LET'S KO Project

실제 프로젝트 시작전까지 비지니스 로직를 제외한 개발환경 구축 및 간단한 CRUD Sample을 만들어 보는 것이다.

차례

- 마치며

개발환경 구성 및 설치

개발환경 구성

IDE 개발툴은 개발자의 생산성과 편의성을 크게 증대 시킨다고 생각한다. 사실 이제 IDE을 쓰지 않고 자바 개발을 한다는 생각은 하기 힘들다.

자바 IDE는 크게 오픈소스인 Eclipse, NetBeans와 상용 IDE인 IntelliJ IDEA(이하 IntelliJ)가 있다.

이번 프로젝트는 IntelliJ를 사용했다. IntelliJ는 상용 IDE로써 다양한 프레임워크를 신속히 지원하고 많은 부분이 Bundle로 제공하고 있어

오픈소스인 Eclipse보다 설정이 적다. (그렇다고 설정을 안하고 쓸수 있다는 것은 아니다.) 플렛폼은 Windows, Linux, Mac을 지원한다.

여기서는 Windows를 선택했다. Windows로 간 이유중 하나는 자바 버전이다.

지금은 오라클에서 맥용도 지원을 하지만 예전 버전(1.6미만 버전)은 여전히 구하기 쉽지 않다.

그리고 SQLServer를 사용한것도 Windows쪽을 선택하게 한 이유이다.

```
개발환경
    ├─ IDE: IntelliJ IDEA13
    ├─ Java: 1.7
    ├─ 형상관리: Git
    ├─ 빌드: Maven
    ├─ Web Application Server: Tomcat7.0
    └─ DB: SQLServer 2008
```

```
Backend 기술 스택
    ├─ SpringFramework
    ├─ SpringSecurity
    ├─ Hibernate
    └─ Tiles
```

```
Frontend 기술 스택
    ├─ jQuery
    ├─ jQuery-ui
    ├─ requirejs
    ├─ jqgrid
    └─ bootstrap
```

설치에 앞서서 개인적으로 프로젝트와 관련된 JDK, IDE, Maven, Web Application Server등은 아래와 같이 한곳에 모아둔다.

```
C:\JavaDE
    ├─ java
    │    ├─ jdk1.6.0_45
    │    ├─ jdk1.7.0_45
    │    └─ jdk1.7.0_45(x64)
    ├─ JetBrains
    │    └─ IntelliJ IDEA 13.0
    ├─ maven
    │    └─ apache-maven-3.1.1
    └─ tomcat
         ├─ apache-tomcat-6.0.37
         └─ apache-tomcat-7.0.42
```

추가 적으로 설명의 편의성을 위해서 각각의 위치를 먼저 기술한다.

```
1.  [PROJECT_HOME]: LETS-KO (프로젝트 홈 폴더)
2.  [MODULE_HOME]: LETS-KO/First (모듈 홈 폴더)
3.  [WEB_HOME]: LETS-KO/First/web (웹 소스 홈 폴더)
4.  [WEB_CONFIG_HOME]: LETS-KO/First/web/WEB-INF/config (웹 설정 폴더)
5.  [JAVA_SRC_HOME]: LETS-KO/First/src/main/java (자바 소스 홈 폴더)
6.  [CONTEXT_CONFIG_HOME]: LETS-KO/First/src/main/java/resources/config (Cont
ext 설정 홈 폴더)
8.  [TOMCAT_HOME]: ['Tomcat' 설치위치]
9.  [MAVEN_HOME]: ['Maven' 설치위치]
10. [JDK_HOME]: ['JDK' 설치위치]
```

IntelliJ IDEA13 설치

IntelliJ IDEA13 Windows 용을 다운로드한다.

IntelliJ 설정은 기본설정, 자바설정을 참고한다.

Git 설치

Git은 리눅스 커널을 개발 관리하기 위해서 리누스 토발즈가 만든 형상관리 툴이다.

기본적으로 Linux계열에서 동작하며 Windows에서 사용하려면 msysgit를 사용하면 된다.

현재 최신 버전은 1.8.4가 최신 버전이다. msysgit설치 후 IntelliJ와 연동 설정을 한다.

IntelliJ와 연동은 IntelliJ-Git 설정을 참고한다.

Maven 설치

의존성 관리만으로도 Maven을 설치할만한 가치는 충분이 크다. Maven 현재 최신 버전은 3.1.1 이다.

Maven 3.1.1 (Binary zip)을 다운로드 한후, 적당한 폴더에 압축해제한다.

`[IntelliJ Menu]` -> `File` -> `settings...` -> `Maven` -> `Maven Home directory [선택]` 압축해제한 곳을 선택한다.

Web Application Server

Web Application Server(이하 WAS)는 여러 밴더가 존재한다.

여러가지 현실적 이유때문에 개발환경과 배포환경의 WAS가 일치하지 않는 경우가 있다.

사실 WAS는 표준 스펙이 존재하기 때문에 밴더별로 스펙만 맞춰주면 동일한 결과를 보장해야 한다.

Tomcat 7.0 Version은 여기를 참고한다.

개발은 일반적으로 많이 사용하는 Tomcat을 이용한다. Tomcat 7.0 다운로드

Database 설치

이 프로젝트에서는 Hibernate를 사용하기 때문에 사실 특정 DataBase를 설치할 필요는 없다. 취양에 맞춰 설치하자.

프로젝트 생성 및 설정

Java Project 생성

이번 프로젝트는 새로운 프로젝트를 생성하기 보다는 GitHub에서 Checkout해서 생성한다.

1. 처음 IntelliJ를 실행하면은 Dashboard가 나타난다. `Check out from version control` -> `GitHub [선택]` 한다.
2. 다음 단계로 Clone Repository 다이알로그가 나타나는데 `Git Repository URL` -> `https://github.com/daejoon/LETS-KO.git [입력]` -> `Parent Directory` -> `C:\Users\{계정이름}\IdeaProjects [입력]` -> `Directory Name` -> `LET-KO [입력]` -> `Clone [클릭]` 한다.
3. Import Project 다이알로그 창이 나타나면 `Create project from existing sources` -> `Next [클릭]` -> `Next [클릭]` -> `Unmark All [클릭]` -> `Finish [클릭]` 한다.
4. `File` -> `Import Module` -> `First.iml [선택]` -> `OK [클릭]` 한다.

Run/Debug Configurations 설정

1. `Run` -> `Edit Configurations [선택]` 한다.

2. Run/Debug Configurations 다이알로그가 나타난다. `+` -> `Tomcat Server` -> `Local [클릭]` 한다.

3. `Name` -> `First - Tomcat 7.0 [입력]` -> `Application Server` -> `Tomcat 7.0 [선택]` 한다.

4. `Fix` -> `First:war exploded [선택]` -> `OK [클릭]` 한다.

5. `File` -> `Project Structure...` -> `Project Settings` -> `Modules` -> `First [선택]` -> `Dependencies` -> `+ [클릭]` -> `Library...` -> `Application Server Libraries` -> `Tomcat 7.0 [선택]` -> `Add Selected [클릭]` 하여 WAS에 의존적인 라이브러리를 링크한다.

Git 설정

`File` -> `Settings...` -> `Version Control` -> `Ignored Files` -> `+ [클릭]` -> `Ignore all files under` -> `... [선택]` -> `.idea [선택]` -> `Ok [클릭]`

Tomcat 설정

UTF-8 설정

`[TOMCAT_HOME]/conf/server.xml` 에 useBodyEncodingForURI="true", URIEncoding="UTF-8"을 추가한다.

```
<Connector port="8080" protocol="HTTP/1.1"
           connectionTimeout="20000"
           redirectPort="8443"
           useBodyEncodingForURI="true"
           URIEncoding="UTF-8" />
```

Jndi 설정

`[TOMCAT_HOME]/conf/context.xml` 에 Resource 엘리먼트를 추가한다.

```
<Resource name="[jndi이름: 예)jdbc/letsko_ds01]"
          auth="Container"
          type="javax.sql.DataSource"
          username="[DB 로그인 아이디]"
          password="[DB 로그인 비밀번호]"
          driverClassName="[DB 드라이버클래스네임: 예)com.microsoft.sqlserver.jdbc.SQLServerDriver]"
          url="[DB Url: 예)jdbc:sqlserver://localhost:1433;databaseName=lets_ko;integratedSecurity=false;]"
```

```
            maxActive="10"
            maxIdle="5"/>
```

`[CONTEXT_CONFIG_HOME]/default-config.xml` 에 Jndi를 설정한다.

```
<dataSources>
    <ds01>
        <jndiName>jdbc/letsko_ds01</jndiName>
        <resourceRef>true</resourceRef>
    </ds01>
</dataSources>
```

`[CONTEXT_CONFIG_HOME]/spring/context-datasource.xml` 의 Jndi를 설정한다.

```
<jee:jndi-lookup id="dataSource" jndi-name="${dataSources.ds02.jndiName}" />
```

Web Application 설정

UTF-8 설정

이 프로젝트의 기본 인코딩은 UTF-8이다.

`[WEB_HOME]/WEB-INF/web.xml` 에 Spring CharacterEncodingFilter를 이용해서 UTF-8을 설정한다.

```
<!-- Encoding Filter -->
<filter>
    <filter-name>encodingFilter</filter-name>
    <filter-class>org.springframework.web.filter.CharacterEncodingFilter</fil
ter-class>
    <init-param>
        <param-name>encoding</param-name>
        <param-value>UTF-8</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>encodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

Log4j 설정

Log4j의 설정 파일을 읽을수 있게 web.xml에 추가한다.

```
<!-- log4j ContextLoader -->
<context-param>
    <param-name>log4jConfigLocation</param-name>
```

```
    <param-value>classpath:config/log4j/log4j.xml</param-value>
</context-param>
<listener>
    <listener-class>org.springframework.web.util.Log4jConfigListener</listene
r-class>
</listener>
```

Log4j 설정 파일은 `[CONTEXT_CONFIG_HOME]/log4j/log4j.xml` 을 참고한다.

SpringFramework

SpringFramework 설정

SpringFramework(이하 Spring) 설정은 Root Context 설정과, Servlet Context 설정으로 나뉜다.

Root Context 설정은 Spring 전반적인 설정이고, Servlet Context 설정은 웹과 관련된 설정이다.

Root Context 설정은 Servlet Context로 상속된다.

Servlet Context 설정은 `[SERVLET_CONFIG_HOME]/springmvc` 폴더 아래 파일을 참고한다.

Root Context 설정은 `[CONTEXT_CONFIG_HOME]/spring` 폴더 아래 파일을 참고한다.

`[WEB_HOME]/WEB-INF/web.xml` 의 Root Context 위치 설정

```
<!-- SpringFramework ContextLoader -->
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>classpath*:config/spring/context-*.xml</param-value>
</context-param>
<listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</li
stener-class>
</listener>
```

`[WEB_HOME]/WEB-INF/web.xml` 의 Servlet Context 위치 설정

```
<!-- Servlet Dispatcher -->
<servlet>
    <servlet-name>action</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet
-class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>
            /WEB-INF/config/springmvc/servlet-*.xml
        </param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
```

```
</servlet>
<servlet-mapping>
    <servlet-name>action</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>
```

user-config.xml, default-config.xml 설정

Spring의 org.springframework.beans.factory.config.PropertyPlaceholderConfigurer을 사용하면 properties를 스프링 설정에 사용할수 있다.

여기에 org.springmodules을 추가하면 properties를 사용하지 않고 xml로 대체할 수 있다.

이 프로젝트는 org.springmodules를 사용했다.

`[CONTEXT_CONFIG_HOME]/spring/context-common.xml` 에 CompositeConfiguration을 이용하여 xml 설정파일을 여러개 올릴수 있다. 두 설정 파일 중에 같은 엘리먼트가 존재하면 먼저 올린 user-config.xml 파일의 엘리먼트가 우선한다.

```
<!-- 환경 설정 xml 파일을 로딩한다. -->
<bean id="configuration" class="org.apache.commons.configuration.CompositeConfiguration">
    <constructor-arg>
        <list>
            <bean class="org.apache.commons.configuration.XMLConfiguration">
                <constructor-arg type="java.lang.String">
                    <value>config/user-config.xml</value>
                </constructor-arg>
            </bean>
            <bean class="org.apache.commons.configuration.XMLConfiguration">
                <constructor-arg type="java.lang.String">
                    <value>config/default-config.xml</value>
                </constructor-arg>
            </bean>
        </list>
    </constructor-arg>
</bean>
```

이렇게 작성한 이유는 개발시에 공통 부분과 개개인 설정이 분리됨으로 해서 개발의 편의성이 증대되고 실제 배포시에는 default-config.xml만 배포함으로써

배포 환경과 개발 환경을 분리하여 관리할수 있기 때문이다.

따라서 여기 GitHub에도 user-config.xml 파일을 업로드 하지 않았다. 차후에 Clone한후 user-config.xml 파일을 추가하면 된다.

사실 user-config.xml 파일과 default-config.xml 파일의 엘리먼트는 일치하지 않아도 상관없으니 경험상 일치하는게 실수 방지에 좋고 편하다.

되도록이면 두 파일의 엘리먼트를 일치시키고 개별적 적용 엘리먼트만 user-config.xml 파일에서 수정하자.

`[CONTEXT_CONFIG_HOME]/spring/context-common.xml` 에 CompositeConfiguration을 이용하여 설정 xml 파일들을 PropertyPlaceholderConfigurer에 연결시켜준다.

```
<bean class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
    <property name="properties">
        <bean class="org.springmodules.commons.configuration.CommonsConfigurationFactoryBean">
            <property name="configurations">
                <list>
                    <ref bean="configuration" />
                </list>
            </property>
        </bean>
    </property>
</bean>
```

이 설정을 함으로써 context-*.xml 파일들에서 properties를 사용할수 있다.

또한 이렇게 사용한 user-config.xml과 default-config.xml 파일은 dd2.com.util.CofingUtil을 통해서 런타임시에 접근할수 있다.

사용 방법은 xml에서 properties를 사용하듯이 dot 접근방법을 쓴다.

환경설정 xml 파일 작성 예

```
<!-- sampel-config.xml -->
<?xml version="1.0" encoding="UTF-8"?>
<config>
    <!-- mode -->
    <mode>
        <!-- mode.type : debug / release -->
        <type>debug</type>
    </mode>
</config>
```

스프링 설정 파일에서 접근

```
<bean id="testSampleBean" class="org.testSample.TestBean">
    <property name="test" value="${mode.type}" />
</bean>
```

자바코드에서 접근

```
String type = ConfigUtil.getString("mode.type");
```

위의 두 예제에서 봤듯이 최상위 config 엘리먼트는 생략가능하다.

SpringSecurity

SpringSecurity 설정

SpringSecurity를 사용함으로써 많은 부분의 권한관리를 줄일수 있다.

`[WEB_HOME]/WEB-INF/web.xml` 의 DelegatingFilterProxy filter 설정해야 SpringSecurity가 동작한다.

```
<!-- Spring Security Filter -->
<filter>
    <filter-name>springSecurityFilterChain</filter-name>
    <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
</filter>
<filter-mapping>
    <filter-name>springSecurityFilterChain</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

SpringSecurity의 세부 설정은 `[CONTEXT_CONFIG_HOME]/spring/context-security.xml` 을 확인한다.

Hibernate

Hibernate 설정

`[CONTEXT_CONFIG_HOME]/spring/context-datasource.xml` 의 LocalSessionFactoryBean Bean을 설정한다. LocalSessionFactoryBean을 사용하면 Hibernate Annotation을 사용할수 있다.

```
<!-- hibernate sessionFactory -->
<bean id="sessionFactory" class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">
    <property name="dataSource" ref="dataSource" />
    <property name="packagesToScan" value="dd2.local" />
    <property name="hibernateProperties">
        <props>
            <prop key="hibernate.dialect">${hibernate.dialect}</prop>
            <prop key="hibernate.show_sql">${hibernate.show_sql}</prop>
            <prop key="hibernate.format_sql">${hibernate.format_sql}</prop>
            <prop key="hibernate.connection.useUnicode">${hibernate.connection.useUnicode}</prop>
            <prop key="hibernate.connection.characterEncoding">${hibernate.connection.characterEncoding}</prop>
            <prop key="hibernate.connection.charSet">${hibernate.connection.charSet}</prop>
            <prop key="hibernate.hbm2ddl.import_files">${hibernate.hbm2ddl.import_files}</prop>
```

```
            <prop key="hibernate.hbm2ddl.auto">${hibernate.hbm2ddl.auto}</pro
p>
        </props>
    </property>
</bean>
```

Hibernate 설정은 default-config.xml, user-config.xml에서 설정값을 가져온다.

hibernate.dialect는 SQLServer를 사용하기 때문에 `org.hibernate.dialect.SQLServerDialect` 로 설정 한다. hibernate.showsql, hibernate.formatsql을 `true` 로 설정하여 개발시에 console 화면에 정렬해서 보여준다.

```
<!-- hibernate -->
<hibernate>
    <dialect>org.hibernate.dialect.SQLServerDialect</dialect>
    <show_sql>true</show_sql>
    <format_sql>true</format_sql>
    <connection>
        <useUnicode>true</useUnicode>
        <characterEncoding>UTF-8</characterEncoding>
        <charSet>UTF-8</charSet>
    </connection>
    <hbm2ddl>
        <import_files>config/hibernate/sql/initial_data.sql</import_files>
        <auto>validate</auto>
    </hbm2ddl>
</hibernate>
```

hibernate.hbm2ddl.auto의 값은 create, create-drop, update, validate 값이 올수 있다.

```
create: Entity를 기초로 테이블을 생성한다.
create-drop: Entity를 기초로 테이블을 생성하고 프로그램 종료시에 삭제한다. 여기서는 WAS 종료
시에 삭제된다.
update: Entity와 테이블을 비교하여 변경 부분을 적용한다. (권한에 따라 안될수도 있다.)
validate: Entity와 테이블을 비교만 한다.
```

대부분 프로젝트를 만들때 데이터베이스 테이블을 미리 구성하고 그다음에 Entity를 생성한다. 따라서 이 프로젝트와 같이 테이블을 역으로 생성하는 경우는 드물다.

여기서는 create로 테이블을 생성후에 설정값을 validate로 변경하자.

그렇게 하지 않는다면 WAS가 실행될때마다 테이블이 다시 생성되어서 기존 데이터가 삭제된다.

Tiles

Tiles 설정

Tiles는 Servlet Context 영역으로 `[WEB_CONFIG_HOME]/springmvc/servlet-tiles.xml` 에

서 설정한다.

```
<!-- tiles configurer -->
<bean id="tilesConfigurer" class="org.springframework.web.servlet.view.tiles3
.TilesConfigurer">
    <property name="completeAutoload" value="true"/>
    <property name="useMutableTilesContainer" value="true" />
    <property name="checkRefresh" value="true" />
    <property name="definitions">
        <list>
            <value>/WEB-INF/config/tiles/**/tiles-*.xml</value>
        </list>
    </property>
</bean>
```

completeAutoload 프로퍼티의 값을 true로 설정하면 tiles 설정파일에서 EL, REGEXP를 사용할수 있다. `[MODULE_HOME]/POM.xml` 에 tiles-extras, tiles-el를 추가한다.

```
<dependency>
    <groupId>org.apache.tiles</groupId>
    <artifactId>tiles-extras</artifactId>
    <version>${tiles.version}</version>
</dependency>
<dependency>
    <groupId>org.apache.tiles</groupId>
    <artifactId>tiles-el</artifactId>
    <version>${tiles.version}</version>
</dependency>
```

Tiles설정은 `[WEB_CONFIG_HOME]/tiles/tiles-definitions.xml` 를 참조한다.

```
<tiles-definitions>
    <!-- default template -->
    <definition name="defaultTpl" template="/WEB-INF/view/tiles/template/defa
ult.layout.jsp">
        <put-attribute name="title"         value="" type="string" />
        <put-attribute name="head"          value="/WEB-INF/view/tiles/attrib
ute/head.attr.jsp" />
        <put-attribute name="javascript"    value="/WEB-INF/view/tiles/attrib
ute/javascript.attr.jsp" />
        <put-attribute name="top"           value="/WEB-INF/view/tiles/attrib
ute/top.attr.jsp" />
        <put-attribute name="left"          value="/WEB-INF/view/tiles/attrib
ute/left.attr.jsp" />
        <put-attribute name="contents"      value="/WEB-INF/view/tiles/attrib
ute/contents.attr.jsp" />
        <put-attribute name="bottom"        value="/WEB-INF/view/tiles/attrib
ute/bottom.attr.jsp" />
    </definition>

    <!-- index template -->
    <definition name="indexTpl" template="/WEB-INF/view/tiles/template/index.
layout.jsp">
```

```
        <put-attribute name="title"         value="" type="string" />
        <put-attribute name="head"          value="/WEB-INF/view/tiles/attribute/head.attr.jsp" />
        <put-attribute name="javascript"    value="/WEB-INF/view/tiles/attribute/javascript.attr.jsp" />
        <put-attribute name="top"           value="/WEB-INF/view/tiles/attribute/top.attr.jsp" />
        <put-attribute name="contents"      value="/WEB-INF/view/tiles/attribute/contents.attr.jsp" />
        <put-attribute name="bottom"        value="/WEB-INF/view/tiles/attribute/bottom.attr.jsp" />
    </definition>
</tiles-definitions>
```

tiles-definitions.xml에서 `<definition name="defaultTpl" \>` 구성 예

```
1. defaultTpl의 전체적인 틀(템플릿)은 "/WEB-INF/view/tiles/template/default.layout.jsp"에 구성되어 있다.
2. default.layout.jsp는 title, head, javascript, top, left, contents, bottom 부분으로 구성되어 있다.
    2.1. title: 웹페이지의 제목이 오는 부분
    2.2. head: html의 <head></head> 태그 안에 부분
    2.3. javascript: 공통으로 사용되는 자바스크립트가 인클루드 되는 부분
    2.4. top: 상단 공통메뉴
    2.5. left: 왼쪽 공통메뉴
    2.6. contents: 각 페이지별로 화면구성
    2.7. bottom: 회사소개, Copyright 구성
```

결국 `<definition name="defaultTpl" \>`을 상속 받는다는 것은 위 공통 구성을 사용한다는 것이고 그중 title, contents부분만 교체해 줌으로 해서 페이지 별 화면 구성을 달리 할 수 있다.

`[WEB_CONFIG_HOME]/tiles/tiles-definitions.xml`의 예외 같이 `<definition name="defaultTpl" \>`을 상속받아 title, contents 부분만 오버라이딩 해주면 된다.

만약 특별한 위와 다른 구조의 레이아웃이 필요하면 tiles-definitions.xml에 기본 템플릿을 추가한후 상속받아 사용하면 된다.

```
<tiles-definitions>
    <!-- 왼쪽 메뉴가 있는 템플릿 -->
    <definition name="REGEXP:(.*)\.defaultTpl" extends="defaultTpl">
        <put-attribute name="title"     expression="${title}" />
        <put-attribute name="contents"  value="/WEB-INF/view/{1}.jsp" />
    </definition>
    <!-- 왼쪽 메뉴가 없는 템플릿 -->
    <definition name="REGEXP:(.*)\.indexTpl" extends="indexTpl">
        <put-attribute name="title"     expression="${title}" />
        <put-attribute name="contents"  value="/WEB-INF/view/{1}.jsp" />
    </definition>
</tiles-definitions>
```

웹페이지 호출 경로는 아래와 같다.

```
URL Call -> Controller -> Service -> DAO  -----
                                              |
                                           DataBase
                                              |
웹페이지   <- Controller <- Service <- DAO <-----
```

여기서 Controller에서 웹페이지로 호출될때 Springframework에서는 ViewResolver를 이용해서 출력 포맷을 지정할수 있다.

`[WEB_CONFIG_HOME]/springmvc/servlet-view.xml` 파일에서 tiles를 이용하기 위해서 viewresolver에 등록했다.

SpringFramework에서 뷰를 지정하는 순서는 ContentNegotiatingViewResolver 패턴매칭 알고리즘에 의해서 몇가지 후보군을 고른후

defaultViews의 View를 추가하여 총 후보군을 설정 매칭후 결과를 리턴한다.

매칭 결과 일치하는 View를 찾을수 없으면 404에러가 발생한다.

Spring View Resolver 설정

```
<bean class="org.springframework.web.servlet.view.ContentNegotiatingViewResol
ver">
    <property name="order" value="0" />
    <property name="contentNegotiationManager">
        <bean class="org.springframework.web.accept.ContentNegotiationManager
">
            <constructor-arg>
                <bean class="org.springframework.web.accept.PathExtensionCont
entNegotiationStrategy">
                    <constructor-arg>
                        <map>
                            <entry key="html"   value="text/html"/>
                            <entry key="json"   value="application/json" />
                            <entry key="jsonp"  value="javascript/json" />
                            <entry key="xml"    value="application/xml" />
                        </map>
                    </constructor-arg>
                </bean>
            </constructor-arg>
        </bean>
    </property>

    <!-- ViewResolvers -->
    <property name="viewResolvers">
        <list>
            <bean class="org.springframework.web.servlet.view.BeanNameViewRes
olver" />
            <bean class="org.springframework.web.servlet.view.tiles3.TilesVie
wResolver">
                <property name="viewClass"      value="org.springframework.we
b.servlet.view.tiles3.TilesView" />
                <property name="contentType"    value="text/html" />
```

```
            </bean>
            <bean class="org.springframework.web.servlet.view.InternalResourc
eViewResolver">
                <property name="viewClass"      value="org.springframework.we
b.servlet.view.JstlView" />
                <property name="prefix"         value="/WEB-INF/view/" />
                <property name="suffix"         value=".jsp" />
            </bean>
        </list>
    </property>

    <!-- DefaultViews -->
    <property name="defaultViews">
        <list>
            <ref bean="jacksonJsonView" />
            <ref bean="jsonpView" />
            <ref bean="marshallingView" />
        </list>
    </property>
</bean>
```

Controller에서 호출하는 방법은 return값으로 tiles name을 넘기는 방법을 사용한다.

`[JAVA_SRC_HOME]/dd2/local/busi/main/web/MainController.java` 참고

```
... 생략

@Controller
@RequestMapping("/main/*")
public class MainController extends CommonController {
    private static final Log logger = LogFactory.getLog(MainController.class)
;
    private static final String BASE_URL = "main/";

    @Override
    public String getBaseUrl() {
        return BASE_URL;
    }

    @RequestMapping("index")
    public String doIndex() {
        return getBaseUrl() + "index.indexTpl";
    }

    @RequestMapping("dashboard")
    public String doDashboard() {
        return getBaseUrl() + "dashboard.defaultTpl";
    }
}
```

Tiles 템플릿은 indexTpl, defaultTpl 두가지 종류가 있고 필요에 따라서 추가한후 사용하면 된다.

추가는 `[WEB_CONFIG_HOME]/tiles/tiles-definitions.xml` 에 추가한다.

Frontend

Web App 구성

전체적인 웹 구성

```
ROOT-HOME
    ├─ static
    │    ├─ css: WebApp에서 사용하는 css파일
    │    ├─ images: WebApp에서 사용하는 이미지
    │    ├─ js
    │    │    ├─ bootstrap
    │    │    │    └─ bootstrap.local.js: 부트스트랩 설정
    │    │    ├─ extend
    │    │    │    └─ extendjs.js: 자바스크립트 확장
    │    │    ├─ jqgrid
    │    │    │    └─ jquery.jqgrid.local.js: jqgrid 설정
    │    │    ├─ jquery
    │    │    │    └─ jquery.local.js: jquery 설정
    │    │    ├─ underscore
    │    │    │    └─ underscore.local.js: underscore 설정
    │    │    ├─ local.js: WebApp 공통으로 사용하는 기능
    │    │    └─ main.js: requirejs 설정
    │    └─ lib
    │         ├─ bootstrap
    │         │    └─ 3.0.3: bootstarp 버전
    │         ├─ font-awesome
    │         │    └─ 4.0.3: font-awesome 버전
    │         ├─ jqgrid
    │         │    └─ 4.5.4: jqgrid 버전
    │         ├─ jquery
    │         │    └─ 1.10.2: jquery 버전
    │         ├─ jquery-ui
    │         │    └─ 1.10.3: jquery-ui 버전
    │         ├─ json
    │         │    └─ 2013.05.26: json 버전
    │         ├─ require
    │         │    └─ 2.1.9: require 버전
    │         └─ underscore
    │              └─ 1.5.2: underscore 버전
    ├─ WEB-INF
    │    ├─ config
    │    │    ├─ springmvc: SpringFramework Servlet Context 설정
    │    │    └─ tiles: Tiles 설정
    │    ├─ view: .jsp 뷰의 모음
    │    └─ web.xml: WebApp web.xml
    │
    └─ welcome.jsp: index 페이지
```

Bootstrap 설정

부트스트랩은 css Framework으로 반응형 웹을 지원하며 CSS 고민을 덜고 웹사이트 개발을 할수 있게 해주는 아주 고마운 라이브러리다.

부트스트랩을 다운로드한다. 현재 최신 버전은 3.0.3 이다.

`[WEB_HOME]/static/lib/bootstrap/3.0.3` 에 압축을 해제한다.

부트스트랩 개별 설정은 `[WEB_HOME]/static/js/bootstrap/bootstrap.local.js` 을 참고한다.

부트스트랩을 사용하기 위해서는 각 페이지 마다 .css, .js를 포함해야 한다.

이 프로젝트에서는 Tiles를 이용하여 부트스트랩 라이브러리를 포함시킨다.

`[WEB_HOME]/WEB-INF/view/tiles/attribute/head.attr.jsp` 에 css를 추가한다.

```
<!-- stylesheet -->
<link href="${pageContext.request.contextPath}/static/lib/bootstrap/3.0.3/css
/bootstrap.css" rel="stylesheet">
<link href="${pageContext.request.contextPath}/static/lib/bootstrap/3.0.3/css
/bootstrap-theme.css" rel="stylesheet">
<link href="${pageContext.request.contextPath}/static/lib/font-awesome/4.0.3/
css/font-awesome.css" rel="stylesheet">
```

부트스트랩 라이브러리와 함께 부트스트랩 기반의 font-awesome 라이브러리도 같이 포함시킨다. font-awesome 다운로드

`[WEB_HOME]/static/lib/font-awesome/4.0.3` 에 압축을 해제한다.

부트스트랩의 사용법은 Bootstrap Getting Started 문서를 참고한다.

requirejs 설정

requirejs는 웹을 모듈화 할수 있게 해주는 프레임웍이다. 자바스크립트 파일들을 requirejs 모듈화 규칙에 따라서 제작하면 각 자바스크립들에 대한 의존성 관리를 할 수 있다.

requirejs을 다운로드 한후 `[WEB_HOME]/static/lib/require/2.1.9` 에 압축을 해제한다.

참고: JavaScript 모듈화를 위한 RequireJS 핵심정리

requierjs를 사용하기 위해서는 처음 엔트리 포인트가 필요하다. 이프로젝트에서는 head.attr.js에 작성하고 공통적으로 사용했다.

`[WEB_HOME]/WEB-INF/view/tiles/attribute/head.attr.jsp` require.src.js 파일을 추가하고 requirejs 환경설정 파일인 main.js 파일로 추가한다.

```
<!-- script -->
```

```
<script src="${pageContext.request.contextPath}/static/lib/require/2.1.9/require.src.js"></script>
<script src="${pageContext.request.contextPath}/static/js/main.js"></script>
```

환경설정은 `[WEB_HOME]/static/js/main.js` 을 참고한다.

```
/**
 * user strict 명령은 엄격하게 JavaScript 룰을 적용하라는 의미이다.
 * 일부 브라우저의 경우 use strict 명령을 통해 보다 빠르게 동작하는 경우도 존재하는 것 같다.
 * 잘못된 부분에 대한 검증도 보다 엄격하게 동작한다.
 * 하지만, 일부 라이브러리의 경우 use strict 명령을 사용하면 동작하지 않는 경우도 있으므로 주의해야 한다.
 */
;"use strict"
require.config({
    baseUrl: "/static/js",
    paths: {
        // local paths
        "backbone.local"            : "backbone/backbone.local",
        "bootstrap.local"           : "bootstrap/bootstrap.local",
        "dynatree.local"            : "dynatree/jquery.dynatree.local",
        "extend"                    : "extend/extendjs",
        "jqgrid.local"              : "jqgrid/jquery.jqgrid.local",
        "jquery.local"              : "jquery/jquery.local",
        "underscore.local"          : "underscore/underscore.local",
        "local"                     : "local",

        // lib paths
        "angular"                   : "../lib/angular/1.2.6/angular",
        "angular-resource"          : "../lib/angular/1.2.6/angular-resource",
        "angular-locale"            : "../lib/angular/1.2.6/i18n/angular-locale_ko",
        "backbone"                  : "../lib/backbone/1.1.0/backbone",
        "bootstrap"                 : "../lib/bootstrap/3.0.3/js/bootstrap",
        "ckeditor"                  : "../lib/ckeditor/4.3/ckeditor",
        "domReady"                  : "../lib/domReady/2.0.1/domReady",
        "dynatree"                  : "../lib/dynatree/1.2.4/src/jquery.dynatree",
        "highlight"                 : "../lib/highlight/7.5/highlight.pack",
        "holder"                    : "../lib/holder/2.2/holder",
        "html5shiv"                 : "../lib/html5shiv/3.7.0/html5shiv.src",
        "jqgrid"                    : "../lib/jqgrid/4.5.4/src/jquery.jqGrid",
        "jqgrid-locale"             : "../lib/jqgrid/4.5.4/src/i18n/grid.locale-kr",
        "jquery"                    : "../lib/jquery/1.10.2/jquery-1.10.2",
        "jquery-form-validator"     : "../lib/jquery-form-validator/2.1.27/form-validator/jquery.form-validator",
        "jquery-ui"                 : "../lib/jquery-ui/1.10.3/ui/jquery-ui",
        "jquery-ui-datepicker"      : "../lib/jquery-ui/1.10.3/ui/jquery.ui.datepicker",
        "jquery-ui-datepicker-lang" : "../lib/jquery-ui/1.10.3/ui/i18n/jquery.ui.datepicker-ko",
        "json"                      : "../lib/json/2013.05.26/json2",
```

```
    "respond"                   : "../lib/respond/1.4.0/respond.src",
    "underscore"                : "../lib/underscore/1.5.2/underscore"
},
shim: {
    "angular": {
        deps: ["angular-locale"],
        exports: "angular"
    },
    "backbone": {
        deps: ["underscore", "jquery"],
        exports: "Backbone",
        init: function() {
            return this.Backbone.noConflict();
        }
    },
    "bootstrap": {
        deps: ["jquery"]
    },
    "ckeditor": {
        deps: ["jquery"],
        exports: "CKEDITOR"
    },
    "dynatree": {
        deps: ["jquery", "jquery-ui"]
    },
    "highlight": {
        exports: "hljs"
    },
    "holder": {
        exports: "HOLDER"
    },
    "jqgrid": {
        deps: ["jquery", "jquery-ui", "jqgrid-locale"]
    },
    "jqgrid-locale": {
        deps: ["jquery"]
    },
    "jquery": {
        deps: ["json"],
        exports: "jQuery",
        init: function() {
            return this.jQuery.noConflict(true);
        }
    },
    "jquery-form-validator": {
        deps: ["jquery"]
    },
    "jquery-ui": {
        deps: ["jquery"]
    },
    "jquery-ui-datepicker": {
        deps: ["jquery-ui"]
    },
    "jquery-ui-datepicker-lang": {
        deps: ["jquery-ui-datepicker"]
    },
```

```
        "json": {
            exports : "JSON"
        },
        "underscore": {
            exports: "_",
            init: function() {
                return this._.noConflict();
            }
        }
    },
    waitSeconds: 15
});
```

Sim Configuraion 참고 문서

jQuery 설정

말이 필요없는 자바스크립트 라이브러리이다. 다운로드한다.

`[WEB_HOME]/static/lib/jquery/1.10.2` 에 압축을 해제한다.

환경설정은 `[WEB_HOME]/static/js/jquery/jquery.local.js` 을 참고한다.

```
;define([
    "jquery",
    "json"
],
function($, JSON) { "use strict";
    /**
     * ajax 기본 셋팅
     */
    $.ajaxSetup({
        type        : 'POST',
        async       : true,
        cache       : false,
        contentType : 'application/json; charset=utf-8',
        dataType    : 'json',
        error       : function(xhr,e) {
            if (xhr.status == 0) {
                alert('You are offline!!\n Please Check Your Network.');
            }
            else if (xhr.status == 404) {
                alert('Requested URL not found.');
            }
            else if (xhr.status == 500) {
                alert('Internel Server Error.\n');
            }
            else if (e.toLowerCase() == 'parsererror') {
                alert('Error.\nParsing JSON Request failed.');
            }
            else if (e.toLowerCase() == 'timeout') {
                alert('Request Time out.');
            }
```

```
            else {
                alert('Unknow Error.\n'+xhr.responseText);
            }
        },
        beforeSend : function(xhr, setting) {
            if ( setting && setting.async == true ) {
            }
        },
        complete : function(xhr, e) {
        }
    });//$.ajaxSetup({

    $.ajaxPrefilter("json", function (options, originalOptions, jqXHR) {
        if (options.type.toUpperCase() == "POST" && originalOptions["data"] &
& ($.isPlainObject(originalOptions.data) || $.isArray(originalOptions.data))
) {
            options.data = JSON.stringify(originalOptions.data);
        }
    });

    $.fn.serializeObject = function() {
        var o = {},
            a = this.serializeArray();

        $.each(a, function() {
            if ( o[this.name] ) {
                if ( !o[this.name].push ) {
                    o[this.name] = [o[this.name]];
                }
                o[this.name].push(this.value || '');
            } else {
                o[this.name] = this.value || '';
            }
        });
        return o;
    };

    /**
     * 토글 이벤트 구현
     * @param a 첫번째 클릭 이벤트
     * @param b 두번째 클릭 이벤트
     * @returns {*}
     */
    $.fn.clickToggle = function(a, b) {
        return this.each(function() {
            var clicked = false;
            $(this).on("click", function() {
                if (clicked) {
                    clicked = false;
                    return b.apply(this, arguments);
                } else {
                    clicked = true;
                    return a.apply(this, arguments);
                }
            });
        });
```

```
    };

    $.createGUID = function() {
        var S4 = function() {
            return ((Math.random()+1)*10000|0).toString(16).substring(1);
        };
        return "" + (new Date()).getTime() + "-" + (S4()+S4()+"-"+S4()+"-"+S4
()+"-"+S4()+"-"+S4()+S4()+S4()).toUpperCase();
    };
});
```

프로젝트에서 jquery API와 관련된 기본적인 설정과 Custom API를 추가해서 사용하면 된다.

jQuery Documentation

jQueryUi 설정

jquery를 이용한 각종 UI 컴포넌트 집합이다. jqgrid에서 사용하기 때문에 필요하다. 다운로드한다.

`[WEB_HOME]/static/lib/jquery-ui/1.10.3` 에 압축 해제 한다.

jquery-ui는 css와 javascript 파일로 구성되어 있다. css는 head.attr.jsp 파일에 공통적으로 포함시킨다.

```
<link href="${pageContext.request.contextPath}/static/lib/jquery-ui/1.10.3/th
emes/base/jquery-ui.css" rel="stylesheet">
```

javascript 파일은 `[WEB_HOME]/static/js/main.js` 파일에 설정한다.

```
require.config({
    baseUrl: "/static/js",
    paths: {
        ... 생략
        "jquery-ui"                 : "../lib/jquery-ui/1.10.3/ui/jquery-ui",
        "jquery-ui-datepicker"      : "../lib/jquery-ui/1.10.3/ui/jquery.ui.d
atepicker",
        "jquery-ui-datepicker-lang" : "../lib/jquery-ui/1.10.3/ui/i18n/jquery
.ui.datepicker-ko",
        ... 생략
    },
    shim: {
        ... 생략
        "jquery-ui": {
            deps: ["jquery"]
        },
        "jquery-ui-datepicker": {
            deps: ["jquery-ui"]
        },
        "jquery-ui-datepicker-lang": {
            deps: ["jquery-ui-datepicker"]
        },
        ... 생략
    },
```

```
    waitSeconds: 15
});
```

jqGrid 설정

jquery plugin 형식의 웹 그리드 이다. 다운로드한다.

`[WEB_HOME]/static/lib/jqgrid/4.5.4` 에 압축 해제한다.

환경설정은 `[WEB_HOME]/static/js/jqgrid/jquery.jqgrid.local.js` 을 참고한다.

```
;define([
    "jquery",
    "jqgrid",
    "jquery-ui-datepicker-lang"
],
function($) { "use strict";
    var _id = $.createGUID();
    var _guidName = "_local_jqgrid_guid_";
    var _userCacheName = "userCache";
    var _userPostDataName = "userPostData";

    var _DataConvert = function(data) {
        var $t = $(this),
            datatype  = $t.jqGrid('getGridParam', 'datatype'),
            userCache = $t.jqGrid('getGridParam', _userCacheName) || false,
            userPostData  = $t.jqGrid('getGridParam', _userPostDataName) || {
},
            newData   = "";

        if ( typeof data[_userCacheName] === "undefined" ) {
            data[_userCacheName] = userCache;
        }

        if ( typeof data[_guidName] === "undefined" ) {
            data[_guidName] = _id;
        }

        if ( typeof data[_userPostDataName] === "undefined" ) {
            data[_userPostDataName] = userPostData;
        }

        switch (datatype.toLowerCase()) {
            case "json":
                newData = JSON.stringify(data);
                break;
            default :
                newData = data;
                break;
        }
        return newData;
    };
```

```
    /**
     * jqGrid default options
     */
    $.extend($.jgrid.defaults, {
        userCache: false,
        autowidth: true,
        viewrecords: true,
        gridview: true,// true로 하면은 treeGrid, subGrid, or the afterInsertRo
w event. 사용할수 없다. 대신 속도는 빠르다.
        prmNames: {
            page: "pageNumber",
            rows: "pageSize"
        },
        jsonReader: {
            page: "page",
            total: "total",
            records: "records",
            root: "rows",
            id: function(obj) {return (obj.idName||"0");},
            repeatitems: false,// json을 데이터로 사용한다.
            cell: "cell",// repeatitems: false 이면 무시된다.
            userdate: "userdata"
        },
        loadError: function(xhr,st,err) {
            alert("Server Failure" + "\nType: "+st+"\nResponse: "+ xhr.status
 + " "+xhr.statusText );
        },
        mtype: "POST",//Defines the type of request to make ("POST" or "GET")
        datatype: "local",
        ajaxGridOptions: { contentType: "application/json" },
        ajaxRowOptions: { contentType: "application/json" },
        ajaxCellOptions: { contentType: "application/json" },
        serializeGridData: _DataConvert,
        serializeCellData: _DataConvert,
        cellsubmit: 'clientArray',// 셀 수정했을때 자동으로 서버에 전송되지 않는다.
        cellEdit: false,
        rowNum : 10,
        rowList: [10, 15, 25, 50, 100, 200]
    });

    /**
     * jqGrid edit options
     */
    $.extend($.jgrid.edit, {
        ajaxEditOptions: { contentType: "application/json" },
        recreateForm: true,
        serializeEditData: _DataConvert
    });

    /**
     * jqGrid dell options
     */
    $.extend($.jgrid.del, {
        ajaxDelOptions: { contentType: "application/json" },
        recreateForm: true,
        serializeDelData: _DataConvert
```

```
});

/**
 * jqGrid 확장 메소드
 */
$.jgrid.extend({
    addNewRow: function() {
        var $that = this;
        return $that.each(function() {
            var $t = $(this);
            $t.addRowData("", {});
        });
    },
    getSelectedRowId: function() {
        var $t = this,
            selectedId = $t.find("tr.selected-row").prop("id");

        if ( selectedId == undefined  || selectedId == null ) {
            return null;
        }
        return selectedId;
    },
    setSelectedRowById: function(rowid) {
        var $that = this;

        return  $that.each(function() {
            var $t = $(this),
                iRow = 0,
                iCol = 0;
            $.each($t.getDataIDs(), function(idx, value) {
                if ( parseInt(value) == parseInt(rowid) ) {
                    iRow = idx;
                    return false;
                }
            });
            var $tr = $t.find("tr[id=" + rowid + "]");
            $tr.addClass("selected-row ui-state-hover");
            $t[0].p.selrow = rowid;
            $t[0].p.iRow = iRow+1;
            $t[0].p.iCol = iCol;
        });
    },
    getSelectedRowData: function() {
        var $t = this,
            id = $t.getSelectedRowId();

        if ( id == null ) {
            return [];
        }
        return $t.getRowData(id);
    },
    getMultiSelectedRowIDs: function() {
        var $t = this,
            isMultiSelect = $t.getGridParam('multiselect'),
            result = [];
        if ( isMultiSelect == true ) {
```

```
            $t.find("tr[aria-selected=true]").each(function() {
                var $tr = $(this);
                var id = $tr.prop("id");
                result.push(id);
            });
        }
        return result;
    },
    getMultiSelectedRowData: function() {
        var $t = this,
            isMultiSelect = $t.getGridParam('multiselect'),
            result = [];
        if ( isMultiSelect == true ) {
            var ids = $t.getMultiSelectedRowIDs();
            $.each(ids, function(idx, id) {
                result.push( $t.getRowData(id));
            });

        }
        return result;
    },
    getColModel: function(name) {
        var $t = this,
            colModel = $t.getGridParam("colModel");
        if ( arguments.length == 0 ) {
            return colModel;
        } else if ( $.isNumeric(name)) {
            return colModel[name];
        } else {
            var colModelRow = {};
            $.each(colModel, function(idx, value) {
                if ( value.name == name ) {
                    colModelRow = value;
                    return false;
                }
            });
            return colModelRow;
        }
    },
    setColModel: function(updateColModel) {
        // 속도를 빠르게 하기 위해서 해쉬를 이용한다.
        var mapUpdateColModel = {};
        $.each(updateColModel, function (idx, value) {
            mapUpdateColModel[value.name] = value;
        });

        return this.each(function () {
            var $t         = $(this),
                colModel    = $t.getGridParam("colModel"),
                newColModel = [];
            $.each(colModel, function (idx, value) {
                if ( mapUpdateColModel[value.name] ) {
                    newColModel.push(mapUpdateColModel[value.name]);
                } else {
                    newColModel.push(value);
                }
```

```
                });
                $t.setGridParam("colModel", newColModel);
            });
        }
    })

    /**
     * jqGrid Static 메소드 모음
     */
    $.extend( ($.jqGridStatic = $.jqGridStatic || {}), {
        /**
         * 날짜 포맷에 대한 설정
         */
        userDateSetting: {
            sorttype: 'date',
            formatter: function(cellValue /* cell의 값 */, option /* colModel
option  */, rowObject /* 현재 row 값 */) {
                var date = null;

                if ( cellValue == undefined || cellValue == null ) {
                    return "";
                }

                if ( /^\d+$/gi.test(cellValue.toString()) == true ) {
                    date = new Date( parseInt(cellValue)  );
                    return date.format("yyyy-MM-dd");
                } else {
                    return cellValue.toString();
                }
            },
            editable: true,
            edittype: 'text',
            editoptions: {
                size: 12,
                maxlengh: 12,
                dataInit: function (element) {
                    $(element).datepicker({ dateFormat: 'yy-mm-dd' });
                }
            },
            editrules: {
                date: true
            }
        },
        /**
         * jqGrid에서 사용하는 editoptions 중에 Select를 만들어준다.
         * @param setting {defaultOptionUsed, defaultOption, data, valueName,
 displayName, defaultValue}
         * @param option
         * @returns {*|Object}
         */
        selectEditOptions: function(setting , option ) {
            var ret = $.extend({}, option || {});
            if ($.isPlainObject(setting)) {
                var valueString = "";
                var newSetting = $.extend({
                    defaultOptionUsed: true,
```

```
                    defaultOption: {
                        value: null,
                        display: "♦ 선택해주세요 ♦"
                    },
                    data: [],
                    valueName: "",
                    displayName: "",
                    defaultValue: null
                }, setting);

                if ( newSetting.defaultOptionUsed == true ) {
                    if ( $.isPlainObject(newSetting.defaultOption) ) {
                        valueString += "" + newSetting.defaultOption.value +
":" + newSetting.defaultOption.display;
                    } else  {
                        valueString += newSetting.defaultOption;
                    }
                }

                $.each(newSetting.data, function(idx, value) {
                    valueString += ";" + value[newSetting.valueName] + ":" +
value[newSetting.displayName];
                });
                if ( valueString.charAt(0) == ";" ) {
                    valueString = valueString.substring(1);
                }

                ret["value"] = valueString;
                ret["defaultValue"] = "" + newSetting.defaultValue;
            }
            return ret;
        },
        /**
         * [{},{},{},...] 데이터셋 구조에서 해당 Row를 Key Value로 찾아서 리턴한다.
         * @param data {arrayData}
         * @param obj {key,value}
         * @returns {*|Object}
         */
        eachRowByKeyValue: function(data, obj) {
            var ret = {};
            $.each(data, function(idx, value) {
                if ( value[obj.key] && value[obj.key] == obj.value ) {
                    ret = value;
                    return false;
                }
            });
            return ret;
        }
    });
});
```

jqGrid Documentation 참고문서

데모페이지, 그리드 옵션, 메소드, 이벤트, ColModel 옵션, 데이터 조작, Form Editing, 작성방법

Sample

예제로 사용할 테이블 이다. 테이블을 생성한다.

```
CREATE TABLE SAMPLE (
    ID INT NOT NULL IDENTITY(1, 1),
    NAME VARCHAR(255) NOT NULL,
    AGE INT NUL NULL,
    DESCRIPTION VARCHAR(255) NULL,
    PRIMARY KEY(ID)
);
```

`[JAVA_SRC_HOME]/dd2/local/entity/SampelEntity.java` 을 생성한다.

```
package dd2.local.entity;

import javax.persistence.*;

@javax.persistence.Table(name = "SAMPLE")
@Entity
public class SampleEntity {
    private Long id;
    private String name;
    private Long age;
    private String description;

    public SampleEntity() {
    }

    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "ID")
    @Id
    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    @Column(name = "NAME", nullable = false)
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    @Column(name = "AGE", nullable = false)
    public Long getAge() {
        return age;
    }
```

```
    public void setAge(Long age) {
        this.age = age;
    }

    @Column(name = "DESCRIPTION", nullable = true)
    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }
}
```

참고 문서: Hibernate Documentation

`[WEB_HOME]/WEB-INF/view/sample` 폴더를 생성한다.

`[WEB_HOME]/WEB-INF/view/sample/list.jsp` 파일을 생성한다.

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ page trimDirectiveWhitespaces="true" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core"              prefix="c"
     %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/functions"        prefix="fn"
   %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt"              prefix="fmt"
   %>
<%@ taglib uri="http://tiles.apache.org/tags-tiles"            prefix="tiles
"  %>
<%@ taglib uri="http://www.springframework.org/tags"           prefix="sprin
g" %>
<%@ taglib uri="http://www.springframework.org/security/tags"  prefix="sec"
   %>
<div class="panel panel-default">
    <div class="panel-heading">${title}</div>
    <div class="panel-body">
        <table id="list"><tr><td></td></tr></table>
        <div id="pager"></div>
    </div>
</div>

<script type="text/javascript">
;require([
    "jquery",
    "local"
],
function($, LOCAL) { $(document).ready(function() {
    var $grid = $("#list");

    $grid.jqGrid({
        userCache: true,
        caption: "Sample List",
        multiselect: true,
```

```
        url: LOCAL.url("/sample/records"),
        editurl: LOCAL.url("/sample/recordEdit"),
        cellurl: LOCAL.url("/sample/recordEdit"),
        height: 300,
        colNames:['id', 'name', 'age', 'description'],
        colModel:[
            {name:'id', index:'id', hidden: true },
            $.extend(true, {name:'name', index:'name', width:100, editable: t
rue, frozen: true}, $.jqGridStatic.userSearchSetting),
            $.extend(true, {name:'age', index:'name', width:100, editable: tr
ue, frozen: true }, $.jqGridStatic.userSearchSetting),
            $.extend(true, {name:'description', index:'description', width:20
0, editable: true}, $.jqGridStatic.userSearchSetting)
        ],
        pager : "#pager",
        sortname: "name",
        sortorder: "desc",
        loadComplete: function(data) {
            var $t = $(this);
            if ( data.rows.length > 0 ) {
                $t.setSelectedRowById($t.getDataIDs()[0]);
            }
        },
        onSelectRow: function (id) {
        },
        ondblClickRow: function(id, iRow, iCol, e) {
        }
    }).jqGrid('navGrid', '#pager', {
        edit: true,
        add: true,
        del: true,
        search: true,
        refresh: true
    });

    $grid.setGridParam({
        datatype: "json"
    }).trigger("reloadGrid");
});});
</script>
```

`[JAVA_SRC_HOME]/dd2/local/busi/sample/web/SampleController.java` 파일을 생성한다.

```
package dd2.local.busi.sample.web;

import dd2.local.busi.com.web.CommonController;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
@RequestMapping("/sample/*")
public class SampleController extends CommonController {
```

```
    private static final Log logger = LogFactory.getLog(SampleController.clas
s);
    private static final String BASE_URL = "sample/";

    @Override
    protected String getBaseUrl() {
        return BASE_URL;
    }

    ... 생략
}
```

`@RequestMapping("/sample/*")`을 선언 함으로 해서 해당 URL의 `/sample/`로 시작하는 모든

URL은 SampleController.java가 호출된다.

CommonController를 상속 받으면 getBaseUrl 메소드를 오버라이드 해줘야 한다.

CommonController는 `@RequestMapping( value = "comm/{tilesName}")`, `@RequestMapping( value = "comm/{tilesName}/{name}/{value}")`

두개지 형태의 URL을 처리해준다.

```
... 생략

/**
 * ../comm/{titles} 의 url 호출
 * @param request
 * @param model
 * @param tilesName
 * @return
 */
@RequestMapping( value = "comm/{tilesName}")
public String doCommPage(
        HttpServletRequest request,
        ModelMap model,
        @PathVariable(value = "tilesName") String tilesName
) {
    return getBaseUrl() + tilesName + ".defaultTpl";
}

/**
 * ../comm/{titles}/{name}/{value} 의 url 호출
 * @param request
 * @param model
 * @param tilesName
 * @param name
 * @param value
 * @return
 */
@RequestMapping( value = "comm/{tilesName}/{name}/{value}")
public String doCommPageAndSingleVar(
        HttpServletRequest request,
```

```
        ModelMap model,
        @PathVariable(value = "tilesName") String tilesName,
        @PathVariable(value = "name") String name,
        @PathVariable(value = "value") String value
) {
    model.put(name, value);
    return getBaseUrl() + tilesName + ".defaultTpl";
}

... 생략
```

로컬에서 작업하면 `http://localhost:8080/sample/comm/list` URL로 제대로 호출되는지 확인한다.

호출이 정상적으로 된다면 왼쪽 메뉴에 등록하자

`[WEB_HOME]/WEB-INF/view/tiles/attribute/left.attr.jsp` 에 URL을 추가한다.

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ page trimDirectiveWhitespaces="true" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core"              prefix="c"
     %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/functions"        prefix="fn"
   %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt"              prefix="fmt"
   %>
<%@ taglib uri="http://tiles.apache.org/tags-tiles"            prefix="tiles
"  %>
<%@ taglib uri="http://www.springframework.org/tags"           prefix="sprin
g" %>
<%@ taglib uri="http://www.springframework.org/security/tags"  prefix="sec"
   %>
<div id="left-menu" class="list-group">
    ... 생략
    <a class="list-group-item" href="<c:url value='/sample/comm/list/title/Sa
mple' />">Sample</a>
</div>
    ... 생략
```

이제 왼쪽 메뉴에서도 접근 가능하다.

`[JAVA_SRC_HOME]/dd2/local/busi/sample/service/dao/SampleDAO.java` 파일을 생성한다.

```
package dd2.local.busi.sample.service.dao;

import dd2.com.dao.GenericDAO;
import dd2.com.jqgrid.JqGridRequest;
import dd2.com.jqgrid.JqGridResponseGeneric;
import dd2.local.entity.SampleEntity;

public interface SampleDAO extends GenericDAO<SampleEntity, Long> {
    JqGridResponseGeneric<SampleEntity> list(JqGridRequest request);
```

```
}
```

GenericDAO는 Hibernate를 편하게 사용하기 위한 기본적인 메소드를 정의 하고 있다.

`[JAVA_SRC_HOME]/dd2/com/dao/GenericDAO.java`

```
... 생략

public interface GenericDAO<T, ID extends Serializable> {
    T findById(ID id, boolean lock);
    T findById(ID id);
    List<T> findAll();
    List<T> findByExample(T exampleInstance, String... excludeProperty);
    T save(T entity);
    T update(T entity);
    T saveOrUpdate(T entity);
    void delete(T entity);
    void deleteById(ID id);
    void deleteByIds(ID[] ids);
    void flush();
    void clear();
}
```

GenericDAO는 인터페이스 이고 구현은 `[JAVA_SRC_HOME]/dd2/com/dao/GenericDAOHibernate.java` 에 구현되어 있다.

`[JAVA_SRC_HOME]/dd2/local/busi/sample/service/dao/SampleDAOHibernate.java` 파일을 생성한다.

```
... 생략

@Repository
@SuppressWarnings("unchecked")
public class SampleDAOHibernate extends GenericHibernateDAO<SampleEntity, Long> implements SampleDAO {
    private static final Log logger = LogFactory.getLog(SampleDAOHibernate.class);

    @Override
    public JqGridResponseGeneric<SampleEntity> list(JqGridRequest request) {
        JqGridResponseGeneric<SampleEntity> response = new JqGridResponseGeneric<>();

        Criterion criterion = JqGridRestrictionForHibernate.create(request);
        Order order = JqGridOrderForHibernate.create(request);

        Criteria rowCriteria = this.getCriteria()
                .setProjection(Projections.projectionList()
                        .add(Projections.rowCount())
                );
        if ( criterion != null ) {
            rowCriteria = rowCriteria.add(criterion);
        }
```

```
        if ( order != null ) {
            rowCriteria = rowCriteria.addOrder(order);
        }
        int total = Integer.parseInt(rowCriteria.uniqueResult().toString());


        Criteria listCriteria = this.getCriteria();
        if (criterion != null) {
            listCriteria = listCriteria.add(criterion);
        }
        if (order != null) {
            listCriteria = listCriteria.addOrder(order);
        }
        List<SampleEntity> sampleEntityList = listCriteria
                .setFirstResult(request.getPageNumber() - 1)
                .setMaxResults(request.getPageSize())
                .list();

        response.setTotal(total);
        response.setPage(request.getPageNumber());
        response.setRecords(request.getPageSize());
        response.setRows(sampleEntityList);

        return response;
    }
}
```

Hibernate Criteria를 이용해서 작성한다. `JqGridRestrictionForHibernate`, `JqGridOrderForHibernate` 유틸리티 클래스를 이용해서 작성을 편하게 했다.

조회한 결과물은 JSON으로 리턴하기 위해 `JqGridResponseGeneric<SampleEntity>`에 담는다.

`[JAVA_SRC_HOME]/dd2/local/busi/sample/service/dao/SampleService.java` 인터페이스 파일을 생성한다.

```
package dd2.local.busi.sample.service;

import dd2.com.jqgrid.JqGridRequest;
import dd2.com.jqgrid.JqGridResponseGeneric;
import dd2.com.service.GenericService;
import dd2.local.entity.SampleEntity;

public interface SampleService extends GenericService<SampleEntity, Long> {
    JqGridResponseGeneric<SampleEntity> list(JqGridRequest request);
}
```

`[JAVA_SRC_HOME]/dd2/local/busi/sample/service/dao/SampleServiceHibernate.java` 구현 파일을 생성한다.

```
... 생략

@Service
public class SampleServiceHibernate extends GenericHibernateService<SampleEnt
```

```
ity, Long> implements SampleService  {
    private static final Log logger = LogFactory.getLog(SampleServiceHibernat
e.class);

    @Autowired
    private SampleDAO sampleDAO;

    @Override
    protected GenericDAO<SampleEntity, Long> getGenericDAO() {
        return sampleDAO;
    }

    @Transactional
    @Override
    public JqGridResponseGeneric<SampleEntity> list(JqGridRequest request) {
        return sampleDAO.list(request);
    }
}
```

GenericHibernateService 클래스를 상속받음으로 기본적인 기능을 바로 사용할수 있다.

```
public abstract class GenericHibernateService<T, ID extends Serializable> imp
lements GenericService<T, ID> {

    /**
     * 기본 CRUD 가능하게 하는 엔티티 DAO 이다.
     * 상속 받은 쪽에서 구현해 줘야 한다.
     * @return
     */
    protected abstract GenericDAO<T, ID> getGenericDAO();

    @Transactional
    @Override
    public T findById(ID id) {
        return this.getGenericDAO().findById(id);
    }

    @Transactional
    @Override
    public T findById(ID id, boolean lock) {
        return this.getGenericDAO().findById(id, lock);
    }

    @Transactional
    @Override
    public List<T> findAll() {
        return this.getGenericDAO().findAll();
    }

    @Transactional
    @Override
    public List<T> findByExample(T exampleInstance, String... excludeProperty
) {
        return this.getGenericDAO().findByExample(exampleInstance, excludePro
perty);
```

```
    }

    @Transactional
    @Override
    public T save(T entity) {
        return this.getGenericDAO().save(entity);
    }

    @Transactional
    @Override
    public T update(T entity) {
        return this.getGenericDAO().update(entity);
    }

    @Transactional
    @Override
    public T saveOrUpdate(T entity) {
        return this.getGenericDAO().saveOrUpdate(entity);
    }

    @Transactional
    @Override
    public void delete(T entity) {
        this.getGenericDAO().delete(entity);
    }

    @Transactional
    @Override
    public void deleteById(ID id) {
        this.getGenericDAO().deleteById(id);
    }

    @Transactional
    @Override
    public void deleteByIds(ID[] ids) {
        this.getGenericDAO().deleteByIds(ids);
    }
}
```

GenericHibernateService 클래스를 상속 받고 다시 SampleService 인터페이스를 상속받은 이유는 템플릿 메소드 패턴을

이용해서 GenericHibernateDAO의 기능을 Controller 단까지 끌고 갈려고 하기 때문이다.

만약 GenericHibernateService 상속 받지 않는다면 엔티티의 간단한 CRUD를 Service 클래스 마다 구현해 주어야 한다.

`[JAVA_SRC_HOME]/dd2/local/busi/sample/web/SampleController.java` 를 수정한다.

```
... 생략

@Controller
@RequestMapping("/sample/*")
```

```
public class SampleController extends CommonController {
    private static final Log logger = LogFactory.getLog(SampleController.clas
s);
    private static final String BASE_URL = "sample/";

    @Autowired
    private SampleService sampleService;

    @Override
    protected String getBaseUrl() {
        return BASE_URL;
    }

    @RequestMapping( value = "records", method = RequestMethod.POST )
    public @ResponseBody
    JqGridResponseGeneric<SampleEntity> records( @RequestBody JqGridRequest j
qGridRequest ) {
        return sampleService.list(jqGridRequest);
    }

    @RequestMapping( value = "recordEdit", method = RequestMethod.POST )
    public @ResponseBody
    Map<String,Object> recordEdit( @RequestBody Map<String, Object> params )
{
        if ( params != null ) {
            JqGridCrudUtil<SampleEntity, Long>
                    jqgrid = new JqGridCrudUtil<>(SampleEntity.class, Long.cl
ass, params);

            switch (jqgrid.getOper()) {
                case ADD:
                {
                    SampleEntity newEntity = jqgrid.createEntityWithDataBindi
ng();
                    sampleService.save(newEntity);
                }
                break;
                case EDIT:
                {
                    Long id = jqgrid.getId();
                    SampleEntity entity = jqgrid.createEntityWithDataBinding(
);

                    SampleEntity updateEntity = sampleService.findById(id);

                    updateEntity.setAge(entity.getAge());
                    updateEntity.setName(entity.getName());
                    updateEntity.setDescription(entity.getDescription());

                    sampleService.deleteById(id);
                    sampleService.save(updateEntity);
                }
                break;
                case DELETE:
                {
                    Long[] ids = jqgrid.getIds();
```

```
                    sampleService.deleteByIds(ids);
                }
                break;
                default: break;
            }
        }// if ( params != null ) {

        return new HashMap<>();
    }
}
```

recordEdit 메소드 안에서 Insert, Update, Delete를 모두 해준다.

JqGridCrudUtil 클래스는 ajax JSON으로 받은 데이터를 SampleEntity로 변환하는 작업을 해준다.

이렇게 함으로 해서 기본적인 그리드는 list 메소드만 구현을 하고 Insert, Update, Delete는 상위 클래스에

구현해 놓을 것을 바로 사용할수 있다.

다시 페이지를 호출하고 CRUD를 해보자!

마치며

마치며

앞으로도 문서는 계속 보강하겠습니다. 이 프로젝트가 저를 비롯한 다른 개발자 분들에게 도움이 되었으면 합니다.

읽어 주셔서 감사합니다.