= propertyService.getString("temp.dir");
06.    String fileFullPath = filePath+File.separator+fileName;
07.//2 dao    (sqlmap    resultClass="java.util.LinkedHashMap"    )
08.    List dataList = dao.selectList("code.findCodeByCondition", null);
09.
10.//3 List
11.    List cellInfoList = new ArrayList();
12.    cellInfoList.add("");
13.    cellInfoList.add("");
14.    cellInfoList.add("");
15.    cellInfoList.add("");
16.    cellInfoList.add("");
17.    cellInfoList.add("");
18.    cellInfoList.add("");
19.    cellInfoList.add("");
20.    cellInfoList.add("");
21.    cellInfoList.add("");
22.
23.//4. excelUtil
24.    ExcelUtil excelUtil= new ExcelUtil();
25.    excelUtil.makeExcelFileFromList(fileFullPath, cellInfoList, dataList);

```



참고자료



참고문헌

*

첨부파일

이름	크기	생성자	Creation Date	댓글
 예전국가세정엑셀.zip	3 kB	김민아	3월 02, 2010 15:26	예전국가세정엑셀파일생성소스
 엑셀파일다운로드_개선후.zip	3 kB	김민아	3월 08, 2010 10:53	

Copyright © 2009 Daewoo Information Systems Co., Ltd.

10. property 서비스



목차

- 개요
- 적용방법
 - applicationContext 파일에 propertyService 등록하기
 - 서비스에서 propertyService 사용하기
- 참고자료

문서정보

- 제목:property 서비스
- 최초작성자: 김민아
- 최초작성일: 2009/11/26
- 이문서는 사용자에게 8번 보여졌습니다.

개요

property 서비스란 설정파일이나 외부파일에 property를 정의하여 자바 코드에서 필요 시 정의된 property를 가져와 쓰는 것을 말한다.(예 파일업로드 시 파일저장 디렉토리)

현재 JCF에서는 spring의 bean으로 propertyService를 등록하여 사용한다.

적용방법

applicationContext 파일에 propertyService 등록하기

CappllicationContext.xml

```

01.<bean name="propertyService" class="jcf.util.properties.PropertyServiceImpl"
02.    destroy-method="destroy">
03.    <property name="extFileName">
04.        <set>
05.            <map>
06.                <entry key="encoding" value="UTF-8"/>
07.                <entry key="filename" value="classpath:/app.properties"/>
08.            </map>
09.        </set>
10.    </property>
11.</bean>

```

위와 같이 jcf-uti 라이브러리에 있는 jcf.util.properties.PropertyServiceImpl 에 대해 propertyService 라는 이름으로 bean으 등록하고 property 이름은 extFileName, 그 안에 key는 filename , value에는 property를 지정한 외부파일을 지정한다.

app.properties

```

1.upload.real.dir=C:\jcf
2.upload.temp.dir=C:\temp

```

서비스에서 propertyService 사용하기

```
01. @Controller
02. public class FileController {
03.
04.     @Resource(name="propertyService")
05.     private PropertyService propertyService;
06.
07.
08.     @RequestMapping
09.     public void upload(HttpServletRequest request,
10.         HttpServletResponse response) throws Exception{
11.         //
12.         String tempDir= propertyService.getString("upload.temp.dir");
13.     }
```

1. 클래스의 전역변수로 PropertyService 를 private로 선언하고 @Resource (name="propertyService") 라는 어노테이션을 설정한다.
2. 메소드 안에서 propertyService.getString 메소드로 app.properties 파일에 선언한 프로퍼티 value를 가져온다.

참고자료



참고문헌

*<http://www.egovframe.org/wiki/doku.php?id=egovframework:rte:fdl:property>