

**Wind Lidar 개발 환경 구성**



2017.02

본 문서는 한진정보통신에서 문서의 저작권을 소유하고 있으며, 제공 목적 이외의 용도로 사용할 수 없습니다. 본 문서와 관련하여 전체 또는 부분적으로 복제하거나 복사 또는 한진정보통신의 승인 없이는 어떤 사람에게 전송 될 수 없습니다

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 개 정 현 황 | | | | | |
| 개정번호 | 일 자 | 개 정 사 유 | 작 성 | 검 토 | 승 인 |
| V 1.0 | 2017.02.08 |  | 정민섭 |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

목 차

[1. FTP 설치 ４](#_Toc478032662)

[1.1 FileZilla Server 설치 ４](#_Toc478032663)

[1.2 FileZilla Server 실행 ６](#_Toc478032664)

[1.2.1 FTP Server 설정 ７](#_Toc478032665)

[1.2.2 FTP 사용자 등록 ８](#_Toc478032666)

[1.2.3 FTP 사용자 홈 디렉토리 설정 ９](#_Toc478032667)

[2. MariaDB 10.1 Series 설치 １０](#_Toc478032668)

[2.1 MariaDB Download １０](#_Toc478032669)

[2.2 MariaDB 설치 １０](#_Toc478032670)

[2.2.1 다운로드 받은 파일 실행 １０](#_Toc478032671)

[2.2.2 설치 완료 테스트 ２１](#_Toc478032672)

[2.3 MariaDB 설정 ２１](#_Toc478032673)

[2.3.1 Maria Db UTF-8 설정 ２２](#_Toc478032674)

[2.3.2 사용자 데이터베이스 생성 ２３](#_Toc478032675)

[2.3.3 사용자 계정 생성 ２４](#_Toc478032676)

[2.3.1 사용자 DB 권한 설정 ２４](#_Toc478032677)

[2.3.2 사용자 DB 접속 테스트 ２５](#_Toc478032678)

[3. MariaDB Table 생성 ２５](#_Toc478032679)

[3.1 MariaDB UTF-8 설정 ２５](#_Toc478032680)

[3.1.1 데이터베이스 문자셋 확인 ２５](#_Toc478032681)

[3.1.2 UTF-8 문자셋 설정 ２６](#_Toc478032682)

[3.2 테이블 생성 ２７](#_Toc478032683)

[4. MariaDB 개발 환경 구성 ２８](#_Toc478032684)

[4.1 Mysql-connector-net 설치 ２８](#_Toc478032685)

[4.2 Visual Studio에서 Mysql 참조하기 ２９](#_Toc478032686)

[5. INI 파일 구성 ３０](#_Toc478032687)

[5.1 WindLidar 서버 ３０](#_Toc478032688)

[5.2 WindLidar 클라이언트 ３２](#_Toc478032689)

[6. Visual Studio 2013에서 GitHub 연결 ３３](#_Toc478032690)

[6.1 GitHub 레파지토리 URL 확인 ３４](#_Toc478032691)

[6.2 Visual Studio에서 GitHub 연결 ３４](#_Toc478032692)

[6.3 Source Tree tool for GitHub ４０](#_Toc478032693)

[7. 웹 개발 환경 구성 ４０](#_Toc478032694)

[7.1 프로젝트 디렉토리 정의 ４０](#_Toc478032695)

[7.2 웹 개발 환경 버전 ４１](#_Toc478032696)

[7.2.1 Eclipse 다운로드 받기 ４１](#_Toc478032697)

[7.2.2 JDK 다운로드 하기 ４１](#_Toc478032698)

[7.2.3 Tomcat 다운로드 ４５](#_Toc478032699)

[7.2.4 Maven 다운로드 ４６](#_Toc478032700)

[7.3 웹 개발 환경 설정 ４７](#_Toc478032701)

[7.3.1 Tomcat Installation Identification ４７](#_Toc478032702)

[7.3.2 Maven Setup ５１](#_Toc478032703)

[7.3.3 STS (Spring Tool Suite) 설치 ５４](#_Toc478032704)

[7.3.4 Maven Integration Installation ５８](#_Toc478032705)

[7.3.5 Environment Variable Setup ５９](#_Toc478032706)

[8. Web Development with spring framework ６１](#_Toc478032707)

[8.1 Spring framework 개요 ６１](#_Toc478032708)

[8.2 Eclipse에서 Spring Project 생성 ６１](#_Toc478032709)

[8.2.1 Spring Project 생성 ６１](#_Toc478032710)

[8.2.2 Maven Dependencies ６４](#_Toc478032711)

[8.2.3 POM.xml ６５](#_Toc478032712)

[8.2.4 UTF-8 설정 ７２](#_Toc478032713)

[8.2.5 생성된 프로젝트 실행 ７２](#_Toc478032714)

[8.3 Mybatis 연동하기 ７６](#_Toc478032715)

[8.3.1 Mybatis개요 ７６](#_Toc478032716)

[8.3.2 Mybatis 라이브러리 ７７](#_Toc478032717)

[8.3.3 MyBatis와 데이터베이스 연결 설정 ７７](#_Toc478032718)

[8.3.4 MyBatis와 Spring 연결 ７９](#_Toc478032719)

[8.3.5 DAO 작성 ８０](#_Toc478032720)

[8.4 Application Development with MyBatis ８２](#_Toc478032721)

[8.4.1 List 데이터 가져오기 ８２](#_Toc478032722)

[8.4.2 HandlerMethodArgumentResolver 적용 ９０](#_Toc478032723)

[8.4.3 사용자 관리 프로그램 개발 ９６](#_Toc478032724)

# FTP 설치

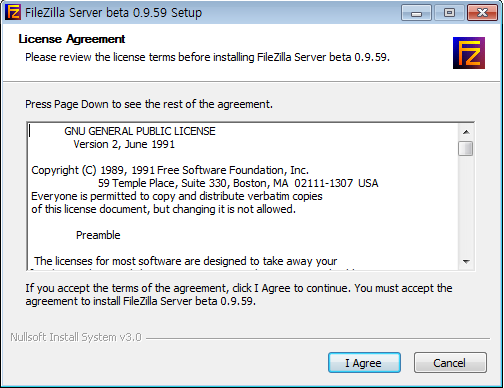
## FileZilla Server 설치

WindLidar 데이터를 수신하기 위해서는 Workstation에 FTP Server를 설치해야 한다. 무료로 배포되는 FTP Server로서 FileZilla Server가 있다. 이 프로그램을 설치한다.

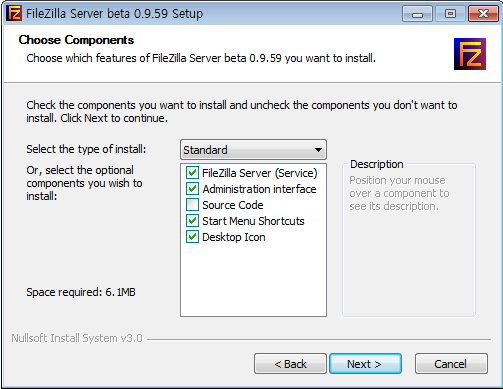
* 설치할 파일

FileZilla\_Server-0\_9\_59.ext (window 용)

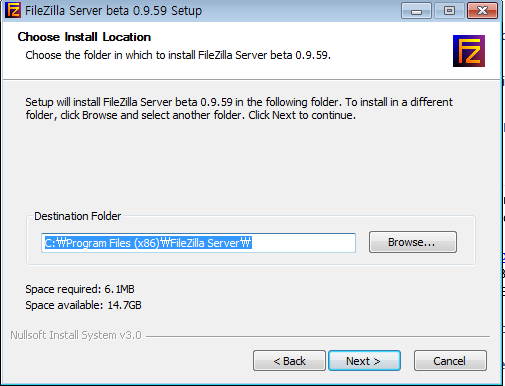
* 설치



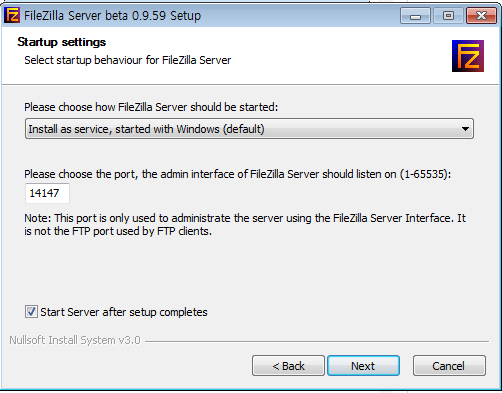
“I Agree” 동의 버튼을 클릭해서 다음으로 넘어간다.



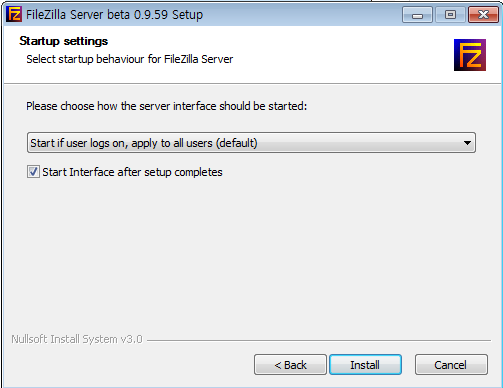
Select the type of install을 Standard로 선택하고 Install component를 “FileZilla Server(Service), “Administration interface”, “Start Menu Shortcuts”, “Desktop Icon”을 체크한 후 Next 버튼을 클릭한다.



설치할 폴더를 선택하고 Next 버튼을 클릭한다.

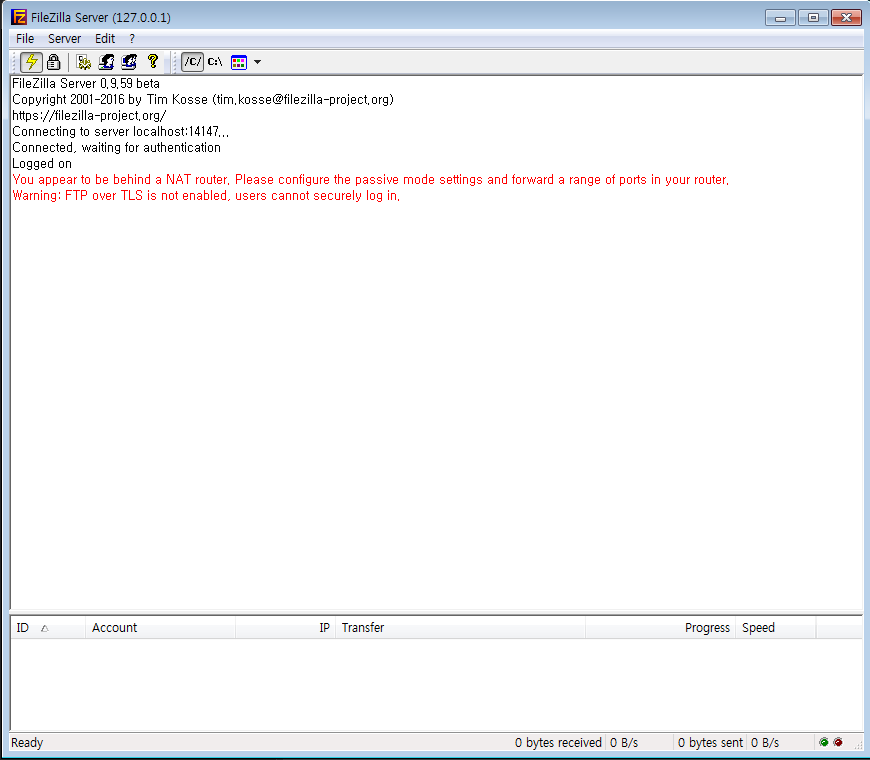


위의 Port 번호는 FTP Port 번호가 아니다. 관리자 접속 Port이다. 정보만 확인하고 Next 버튼을 클릭한다.



“Install” 버튼을 클릭해서 설치를 수행한다.

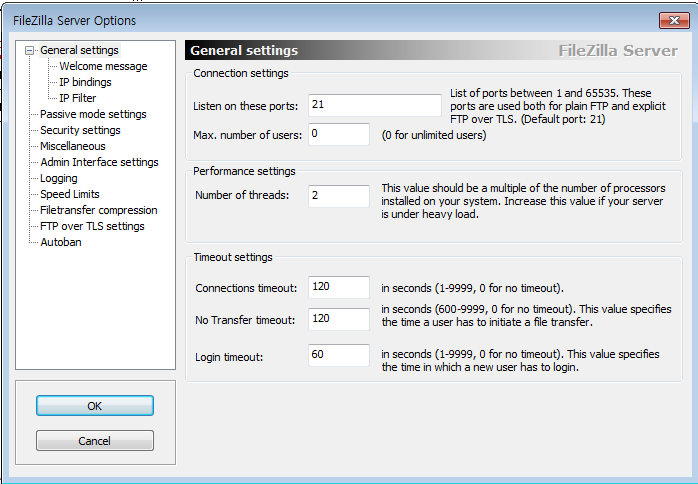
## FileZilla Server 실행



FileZilla Server를 가동하고 최초 가동 시 패스워드를 입력란이 나오면 관리자 패스워드를 신규 입력한다. 입력이 완료되고 확인을 누르면 위와 같은 화면이 현시된다.

### FTP Server 설정

Edit 메뉴에서 Setting 메뉴를 클릭한다.

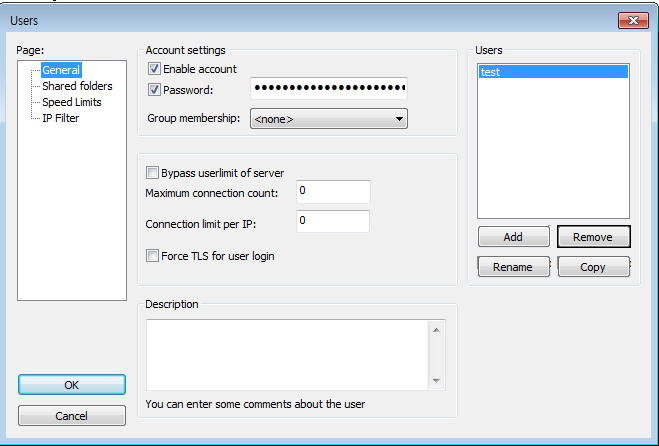


FTP Port, Max number of Users, Number of threads 등을 설정한다.

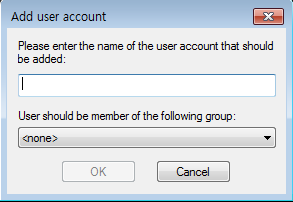
그 외에 Security settings 등 다양한 설정을 수행할 수 있다.

### FTP 사용자 등록

Edit메뉴에서 User를 클릭해서 사용자 생성 화면을 현시한다.



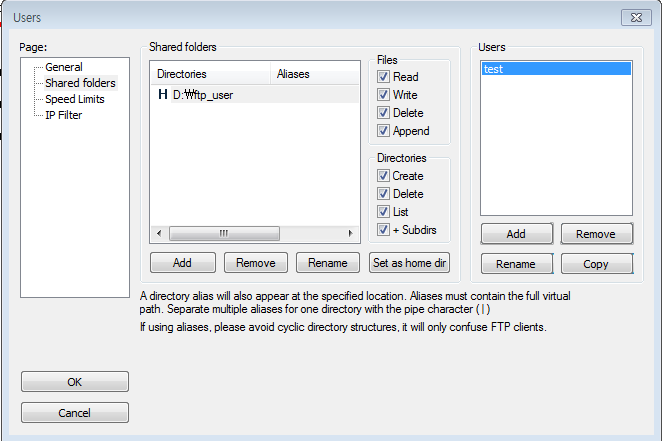
Users 란에서 Add 버튼을 클릭해서 사용자를 등록한다.



사용자 아이디를 입력하고 OK버튼을 클릭하면 사용자가 등록된다. 등록된 사용자에 대해서 Account settings 란에서 패스워드를 입력한다.

### FTP 사용자 홈 디렉토리 설정

Shared folders 페이지에서 사용자 폴더와 File, Directory에 대한 권한을 설정한다.



MariaDB 10.1 Series 설치

## MariaDB Download

* 공식 홈페이지

www.mariadb.org

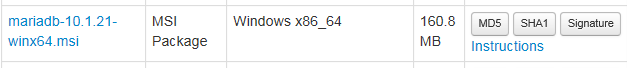
* 다운로드 홈페이지

<https://downloads.mariadb.org/>

* 다운로드 버전

MariaDB 10.1 Series

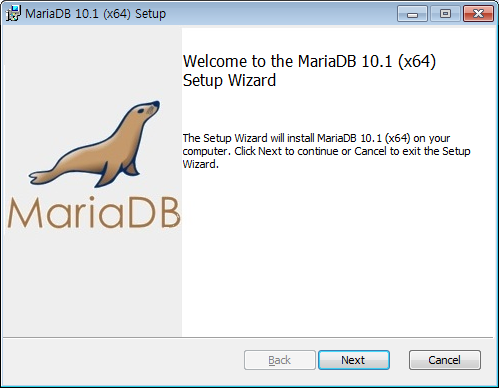




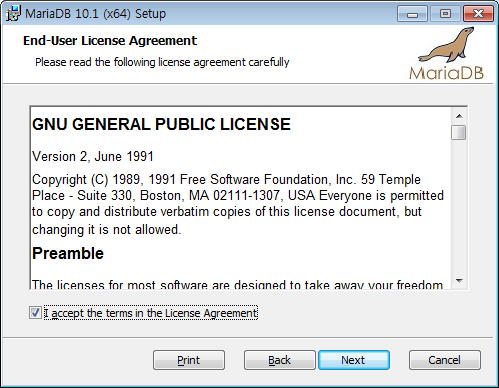
## MariaDB 설치

### 다운로드 받은 파일 실행

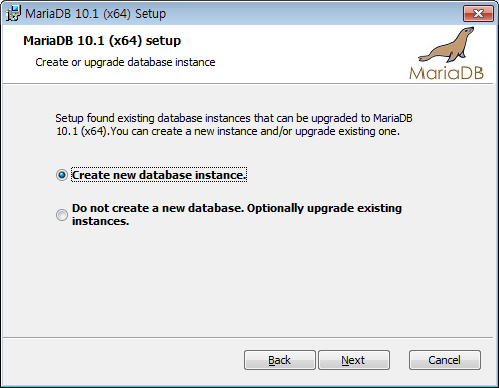
mariadb-10.1.21-winx64.msi 파일을 실행한다.



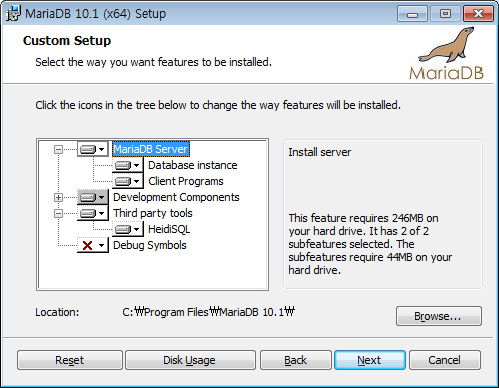
[Next] 버튼을 클릭한다.



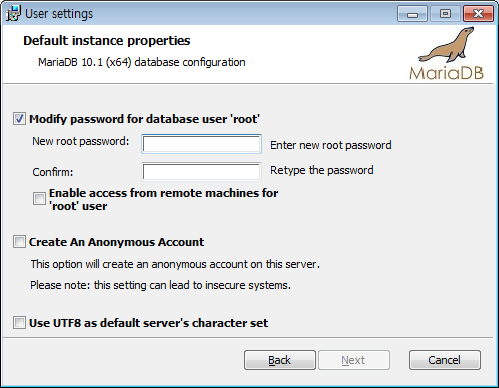
“I accept the terms in the License Agreement” 체크박스에 체크하고 [Next]버튼을 클릭한다.



“Create new database instance” 옵션박스를 체크하고 [Next]버튼을 클릭한다.

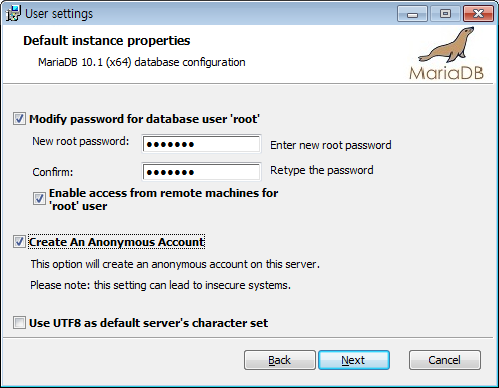


설치 위치를 확인하고 [Next]버튼을 클릭한다.

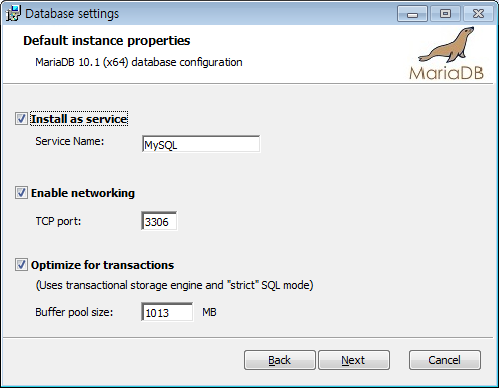


‘root’ 사용자의 패스워드를 설정한다. 이 패스워드는 반드시 기억해야 한다.

패스워드는 ‘hist123’으로 설정한다. 또한 원격 머신에서 접속 가능하도록 체크한다. 그리고 Anonymouse 계정도 등록한다.



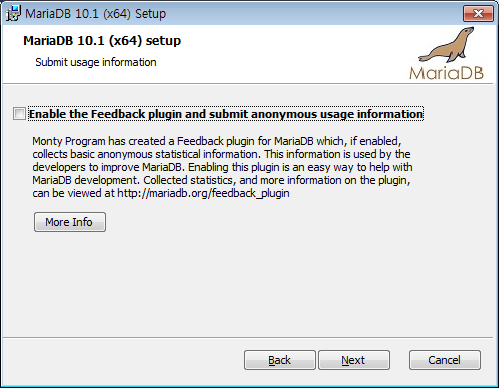
[Next] 버튼을 클릭한다.



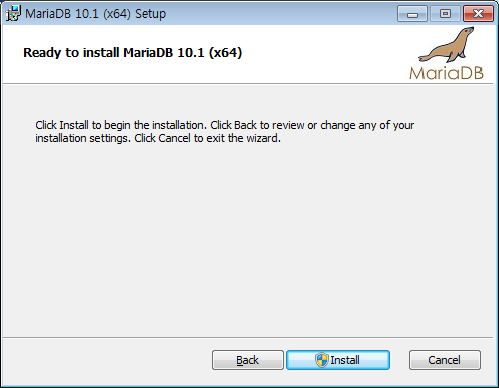
‘Install as service’, ‘TCP port’, ‘Optimize for transactions’을 체크 및 확인 후 [Next] 버튼을 클릭한다

Test 환경

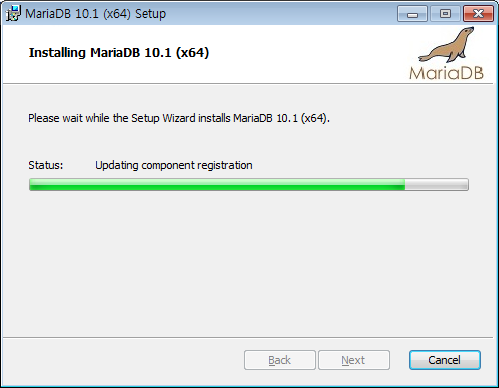
* Service Name : MariaDB
* TCP port : 3303

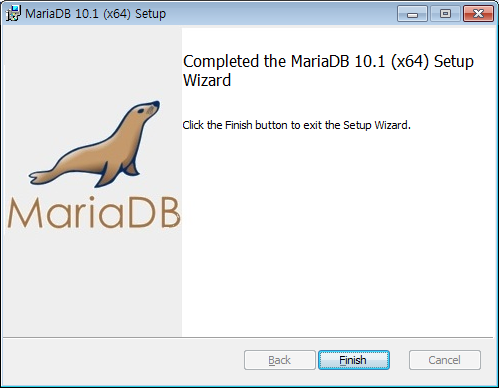


[Next] 버튼을 클릭한다.



[Install] 버튼을 클릭한다.





[Finish] 버튼을 클릭해서 설치를 마친다.

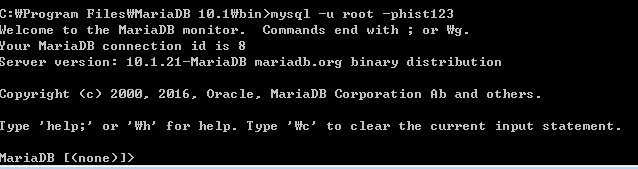


설치가 완료되면 위와 같이 메뉴가 구성된다.

### 설치 완료 테스트

C:> mysql –u root –phist123

command 창에서 위와 같이 입력 후 Enter key를 눌렀을 때 아래와 같이 나오면 설치가 완료된 것이다.



## MariaDB 설정

* 데이터베이스명 : windlidarDB
* 사용자 명 : histUser
* 패스워드 : hist123

### Maria Db UTF-8 설정

설치 디렉토리에서 data 디렉토리에 보면 my.ini 파일이 있다. 이 파일을 열어서 아래와 같이 수정한다.

|  |
| --- |
| [mysqld]  datadir=C:/Program Files/MariaDB 10.1/data  port=3303  sql\_mode="STRICT\_TRANS\_TABLES,NO\_ENGINE\_SUBSTITUTION"  default\_storage\_engine=innodb  innodb\_buffer\_pool\_size=1013M  innodb\_log\_file\_size=50M  character-set-client-handshake  init\_connect="SET collation\_connection=utf8\_general\_ci"  init\_connect="SET NAMES utf8"  character-set-server=utf8  collation-server=utf8\_general\_ci  [client]  port=3303  plugin-dir=C:/Program Files/MariaDB 10.1/lib/plugin  default-character-set=utf8  [mysql]  default-character-set=utf8  [mysqldump]  default-characer-set=utf8 |

### 사용자 데이터베이스 생성

Database 명 : **windlidarDB**

|  |
| --- |
| create database windlidarDB DEFAULT CHARACTER SET utf8 collate utf8\_general\_ci;  show databases;  create user histUser;  set password for 'histUser' = password('hist123');  grant all privileges on windlidarDB.\* to 'histUser'@'%';  grant select, insert, update, delete, create, drop, alter on \*.\* to 'histUser'@'%';  flush privileges; |

|  |
| --- |
| MariaDB[<none>]> use mysql;  Database changed  MariaDB [mysql]> create database **windlidarDB;**  **Query OK, 1 row affected <0.00 sec>**  **MariaDB [mysql]> show databases;** |

### 사용자 계정 생성

|  |
| --- |
|  |

### 사용자 DB 권한 설정

|  |
| --- |
| MariaDB [mysql]> grant all privileges on windlidarDB.\* to ‘histUser’@’%’;  Query OK, 0 rows affected <0.00 sec> |

grant select, insert, update, delete, create, drop, alter on \*.\* to 'histUser'@'%';

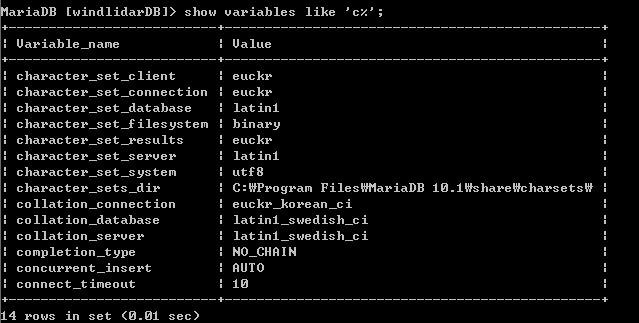
### 사용자 DB 접속 테스트

|  |
| --- |
|  |

MariaDB Table 생성

## MariaDB UTF-8 설정

### 데이터베이스 문자셋 확인



### UTF-8 문자셋 설정

설치 디렉토리에서 data 디렉토리에 보면 my.ini 파일이 있다. 이 파일을 열어서 아래와 같이 수정한다.

|  |
| --- |
| [mysqld]  datadir=C:/Program Files/MariaDB 10.1/data  port=3303  sql\_mode="STRICT\_TRANS\_TABLES,NO\_ENGINE\_SUBSTITUTION"  default\_storage\_engine=innodb  innodb\_buffer\_pool\_size=1013M  innodb\_log\_file\_size=50M  character-set-client-handshake=FALSE  init\_connect="SET collation\_connection = utf8\_general\_ci"  init\_connect="SET NAMES utf8"  default-character-set = utf8  character-set-server = utf8  collation-server = utf8\_general\_ci  [client]  port=3303  plugin-dir=C:/Program Files/MariaDB 10.1/lib/plugin  default-character-set = utf8  [mysql]  default-character-set = utf8  [mysqldump]  default-characer-set = utf8 |

## 테이블 생성

|  |
| --- |
| -- User table  create table t\_usr\_info  (  id varchar(20) primary key not null,  name varchar(20) not null,  pass varchar(60) not null,  email varchar(120) ,  auth\_chk varchar(1),  last\_dt datetime,  reg\_dt datetime not null  ) default character set utf8 collate utf8\_general\_ci;  -- 파일 전송  create table t\_rcv\_info  (  no int(11) primary key auto\_increment not null,  s\_code varchar(3),  s\_year varchar(4),  s\_mon varchar(2),  s\_day varchar(2),  s\_hour varchar(2),  s\_min varchar(2),  s\_sec varchar(2),  file\_cnt int(11),  use\_chk varchar(1) not null,  file\_name1 varchar(200),  file\_name2 varchar(200),  reg\_dt datetime,  upt\_dt datetime  )default character set utf8 collate utf8\_general\_ci; |

MariaDB 개발 환경 구성

## Mysql-connector-net 설치

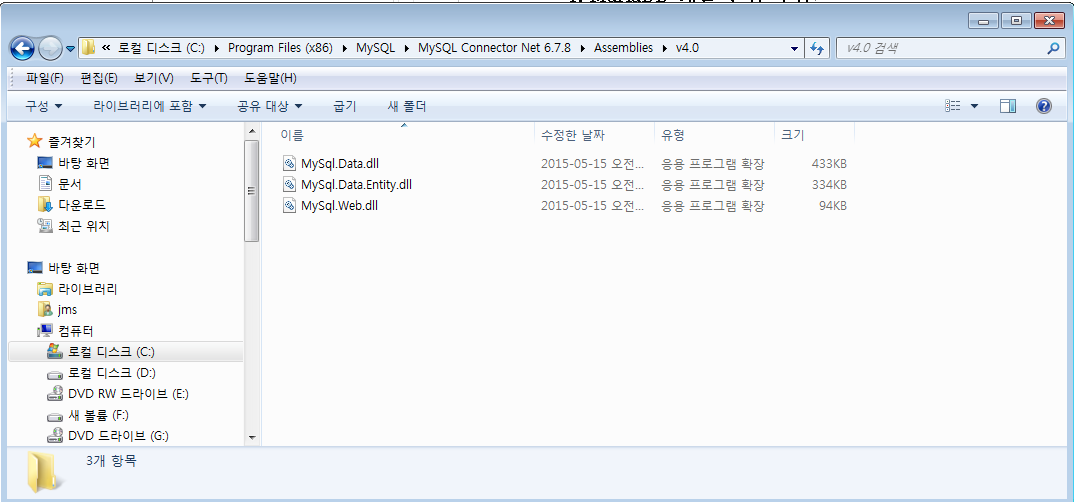
C#으로 개발 위해 Mysql connector가 필요하다. 해당 파일은 아래에서 다운로드 받는다.

<http://download.softagency.net/MySQL/Downloads/Connector-Net/>

다운로드 파일 : mysql-connector-net-6.7.9.msi

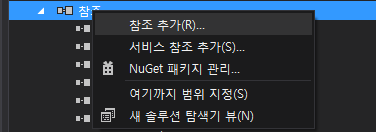
설치가 완료되면 아래의 폴더에 설치된다.

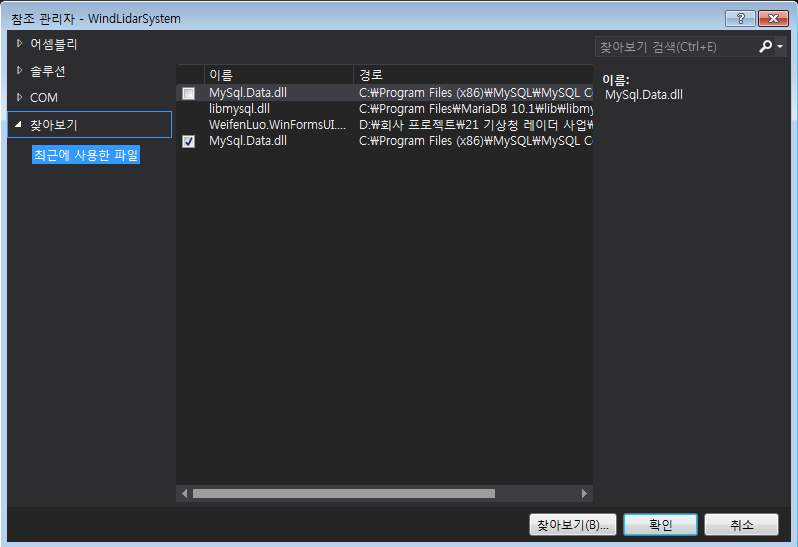
C:\Program Files (x86)\MySQL\MySQL Connector Net 6.7.8\Assemblies\v4.0



## Visual Studio에서 Mysql 참조하기

Visual Studi에서 솔루션 탐색기 > 참조를 클릭한 후 해당 참조란에 Mysql 라이브러리를 등록한다.





[찾아보기] 버튼을 클릭한 후 Mysql.Data.dll을 추가한다. Net 버전 4.0을 선택한다. (Visual Studio 2013 버전)

using문은 아래와 같이 참조한다.

using MySql.Data.MySqlClient;

INI 파일 구성

## WindLidar 서버

WindLidar 서버 프로그램은 기본 환경정보를 ini파일에 등록해서 사용한다.

위치는 D:\WindLidarServer.ini 에 위치한다.

|  |
| --- |
| [WindLidarSystem]  ListenPort=10001  ClientPort=10002  DataClientPort=10003  LocalPort=10004  UWA\_FTP\_IP=localhost  UWA\_FTP\_PORT=21  UWA\_FTP\_ID=server  UWA\_FTP\_PASS=server123  DB\_NAME=windlidarDB  DB\_USER=histUser  DB\_PASS=hist123  DB\_HOST=localhost  DB\_PORT=3303  FTP\_THREAD\_TIME=10  STS\_THREAD\_TIME=180  SOURCE\_PATH=D:\\ftp\_user  BACKUP\_PATH=D:\\backup |

|  |  |  |
| --- | --- | --- |
| 파라미터 | 값 | 설명 |
| ListenPort | 10001 | Listen Port로 클라이언트는 이 Port로 접속한다. |
| ClientPort | 10002 | Client에게 보낼 때 사용하는 Port |
| LocalPort | 10004 | Client에게 보낼 때 사용하는 로컬 Port |
| UWA\_FTP\_IP | localhost | UWA FTP Server IP Address |
| UWA\_FTP\_PORT | 21 | UWA FTP Server Port |
| UWA\_FTP\_ID | Server | UWA FTP User ID |
| UWA\_FTP\_PASS | server123 | UWA FTP User Password |
| DB\_NAME | windlidarDB | WindLidar 데이터베이스명 |
| DB\_USER | histUser | 데이터베이스 사용자명 |
| DB\_PASS | hist123 | 데이터베이스 사용자 패스워드 |
| DB\_HOST | localhost | 데이터베이스 호스명 |
| DB\_PORT | 3303 | 데이터베이스 Port |
| FTP\_THREAD\_TIME | 10 | 데이터 전송 Thread time으로 10 second |
| STS\_THREAD\_TIME | 180 | 클라이언트 프로그램의 상태 체크 Thread time(second) |
| DataClientPort | 10003 | Client의 수신포트로 관측 데이터에 대한 수신 결과를 전송한다. |
| SOURCE\_PATH | D:\\ftp\_user | 서버의 FTP Folder(프로그램에서 직접 참조하는 디렉토리) |
| BACKUP\_PATH | D:\\backup | 관측 데이터를 백업할 디렉토리 |

## WindLidar 클라이언트

WindLidar 클라이언트 프로그램은 기본 환경정보를 ini파일에 등록해서 사용한다.

위치는 D:\WindLidarClient.ini 에 위치한다.

|  |
| --- |
| [WindLidarClient]  SndLocalPort=10009  FTP\_IP=localhost  FTP\_PORT=21  FTP\_ID=test  FTP\_PASS=test123  ST\_IP=localhost  ST\_PORT=10001  ST\_RCV\_PORT=10002  FT\_RCV\_PORT=10003  ST\_CODE=13211  SLEEP\_TIME=60  FTP\_URI=ftp://  SOURCE\_PATH=D:\\KoreaLidar  BACKUP\_PATH=D:\\KoreaLida |

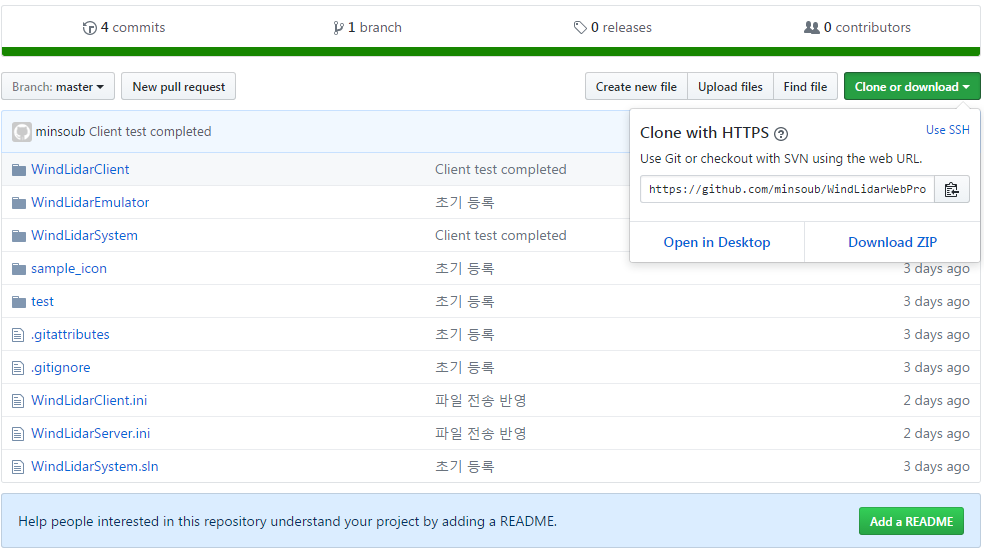
|  |  |  |
| --- | --- | --- |
| 파라미터 | 값 | 설명 |
| SndLocalPort | 10009 | 클라이언트에서 상태 데이터를 보낼 때 사용하는 Local Port |
| FTP\_IP | localhost | 수집 서버 FTP Address |
| FTP\_PORT | 21 | 수집 서버 FTP Port |
| FTP\_ID | test | 수집 서버 FTP User ID |
| FTP\_PASS | test123 | 수집 서버 FTP User Password |
| ST\_IP | localhost | 클라이언트 프로세스 상태를 보낼 서버 IP |
| ST\_PORT | 10001 | 클라이언트 프로세스 상태를 보낼 서버 Port |
| ST\_RCV\_PORT | 10002 | 상태값 전송에 대한 응답 수신 Port |
| FT\_RCV\_PORT | 10003 | 관측 데이터 전송에 대한 응답 수신 Port |
| ST\_CODE | 13211 | 사이트 코드  클라이언트 프로그램은 이 코드로 자기의 위치를 확인한다. |
| SLEEP\_TIME | 60 | Sleep time |
| FTP\_URI | ftp:// | FTP URI |
| SOURCE\_PATH | D:\\KoreaLidar | 윈드라이더가 데이터를 저장하는 위치 |
| BACKUP\_PATH | D:\\KoreaLidar | 전송 후 데이터를 백업할 저장 위치 |

Visual Studio 2013에서 GitHub 연결

윈드라이더 자료 전송 프로그램에서 클라이언트/서버 프로그램은 Visual Studio 2013 툴에서 C#으로 개발하였다. 따라서 소스 관리를 위해서 GitHub에 연결해서 관리한다.

## GitHub 레파지토리 URL 확인

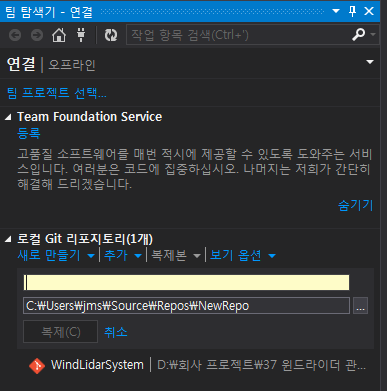
GitHub에 가입하고 나서 레파지토리 생성 후 레파지토리 URL을 확인한다.



<https://github.com/minsoub/WindLidarProject.git>

## Visual Studio에서 GitHub 연결

Visual Studio에서 팀 탐색기 탭을 클릭해서 로컬 Git 리포지토리에서 복제본을 클릭한다.

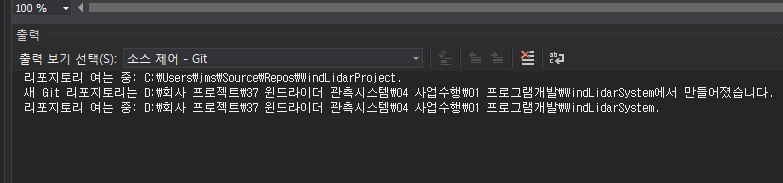


입력란에 위에서 카피한 <https://github.com/minsoub/WindLidarProject.git>

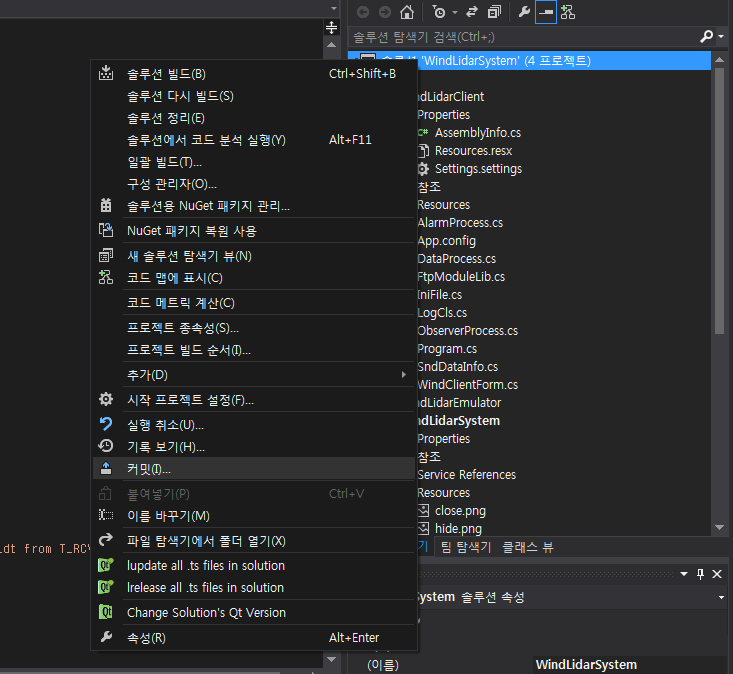
을 입력한다. 입력 후 복제할 디렉토리를 지정한다. 지정 후 [복제] 버튼을 클릭해서 복제를 수행한다.

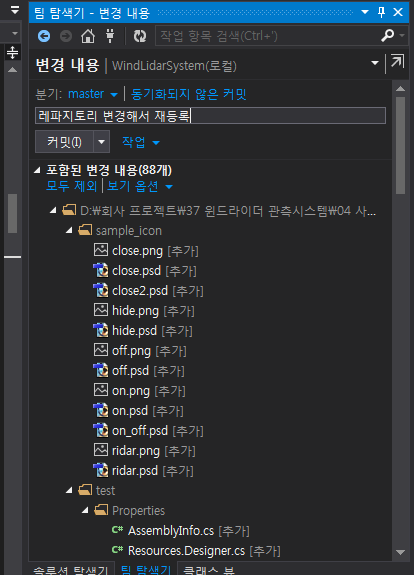
복제가 완료되었다면 기존 소스가 있으면 소스를 등록한다.

솔루션 탐색기에서 솔루션을 선택해서 소스 버전 컨트롤 메뉴를 클릭한다.

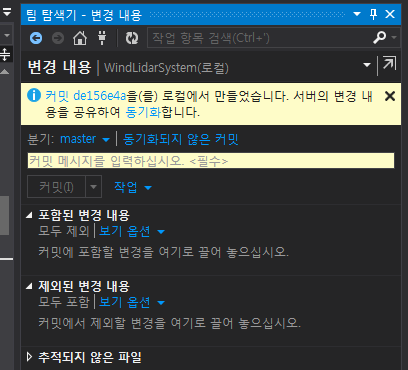


레파지토리를 열어서 등록한다. 등록이 완료되면 소스를 커밋한다.

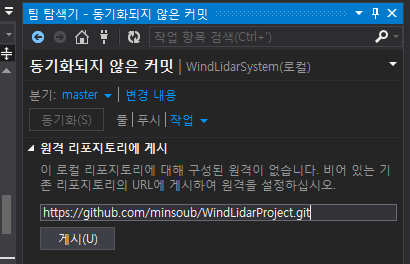




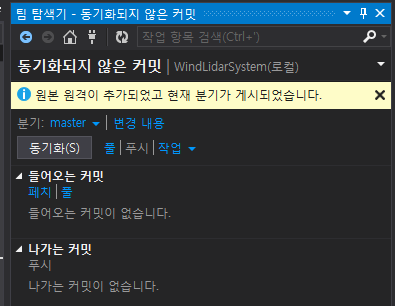
커밋 내용을 입력하고 커밋 버튼을 클릭한다.



동기화 여부를 물으면 동기화를 클릭합니다.

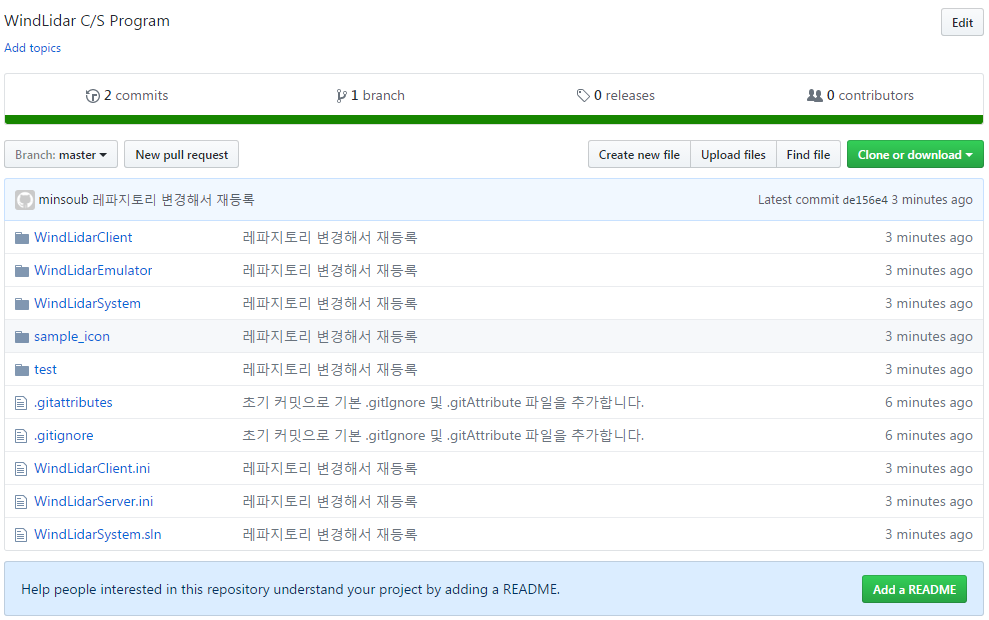


원격 리포지토리에 게시가 나오면 URL을 입력하고 게시 버튼을 클릭한다.



동기화 버튼을 클릭하여 작업을 마무리한다.

GitHub에서 올바르게 등록되었는지 확인한다.

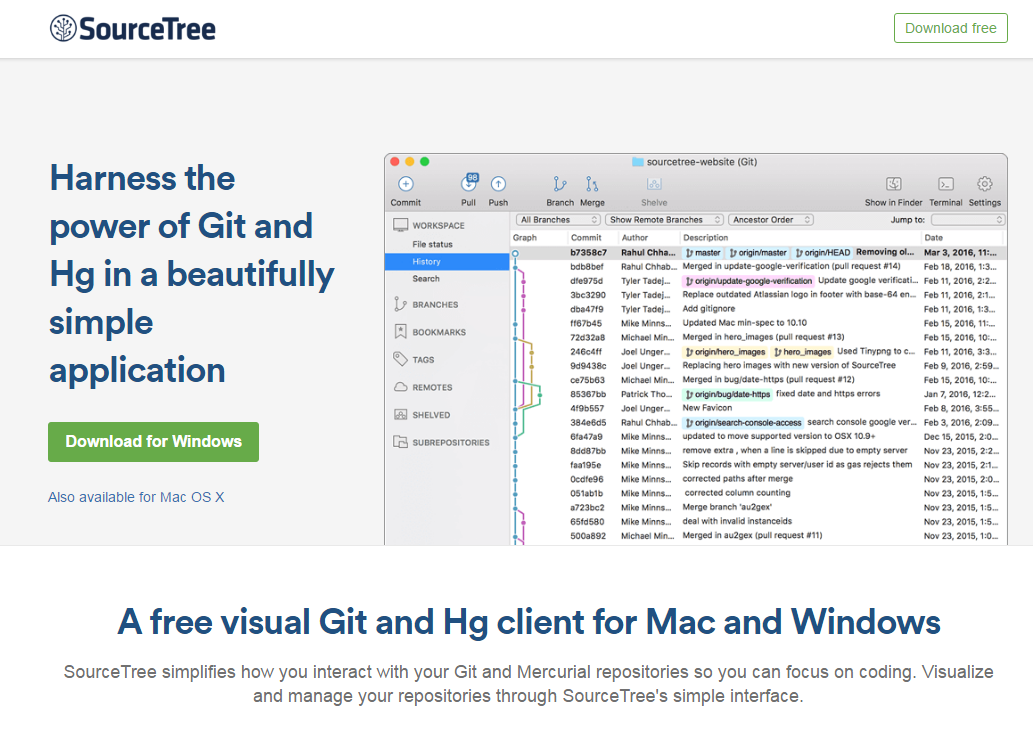


## Source Tree tool for GitHub

GitHub를 Visual적으로 보기 위해서 Source Tree 툴을 설치한다.

Source Tree 툴은 아래 URL로 접속해서 다운로드 받는다.

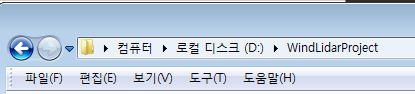
<https://www.sourcetreeapp.com/>



웹 개발 환경 구성

## 프로젝트 디렉토리 정의

프로젝트를 한 곳에 관리하기 위해서 디렉토리를 D:\WindLidarProject 폴더로 정의한다. (C:\ProjectWorkspace\WindLidarProject)

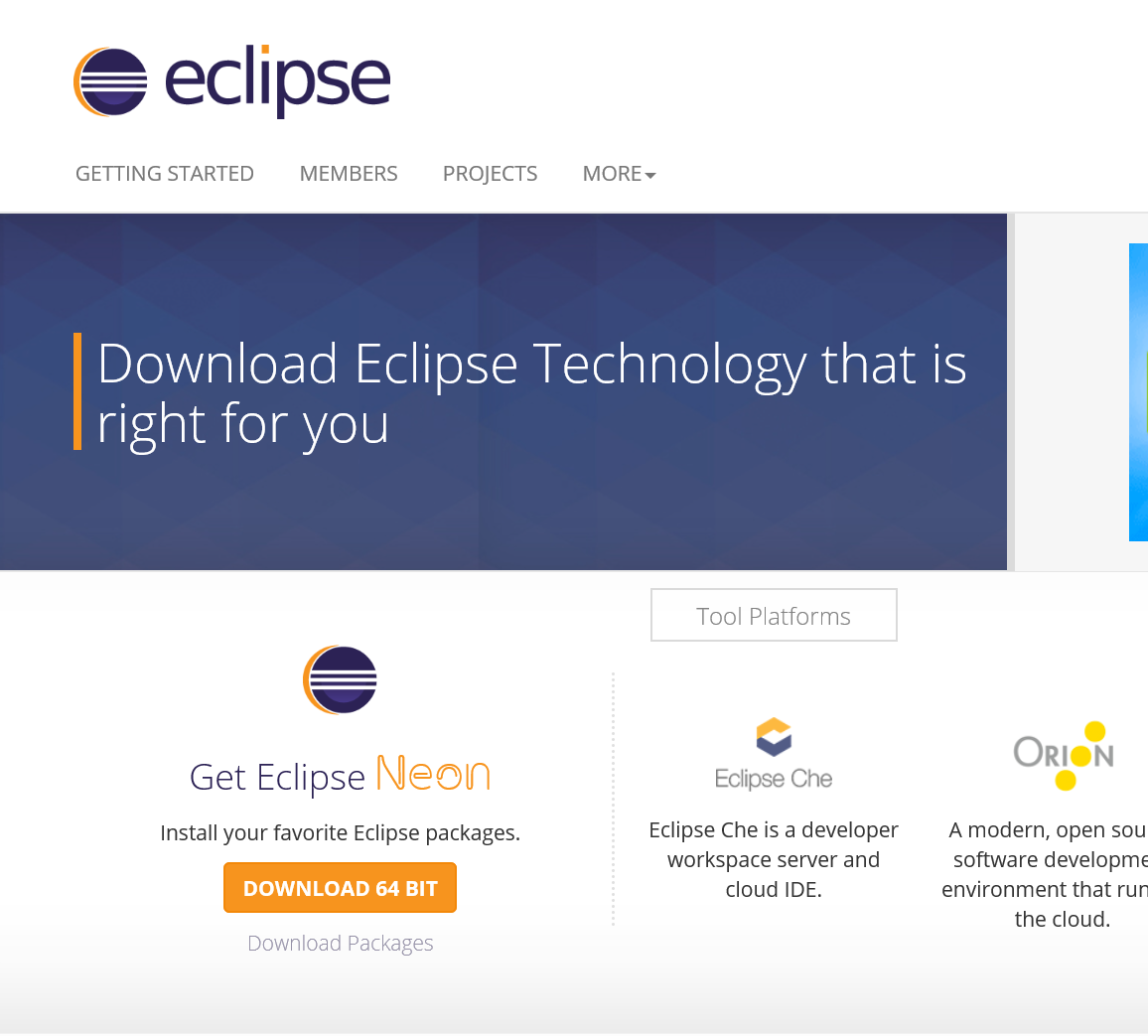


## 웹 개발 환경 버전

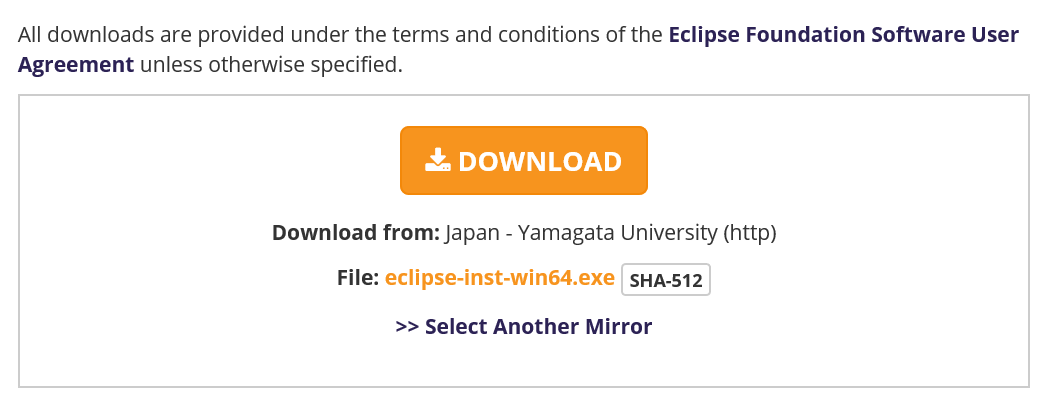
|  |  |
| --- | --- |
| OS | Window 7, 8, 9, 10 |
| IED | Eclipse Neon |
| JDK | JAVA SE 8u121 |
| Tomcat | Tomcat 7.0.75 |
| Maven | Maven 3.3.9 |
| MariaDB | MariaDB 10.1 Series |

### Eclipse 다운로드 받기

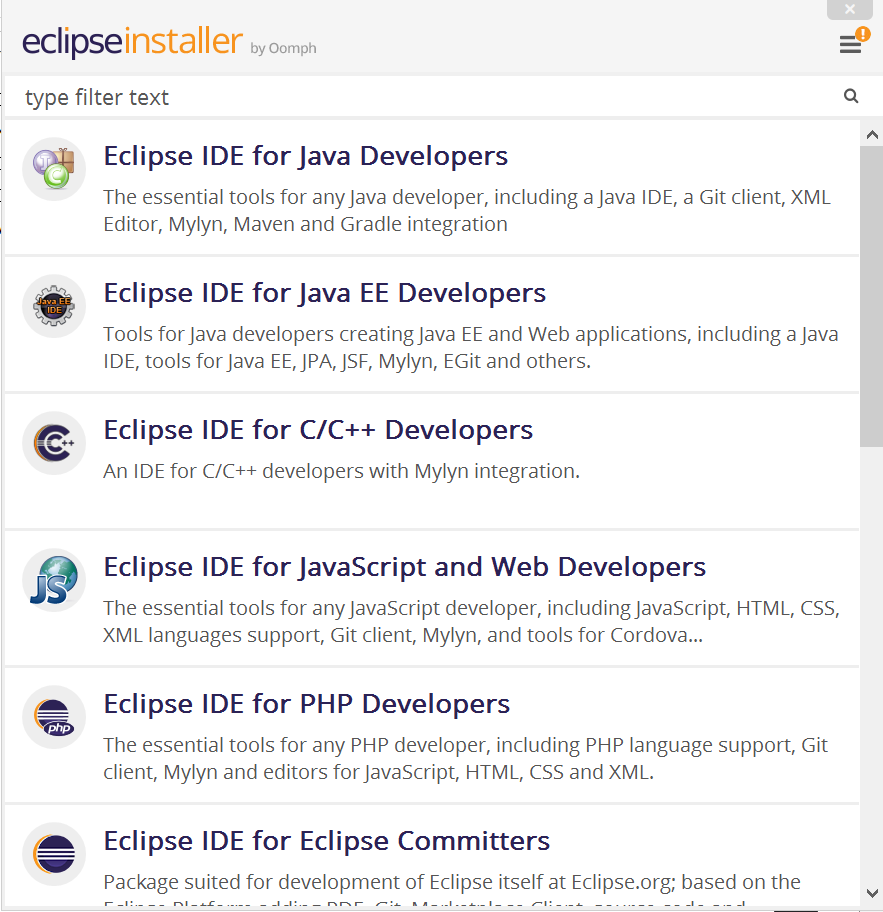
Eclipse.org에 접속해서 윈도우용 Eclipse IDE for JAVA EE Developers 를 다운로드 받는다.



Get Eclipse Neon에서 [DOWNLOAD 64 BIT] 버튼을 클릭한다.



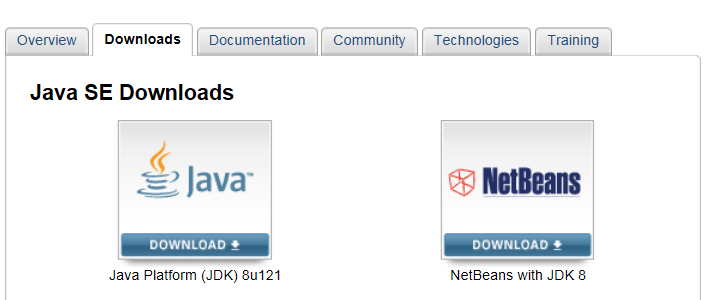
Eclipse-inst-win64.exe를 다운로드 받는다. Elicpse-inst-win64.exe 파일은 Eclipse 인스톨 도움 파일이다. 설치하기 전에 JDK를 먼저 설치해야 한다.



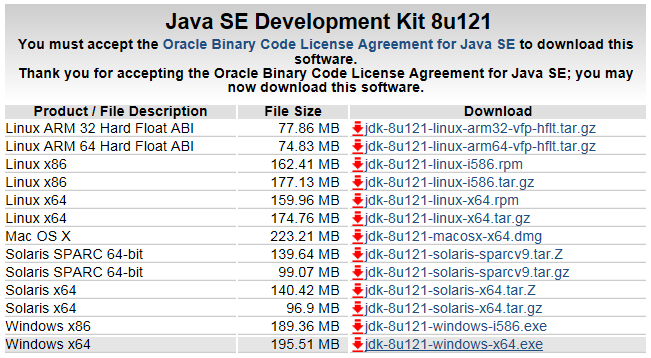
인스톨 프로그램을 실행시키면 위와 같은 화면이 나온다. 위에서 Eclipse IDE for Java EE Developers 다운로드 받아서 설치한다.

### JDK 다운로드 하기

<http://www.oracle.com/technetwork/java/javase/downloads/index.html> 접속

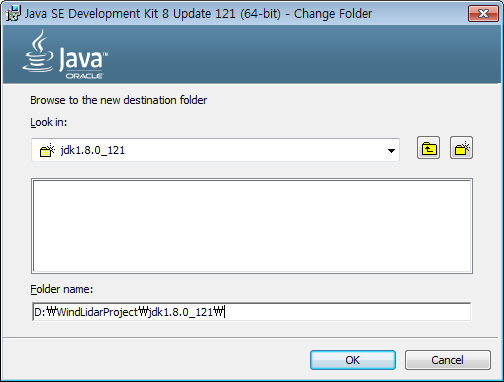


Java Plaftform (JDK) 8u121 를 클릭하면 아래의 화면이 출력된다.

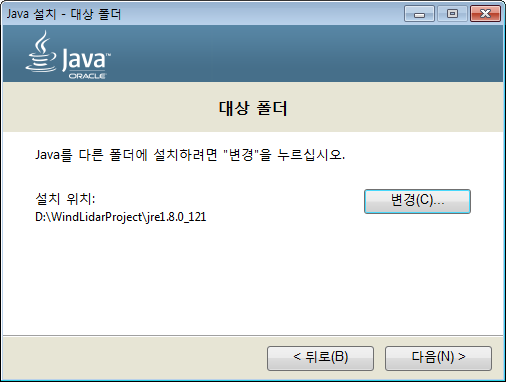


Windows x64 버전을 다운로드 받는다. => jdk-8u121-windows-x64.exe

다운로드 파일을 실행시켜 설치를 진행한다. 설치 시 위에서 설정한 기본 디렉토리에 설치한다.

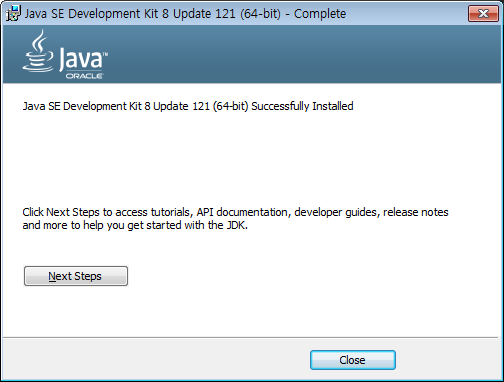


D:\WindLidarProject\java\jdk1.8.0\_121 폴더에 설치한다.



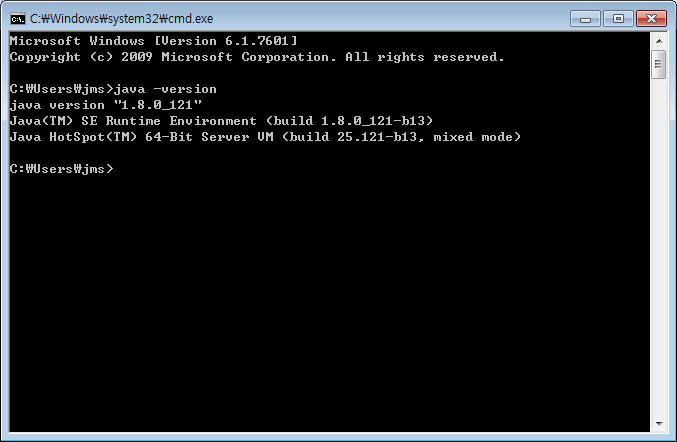
설치 진행중 Jre 설치 폴더가 나오면 위에서 설정한 폴더와 마찬가지로 설정한다.

D:\WindLidarProject\java\jre1.8.0\_121 폴더에 설치한다.



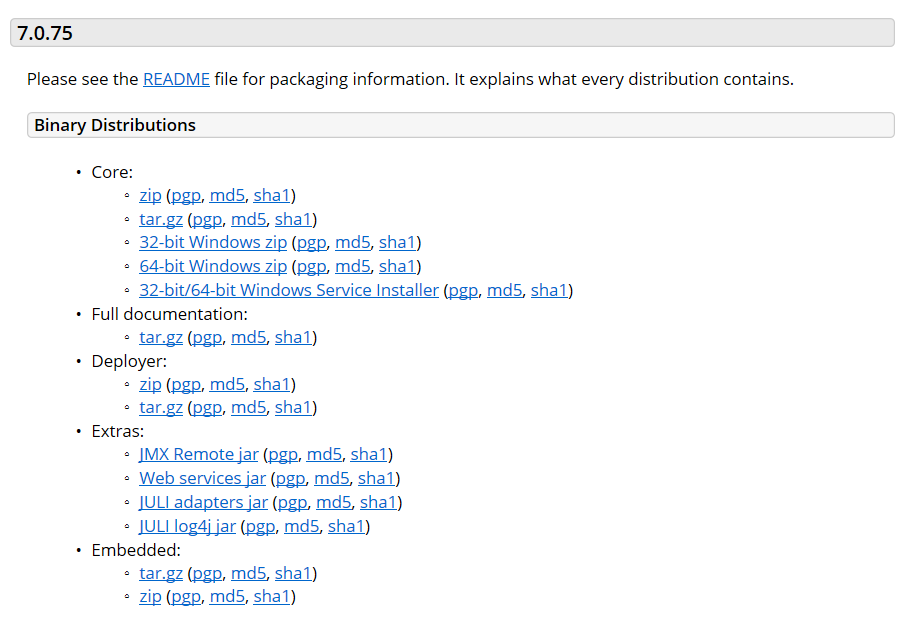
윈도우용은 기본적으로 자바를 설치하면 자동적으로 Path가 설정된다.

윈도우 Command창에서 java- version으로 확인해 본다.



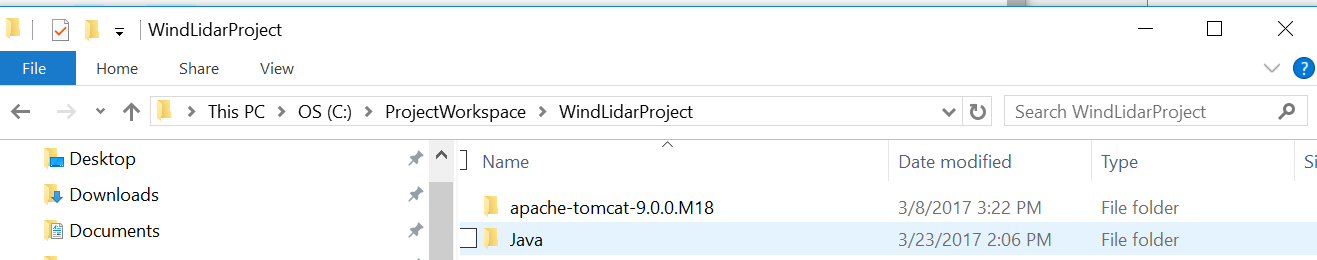
### Tomcat 다운로드

<http://tomcat.apache.org/download-70.cgi> URL에 접속한다. 접속페이지 아래 Binary Distributions 항목에 보면 64-bit Windows.zip이 있다. 이 파일을 다운로드 받아서 프로젝트 폴더에 복사한다.



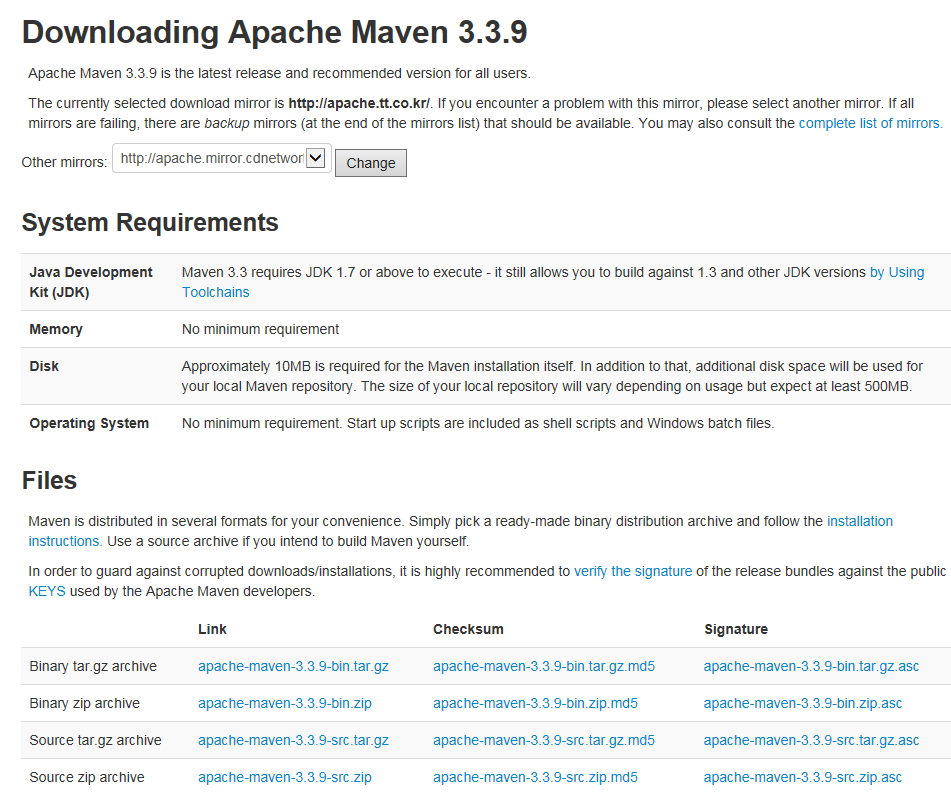
D:\WindLidarProject\apache-tomcat-7.0.75 폴더

최신버전은 apache-tomcat-9.0.0.M18 버전이다. 압축파일을 다운로드 받은 후 압축을 풀어서 폴더를 프로젝트 폴더에 복사한다.



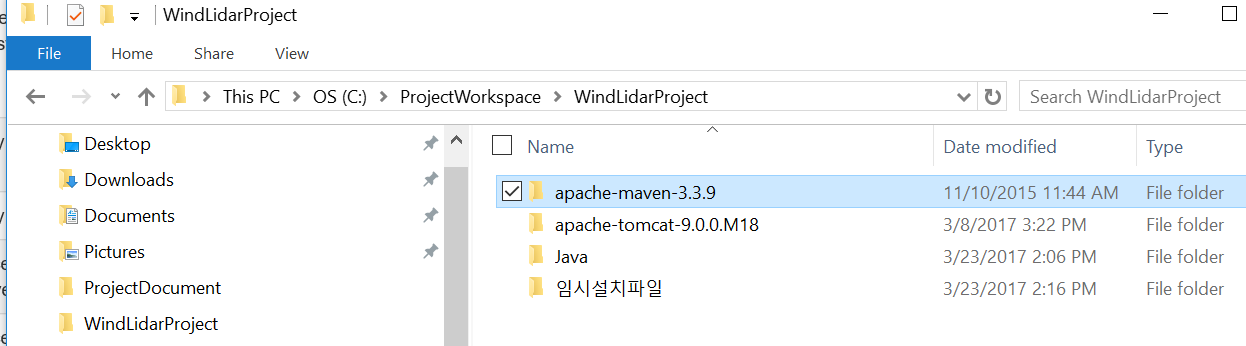
### Maven 다운로드

<https://maven.apache.org/download.cgi> URL에 접속한다.



Binary zip archive에서 **apache-maven-3.3.9-bin.zip** 파일을 다운로드 받는다.

압축을 푼 후 폴더를 D:\WindLidarProject 폴더에 복사한다.

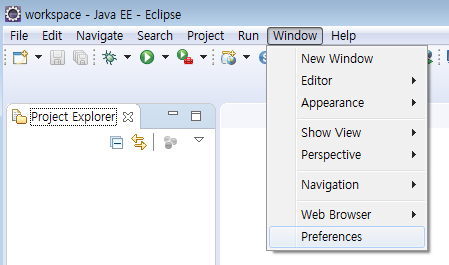


## 웹 개발 환경 설정

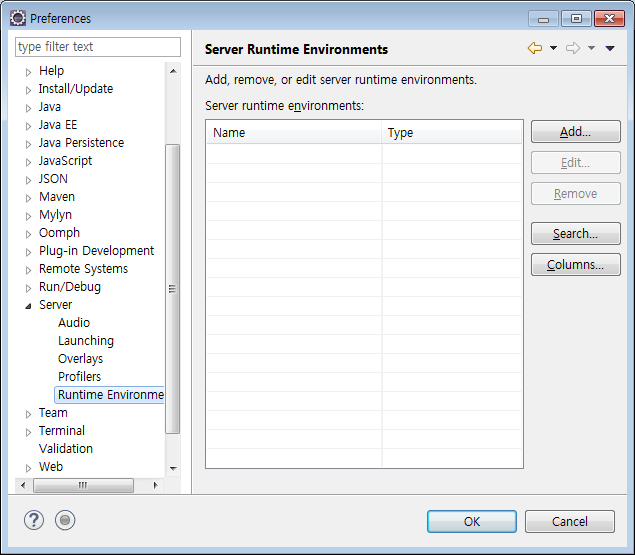
### Tomcat Installation Identification

Eclipse를 실행하여 아래와 같이 설정한다.

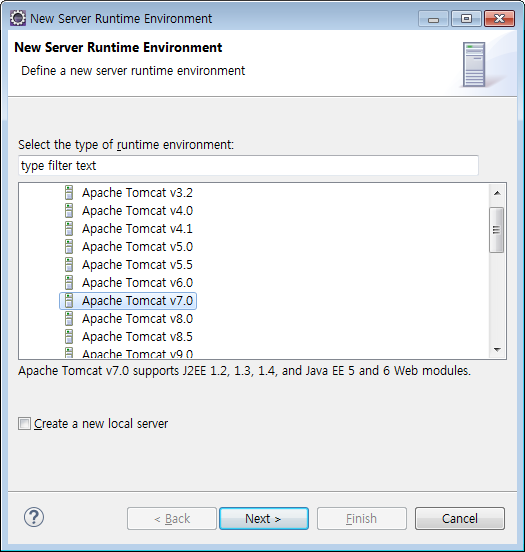
상단 메뉴에서 Window->Preference를 선택



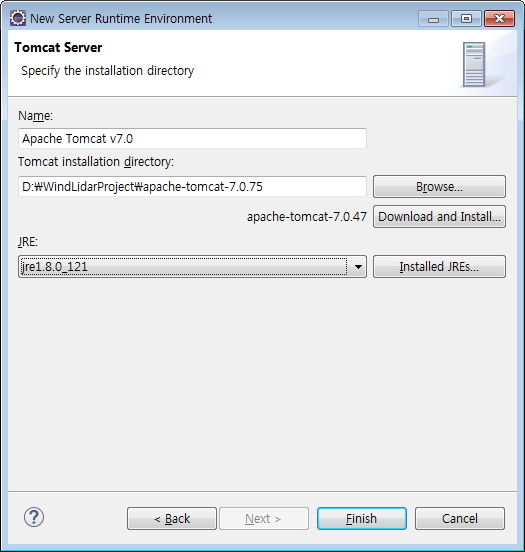
Server->Runtime Environment를 선택한다.



Add 버튼을 추가한다.



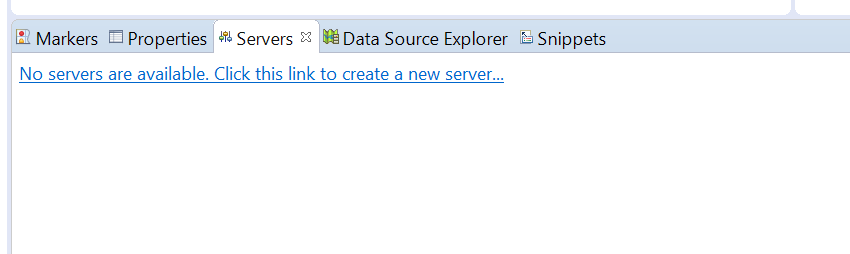
Apache Tomcat v7.0을 선택하고 ‘Next’버튼을 클릭한다.



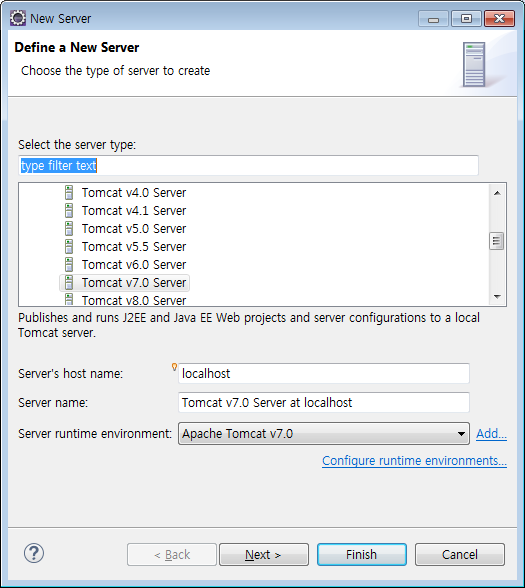
Finish 버튼을 클릭해서 설정을 완료한다.

Window->Show View에서 Servers를 선택해서 확인한다.

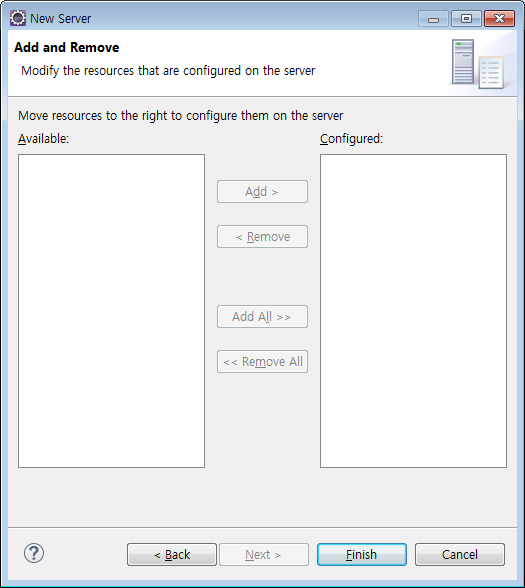
[Servers]항목이 생기면 바로 아래 No servers are available. Click this link to create a new server.. 항목이 생긴다. 이 항목을 클릭한다.



No servers are available. Click this link to create a new server를 클릭한다.



‘Next’ 버튼을 클릭하면 아래와 같은 화면이 나온다.



현재 프로젝트를 생성한 것이 없어 Available 항목이 나오지 않는다. 프로젝트를 하나 생성해서 해보면 항목에 나올 것이다.

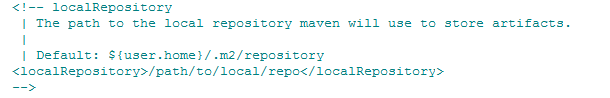
### Maven Setup

Maven은 프로젝트 라이브러리를 관리하기 위해서 사용한다.

#### Maven Repository Change

Maven을 사용할 경우, 어떤 라이브러리를 사용한다고 명시하면, 인터넷에서 Repository를 관리하면 그곳에서 필요한 라이브러리를 자동으로 찾아서 추가해준다. 이렇게 사용할 라이브러리를 다운받고, 참조할 위치를 변경해주려고 한다.

D:\WindLidarProject\apache-maven-3.3.9\conf 폴더를 이동하여 settings.xml파일을 연다.



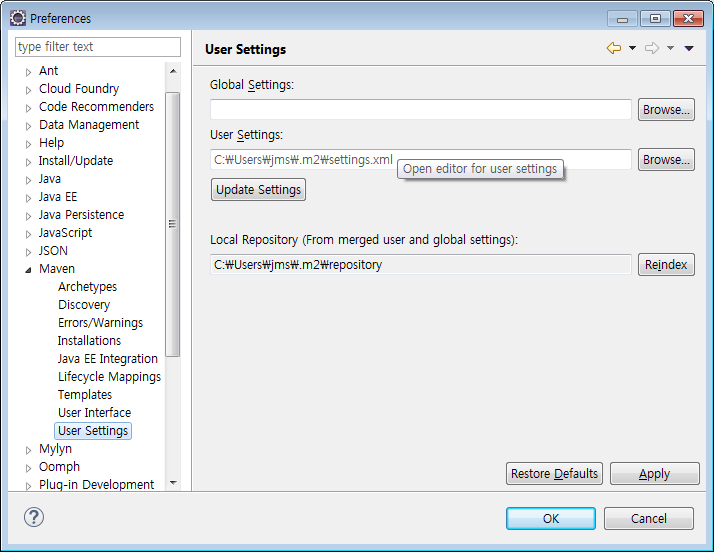
localRepository 의 부분이 있다. 기본적으로 설정하지 않으면 디폴트로 사용자 폴더에 .m2라는 폴더 밑에 repository 라는 폴더에서 라이브러리가 관리된다. 개발환경이 설치되어 있는 폴더에서 모든 것을 관리하려면 repository 위치를 변경해야 한다.

D:\WindLidarProject\apache-maven-3.3.9\repoistory 폴더를 생성한다. 그리고 나서 settings.xml 에서 <localRepository> 부분을 주석 해제하고 변경된 폴더로 변경한다.



#### Eclipse 내에서 Maven 설정

* Window > Preference 를 선택
* Maven > User Settings를 선택



User Settings에서 ‘Browse’ 버튼을 클릭한다.

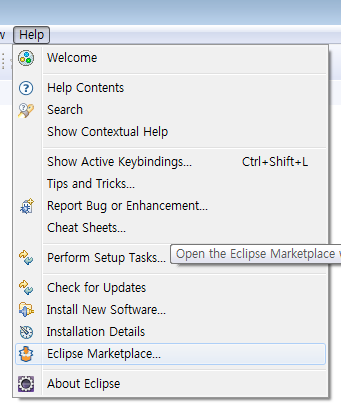
6.3.2.1에서의 Maven Repository 변경에서 설정한 settings.xml 파일을 선택한다.

D:\WindLidarProject\apache-maven-3.3.9\conf\settings.xml

### STS (Spring Tool Suite) 설치

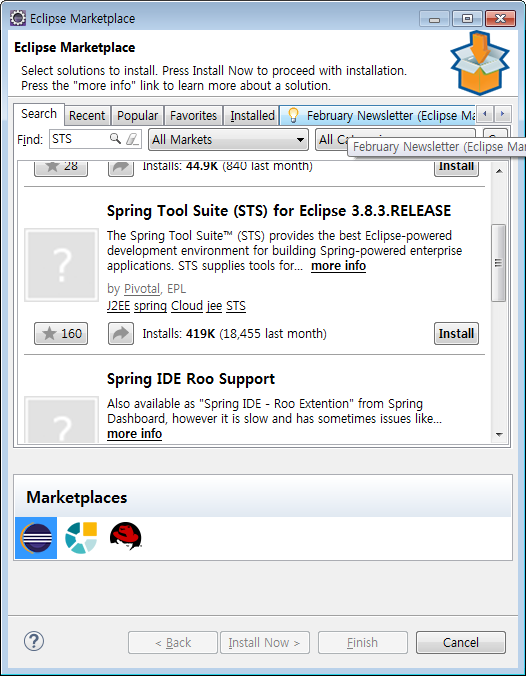
STS는 이클립스에서 스프링 개발을 편리하게 해주는 개발도구이다.

설치 방법

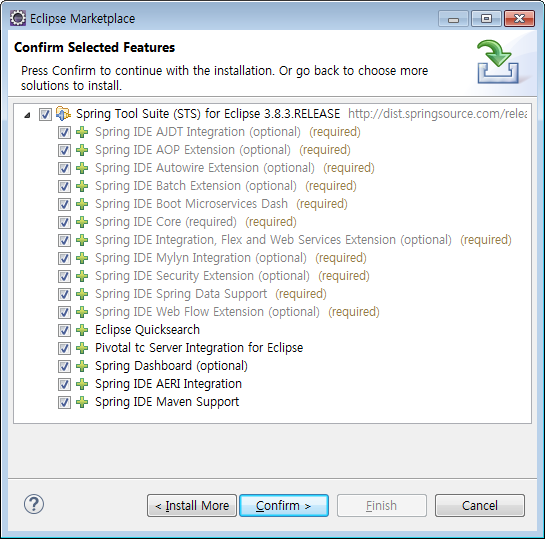


이클립스 상단 메뉴에서 Help > Eclipse Marketplace 메뉴를 선택한다.

Marketplace에서 STS를 검색한다.



Spring Tool Suite (STS) for Eclipse 3.8.3.RELEASE 를 인스톨한다.

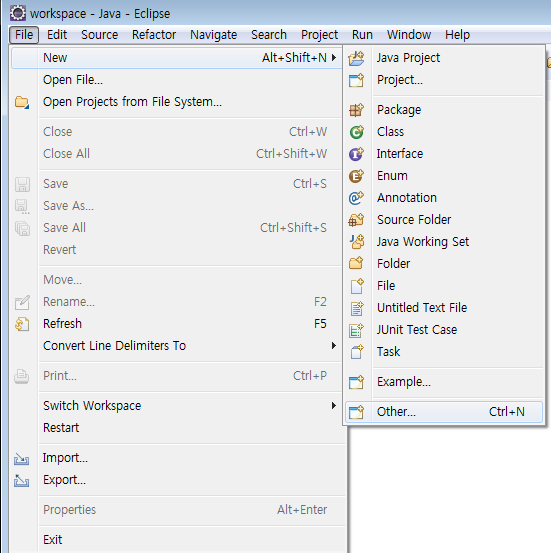


Conform 버튼을 클릭해서 설치를 마무리 한다.

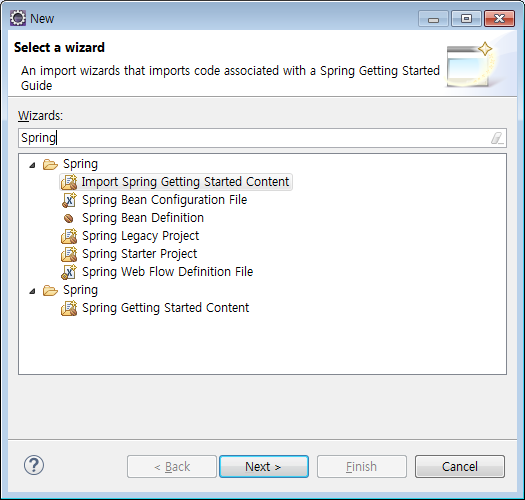
* 설치 확인

STS가 잘 설치되어 있는지 확인하기 위해서 아래와 같이 수행한다.

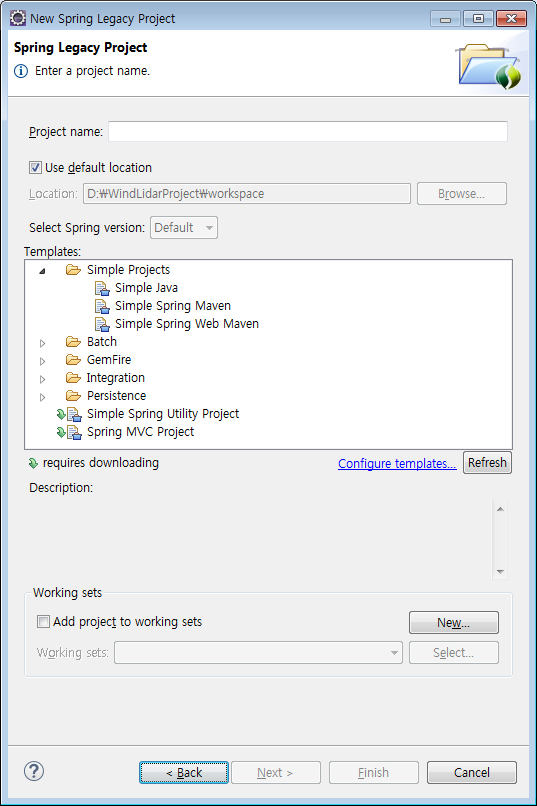
1. File > New > others를 선택



1. Wizards 칸에 Spring을 친 후 Spring Legacy Project를 선택한다.

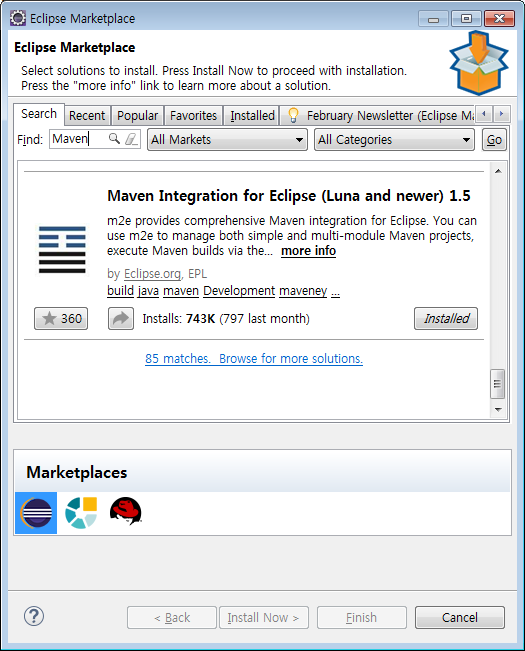


1. Next 버튼을 누르면 Spring Project가 나온다.



### Maven Integration Installation

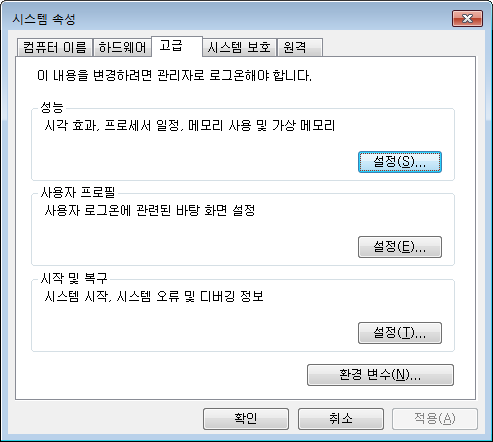
1. Eclipse에서 Help > Marketplace > Maven을 검색



1. Maven Integration for Eclipse 을 선택해서 다운로드

Maven Integration for Eclipse (Luna and newer) 1.5를 선택해서 설치한다.

### Environment Variable Setup

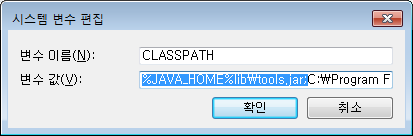


아래 사항은 고급에서 환경 변수로 들어가서 설정한다.

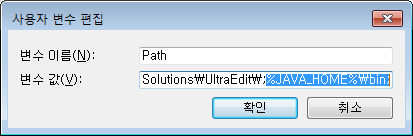
1. JAVA\_HOME 환경 설정



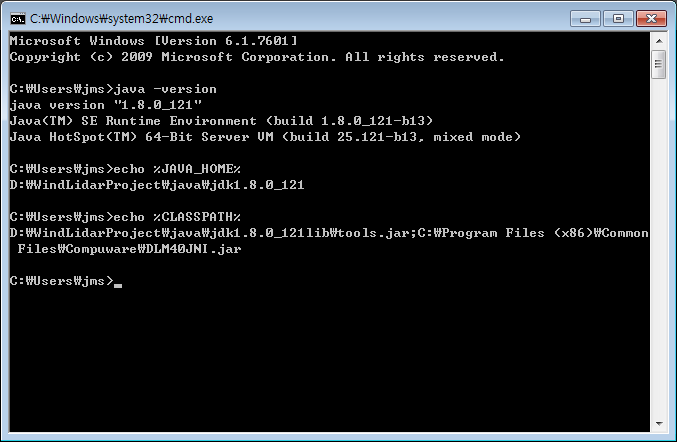
1. CLASSPATH 변수 설정



1. Java PATH 등록



아래 설정이 올바르게 되었는지 확인한다.



Web Development with spring framework

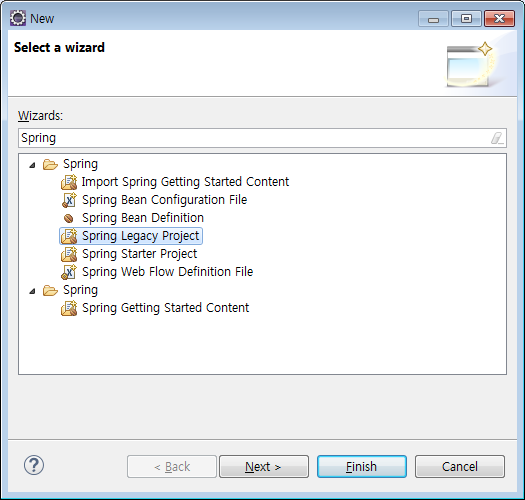
## Spring framework 개요

1. 스프링은 POJO(Plain Old Java Object) 방식의 프레임워크로서, 일반적인 J2EE 프레임워크에 비행 특정 라이브러리를 사용할 필요가 없어서 개발이 쉬우며, 기존 라이브러리의 지원이 용이하다.
2. 스프링은 관점지향 프로그래밍, AOP(Aspect Oriented Programing)을 지원한다. 트랜잭션, 로깅, 보안 등 여러 모듈, 여러 계층에서 적용되는데, 이런 코드들을 실제 비즈니스 로직과 분리할 수 있도록 도와준다. AOP가 OOP(Object Oriented Programing)을 대체하는 기술로 생각되기도 했지만, 실제로 AOP는 OOP스럽게 보완해 주는 기술이다.
3. 스프링은 의존성 주입, DI(Dependency Injection)를 지원한다. 이는 객체간의 의존관계를 관리하는 기술이라 생각하면 된다. 어떤 객체가 필요로 하는 객체를 자기 자신이 직접 생성하는 것이 아니라, 외부에 있는 다른곳에서 자신이 필요로 하는 객체를 주입받는 것을 말한다.
4. 스프링은 제어 반전, IoC (Inversion of Controller)를 지원한다. 컨트롤의 제어권이 개발자가 아니라 프레임워크에 있음을 말한다. 즉, 객체의 생성부터 모든 생명주기의 관리까지 객체의 제어권이 바뀐 것을 의미한다.

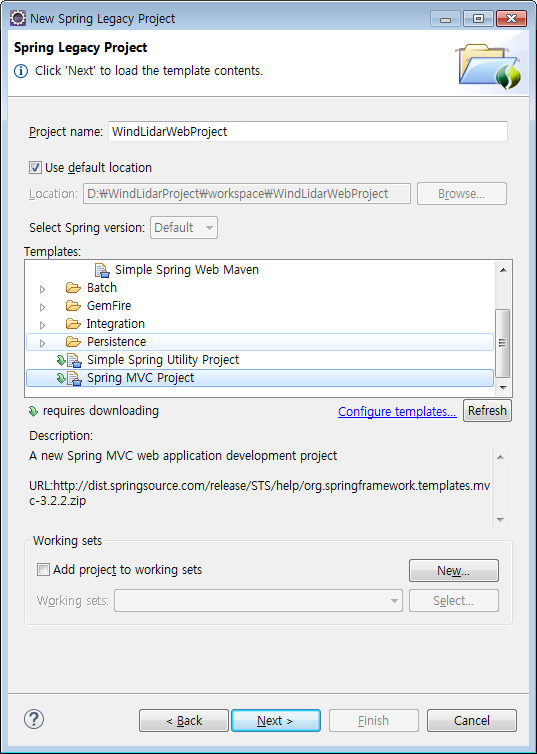
## Eclipse에서 Spring Project 생성

### Spring Project 생성

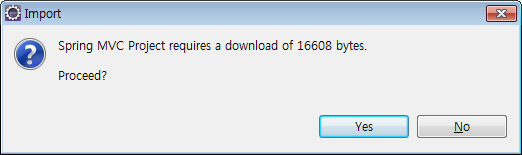
1. Eclipse에서 File > New > Other를 선택해서 Wizards에 Spring을 검색한다.
2. 검색된 Spring 항목에서 Spring Legacy Project를 선택하고 Next 버튼을 클릭한다.



1. 프로젝트명을 적고 Next 버튼을 클릭한다.

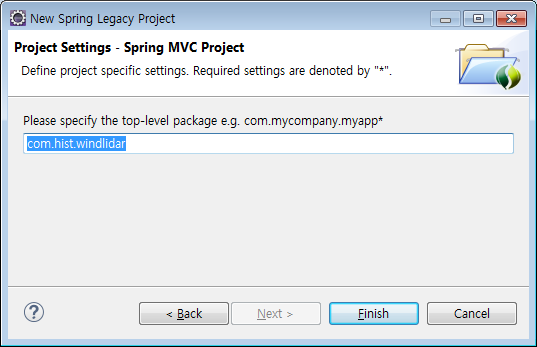


프로젝트명에 “WindLidarWebPorject” 명을 적고 Templates에 Spring MVC Project를 선택하고 ‘Next’ 버튼을 클릭한다.



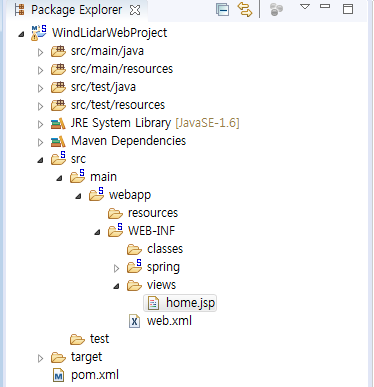
Spring MVC Project를 선택하고 ‘Next’ 버튼을 클릭하면 위와 같은 화면이 출력된다. ‘Yes’ 버튼을 클릭해서 필요한 라이브러리를 다운로드 받는다.

1. Package명을 입력한다.



패키지명 : com.hist.windlidar

1. Finish 버튼을 클릭해서 프로젝트를 생성한다. 생성되면서 인터넷에서 스프링 프로젝트에 필요한 라이브러리를 자동으로 다운로드 한다.



생성된 프로젝트

만일 컴파일 에러가 발생하면 여러가지 문제점이 있겠지만 Maven에서 의존성 라이브러리를 다운로드 받으면서 네트워크상에 문제로 인해 제대로 다운로드 받지 못해 에러가 발생할 수 있다. 이경우는 해당 jar를 Maven repository에서 삭제를 하고 Eclipse에서 maven 메뉴에서 Update Project를 수행하면 다시 라이브러리를 다운로드 받게 된다.

### Maven Dependencies

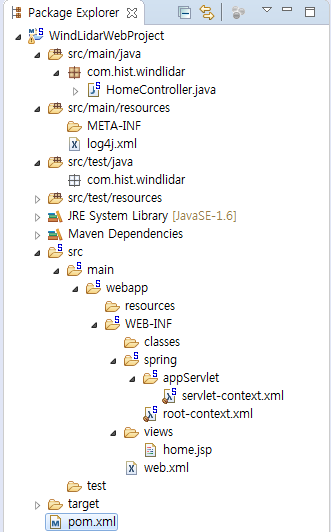


프로젝트를 생성면서 다운로드 받은 라이브러리이다.

### POM.xml

POM.xml파일은 프로젝트에서 필요한 라이브러리를 관리하는 파일이다. POM.xml 파일에서 <dependency></dependency> 태그를 확인할 수 있는데, 이 태그가 하나의 라이브러리를 의미한다. 만약 라이브러리를 추가하고 싶으면, <dependency></dependency>라는 태그를 추가함으로써, 새로운 라이브러리를 추가할 수 있다.

1. POM.xml 확인



1. POM.xml 내용

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4\_0\_0.xsd">  <modelVersion>4.0.0</modelVersion>  <groupId>com.hist</groupId>  <artifactId>windlidar</artifactId>  <name>WindLidarWebProject</name>  <packaging>war</packaging>  <version>1.0.0-BUILD-SNAPSHOT</version>  <properties>  <java-version>1.6</java-version>  <org.springframework-version>3.1.1.RELEASE</org.springframework-version>  <org.aspectj-version>1.6.10</org.aspectj-version>  <org.slf4j-version>1.6.6</org.slf4j-version>  </properties>  <dependencies>  <!-- Spring -->  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-context</artifactId>  <version>${org.springframework-version}</version>  <exclusions>  <!-- Exclude Commons Logging in favor of SLF4j -->  <exclusion>  <groupId>commons-logging</groupId>  <artifactId>commons-logging</artifactId>  </exclusion>  </exclusions>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-webmvc</artifactId>  <version>${org.springframework-version}</version>  </dependency>    <!-- AspectJ -->  <dependency>  <groupId>org.aspectj</groupId>  <artifactId>aspectjrt</artifactId>  <version>${org.aspectj-version}</version>  </dependency>    <!-- Logging -->  <dependency>  <groupId>org.slf4j</groupId>  <artifactId>slf4j-api</artifactId>  <version>${org.slf4j-version}</version>  </dependency>  <dependency>  <groupId>org.slf4j</groupId>  <artifactId>jcl-over-slf4j</artifactId>  <version>${org.slf4j-version}</version>  <scope>runtime</scope>  </dependency>  <dependency>  <groupId>org.slf4j</groupId>  <artifactId>slf4j-log4j12</artifactId>  <version>${org.slf4j-version}</version>  <scope>runtime</scope>  </dependency>  <dependency>  <groupId>log4j</groupId>  <artifactId>log4j</artifactId>  <version>1.2.15</version>  <exclusions>  <exclusion>  <groupId>javax.mail</groupId>  <artifactId>mail</artifactId>  </exclusion>  <exclusion>  <groupId>javax.jms</groupId>  <artifactId>jms</artifactId>  </exclusion>  <exclusion>  <groupId>com.sun.jdmk</groupId>  <artifactId>jmxtools</artifactId>  </exclusion>  <exclusion>  <groupId>com.sun.jmx</groupId>  <artifactId>jmxri</artifactId>  </exclusion>  </exclusions>  <scope>runtime</scope>  </dependency>  <!-- @Inject -->  <dependency>  <groupId>javax.inject</groupId>  <artifactId>javax.inject</artifactId>  <version>1</version>  </dependency>    <!-- Servlet -->  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>servlet-api</artifactId>  <version>2.5</version>  <scope>provided</scope>  </dependency>  <dependency>  <groupId>javax.servlet.jsp</groupId>  <artifactId>jsp-api</artifactId>  <version>2.1</version>  <scope>provided</scope>  </dependency>  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>jstl</artifactId>  <version>1.2</version>  </dependency>    <!-- Test -->  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>4.7</version>  <scope>test</scope>  </dependency>  </dependencies>  <build>  <plugins>  <plugin>  <artifactId>maven-eclipse-plugin</artifactId>  <version>2.9</version>  <configuration>  <additionalProjectnatures>  <projectnature>org.springframework.ide.eclipse.core.springnature</projectnature>  </additionalProjectnatures>  <additionalBuildcommands>  <buildcommand>org.springframework.ide.eclipse.core.springbuilder</buildcommand>  </additionalBuildcommands>  <downloadSources>true</downloadSources>  <downloadJavadocs>true</downloadJavadocs>  </configuration>  </plugin>  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-compiler-plugin</artifactId>  <version>2.5.1</version>  <configuration>  <source>1.6</source>  <target>1.6</target>  <compilerArgument>-Xlint:all</compilerArgument>  <showWarnings>true</showWarnings>  <showDeprecation>true</showDeprecation>  </configuration>  </plugin>  <plugin>  <groupId>org.codehaus.mojo</groupId>  <artifactId>exec-maven-plugin</artifactId>  <version>1.2.1</version>  <configuration>  <mainClass>org.test.int1.Main</mainClass>  </configuration>  </plugin>  </plugins>  </build>  </project> |

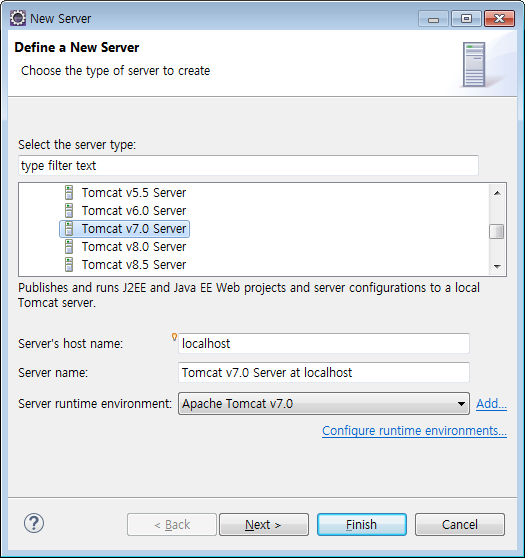
### UTF-8 설정

web.xml에서 아래의 내용을 등록한다.

|  |
| --- |
| <filter>  <filter-name>encodingFilter</filter-name>  <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>  <init-param>  <param-name>encoding</param-name>  <param-value>utf-8</param-value>  </init-param>  <init-param>  <param-name>forceEncoding</param-name>  <param-value>true</param-value>  </init-param>  </filter>  <filter-mapping>  <filter-name>encodingFilter</filter-name>  <url-pattern>/\*</url-pattern>  </filter-mapping> |

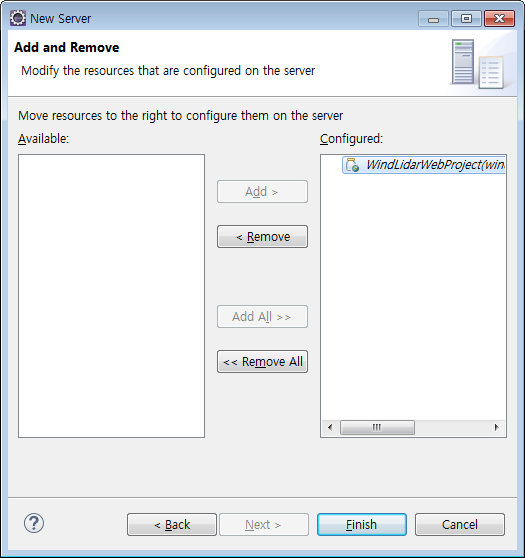
### 생성된 프로젝트 실행

#### Servers에서 새로운 서버 생성



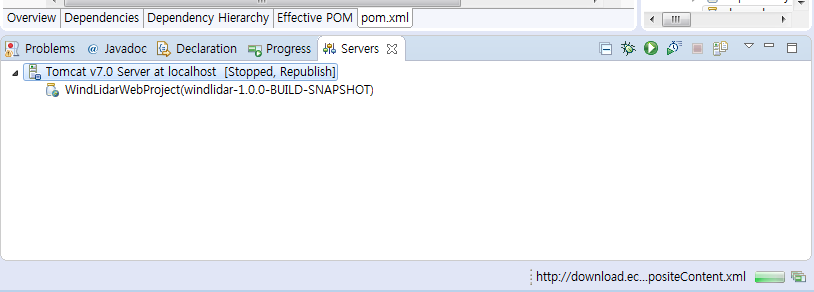
Tomcat v7.0 Server를 선택하고 [Next] 버튼을 클릭한다.

#### 이용 가능한 프로젝트 선택



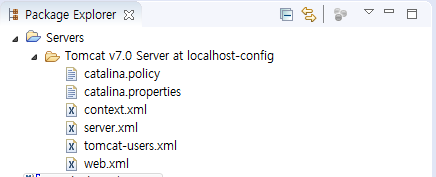
WindLidarWebProject를 선택해서 Add 버튼을 클릭한 후 [Finish] 버튼을 클릭한다.

#### Server 가동

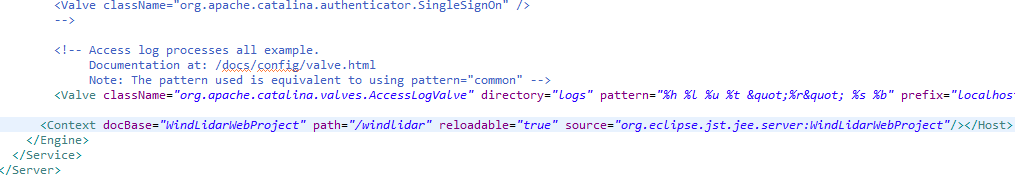


추가된 Server에서 상단 아이콘 메뉴에서 화살표 버튼을 클릭해서 서버를 가동시킨다.

#### Server 기동 확인



Servers 폴더에서 server.xml 파일을 열어서 등록된 프로젝트를 확인한다.

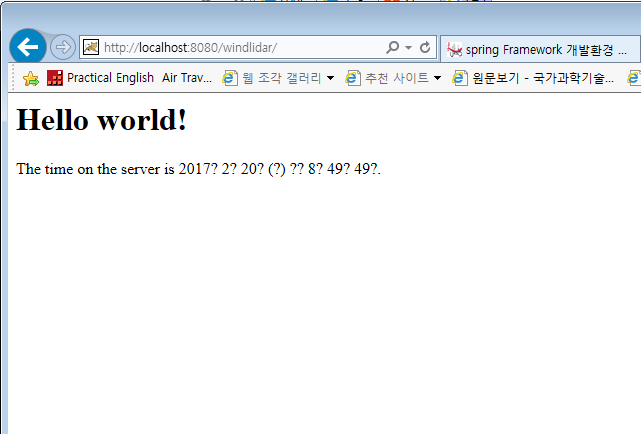


Server path가 /windlidar 로 등록되어 있다. 브라우저에서 아래와 같이 접속을 시도해 본다.

<http://localhost:8080/windlidar>

배치폴더는 다음과 같다.

D:\WindLidarProject\workspace\.metadata\.plugins\org.eclipse.wst.server.core\tmp1\wtpwebapps



## Mybatis 연동하기

### Mybatis개요

Mabatis는 개발자가 지정한 SQL, 저장프로시저, 그리고 몇가지 고급 매핑을 지원한느 퍼시스턴스 프레임워크이다. MyBatis는 JDBC 코드와 수동으로 셋팅하는 파라미터와 결과 매핑을 제거한다. MyBatis는 데이터베이스 레코드에 원시타입과 Map 인터페이스 그리고 자바 POJO를 설정하고 매핑하기 위해 XML과 어노테이션을 사용할 수 있다.

### Mybatis 라이브러리

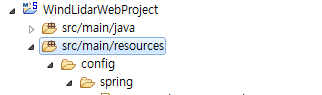
스프링에서 MyBatis 를 사용하려면 라이브러리가 필요하다. 이 프로젝트에서는 Maven을 이용해서 라이브러리를 관리하기 때문에, 라이브러리의 추가는 pom.xml에 추가한다. 아래와 같이 설정한다.

|  |
| --- |
| <!-- MyBatis library -->  <dependency>  <groupId>org.mybatis</groupId>  <artifactId>mybatis</artifactId>  <version>3.2.2</version>  </dependency>  <dependency>  <groupId>org.mybatis</groupId>  <artifactId>mybatis-spring</artifactId>  <version>1.2.0</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-jdbc</artifactId>  <version>${org.springframework-version}</version>  </dependency>  <dependency>  <groupId>commons-dbcp</groupId>  <artifactId>commons-dbcp</artifactId>  <version>1.4</version>  </dependency>    <!-- database library -->  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>5.1.31</version>  </dependency> |

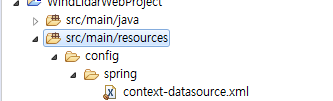
### MyBatis와 데이터베이스 연결 설정

아래와 같이 MyBatis와 데이터베이스 연결설 정을 위해서 수행한다.

1. src/main/resource 폴더 밑에 config > spring 폴더를 생성한다.



1. spring 폴더 밑에 context-datasource.xml 파이을 생성한다.



1. web.xml에서 위의 설정파일을 읽도록 설정되어있는지 확인한다.

web.xml 의 하단에 다음의 내용이 들어 있어야 한다.

|  |
| --- |
| <context-param>  <param-name>contextConfigLocation</param-name>  <param-value>classpath\*:config/spring/context-\*.xml</param-value>  </context-param> |

1. 위의 설정은 최초 서버가 기동 시 해당 위치에 있는 context 파일을 읽어 들이는 것을 의미한다.
2. MyBatis 연결 설정

* context-datasource.xml 에 아래의 내용을 추가한다.

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd*  *http://www.springframework.org/schema/jdbc http://www.springframework/schema/jdbc/spring-jdbc-3.0.xsd"*>    <!-- Root Context: defines shared resources visible to all other web components -->  <bean id=*"dataSource"* class=*"org.apache.commons.dbcp.BasicDataSource"* destroy-method=*"close"*>  <property name=*"driverClassName"* value=*"com.mysql.jdbc.Driver"* />  <property name=*"url"* value=*"jdbc:mysql://localhost:3303/windlidarDB"* />  <property name=*"username"* value=*"histUser"* />  <property name=*"password"* value=*"hist123"* />  </bean>    </beans> |

### MyBatis와 Spring 연결

1. resource 폴더에 mapper 폴더를 생성한다.
2. resource > config > spring 폴더에 context-mappter.xml 파일을 생성한다.
3. context-mapper.xml 파일에 다음의 내용을 작성한다.

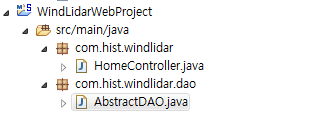
|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd*  *http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd"*>    <bean id=*"sqlSession"* class=*"org.mybatis.spring.SqlSessionFactoryBean"*> <property name=*"dataSource"* ref=*"dataSource"* />  <property name=*"mapperLocations"* value=*"classpath:/mapper/\*\*/\*\_SQL.xml"* />  </bean>    <bean id=*"sqlSessionTemplate"* class=*"org.mybatis.spring.SqlSessionTemplate"*>  <constructor-arg index=*"0"* ref=*"sqlSession"*/>  </bean>  </beans> |

* property의 name과 ref가 dataSource로 정의되어 있다. name은 위에서 등록한 sqlSession 빈(bean)에서 사용할 이름이 dataSource이고 ref의 dataSource는 우리가 context-datasource.xml에서 정의한 빈(bean)을 참조하는 것을 의미한다.
* mapperLocation은 MyBatis를 사용하기 위해서 작성할 SQL문이 위치할 장소이다. 여기서는 classpath:/mapper/\*\*/\*\_SQL.xml 이라고 정의하였습니다.
* web.xml에서 spring context 설정파일을 읽어 오기 위해서 classpath\*:/config/context-\*.xml 라고 정의하였습니다.
* sqlSessionTemplate은 마이바티스 스프링 연동모듈의 핵심이다. SQLSessionTemplate은 SqlSession을 구현하고, 코드에서 SqlSessoin을 대체하는 역할을 한다

### DAO 작성

이제 실제 소스에서 위에서 선언한 SqlSessionTemplate을 사용할 DAO를 생성한다. DAO는 Data Access Object의 약자로 Data Access Object를 Factory 패턴화 시키는 것으로 비즈니스 로직을 모듈화 하는 방법이다.

1. src/main/java 폴더의 com.hist.windlidar 패키지 밑에 dao 패키지를 생성한다.
2. dao 패키지 안에 AbstractDAO.java를 생성한다.



1. 아래의 소스를 작성한다.

|  |
| --- |
| * package com.hist.windlidar.dao; * import java.util.List; * import org.apache.commons.logging.Log; * import org.apache.commons.logging.LogFactory; * import org.mybatis.spring.SqlSessionTemplate; * import org.springframework.beans.factory.annotation.Autowired; * public class AbstractDAO { * protected Log log = LogFactory.getLog(AbstractDAO.class); * @Autowired * private SqlSessionTemplate sqlSession; * protected void printQueryId(String queryId) { * if(log.isDebugEnabled()){ * log.debug("\t QueryId \t: " + queryId); * } * } * public Object insert(String queryId, Object params){ * printQueryId(queryId); * return sqlSession.insert(queryId, params); * } * public Object update(String queryId, Object params){ * printQueryId(queryId); * return sqlSession.update(queryId, params); * } * public Object delete(String queryId, Object params){ * printQueryId(queryId); * return sqlSession.delete(queryId, params); * } * public Object selectOne(String queryId){ * printQueryId(queryId); * return sqlSession.selectOne(queryId); * } * public Object selectOne(String queryId, Object params){ * printQueryId(queryId); * return sqlSession.selectOne(queryId, params); * } * @SuppressWarnings("rawtypes") * public List selectList(String queryId){ * printQueryId(queryId); * return sqlSession.selectList(queryId); * } * @SuppressWarnings("rawtypes") * public List selectList(String queryId, Object params){ * printQueryId(queryId); * return sqlSession.selectList(queryId,params); * }   } |

## Application Development with MyBatis

### List 데이터 가져오기

일반적으로 데이터를 조회해서 화면에 데이터를 출력하는 샘플 프로그램이다. 기능 구현의 순서는 아래와 같다.

* Controller > Service > ServiceImpl > DAO > SQL xml > JSP

#### Controller 구현

1. src/main/java 폴더에서 패키지를 생성한다. com.hist.windlidar.controller 패키지를 생성한다.
2. 생성된 패키지에서 자바 소스를 생성한다. WindLidarController.java
3. 자바 소스파일에 아래와 같이 코딩을 수행한다.

|  |
| --- |
| **package** com.hist.windlidar.controller;  **import** java.util.List;  **import** java.util.Map;  **import** javax.annotation.Resource;  **import** org.apache.log4j.Logger;  **import** org.springframework.stereotype.Controller;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.servlet.ModelAndView;  **import** com.hist.windlidar.service.WindLidarService;  @Controller  **public** **class** WindLidarController {  Logger log = Logger.*getLogger*(**this**.getClass());    @Resource(name="windLidarService")  **private** WindLidarService windLidarService;    // 아래 URL 호출은 다음과 같이 호출되어야 한다.  // http://localhost:8080/windlidar/windlidar/windLidarList.do  @RequestMapping(value="/windlidar/windLidarList.do")  **public** ModelAndView openWindLidarList(Map<String, Object> commandMap) **throws** Exception {  ModelAndView mv = **new** ModelAndView("/windLidar/windLidarList"); // JSP File    List<Map<String, Object>> list = windLidarService.selectWindLidarList(commandMap);    // 결과를 클라이언트에 전달  mv.addObject("list", list);  **return** mv;  }  } |

- 컨트롤러(Controller)는 웹 클라이언트에서 들어온 요청을 해당 비즈니스 로직을 호출하고, 수행결과와 함께 응답을 해주는 Dispatcher 역할을 한다. 클래스의 선언부에 @Controller 어노테이션(Annotation)을 이용하여, Controller 객체임을 선언한다.

- @RequestMapping은 요청 URL을 의미한다. /windlidar/windLidarList.do 라는 주소를 호출하게 되면, 이 주소는 @RequestMapping 어노테이션과 매핑되어, 해당 메서드가 실행된다. ModelAndView mv = new ModelAndView(“/windLidar/windLidarList”);는 우리가 화면에 보여줄 jsp파일을 의미한다. 여기서 주의할 점은 우리가 정의한 Webcontents가 windlidar라면 URL 호출은 다음과 같다. <http://localhost:8080/windlidar/windlidar/windLidarList.do>

- ModelAndView에 정의된 /windLidar/windLidarList의 경우 첫번째 windLidar의 경우 JSP가 있는 폴더명이 될것이고 windLidarList.의 경우 windLidarList.jsp가 될 것이다.

- 다음은 List<Map<String, Object>> list = windLidarService.selectWindLidarList(commandMap); 부분이다. 목록을 보여주기 때문에 목록을 저장할 수 있는 List를 선언하였고 List 형식은 Map<String, Object>인데, 하나의 목록에도 여러가지 정보가 존재한다. 따라서 Map에 저장하려는 것이다. Map은 다시 키(Key)와 값(Value)로 구분되어지는데, 각각의 컬럼의 키와 실제값이 저장된다.

- Controller는 단순히 어떤 주소와 화면을 연결하고, 비즈니스 로직을 호출하는 역할을 한다. 실제로 여러가지 비즈니스 로직은 Service에서 작성한다. 여기서는 단순히 데이터를 조회하는 부분이지만, 상세 조회, 등록 등 이런 기능도 Service에서 처리해주게 된다.

- mv.addObject(“list”, list)는 서비스로직의 결과를 ModelAndView 객체에 담아서 클라이언트, 즉 jsp에서 그 결과를 사용할 수 있도록 한다. mv에 값을 저장한느 것은 map과 똑같이 키와 값으로 구성이 되는데, windLidarServer.selectWindLidarList에서 얻어온 결과를 “list”라는 이름으로 저장해서 리턴한다. jsp 화면에서 이 “list”라는 이름으로 데이터에 접근을 할 수 있다.

#### Service 구현

Service 영역은 두개의 파일로 구성된다. Service 인터페이스와 이 인터페이스를 실제로 구현한 ServiceImpl 클래스로 구성이 되어있다. 이는 Spring의 IoC/DI (Inversion of Control / Dependency Injection) 기능을 이용한 Bean 관리 기능을 사용하기 위함이다.

##### Interface Class 구현

|  |
| --- |
| **package** com.hist.windlidar.service;  **import** java.util.List;  **import** java.util.Map;  **public** **interface** WindLidarService {  List<Map<String, Object>> selectWindLidarList(Map<String, Object> commandMap) **throws** Exception;  } |

* Service 인터페이스는 비즈니스 로직의 수행을 위한 메소드를 정의한다.

##### Interface Implements

|  |
| --- |
| **package** com.hist.windlidar.service;  **import** java.util.List;  **import** java.util.Map;  **import** javax.annotation.Resource;  **import** org.apache.log4j.Logger;  **import** org.springframework.stereotype.Service;  **import** com.hist.windlidar.dao.WindLidarDAO;  @Service("windLidarService")  **public** **class** WindLidarServiceImpl **implements** WindLidarService {  Logger log = Logger.*getLogger*(**this**.getClass());    @Resource(name="windLidarDAO")  **private** WindLidarDAO windLidarDAO;    @Override  **public** List<Map<String, Object>> selectWindLidarList(Map<String, Object> commandMap) **throws** Exception{  // **TODO** Auto-generated method stub  **return** windLidarDAO.selectWindLidarList(commandMap);  }  } |

* ServiceImpl 클래스는 Service 인터페이스를 통해 정의된 메서드를 실제로 구현하는 클래스이다.
* @Service 어노테이션을 이용하여 Service 객체임을 선언하고 이 객체의 이름은 windLidarService라고 선언하였다.
* Controller 클래스에서 @Resource(name=”windLidarService”) 이 부분은 Service 영역의 접근을 위한 선언이다. WindLidarServiceImpl 클래스를 생성하고 그 Service를 “windLidarService”라는 이름으로 선언을 하였는데, 이를 사용하기 위한 선언이다. 여기서 @Resource 어노테이션을 통해서 필요한 빈(Bean)을 수동으로 등록하는 것이다. 그리고 수동으로 등록할 빈의 이름이 “windLidarService”이고 이는 @Service(“windLidarService”)라고 선언했을 때의 그 이름인 것을 확인할 수 있다.
* Service에서는 데이터 접근을 위한 DAO(Data Access Object) 객체를 선언하였다

#### DAO 구현

데이터베이스에 접근하기 위해서 DAO 클래스를 구현한다.

|  |
| --- |
| **package** com.hist.windlidar.dao;  **import** java.util.List;  **import** java.util.Map;  **import** org.springframework.stereotype.Repository;  @Repository("windLidarDAO")  **public** **class** WindLidarDAO **extends** AbstractDAO {  @SuppressWarnings("unchecked")  **public** List<Map<String, Object>> selectWindLidarList(Map<String, Object> map) **throws** Exception  {  // 실행할 쿼리 이름과 매개변수  **return** (List<Map<String, Object>>)selectList("windlidar.selectLidarList", map);  }  } |

* Repository 어노테이션을 통해서 이 클래스가 DAO 임을 선언하고 이름을 windLidarDAO라고 선언하였다.
* WindLidarServiceImpl에서 @Resource(name=”windLidarDAO”)로 bean을 수동으로 등록하였고, 거기서 사용된 빈이 방금 작성한 WindLidarDAO 클래스이다.
* AbstractDAO 클래스를 상속받았다.
* DAO는 데이터베이스에 접근하여 데이터를 조작하는 역할만 수행한다.
* MyBatis 프레임워크를 사용하기 때문에 JDBC 연결 및 쿼리 수행을 편하게 할 수 있다.
* MyBatis는 프로그램의 소스코드에서 SQL을 분리하여 별도의 XML 파일을 저장하고, 이 둘을 연결하는 방식으로 작동한다.
* windlidar.selectLidarList가 쿼리 이름이고 Controller에서부터 계속 넘어온 Map<String, Object> map이 쿼리 실행시 필요한 변수들이다.
* 조회된 결과 값은 List<Map<String, Object>> 형식으로 반환할 수 있도록 형변환을 수행한다.

#### SQL 작성

src/main/resources 폴더 밑에 mapper 폴더를 작성하였다. 여기에 SQL 파일을 만든다.

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd">    <mapper namespace=*"windlidar"*>  <select id=*"selectLidarList"* parameterType=*"hashmap"* resultType=*"hashmap"*>  <![CDATA[  SELECT  NO,  S\_CODE,  S\_YEAR,  S\_MON,  S\_DAY,  S\_HOUR,  S\_MIN,  S\_SEC,  FILE\_CNT,  USE\_CHK,  REG\_DT,  UPT\_DT  FROM  T\_RCV\_INFO  ORDER BY REG\_DT ASC  ]]>  </select>  </mapper> |

* <select> 태그를 이용하여 쿼리가 select 문이라는 것을 명시
* id=”selectLidarList” 부분은 이 쿼리의 id는 selectLidarList라고 정의
* parameterType=”hashmap” 부분은 이 쿼리가 실행될 때 필요한 변수는 HashMap 형태를 의미한다.
* resultType=”hashmap” 부분은 이 쿼리의 결과값은 HashMap에 담겨서 반환된다는 것을 의미한다.
* 타입에 대한 부분은 아래를 참조한다.
* <https://mybatis.github.io/mybatis-3/ko/configuration.html#typeAliases>

#### JSP 구현

servlet-context.xml 파일을 보면 아래와 같이 명시되어 있다.

|  |
| --- |
| <!-- Resolves views selected for rendering by @Controllers to .jsp resources in the /WEB-INF/views directory -->  <beans:bean class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>  <beans:property name=*"prefix"* value=*"/WEB-INF/views/"* />  <beans:property name=*"suffix"* value=*".jsp"* />  </beans:bean> |

JSP 파일의 경우 /WEB-INF/views 디렉토리에 아래에 존재하는 것이다. 우리는 WindLidarController 클래스에서 ModelAndView에 등록된 JSP 파일의 경우 ModelAndView mv = new ModelAndView(“/windLidar/windLidarList”); 로 되어 있다. 따라서 파일의 경로는 /WEB-INF/views/windLidar/windLidarList.jsp 가 되어야 한다. 아래는 JSP 소스이다.

|  |
| --- |
| <%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"* pageEncoding=*"UTF-8"*%>  <!DOCTYPE html>  <html>  <head>  <title>first</title>  <%@ taglib prefix=*"c"* uri=*"http://java.sun.com/jsp/jstl/core"* %>  <%@ taglib prefix=*"fn"* uri=*"http://java.sun.com/jsp/jstl/functions"* %>  </head>  <body>  <h2>WindLidar Data List</h2>  <table style="border:*1px solid #ccc*">  <colgroup>  <col width=*"8%"*/>  <col width=*"8%"*/>  <col width=*"8%"*/>  <col width=*"8%"*/>  <col width=*"8%"*/>  <col width=*"8%"*/>  <col width=*"8%"*/>  <col width=*"8%"*/>  <col width=*"8%"*/>  <col width=*"8%"*/>  <col width=*"8%"*/>  <col width=*"8%"*/>  </colgroup>  <thead>  <tr>  <th scope=*"col"*>No</th>  <th scope=*"col"*>Code</th>  <th scope=*"col"*>Year</th>  <th scope=*"col"*>Month</th>  <th scope=*"col"*>Day</th>  <th scope=*"col"*>HOUR</th>  <th scope=*"col"*>Minute</th>  <th scope=*"col"*>SEC</th>  <th scope=*"col"*>File Count</th>  <th scope=*"col"*>Use Check</th>  <th scope=*"col"*>REG dt</th>  <th scope=*"col"*>UPT dt</th>  </tr>  </thead>  <tbody>  <c:choose>  <c:when test=*"*${fn:length(list) > 0}*"*>  <c:forEach items=*"*${list }*"* var=*"row"*>  <tr>  <td>${row.NO }</td>  <td>${row.S\_CODE }</td>  <td>${row.S\_YEAR }</td>  <td>${row.S\_MON }</td>  <td>${row.S\_DAY }</td>  <td>${row.S\_HOUR }</td>  <td>${row.S\_MIN }</td>  <td>${row.S\_SEC }</td>  <td>${row.FILE\_CNT }</td>  <td>${row.USE\_CHK }</td>  <td>${row.REG\_DT }</td>  <td>${row.UPT\_DT }</td>  </tr>  </c:forEach>  </c:when>  <c:otherwise>  <tr>  <td colspan=*"4"*>조회된 결과가 없습니다.</td>  </tr>  </c:otherwise>  </c:choose>  </tbody>  </table>  </body>  </html> |

### HandlerMethodArgumentResolver 적용

#### HandlerMethodArgumentResolver 개요

HandlerMethodArgumentResolver는 스프링 3.1에서 추가된 인터페이스이다. 스프링 3.1 이전에는 WebArgumentResolver라는 인터페이스였는데, 스프링 3.1 이후부터 HandlerMethodArgumentResolver 라는 이름으로 변경되었다. 이 클래스가 하는 역할은 다음과 같다.

* 스프링 사용 시 컨트롤러(Controller)에 들어오는 파라미터(Parameter)를 수정하거나 공통적으로 추가를 해주어야 하는 경우가 있다. 예를 들어, 로그인을 한 사용자의 아이디나 니넥임을 추가한다고 하면 보통 그런 정보는 세션에 담아 놓고 사용하는데 DB에 그런 정보를 입력할 때에는 결국 세션에서 값을 꺼내와서 파라미터로 추가를 해야 한다.
* 그런 경우가 하나나 두번 정보 있다면 몰라도, 여러 번 사용되는 값을 그렇게 일일이 세션에서 가져오는 것은 상단히 번거로운 일이다.
* HandlerMethodArgumentResolver 클래스는 사용자 요청이 Controller에 도달하기 전에 그 요청의 파라미터들을 수정할 수 있도록 해준다.

#### CommandMap 클래스 생성

request에 담겨있는 파라미터를 Map에 담아주는 역할을 하는 클래스이다. 컨트롤러 클래스에서 public ModelAndView openWindLidarList(Map<String, Object> commandMap) throws Exception 라고 선언을 했었다. 여기서 Map<String, Object> commandMap에 사용자가 넘겨준 파라미터가 저장되어 있다. 여기서 문제는 HandlerMethodArgumentResolver는 컨트롤러의 파라미터가 Map 형식이면 동작하지 않는다. 즉 스프링 3.1에서 HandlerMethodArgumentResolver를 이용하여 그러한 기능을 만들더라도, 컨트롤러의 파라미터가 Map 형식이면 우리가 설정한 클래스가 아닌, 스프링에서는 기본적으로 설정된 ArgumentResolver를 거치게 된다. 이 경우는 스프링의 <mvc:annotation-driven />을 선언하게 되면 위에서 이야기한것처럼 동작하게 된다. 따라서 <mvc:annotation-driven/>을 선언하려면 Map을 그대로 사용할 수 없고, 선언하지 않으면 문제는 없다. 하지만 <mvc:annotation-drive/>을 선언해야 하는 경우도 있기 때문에 Map을 대신할 CommandMap 클래스를 만들어 사용한다.

* com.hist.windlidar.common 패키지를 생성하고 CommandMap.java 클래스 파일을 생성한다.

|  |
| --- |
| **package** com.hist.windlidar.common;  **import** java.util.HashMap;  **import** java.util.Map;  **import** java.util.Map.Entry;  **import** java.util.Set;  **public** **class** CommandMap {  Map<String,Object> map = **new** HashMap<String,Object>();  **public** Object get(String key){  **return** map.get(key);  }    **public** **void** put(String key, Object value){  map.put(key, value);  }  **public** Object remove(String key){  **return** map.remove(key);  }  **public** **boolean** containsKey(String key){  **return** map.containsKey(key);  }  **public** **boolean** containsValue(Object value){  **return** map.containsValue(value);  }  **public** **void** clear(){  map.clear();  }  **public** Set<Entry<String, Object>> entrySet(){  **return** map.entrySet();  }  **public** Set<String> keySet(){  **return** map.keySet();  }  **public** **boolean** isEmpty(){  **return** map.isEmpty();  }  **public** **void** putAll(Map<? **extends** String, ?**extends** Object> m){  map.putAll(m);  }  **public** Map<String,Object> getMap(){  **return** map;  }  } |

#### HandlerMethodArgumentResolver 작성

com.hist.windlidar.resolver 패키지를 생성 후 아래 클래스를 작성한다.

|  |
| --- |
| **package** com.hist.windlidar.resolver;  **import** java.util.Enumeration;  **import** javax.servlet.http.HttpServletRequest;  **import** org.springframework.core.MethodParameter;  **import** org.springframework.web.bind.support.WebDataBinderFactory;  **import** org.springframework.web.context.request.NativeWebRequest;  **import** org.springframework.web.method.support.HandlerMethodArgumentResolver;  **import** org.springframework.web.method.support.ModelAndViewContainer;  **import** com.hist.windlidar.common.CommandMap;  **public** **class** CustomMapArgumentResolver **implements** HandlerMethodArgumentResolver {  @Override  **public** **boolean** supportsParameter(MethodParameter parmeter) {  // **TODO** Auto-generated method stub  **return** CommandMap.**class**.isAssignableFrom(parmeter.getParameterType());  }    @Override  **public** Object resolveArgument(MethodParameter parmeter, ModelAndViewContainer mavContainer, NativeWebRequest webRequest,  WebDataBinderFactory binderFactory) **throws** Exception {  // **TODO** Auto-generated method stub  CommandMap commandMap = **new** CommandMap();    HttpServletRequest request = (HttpServletRequest) webRequest.getNativeRequest();  Enumeration<?> enumeration = request.getParameterNames();    String key = **null**;  String[] values = **null**;  **while**(enumeration.hasMoreElements())  {  key = (String) enumeration.nextElement();  values = request.getParameterValues(key);  **if** (values != **null**)  {  commandMap.put(key, (values.length > 1) ? values : values[0]);  }  }  **return** commandMap;  }  } |

* HandlerMethodArguementResolver 인터페이스를 상속하면 두가지 메소드를 반드시 구현해야 한다. supportParameter 메소드와 resolveArgument 메소드이다.
* supportsParameter 메서드는 Resolver가 적용 가능한지 검사하는 역할을 한다.
* resolveArgument 메서드는 파라미터와 기타 정보를 받아서 실제 객체를 반환한다.
* supportsParameter 메서드는 컨트롤러의 파라미터가 CommandMap 클래스인지 검사하도록 하였다. 추후 Controller의 Map<String, Object> 형식을 CommandMap이라고 변견할 것이다.
* resolveArgement 메소드는 CommandMap 객체를 생성하고 request에 담겨있는 모든 키와 값을 commandMap에 저장하였다.
* request에 있는 값을 iterator를 이용하여 하나씩 가져와서 파라미터를 담고 commandMap을 반환하였다.

#### CustomMapArgumentResoler 등록

CustomerMapArgumentResolver를 등록하기 위해서는 root context 영역에 등록해야 한다. servlet-context.xml 파일에 다음과 같이 등록한다.

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns:context=*"http://www.springframework.org/schema/context"*  xmlns:p=*"http://www.springframework.org/schema/p"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:mvc=*"http://www.springframework.org/schema/mvc"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.0.xsd*  *http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-3.0.xsd*  *http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc.xsd"*>    <context:component-scan base-package=*"first"*></context:component-scan>  <mvc:annotation-driven>  <mvc:argument-resolvers>  <bean class=*"com.hist.windlidar.resolver.CustomMapArgumentResolver"*></bean>  </mvc:argument-resolvers>  </mvc:annotation-driven>  <!-- mvc:interceptors>  <mvc:interceptor>  <mvc:mapping path="/\*\*"/>  <bean id="loggerInterceptor" class="first.common.logger.LoggerInterceptor"></bean>  </mvc:interceptor>  </mvc:interceptors -->  <bean class=*"org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping"*/>  <bean class=*"org.springframework.web.servlet.view.BeanNameViewResolver"* p:order=*"0"* />  <bean id=*"jsonView"* class=*"org.springframework.web.servlet.view.json.MappingJacksonJsonView"* />  <bean  class=*"org.springframework.web.servlet.view.UrlBasedViewResolver"* p:order=*"1"*  p:viewClass=*"org.springframework.web.servlet.view.JstlView"*  p:prefix=*"/WEB-INF/views/"* p:suffix=*".jsp"*>  </bean>  </beans> |

#### Controller 수정

|  |
| --- |
| **package** com.hist.windlidar.controller;  **import** java.util.Iterator;  **import** java.util.List;  **import** java.util.Map;  **import** java.util.Map.Entry;  **import** javax.annotation.Resource;  **import** org.apache.log4j.Logger;  **import** org.springframework.stereotype.Controller;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.servlet.ModelAndView;  **import** com.hist.windlidar.common.CommandMap;  **import** com.hist.windlidar.service.WindLidarService;  @Controller  **public** **class** WindLidarController {  Logger log = Logger.*getLogger*(**this**.getClass());    @Resource(name="windLidarService")  **private** WindLidarService windLidarService;    // 아래 URL 호출은 다음과 같이 호출되어야 한다.  // http://localhost:8080/windlidar/windlidar/windLidarList.do  @RequestMapping(value="/windlidar/windLidarList.do")  **public** ModelAndView openWindLidarList(Map<String, Object> commandMap) **throws** Exception {  ModelAndView mv = **new** ModelAndView("/windLidar/windLidarList"); // JSP File    List<Map<String, Object>> list = windLidarService.selectWindLidarList(commandMap);    // 결과를 클라이언트에 전달  mv.addObject("list", list);  **return** mv;  }    @RequestMapping(value="/windlidar/testMapArgumentResolver.do")  **public** ModelAndView testMapArgumentResolver(CommandMap commandMap) **throws** Exception  {  ModelAndView mv = **new** ModelAndView("");    **if** (commandMap.isEmpty() == **false**)  {  Iterator<Entry<String, Object>> iterator = commandMap.getMap().entrySet().iterator();  Entry<String, Object> entry = **null**;  **while** (iterator.hasNext())  {  entry = iterator.next();  log.debug("key : " + entry.getKey()+", value : " + entry.getValue());  }  }  **return** mv;  }  } |

* 확인할 것은 public ModelAndView testMapArgumentResolve(CommandMap commandMap) throws Exception 이다.
* 파라미터로 등록된 CommandMap에 있는 모든 파라미터를 iterator를 이용하여 출력하였다.
* 서버 가동 후 아래와 같이 테스트 해본다.
* localhost:8080/windlidar/windlidar/testMapArgumentResolver.do?aaa=tmp

### 사용자 관리 프로그램 개발

#### 사용자 등록 페이지 호출

1. JSP 페이지 만들기

userRegister.jsp 파일을 WEB-INF/views/user 폴더 아래에 파일을 생성

|  |
| --- |
| <%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"* pageEncoding=*"UTF-8"*%>  <!DOCTYPE html>  <html lang=*"ko"*>  <head>  <%@ include file=*"/WEB-INF/include/header.jsp"* %>  </head>  <body>  <form id=*"frm"*>  <table class=*"board\_view"*>  <colgroup>  <col width=*"20%"*>  <col width=*"\*"*/>  </colgroup>  <caption>사용자 등록</caption>  <tbody>  <tr>  <th scope=*"row"*>사용자 아이디</th>  <td><input type=*"text"* id=*"id"* name=*"id"* class=*"wdp\_80"*></input></td>  </tr>  <tr>  <th scope=*"row"*>사용자 성명</th>  <td><input type=*"text"* id=*"name"* name=*"name"* class=*"wdp\_100"*></input></td>  </tr>  <tr>  <th scope=*"row"*>사용자 패스워드</th>  <td><input type=*"password"* id=*"pass"* name=*"pass"* class=*"wdp\_80"*></input></td>  </tr>  <tr>  <th scope=*"row"*>사용자 메일주소</th>  <td><input type=*"text"* id=*"email"* name=*"email"* class=*"wdp\_140"*></input></td>  </tr>  <tr>  <th scope=*"row"*>권한 관리</th>  <td>  <input type=*"radio"* id=*"auth\_chk"* name=*"auth\_chk"* value=*"1"* class=*"wdp\_10"*>일반회원</input>  <input type=*"radio"* id=*"auth\_chk"* name=*"auth\_chk"* value=*"2"* class=*"wdp\_10"*>관리자권한</input>  <input type=*"radio"* id=*"auth\_chk"* name=*"auth\_chk"* value=*"0"* class=*"wdp\_10"*>접속차단</input>  </td>  </tr>  </tbody>  </table>  <a href=*"#this"* class=*"btn"* id=*"write"* >사용자 등록</a>  <a href=*"#this"* class=*"btn"* id=*"write"* >정보수정</a>  <a href=*"#this"* class=*"btn"* id=*"list"* >목록으로</a>  </form>  <%@ include file=*"/WEB-INF/include/body.jsp"* %>  <script type=*"text/javascript"*>  $(document).ready(**function**(){    });  </script>  </body>  </html> |

1. Controller 클래스 생성

|  |
| --- |
| **package** com.hist.windlidar.controller;  **import** java.util.Iterator;  **import** java.util.List;  **import** java.util.Map;  **import** java.util.Map.Entry;  **import** javax.annotation.Resource;  **import** org.apache.log4j.Logger;  **import** org.springframework.stereotype.Controller;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.servlet.ModelAndView;  **import** com.hist.windlidar.common.CommandMap;  **import** com.hist.windlidar.service.WindLidarService;  @Controller  **public** **class** UserController {  Logger log = Logger.*getLogger*(**this**.getClass());    @Resource(name="windLidarService")  **private** WindLidarService windLidarService;    /\*\*  \* 사용자 등록 페이지로 이동한다  \* **@param** commandMap  \* **@return**  \* **@throws** Exception  \*/  @RequestMapping(value="/userRegister.do")  **public** ModelAndView userRegister(CommandMap commandMap) **throws** Exception  {  ModelAndView mv = **new** ModelAndView("/user/userRegister");    **return** mv;  }  } |

* com.hist.windlidar.controller 패키지 아래 UserController 클래스 생성.
* @RequestMapping(value=”/userRegister.do”) 선언
* 위의 선언은 URL 접속시 contents명/userRegister.do로 접속하겠다는 뜻.
* userRegister.do로 URL 접속을 시도하면 메소드 맵핑이 userRegister 메소드가 호출되고 이 메소드에서 /user/userRegister View를 호출한다. 이 View는 WEB-INF/views/user/userRegister.jsp 페이지를 호출한다.

1. 페이지 호출하기

<http://localhost:8080/windlidar/userRegister.do>

만일 js파일이나 css 파일의 경로가 호출되지 않으면 아래를 확인한다.

web.xml파일에서 url-pattern이 ‘/’ 경로 표시로 되어 있으면 css, js 등의 경로가 인식이 안된다. 이 점에 대해서 유의한다.

|  |
| --- |
| <servlet-mapping>  <servlet-name>appServlet</servlet-name>  <url-pattern>\*.do</url-pattern>  </servlet-mapping> |

#### 사용자 등록하기

##### JSP 화면

|  |
| --- |
| <%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"* pageEncoding=*"UTF-8"*%>  <!DOCTYPE html>  <html lang=*"ko"*>  <head>  <%@ include file=*"/WEB-INF/include/header.jsp"* %>  </head>  <body>  <form id=*"frm"*>  <table class=*"board\_view"*>  <colgroup>  <col width=*"20%"*>  <col width=*"\*"*/>  </colgroup>  <caption>사용자 등록</caption>  <tbody>  <tr>  <th scope=*"row"*>사용자 아이디</th>  <td><input type=*"text"* id=*"id"* name=*"id"* class=*"wdp\_80"*></input></td>  </tr>  <tr>  <th scope=*"row"*>사용자 성명</th>  <td><input type=*"text"* id=*"name"* name=*"name"* class=*"wdp\_100"*></input></td>  </tr>  <tr>  <th scope=*"row"*>사용자 패스워드</th>  <td><input type=*"password"* id=*"pass"* name=*"pass"* class=*"wdp\_80"*></input></td>  </tr>  <tr>  <th scope=*"row"*>사용자 메일주소</th>  <td><input type=*"text"* id=*"email"* name=*"email"* class=*"wdp\_140"*></input></td>  </tr>  <tr>  <th scope=*"row"*>권한 관리</th>  <td>  <input type=*"radio"* id=*"auth\_chk"* name=*"auth\_chk"* value=*"1"* class=*"wdp\_10"*>일반회원  <input type=*"radio"* id=*"auth\_chk"* name=*"auth\_chk"* value=*"2"* class=*"wdp\_10"*>관리자권한  <input type=*"radio"* id=*"auth\_chk"* name=*"auth\_chk"* value=*"0"* class=*"wdp\_10"*>접속차단  </td>  </tr>  </tbody>  </table>  <a href=*"#this"* class=*"btn"* id=*"write"* >사용자 등록</a>  <a href=*"#this"* class=*"btn"* id=*"update"* >정보수정</a>  <a href=*"#this"* class=*"btn"* id=*"list"* >목록으로</a>  </form>  <%@ include file=*"/WEB-INF/include/body.jsp"* %>  <script type=*"text/javascript"*>  $(document).ready(**function**(){  $("#list").on("click", **function**(e){  e.preventDefault();  fn\_openMemberList();  });  $("#write").on("click", **function**(e){  e.preventDefault();  fn\_memberSave();  });  });    **function** fn\_openMemberList()  {  **var** comSubmit = **new** ComSubmit();  comSubmit.setUrl("<c:url value='/userMemberList.do' />");  comSubmit.submit();  }    **function** fn\_memberSave()  {  **var** comSubmit = **new** ComSubmit("frm");  comSubmit.setUrl("<c:url value='/memberInsert.do' />");        comSubmit.submit();  }  </script>  </body>  </html> |

HTML Form 태그에 정의된 내용들은 모두 소문자로 되어 있으니 파라미터로 넘어가서 서블릿 단에서 데이터를 받을 때 소문자로 처리된다. 이점 유의 해야 한다. 사용자가 사용자 등록 버튼을 클릭하면 fn\_membrSave() 메소드가 호출되어 memberInsert.do 컨트롤러를 호출한다.

##### Controller 클래스

|  |
| --- |
| **package** com.hist.windlidar.controller;  **import** java.util.Iterator;  **import** java.util.List;  **import** java.util.Map;  **import** java.util.Map.Entry;  **import** javax.annotation.Resource;  **import** org.apache.log4j.Logger;  **import** org.springframework.stereotype.Controller;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.servlet.ModelAndView;  **import** com.hist.windlidar.common.CommandMap;  **import** com.hist.windlidar.service.MemberService;  @Controller  **public** **class** UserController {  Logger log = Logger.*getLogger*(**this**.getClass());      @Resource(name="memberService")  **private** MemberService memberService;    /\*\*  \* 사용자 등록 페이지로 이동한다  \* **@param** commandMap  \* **@return**  \* **@throws** Exception  \*/  @RequestMapping(value="/userRegister.do")  **public** ModelAndView userRegister(CommandMap commandMap) **throws** Exception  {  ModelAndView mv = **new** ModelAndView("/user/userRegister");    **return** mv;  }    @RequestMapping(value="/memberInsert.do")  **public** ModelAndView memberInsert(CommandMap commandMap) **throws** Exception  {  ModelAndView mv = **new** ModelAndView("redirect:/memberList.do");    //Map<String, Object> map = commandMap.getMap();  //System.out.println("data : " + map.toString());      memberService.insertMember(commandMap.getMap());    **return** mv;  }    @RequestMapping(value="/memberList.do")  **public** ModelAndView memberList(CommandMap commandMap) **throws** Exception  {  ModelAndView mv = **new** ModelAndView("/user/userList");  List<Map<String, Object>> list = memberService.selectMemberList(commandMap.getMap());    // 결과를 클라이언트에 전달  mv.addObject("list", list);  **return** mv;  }  } |

* UserController 클래스를 정의하고 @Controller로 등록하였다.
* memberInsert.do URL로 호출되면 RequestMapping에 의해서 memberInsert 메소드가 호출된다.
* 이 때 파라미터로는 CommandMap의 사용자 정의 맵을 통해서 넘어온다. 이 정의 맵은 이전에 HandlerMethodArgumentResolver 클래스 정의에서 처럼 파라미터로 넘어오는 부분을 우리가 정의한 커스텀 HandlerMethodArgumentResolver 클래스를 통해서 넘어온다.
* 넘어온 CommandMap을 비즈니스 로직을 담당하는 서비스 클래스에 전달한다.
* 서비스 클래스는 @Resource(name=”memberService”)에 정의했다. 대소문자 구분을 주의해야 한다.
* 서비스 클래스에서 제공하는 insertMember 메소드에 CommandMap을 파라미터로 넘겨준다.
* 완료가 되면 출력 페이지로 ModelAndView 클래스에 redirect:/memberList.do를 인자로 넘겨주었다. 호출되는 페이지는 memberList.do와 연관되는 서블릿으로 전달할 것이다.

##### Service 클래스

* MemberService Interface

|  |
| --- |
| package com.hist.windlidar.service;  import java.util.List;  import java.util.Map;  public interface MemberService {  List<Map<String, Object>> selectMemberList(Map<String, Object> commandMap) throws Exception;  void insertMember(Map<String, Object> map) throws Exception;  } |

* MemberServiceImpl 구현 클래스

|  |
| --- |
| **package** com.hist.windlidar.service;  **import** java.util.List;  **import** java.util.Map;  **import** javax.annotation.Resource;  **import** org.apache.log4j.Logger;  **import** org.springframework.stereotype.Service;  **import** com.hist.windlidar.dao.MemberDAO;  @Service("memberService")  **public** **class** MemberServiceImpl **implements** MemberService {  Logger log = Logger.*getLogger*(**this**.getClass());    @Resource(name="memberDAO")  **private** MemberDAO memberDAO;    @Override  **public** List<Map<String, Object>> selectMemberList(Map<String, Object> commandMap) **throws** Exception {  // **TODO** Auto-generated method stub  **return** memberDAO.selectMemberList(commandMap);  }  @Override  **public** **void** insertMember(Map<String, Object> map) **throws** Exception {  // **TODO** Auto-generated method stub  memberDAO.insertMember(map);  }  } |

* 서비스 클래스는 @Service(“memberService”) 사용해서 서비스 클래스로 등록한다.
* 등록시 대소문자에 유의한다.
* 컨트롤러 클래스에서 호출된 메소드는 DAO 클래스에 파라미터로 전달한다.

##### DAO 클래스

|  |
| --- |
| package com.hist.windlidar.dao;  import java.util.List;  import java.util.Map;  import org.springframework.stereotype.Repository;  @Repository("memberDAO")  public class MemberDAO extends AbstractDAO {    /\*\*  \* Map에서 넘어온 사용자 정보를 가지고 데이터베이스에 등록한다.  \* 쿼리문 : insertMember 쿼리 참조  \*  \* @param map  \* @throws Exception  \*/  public void insertMember(Map<String, Object> map) throws Exception  {  insert("windlidar.insertMember", map);  }    /\*\*  \* 사용자 리스트를 조회해서 Map 에 담아서 리턴한다.  \*  \* @param map  \* @return  \* @throws Exception  \*/  @SuppressWarnings("unchecked")  public List<Map<String, Object>> selectMemberList(Map<String, Object> map) throws Exception  {  return (List<Map<String, Object>>) selectList("windlidar.selectMemberList", map);  }  } |

* 서비스 클래스에 넘어온 Map 데이터를 MyBatis에 연결된 메소드에 전달해서 데이터베이스에 사용자를 등록한다.
* AbstractDAO 클래스에 @Autowired로 연결된 SqlSessionTemplate에서 데이터베이스를 처리하는데 아래 SQL 쿼리에 등록된 아이디와 파라미터로 해서 데이터를 처리한다.
* SqlSessionTemplate 클래스는 context-mapper.xml에 bean으로 등록되어 있다. 등록된 bean을 autwired 로 연견 한 것이다.

##### SQL 쿼리

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd">    <mapper namespace=*"windlidar"*>  <insert id=*"insertMember"* parameterType=*"hashmap"*>  <![CDATA[  INSERT INTO T\_USR\_INFO  (  ID,  NAME,  PASS,  EMAIL,  AUTH\_CHK,  REG\_DT  ) VALUES (  #{id},  #{name},  #{pass},  #{email},  #{auth\_chk},  current\_timestamp  )  ]]>  </insert>  </mapper> |

* DAT 클래스에서 SQL 쿼리문 연결 시 아래와 같이 연결한다.
* insert("windlidar.insertMember", map);

#### 사용자 목록 페이지

##### JSP 화면

|  |
| --- |
| **function** fn\_openMemberList()  {  **var** comSubmit = **new** ComSubmit();  comSubmit.setUrl("<c:url value='/memberList.do' />");  comSubmit.submit();  } |

* 사용자가 [목록으로] 버튼을 클릭했을 때 위의 함수가 호출된다.
* 호출된 함수는 userMemberList.do의 서블릿을 호출하게 된다.
* 이 호출에 대한 URL Mapping은 컨트롤러 클래스에 처리한다.

##### Controller 클래스

|  |
| --- |
| @Controller  **public** **class** UserController {  Logger log = Logger.*getLogger*(**this**.getClass());      @Resource(name="memberService")  **private** MemberService memberService;  @RequestMapping(value="/memberList.do")  **public** ModelAndView memberList(CommandMap commandMap) **throws** Exception  {  ModelAndView mv = **new** ModelAndView("/user/userList");  List<Map<String, Object>> list = memberService.selectMemberList(commandMap.getMap());    // 결과를 클라이언트에 전달  mv.addObject("list", list);  **return** mv;  }  } |

* @Controller를 선언하여 Controller 클래스를 정의한다.
* 비즈니스 로직을 담당할 서비스 클래스를 @Resource를 사용하여 등록한다.
* @Resource(name=”memberService”)
* 사용자가 호출한 URL을 맵핑하기 위한 @RequestMapping을 선언한다.
* memberList.do로 호출된 URL은 memberList 메소드로 호출된다.
* memberList 메소드를 사용자 파라미터를 입력 받아서 Service 클래스에서 데이터를 조회하여 List 객체를 담아서 ModelAndView 클래스에 담아서 리턴한다.
* 리턴되는 페이지는 /user/userList의 jsp 페이지로 호출된다.

##### Service 클래스

|  |
| --- |
| @Service("memberService")  **public** **class** MemberServiceImpl **implements** MemberService {  Logger log = Logger.*getLogger*(**this**.getClass());    @Resource(name="memberDAO")  **private** MemberDAO memberDAO;  @Override  **public** List<Map<String, Object>> selectMemberList(Map<String, Object> commandMap) **throws** Exception {  // **TODO** Auto-generated method stub  **return** memberDAO.selectMemberList(commandMap);  }  } |

* @Service(“memberService”)를 사용하여 Sevice 클래스임을 정의한다. 호출되는 서비스는 대소문자를 구분한다.
* DAO 클래스를 사용하기 위해서 @Resource(name=”memberDAO”)를 선언한다.
* selectMemberList 메소드에서 DAO의 selectMemberList 메소드를 호출하여 데이터베이스에서 데이터를 조회해서 리턴한다.

##### DAO 클래스

|  |
| --- |
| @Repository("memberDAO")  **public** **class** MemberDAO **extends** AbstractDAO {  /\*\*  \* 사용자 리스트를 조회해서 Map 에 담아서 리턴한다.  \*  \* **@param** map  \* **@return**  \* **@throws** Exception  \*/  @SuppressWarnings("unchecked")  **public** List<Map<String, Object>> selectMemberList(Map<String, Object> map) **throws** Exception  {  **return** (List<Map<String, Object>>) selectList("windlidar.selectMemberList", map);  }  } |

* @Repository(“memberDAO”) 를 정의하고 AbstractDAO 인터페이스를 상속 받는다.
* MyBatis를 사용한 쿼리문을 호출하기 위해서 SQLxml에 정의한 아이디를 호출한다.

##### SQL 쿼리

|  |
| --- |
| <select id=*"selectMemberList"* parameterType=*"hashmap"* resultType=*"hashmap"*>  <![CDATA[  SELECT  ID,  NAME,  PASS,  EMAIL,  AUTH\_CHK,  LAST\_DT,  REG\_DT  FROM  T\_USR\_INFO  ORDER BY REG\_DT ASC  ]]>  </select> |

#### 사용자 수정 화면 호출

##### JSP 화면

|  |
| --- |
| <a href=*"#this"* name=*"title"*>${row.ID }</a>  <input type=*"hidden"* id=*"IDX"* value=*"*${row.ID }*"*>  …  $("a[name='title']").on("click", **function**(e){ //제목  e.preventDefault();  fn\_openBoardDetail($(**this**));  });  …  **function** fn\_openBoardDetail(obj){  **var** comSubmit = **new** ComSubmit();  comSubmit.setUrl("<c:url value='/userDetailInfo.do' />");  comSubmit.addParam("IDX", obj.parent().find("#IDX").val());  comSubmit.submit();  } |

* 사용자가 아이디를 클릭하면 fn\_openBoardDetail 스크립트가 호출된다.
* 호출된 스크립트 함수에서 userDetailInfo.do 서블릿을 호출하는데 파라미터로 사용자 아이디를 변수로 IDX를 사용해서 넘겨준다.

##### Controller 클래스

|  |
| --- |
| /\*\*  \* 사용자 상세 정보를 조회한다.  \* 호출 페이지로부터 사용자 아이디로 IDX 변수가 넘어 왔다.  \*  \* **@param** commandMap  \* **@return**  \* **@throws** Exception  \*/  @RequestMapping(value="/userDetailInfo.do")  **public** ModelAndView memberDetailInfo(CommandMap commandMap) **throws** Exception  {  ModelAndView mv = **new** ModelAndView("/user/userRegister");  Map<String, Object> detailInfo = memberService.selectMemberDefailtInfo(commandMap.getMap());    mv.addObject("info", detailInfo);  **return** mv;  } |

* JSP 페이지에서 사용자 상세정보 조회를 위해서 userDetailInfo.do 서블릿을 호출하였다.
* Controller 클래스는 userDetaiInfo.do로 호출된 URL정보를 memberDetailInfo 메소드로 맵핑시킨다.
* 호출된 memberDetailInfo 메소드는 비즈니스 로직을 담당하는 서비스 클래스에 호출해서 상세 정보를 조회하여 Map 객체를 얻어서 ModelAndView 클래스에 “info” 파라미터로 등록한 후 /user/userRegister의 View로 리턴한다.
* /user/userRegister는 JSP페이지가 있는 user폴더의 userRegister.jsp파일이 호출된다.

##### Service 클래스

* MemberService Interface 메소드 선언

|  |
| --- |
| Map<String, Object> selectMemberDefailtInfo(Map<String, Object> commandMap) **throws** Exception; |

* MemberService 구현 클래스

|  |
| --- |
| /\*\*  \* Controller클래스로부터 넘어온 파라미터를 가지고 DAO 클래스에 넘겨준다.  \*  \*/  @Override  **public** Map<String, Object> selectMemberDefailtInfo(Map<String, Object> commandMap) **throws** Exception {  // **TODO** Auto-generated method stub  **return** memberDAO.selectMemberDetail(commandMap);  } |

* Service 클래스는 DAO의 사용자 상세정보 조회 메소드를 호출한다.
* DAO 클래스에서 조회된 상세정보를 Map으로 담아서 리턴한다.

##### DAO 클래스

|  |
| --- |
| /\*\*  \* 사용자의 상세 정보를 조회한다.  \*  \* **@param** map  \* **@return**  \* **@throws** Exception  \*/  @SuppressWarnings("unchecked")  **public** Map<String, Object> selectMemberDetail(Map<String, Object> map) **throws** Exception  {  **return** (Map<String, Object>)selectOne("windlidar.selectMemberDetail", map);  } |

* DAO 클래스는 Service 클래스로부터 호출 받은 메소드에서 MyBatis를 통해서 데이터베이스를 조회한다.
* 조회하는 SQL문은 selectMemberDetail 아이디로 등록된 SQL문을 통해서 조회한다.
* 조회된 데이터는 Map으로 리턴된다.

##### SQL 쿼리

|  |
| --- |
| <select id=*"selectMemberDetail"* parameterType=*"hashmap"* resultType=*"hashmap"*>  <![CDATA[  SELECT  ID,  NAME,  PASS,  EMAIL,  AUTH\_CHK,  LAST\_DT,  REG\_DT  FROM  T\_USR\_INFO  WHERE ID=#{IDX}  ]]>  </select> |

* IDX 변수는 JSP에서 넘어온 변수 값이다.

##### 사용자 수정 화면

|  |
| --- |
| <form id=*"frm"*>  <table class=*"board\_view"*>  <input type=*"hidden"* id=*"IDX"* value=*"*${info.ID }*"*>  <colgroup>  <col width=*"20%"*>  <col width=*"\*"*/>  </colgroup>  <caption>사용자 등록</caption>  <tbody>  <tr>  <th scope=*"row"*>사용자 아이디</th>  <td><input type=*"text"* id=*"id"* name=*"id"* class=*"wdp\_80"* value=*"*${info.ID}*"* <c:if test="${**not empty** info.ID}">disabled</c:if> ></input></td>  </tr>  <tr>  <th scope=*"row"*>사용자 성명</th>  <td><input type=*"text"* id=*"name"* name=*"name"* class=*"wdp\_100"* value=*"*${info.NAME}*"*></input></td>  </tr>  <tr>  <th scope=*"row"*>사용자 패스워드</th>  <td><input type=*"password"* id=*"pass"* name=*"pass"* class=*"wdp\_80"* value=*"*${info.PASS}*"*></input></td>  </tr>  <tr>  <th scope=*"row"*>사용자 메일주소</th>  <td><input type=*"text"* id=*"email"* name=*"email"* class=*"wdp\_140"* value=*"*${info.EMAIL}*"*></input></td>  </tr>  <tr>  <th scope=*"row"*>권한 관리</th>  <td>  <input type=*"radio"* id=*"auth\_chk"* name=*"auth\_chk"* value=*"1"* class=*"wdp\_10"* <c:if test="${info.AUTH\_CHK == '1'}">checked</c:if>>일반회원  <input type=*"radio"* id=*"auth\_chk"* name=*"auth\_chk"* value=*"2"* class=*"wdp\_10"* <c:if test="${info.AUTH\_CHK == '2'}">checked</c:if>>관리자권한  <input type=*"radio"* id=*"auth\_chk"* name=*"auth\_chk"* value=*"0"* class=*"wdp\_10"* <c:if test="${info.AUTH\_CHK == '0'}">checked</c:if>>접속차단  </td>  </tr>  </tbody>  </table>  <c:if test=*"*${**empty** info.ID}*"*><a href=*"#this"* class=*"btn"* id=*"write"*>사용자 등록</a></c:if>  <a href=*"#this"* class=*"btn"* id=*"update"* >정보수정</a>  <c:if test=*"*${**not empty** info}*"*><a href=*"#this"* class=*"btn"* id=*"delete"*>삭제하기</a></c:if>  <a href=*"#this"* class=*"btn"* id=*"list"* >목록으로</a> |

* JSTL 태그 라이브러리 사용에 주의한다.
* 버튼에 대한 제어를 위해서 empty info.ID, not empty info 등을 사용하였다

#### 사용자 정보 수정

##### JSP 화면

|  |
| --- |
| **function** fn\_memberUpdate()  {  **var** comSubmit = **new** ComSubmit("frm");  comSubmit.setUrl("<c:url value='/memberUpdate.do' />");  comSubmit.submit();  } |

* 사용자 수정 페이지에서 수정 버튼을 클릭했을 때 수행되는 메소드이다.
* Controller 클래스 호출을 위해서 URL을 memberUpdate.do 를 사용하였다.
* 이 호출은 Controller 클래스에서의 RequestMapping 과 연결된다.

##### Controller 클래스

|  |
| --- |
| /\*\*  \* 사용자 정보를 업데이트 한다.  \* 파라미터에서 IDX가 Hidden Key로 넘어온다.  \* IDX를 Key로 데이터베이스에서 업데이트를 수행할 수 있도록 한다.  \* **@param** commandMap  \* **@return**  \* **@throws** Exception  \*/  @RequestMapping(value="/memberUpdate.do")  **public** ModelAndView memberUpdate(CommandMap commandMap) **throws** Exception  {  ModelAndView mv = **new** ModelAndView("redirect:/memberList.do");    memberService.updateMember(commandMap.getMap());    **return** mv;    } |

* JSP에서 호출된 memberUpdate.do URL 맵핑을 통해 memberUpdate 메소드가 호출된다.
* 호출된 메소드에서는 Service 클래스에서 updateMember 메소드를 호출한다.
* 호출할 때 JSP 화면에서 Form 태그로 넘어온 CommandMap 객체에서 HashMap을 뽑아서 파라미터로 넘겨준다.
* 서비스 호출이 완료되면 출력 페이지로 redirect:/memberList.do 를 호출해서 memberList.do URL을 호출한다.

##### Service 클래스

|  |
| --- |
| /\*\*  \* 사용자 정보를 업데이트 한다.  \*/  @Override  **public** **int** updateMember(Map<String, Object> commandMap) **throws** Exception {  // **TODO** Auto-generated method stub  **return** memberDAO.updateMember(commandMap);  } |

* Controller 클래스에서 호출된 서비스 클래스는 DAO 클래스의 updateMember 메소드를 호출해서 사용자 정보를 업데이트한다.

##### DAO 클래스

|  |
| --- |
| /\*\*  \* 사용자 정볼르 업데이트 한다.  \* Key id : IDX  \* 수정폼에서 넘어온 파라미터 변수에 대해서 대소문자를 주의한다.  \*  \* **@param** map  \* **@return**  \* **@throws** Exception  \*/  **public** **int** updateMember(Map<String, Object> map) **throws** Exception  {  **return** (Integer) update("windlidar.updateMember", map);  } |

* DAO 클래스는 AbstractDAO 인터페이스를 구현한 updateMember 메소드는 MyBatis에 등록된 SQL문 ID와 파라미터로 데이터베이스 정보를 업데이트 한다.

##### SQL 쿼리

|  |
| --- |
| <update id=*"updateMember"* parameterType=*"hashmap"*>  <![CDATA[  UPDATE T\_USR\_INFO  SET  NAME = #{name}  PASS = #{pass},  EMAIL = #{email},  AUTH\_CHK = #{auth\_chk}  WHERE ID=#{IDX}  ]]>  </update> |

* JSP 폼에서 Hidden으로 넘어온 IDX를 Key로 정보를 업데이트 한다.

#### 사용자 정보 삭제

##### JSP 화면

|  |
| --- |
| **function** fn\_memberDelete()  {  **if** (confirm("삭제를 하시겠습니까?")))  {  **var** comSubmit = **new** ComSubmit("frm");  comSubmit.setUrl("<c:url value='/memberDelete.do' />");  comSubmit.submit();  }  } |

* 사용자가 삭제 버튼을 클릭하면 위의 메소드가 호출된다.
* 호출된 메소드에서는 Form 태그명을 설정하고 폼 태그내에 hidden으로 키값 IDX가 변수로 설정되어 있다.
* URL로 memberDelete.do 를 설정해서 Post 방식으로 호출한다.

##### Controller 클래스

|  |
| --- |
| /\*\*  \* 사용자 정보를 삭제한다.  \* 파라미터에서 IDX가 Hidden으로 Key로 넘어온다.  \* HTML태그에서 name 필드에 등록된 이름으로 맵핑된다.  \*  \* **@param** commandMap  \* **@return**  \* **@throws** Exception  \*/  @RequestMapping(value="/memberDelete.do")  **public** ModelAndView memberDelete(CommandMap commandMap) **throws** Exception  {  ModelAndView mv = **new** ModelAndView("redirect:/memberList.do");  log.info("memberDelete.do called : " + commandMap.getMap().get("IDX"));  memberService.deleteMember(commandMap.getMap());    **return** mv;  } |

* JSP에서 memberDelete.do로 호출된 URL은 RequestMapping을 통해서 memberDelete 메소드가 호출된다.
* 호출된 메소드는 서비스 클래스에 키값으로 넘어온 파라미터를 Hashmap으로 뽑아서 전달한다.
* 서비스에서 비즈니스 처리가 완료되면 ModelAndView에서 설정한 redirect:/memberList.do 설정으로 인한 URL를 호출한다.

##### Service 클래스

|  |
| --- |
| /\*\*  \* 사용자 정보를 삭제한다.  \*/  @Override  **public** **void** deleteMember(Map<String, Object> commandMap) **throws** Exception {  // **TODO** Auto-generated method stub  log.info("deleteMember [User delete] : " + commandMap.get("IDX"));  memberDAO.deleteMember(commandMap);    } |

* Service 클래스는 MemberService 인터페이스에서 정의한 메소드를 구현한다.
* DAO 클래스의 deleteMember 메소드에 파라미터를 넣어서 호출한다.

##### DAO 클래스

|  |
| --- |
| /\*\*  \* 사용자 정볼를 삭제한다.  \* Key ID : IDX  \* JSP 폼에서 넘어온 파라미터 변수에 대해서 대소문자 주의한다.  \*  \* **@param** map  \* **@throws** Exception  \*/  **public** **void** deleteMember(Map<String, Object> map) **throws** Exception  {  delete("windlidar.deleteMember", map);  } |

* Service 클래스에서 호출한 DAO의 deleteMember 메소드에서는 MyBatis와 연결된 메소드 delete 메소드를 호출한다.
* 호출받은 delete 메소드는 SQL 쿼리 아이디를 사용해서 MyBatis와 연계해서 데이터베이스 삭제를 수행한다.

##### SQL 쿼리

|  |
| --- |
| <delete id=*"deleteMember"* parameterType=*"hashmap"*>  <![CDATA[  DELETE FROM T\_USR\_INFO  WHERE ID = #{IDX}  ]]>  </delete> |

* JSP 폼에서 Hidden으로 넘어온 IDX를 Key로 정보를 삭제 한다.