

HONE Smart Platform Hub 설정 가이드

Version	1.0
Date	2018.02.28
Status	Draft
Document ID	〈document id〉
Owner	Hanwha S&C
Approved	2018.02.28

Table of Contents

1	소개.....	4
1.1	목적	4
1.2	운영 환경	4
1.3	Glossary	4
2	개발 환경.....	4
2.1	공통 프로그램.....	4
2.1.1	JAVA.....	4
2.1.2	Eclipse	8
2.2	WAS.....	9
2.2.1	Tomcat	9
3	HONE SMART PLATFORM 개발가이드	15
3.1	온라인 개발가이드	15
3.1.1	HONE Integration.....	15
3.1.2	서비스 생성 방식.....	16
3.1.3	DAO 생성 방식.....	17
3.1.4	Platform 서비스	18
3.1.5	트랜잭션 관리	21
3.1.6	로깅 처리	24
3.2	Sample 소스	27
3.2.1	구성	27
3.2.2	env.xml.....	28
3.2.3	hone-properties.xml	28
3.2.4	applicationContext-server.xml	29
3.2.5	applicationContext-security.xml	29
3.2.6	hone-secure-layer.xml	32
3.2.7	infinispan-config.xml	33
3.2.8	jgroups-tcp-bizhub-sample.xml.xml	34
3.2.9	logback 세팅	36
3.2.10	Service.....	37
4	서버 배포.....	40
4.1	Eclipse 를 이용한 배포.....	41
4.2	기타	43

5	유의사항	43
5.1	Jeus 오류	43
5.2	Weblogic 오류	44
6	MyBatis 세팅	45
6.1	MyBatis 버전 (권장)	45
6.2	상세 설정 예제	46
6.2.1	lib 추가 (/WEB-INF/lib)	46
6.2.2	applicationContext-persistence.xml 수정 예	46
6.2.3	mybatis-config.xml 설정 예	47
6.2.4	xxxxMapper.xml 구현 예	47
6.2.5	LauncherAppDao.java 구현 예	48
	References	49
	Revision History	49

1 소개

1.1 목적

본 문서는 HONE Smart Platform 의 Business Hub 를 로컬 eclipse 를 이용한 개발 방법 설명을 위해 작성되었다.

1.2 운영 환경

Reference	Descriptions

1.3 Glossary

Term	Definition
Business Hub	HONE Smart Platform 의 제품군 중 기간계와 연동을 담당하는 서버

2 개발 환경

2.1 공통 프로그램

다음의 프로그램은 개발 및 실행에 공통으로 필요한 목록으로 반드시 설치한다.

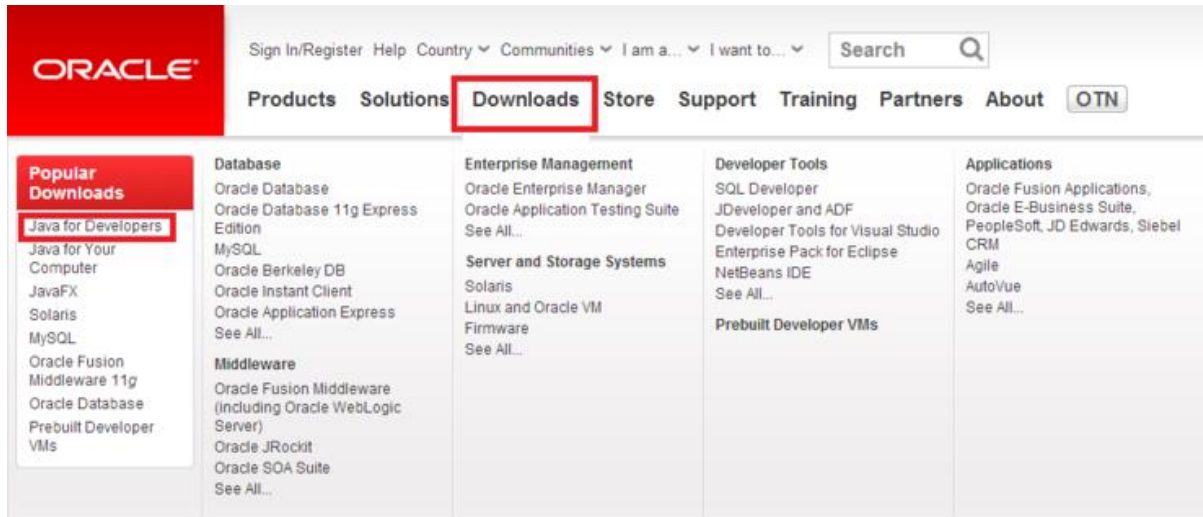
2.1.1 JAVA

2.1.1.1 내려받기

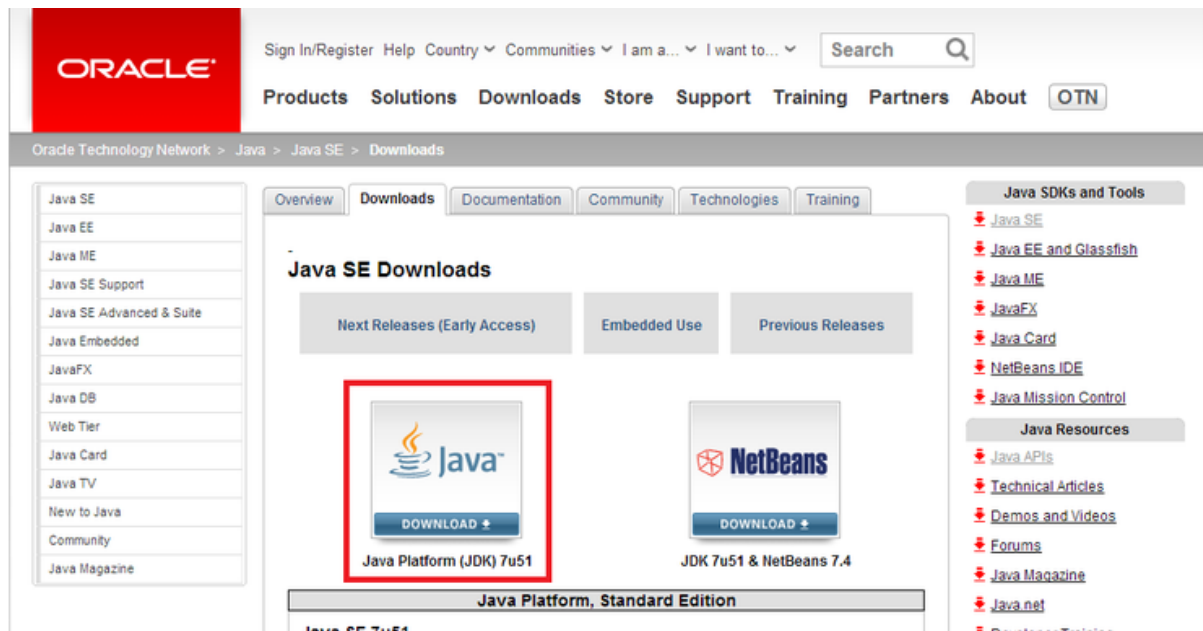
JAVA 는 아래의 경로를 통해 JDK 를 다운로드 받는다.

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

HONE Smart Platform 제품군은 최소 JAVA6 버전에서 개발되었으므로 버전을 확인하여 설치한다. 만약 운영환경에 설치된 JAVA 버전이 상이한 경우 담당자와 협의를 진행한다.



- 1) 브라우저를 이용하여 Oracle 홈페이지에 상단의 “Downloads” 선택 후 우측의 Popular Downloads 목록에서 “Java For Developers”를 선택한다.



- 2) Downloads 화면에서 Java Platform을 선택한다.

Java SE Development Kit 7u51

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

☒ Accept License Agreement ☐ Decline License Agreement

Product / File Description	File Size	Download
Linux ARM v6/v7 Hard Float ABI	67.7 MB	jdk-7u51-linux-arm-vfp-hflt.tar.gz
Linux ARM v6/v7 Soft Float ABI	67.68 MB	jdk-7u51-linux-arm-vfp-sflt.tar.gz
Linux x86	115.65 MB	jdk-7u51-linux-i586.rpm
Linux x86	132.98 MB	jdk-7u51-linux-i586.tar.gz
Linux x64	116.96 MB	jdk-7u51-linux-x64.rpm
Linux x64	131.8 MB	jdk-7u51-linux-x64.tar.gz
Mac OS X x64	179.49 MB	jdk-7u51-macosx-x64.dmg
Solaris x86 (SVR4 package)	140.02 MB	jdk-7u51-solaris-i586.tar.Z
Solaris x86	95.13 MB	jdk-7u51-solaris-i586.tar.gz
Solaris x64 (SVR4 package)	24.53 MB	jdk-7u51-solaris-x64.tar.Z
Solaris x64	16.28 MB	jdk-7u51-solaris-x64.tar.gz
Solaris SPARC (SVR4 package)	139.39 MB	jdk-7u51-solaris-sparc.tar.Z
Solaris SPARC	98.19 MB	jdk-7u51-solaris-sparc.tar.gz
Solaris SPARC 64-bit (SVR4 package)	23.94 MB	jdk-7u51-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	18.33 MB	jdk-7u51-solaris-sparcv9.tar.gz
Windows x86	123.64 MB	jdk-7u51-windows-i586.exe
Windows x64	125.46 MB	jdk-7u51-windows-x64.exe

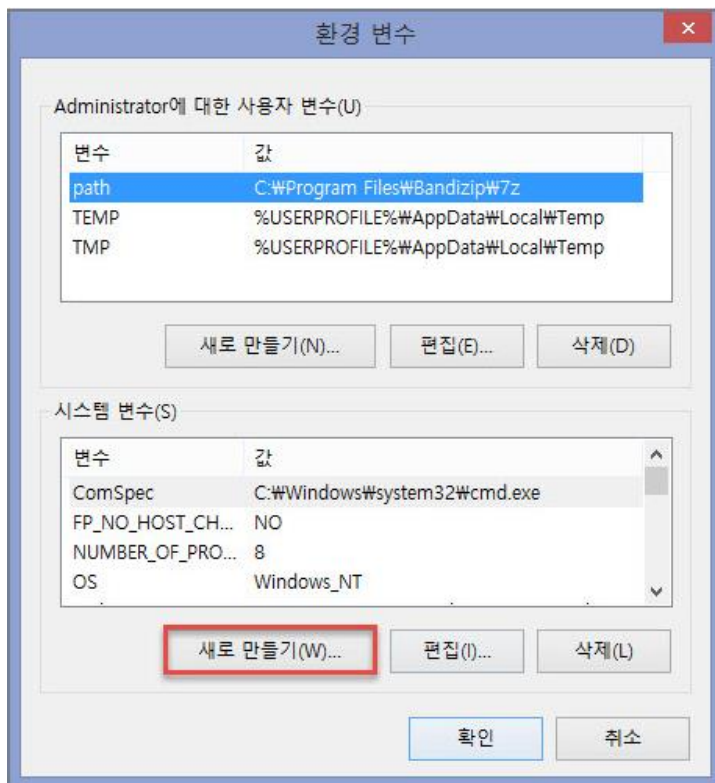
- 3) 상단에 라이선스 관련 승인 여부를 체크하고 설치되는 운영환경에 맞는 버전을 선택하면 다운로드가 시작된다.

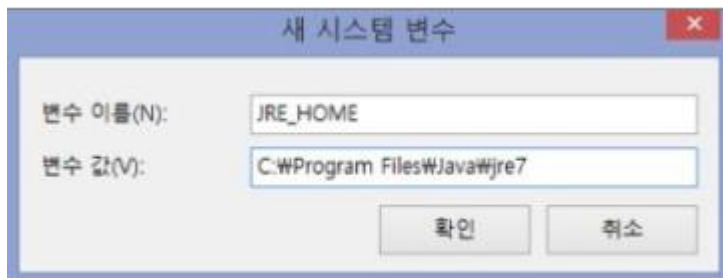
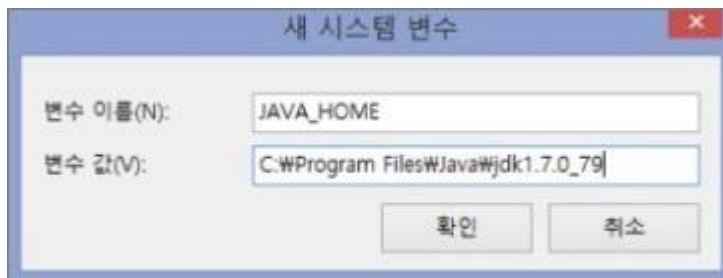
2.1.1.2 설치하기

내려 받은 JDK 파일을 실행하여 설치를 진행한다.

2.1.1.3 설정하기

설치된 JAVA 는 다음과 같이 시스템의 환경변수에 등록하여 설정을 진행한다.





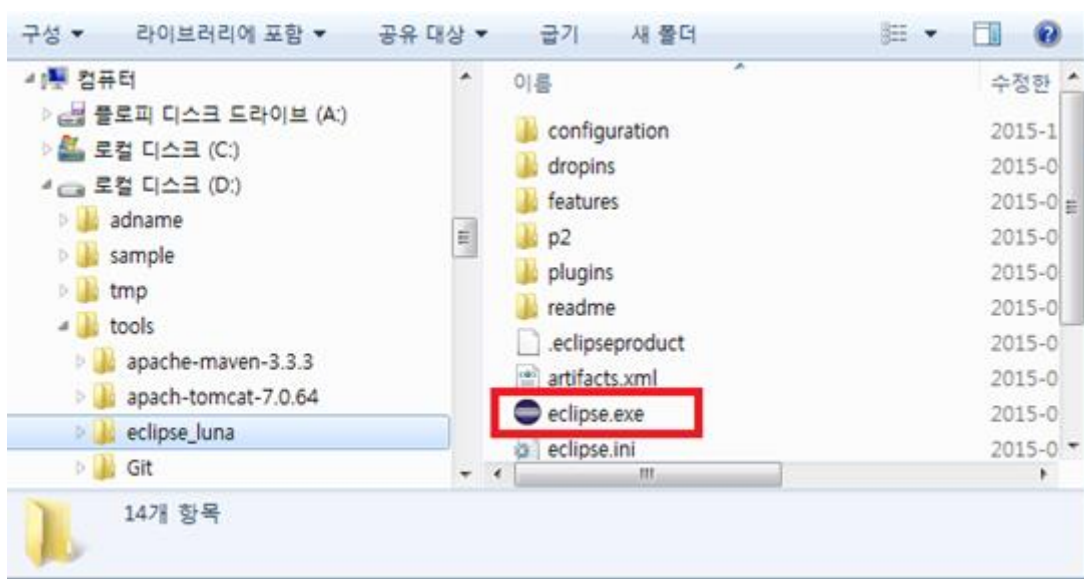
2.1.2 Eclipse

Eclipse 는 Business Hub 소스를 기반으로 빌드 및 내부 테스트 환경을 위해 사용된다.

2.1.2.1 설치하기

Eclipse 는 다음과 같이 설치를 진행한다.

특정 위치 (ex) D:\tools\ eclipse)에 다운로드 받은 설치 파일의 (ex) DEVTOOL_HSP_3.3.6(20180228).zip) 압축을 풀어주면 설치 과정이 완료된다.



eclipse.exe 를 실행하면, eclipse 를 실행할 수 있다.

2.2 WAS

WAS란 Web Application Server의 약어로서 인터넷 상에서 HTTP를 통해 사용자 컴퓨터나 장치에 어플리케이션을 수행해 주는 미들웨어를 의미한다. WAS는 동적 서버 콘텐츠를 수행하는 것으로 일반 웹 서버와 구별되며, 주로 데이터베이스 서버와 같이 수행된다.

일반적으로 사용되는 WAS의 종류는 다음과 같습니다.

WAS 종류	제조 회사	특징
Tomcat	Apache	Servlet Container
JEUS	TmaxSoft	Servlet / EJB Container
WebLogic	Oracle	Servlet / EJB Container
Websphere	IBM	Servlet / EJB Container
Resin	CAUCHO	Servlet Container
IIS	Microsoft	JAVA WAS가 아님

HONE Smart Platform은 UTF-8 기반으로 동작하므로 제품의 정상적인 동작을 위해서 UTF-8 설정을 미리 해야한다.

2.2.1 Tomcat

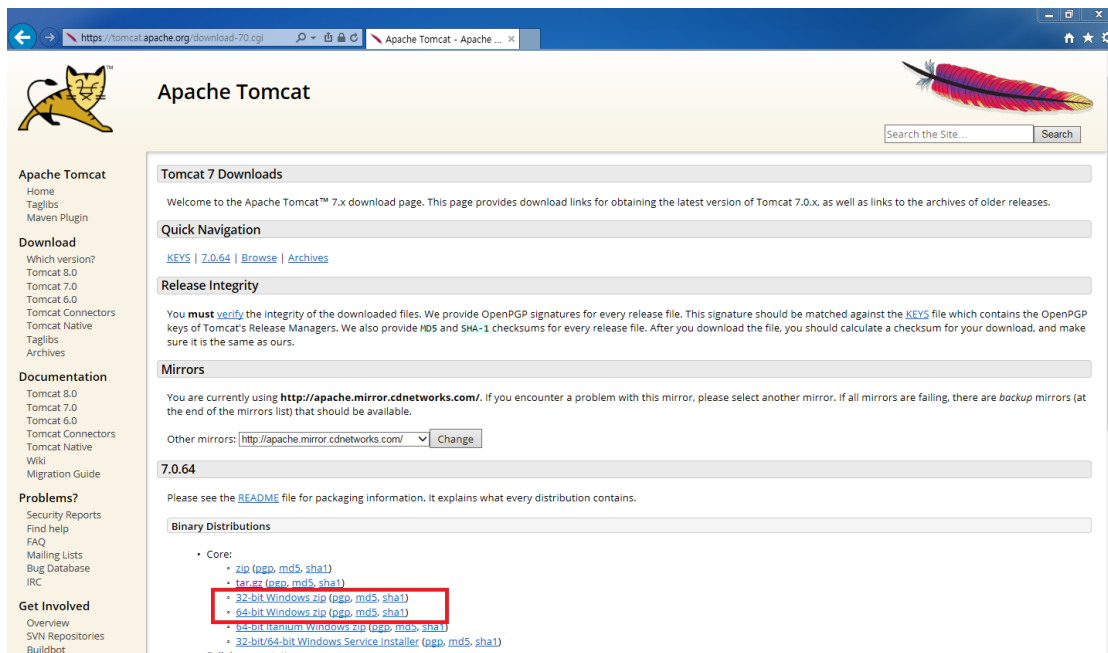
일반적으로 Apache Tomcat은 실제 운영환경에서 사용되는 WAS는 아니며 내부 테스트 환경에서 제한적으로 사용되고 일반적으로 Eclipse 기반으로 운영환경 구성 및 내부 테스트 환경 구성 시 사용된다.

2.2.1.1 내려받기

Tomcat은 아래의 경로를 통해 Tomcat 7.0.64를 다운로드 받습니다.

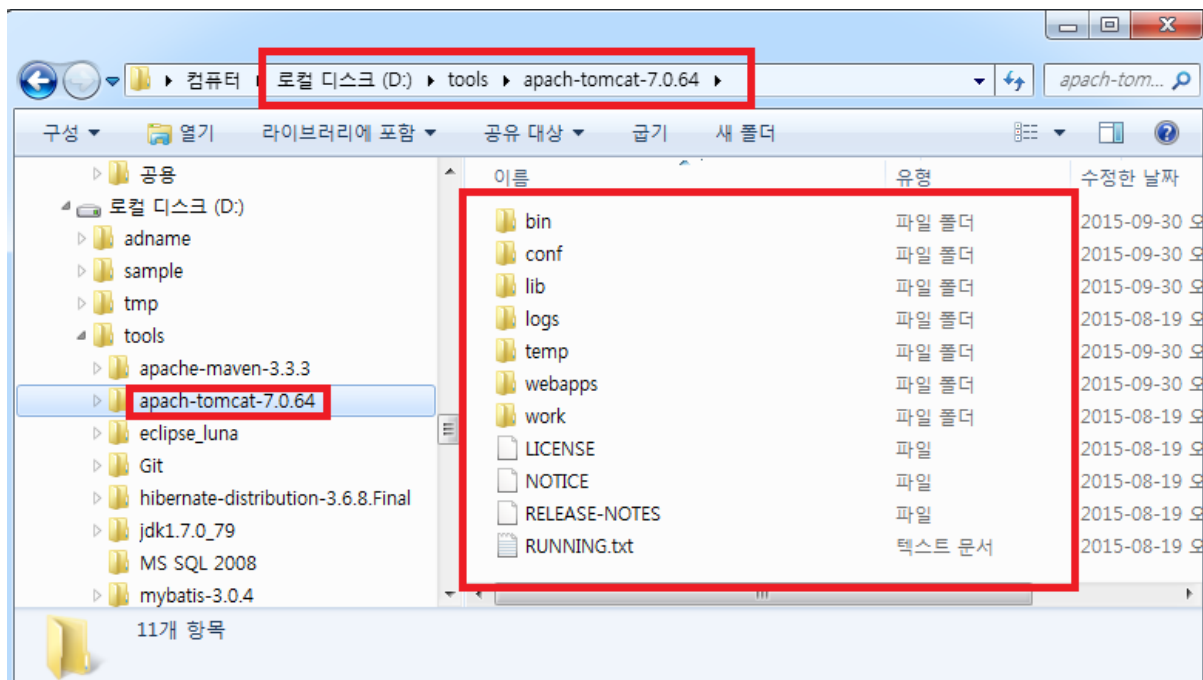
- <https://tomcat.apache.org/download-70.cgi>

브라우저를 이용하여 Apache Tomcat 홈페이지에 좌측의 “Download”의 “Tomcat 7.0”을 선택 후 우측의 화면에서 Core에 zip 설치 파일 중에서 운영환경에 맞는 버전을 선택하면 다운로드가 시작된다.



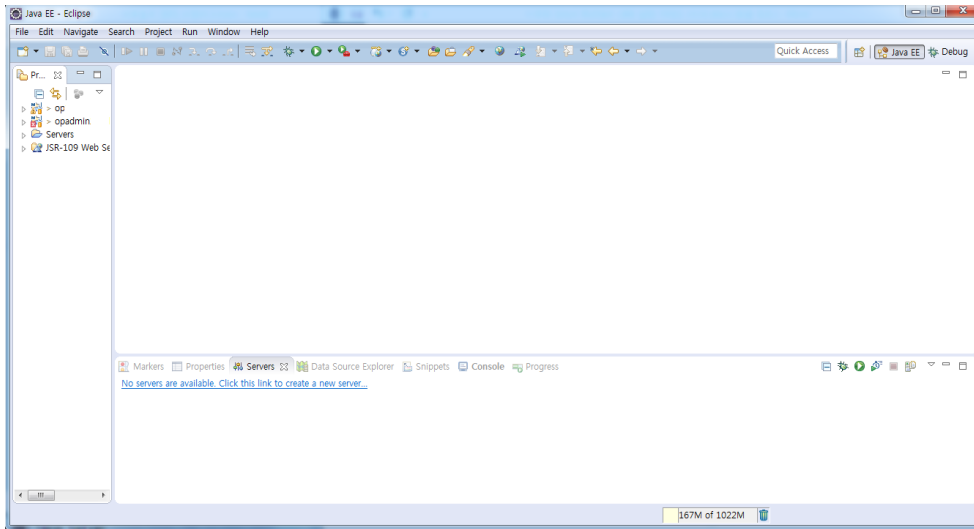
2.2.1.2 설치하기

다운로드 받은 파일을 설치를 원하는 특정 위치에 압축을 해제 합니다. Eclipse 기반의 내부 테스트 환경 구축 시 해당 경로는 Tomcat 설정에 사용됩니다.

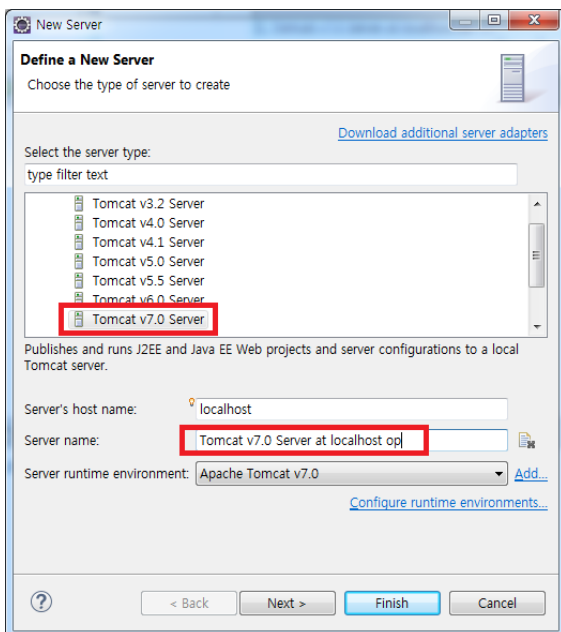


2.2.1.3 실행하기

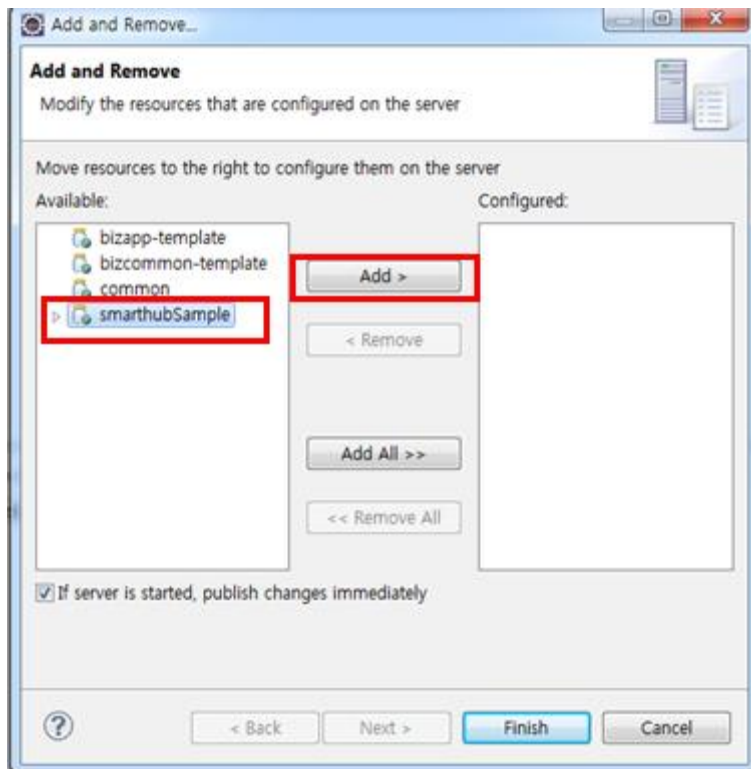
Business Hub 를 Tomcat 에서 실행하기 위해 Server 설정이 필요하다.



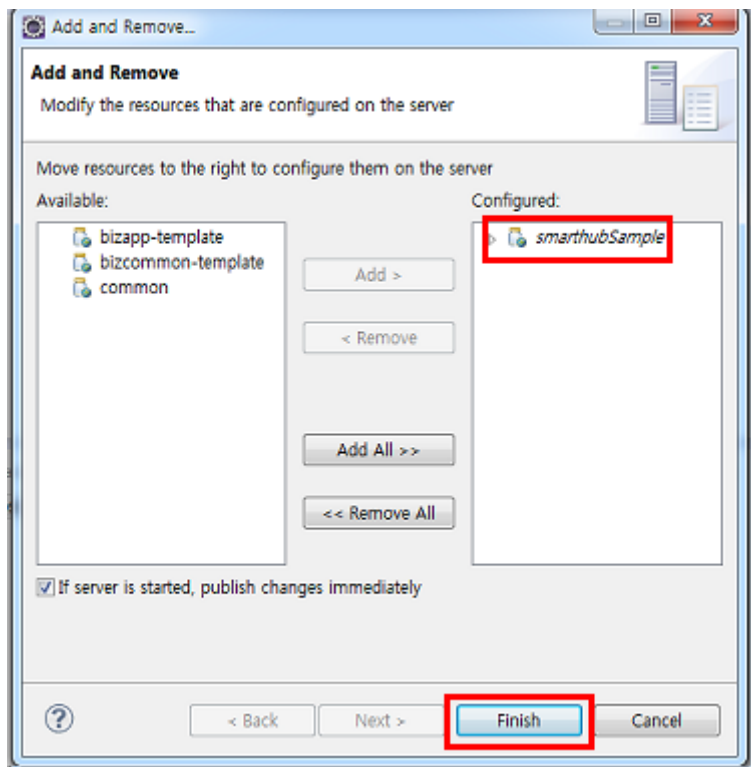
1) Eclipse 에 하단에 Servers 탭을 선택합니다. Servers 에서 새로운 서버를 생성한다. Servers 에서 우클릭하여, new 를 클릭하거나, “NO servers are available. Click this link to create a new server”를 클릭한다.



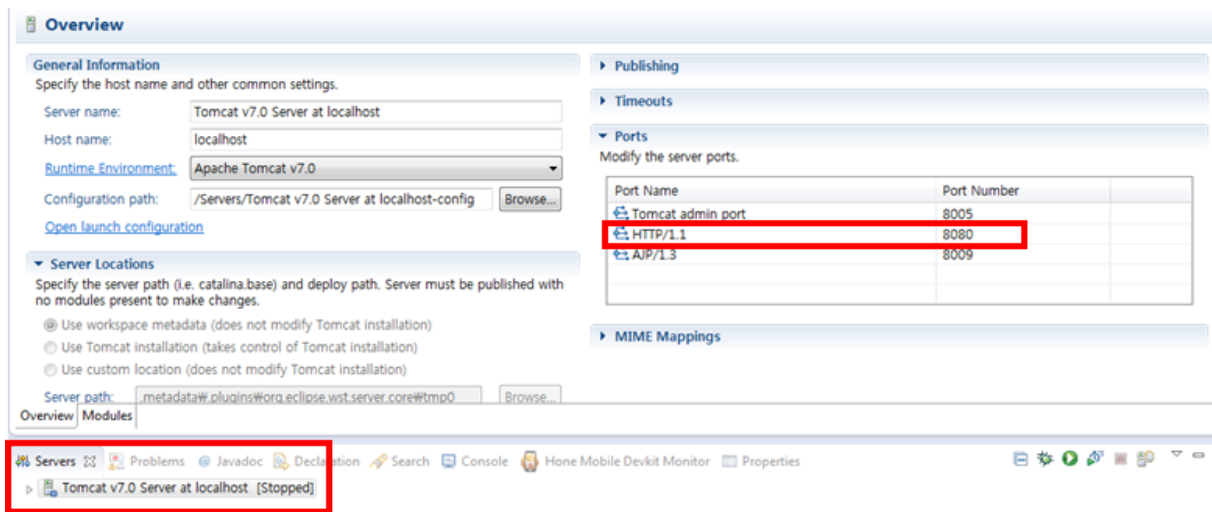
2) New Server 화면에서 Select the server type: 부분에서는 Tomcat v7.0 Server 를 선택한다. Server name:은 Default 이름에 smarthub 를 추가하여 Business Hub 가 실행되는 Server 라는 것을 명시해줍니다. 설정을 완료하면, “Next)”버튼을 클릭한다.



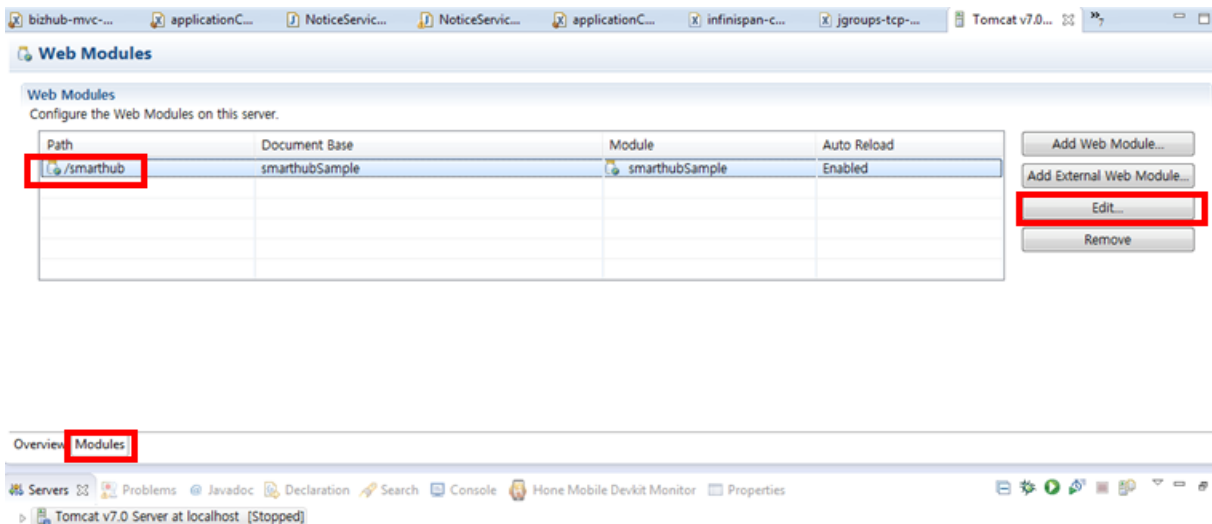
3) 프로젝트를 선택하여 “Add)” 버튼을 클릭한다.



4) 프로젝트가 우측 Configured:에 있는지를 확인하고 “Finish” 버튼을 클릭한다.

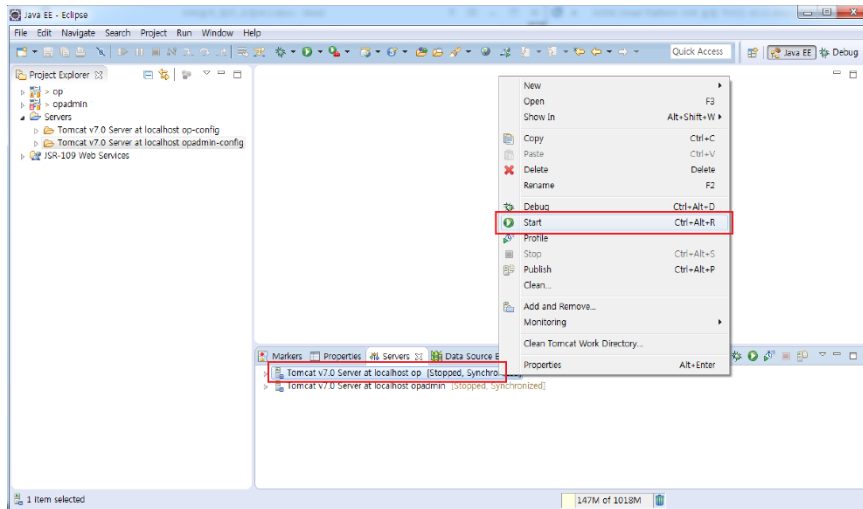


5) server 를 두번 클릭하고 열린 화면에서 8080 로 Port 번호를 변경한다.

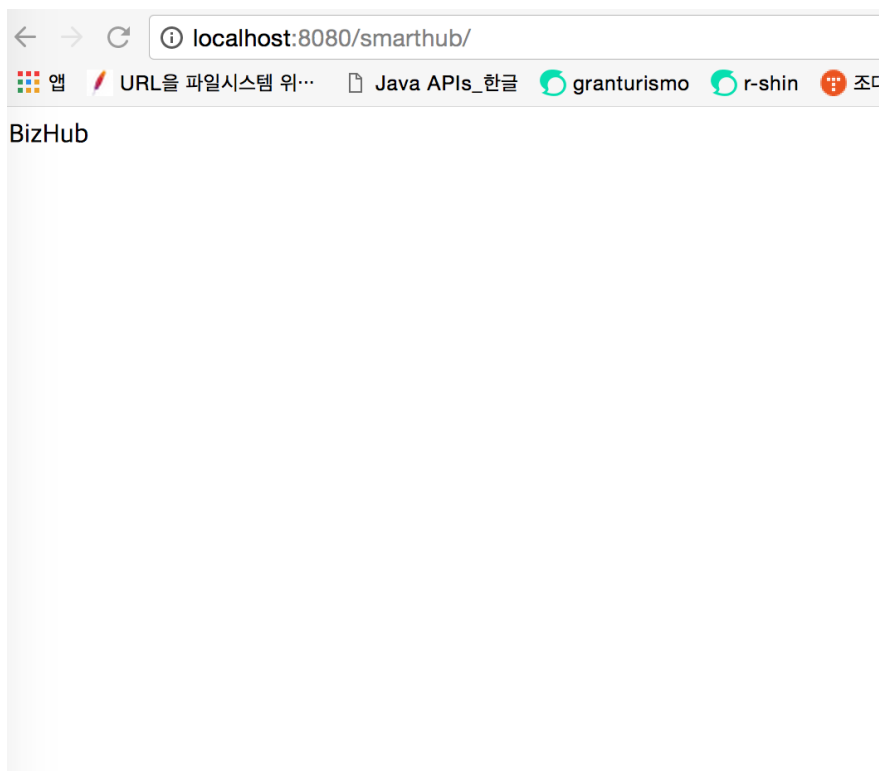


6) server 탭에서 Modules 를 클릭하고 Path 를 smarthub 로 변경 한다.

Eclipse 환경에서 Business Hub 를 실행하는 방법은 다음과 같다.



- 1) Eclipse 하단의 생성한 Server 를 선택하고, 우클릭을 하여 출력된 메뉴에서 Start 를 선택한다.
Tomcat Server 에서 Business Hub 를 실행하게 된다.



- 2) 브라우저에서 URL 주소에 설정한 주소 정보(localhost:8080/smarthub/)를 입력하여 서버가 정상적으로 동작하는 지 확인한다.

3 HONE Smart Platform 개발가이드

3.1 온라인 개발가이드

3.1.1 HONE Integration

Camel 기반의 HONE Integration 기능을 내장하여 레거시 통합에 EIP (Enterprise Integration Patterns) 적용이 가능하다.

3.1.1.1 HONE Integration Flow 작성

Integration Flow 는 Flow Editor 를 이용하여 작성한다.

보다 다양한 Camel 컴포넌트 사용을 위해서는 flow 파일을 직접 편집하거나 Camel DSL 을 사용 할 수 있다.

flow 파일 확장자는 .msf 이며 msf 파일은 integration.xml 의 "msf.namePattern"에 정의된 경로에서 자동 로딩된다.

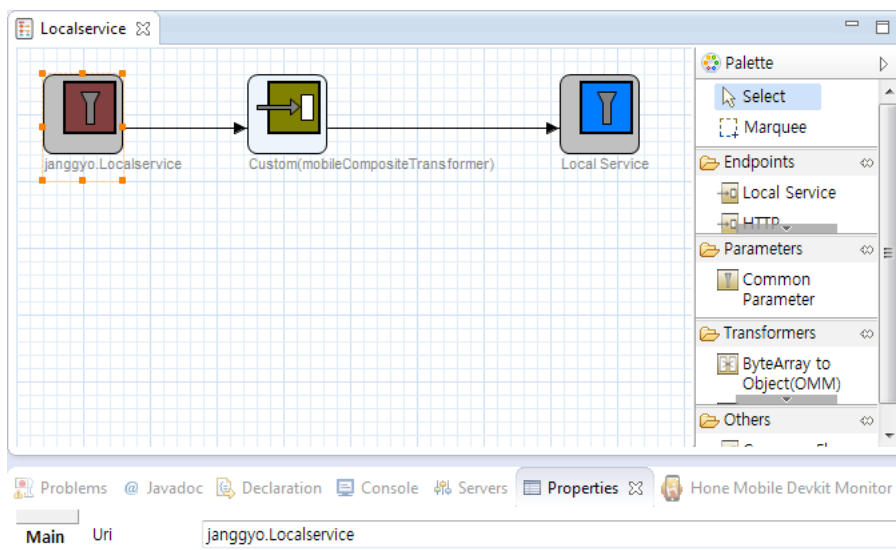
DSL 클래스는 integration.xml 의 "context.option.builderPackage"에 정의된 경로에서 자동 로딩된다.

msf 파일의 기본 설정 위치는 /integration/msf/{applicationName}/xxx.msf .

Business Hub 의 모든 설정은 논리적으로 복수의 어플리케이션 단위로 관리되며, 단말로의 요청도 이 어플리케이션 단위로 호출한다.

로컬 서비스 호출(LocalService.msف가 템플릿 프로젝트에 포함됨)

LocalService.msف는 Business Hub 에 내장된 POJO 기반의 로컬 서비스를 연계하기 위한 flow 를 의미한다.



3.1.1.2 서비스 트랜잭션 ID 매핑

Business Hub 에서 제공 되는 모든 서비스는 단말(-)서버간 통신 규약을 따른다.

서비스 URL 은 다음과 같다.

http(https)://xxx.xxx.xxx.xxx:9999/contextPath/{applicationName}/service/{serviceCategory}/{serviceName}.

서버에서는 TransactionID ({serviceCategory}.{serviceName}) 형태로 관리된다.

서비스 매핑 정보는 msdl 형태의 파일로 관리되면 기본 위치는 아래와 같다.
 /integration/msdl/{applicationName}/xxx.msdl

msdl 문서의 기본 포맷

[TransactionID]

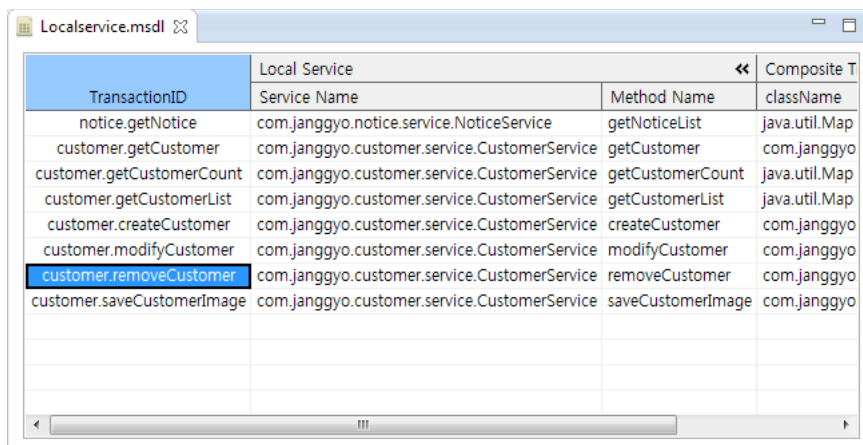
flowId= {msf 파일의 Uri}

[customer.getCustomer]

flowId=Localservice

msdl 파일의 추가 속성 포맷은 mssd 에 정의 되어 있으며, 추가 속성은 해당 camel flow 호출시 property 에 해당 값이 설정 된다

msdl 파일 편집기



TransactionID	Local Service	Method Name	Composite T
	Service Name		className
notice.getNotice	com.janggyo.notice.service.NoticeService	getNoticeList	java.util.Map
customer.getCustomer	com.janggyo.customer.service.CustomerService	getCustomer	com.janggyo
customer.getCustomerCount	com.janggyo.customer.service.CustomerService	getCustomerCount	java.util.Map
customer.getCustomerList	com.janggyo.customer.service.CustomerService	getCustomerList	java.util.Map
customer.createCustomer	com.janggyo.customer.service.CustomerService	createCustomer	com.janggyo
customer.modifyCustomer	com.janggyo.customer.service.CustomerService	modifyCustomer	com.janggyo
customer.removeCustomer	com.janggyo.customer.service.CustomerService	removeCustomer	com.janggyo
customer.saveCustomerImage	com.janggyo.customer.service.CustomerService	saveCustomerImage	com.janggyo

3.1.2 서비스 생성 방식

Service 클래스 작성을 위해서는 Service 인터페이스 작성 후 이에 해당하는 구현 클래스만 작성하면 된다.
 비즈니스 로직을 작성하는 레이어이므로 특별히 상속을 위해 제공하는 상위 클래스는 없다.

아래는 Service 클래스의 간단한 클래스 다이어그램 및 클래스 정의 부분이다.



표준 Bean 등록 방식을 어노테이션 기반으로 하므로 Service 클래스 선언부 위에 @Named 라는 어노테이션을 기술해주어야 한다

```

@Named("package.CustomerService")
public class CustomerServiceImpl implements CustomerService {
    @Inject
    CustomerDao customerDao;

    public Customer getCustomer(Customer customer){
        return customerDao.selectCustomer(customer);
    }
}
  
```

3.1.3 DAO 생성 방식

일반적으로 Spring 프레임워크를 사용해 Dao 를 구현하는 경우 여러 데이터 제어용 API 를 제공해주는 ~Template 류의 클래스를 사용해 데이터 접근 로직을 구현하게 된다.

스프링에서는 JDBC 데이터 접근에 대한 여러 형태의 Template 클래스를 제공하고 있는데 그 종류와 특징은 아래와 같다.

- JdbcTemplate
가장 기본이 되는 클래스로 스프링 초기부터 제공하는 기본 템플릿 클래스
- NamedParameterJdbcTemplate
JdbcTemplate클래스를 포함한 클래스로 쿼리수행 시 변수명을 "?" 대신 "이름"으로 지정해 사용

이러한 Template 클래스들을 DI 등의 작업없이 바로 사용할 수 있도록 ~DaoSupport 류의 상위 추상 클래스를 제공하며 SQL 을 독립적으로 관리하는 DBIO 기반의 개발에 사용할 수 있는 클래스로 DbioDaoSupport 클래스를 제공하고 있다

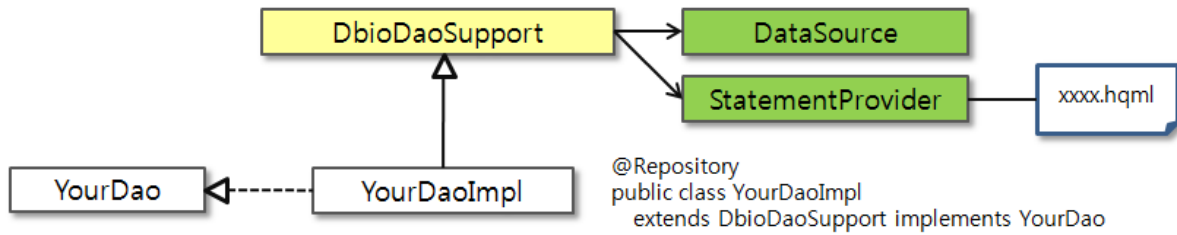
3.1.3.1 기본 가이드

Dao 클래스 작성을 위해서는 Dao 인터페이스 작성후 DbioDaoSupport 를 상속받는 구현 클래스를 작성하면 된다.

이때 상위 클래스인 DbioDaoSupport 내부에서 자동으로 DataSource 를 인식하여 JdbcTemplate 을 생성하므로 별도로 JdbcTemplate 이나 DataSource 에 대한 참조를 주입하지 않아도 기본적으로 지정된 Datasource Bean 을 주입받아 필요한 JdbcTemplate 을 구성해 제공한다.

단, 다수의 DataSource 를 활용하는 경우에는 별도의 DI 방식등을 활용해 DataSource 나 JdbcTemplate 을 직접 주입할 수 있기 때문에 필요 시 적절히 사용하면 된다.

아래는 Dao 클래스의 간단한 클래스 다이어그램 및 클래스 정의 부분이다.



표준 Bean 등록 방식을 어노테이션 기반으로 하므로 Dao 클래스 선언부 위에 @Repository 라는 어노테이션을 기술해주어야 한다

```

@Repository
public class OracleUserDaoImpl extends DbioDaoSupport implements UserDao {
    public User getUser( User user ) {
        String sql = getSql("UserDao.getUser");
        return queryForObject(sql, new
        BeanPropertySqlParameterSource(user), new
        BeanPropertyRowMapper<User>(User.class));
    }
}
  
```

3.1.4 Platform 서비스

3.1.4.1 Property 서비스

애플리케이션 개발 시 외부의 프로퍼티를 로직 내에 주입시켜야 하는 경우가 발생한다. 이런 경우 외부 프로퍼티 파일에 해당 값들을 설정한 후 이 값을 애플리케이션 로직등에서 사용하게 되는데 HONE Smart Platform 에서는 기본적으로 애플리케이션에서 필요한 프로퍼티를 설정파일 및 애플리케이션 내부에서 사용할 수 있는 모듈을 제공하고 있다.

(a) 설정

기본적으로 /WEB-INF/classes/hone-properties.xml에 프로퍼티 값들을 추가 작성할 수 있으며 이곳에 작성된 프로퍼티들은 HONE Smart Platform의 다른 설정 파일에서 참조하거나 애플리케이션 로직 내에서 접근이 가능하다

아래는 hone-properties.xml파일의 작성 예다.

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
    <entry key="hubId">janggyo</entry>
    <entry key="bizAppResourcePath">resources</entry>
//생략
</properties>
  
```

그리고 기본적으로 HONE Smart Platform 에서 주어진 프로퍼티 설정외에 별도의 프로퍼티 파일을 만들고 각각 다른 프로퍼티 아이디로 등록해 사용하는 방법도 가능하다.

```

<util:properties id="HoneProperty" location="classpath:hone-properties.xml"/>
<util:properties id="AppProperty" location="classpath:app-properties.xml"/>
  
```

(b) 활용

설정을 통해 등록된 프로퍼티는 아래와 같이 설정파일이나 로직내에서 참조 가능하다.

설정파일에서는 Spring 에서 제공하는 SPEL 구문을 사용해 접근가능하며 아래와 같은 형태가 된다

applicationContext-datasource.xml_bak 파일을 참고

```
<bean id="databaseInfo" class="sample.DataSourceInfo">
    <property name="databaseType" value="#{HoneProperty['database.type']}" />
</bean>
```

로직내에서는 설정에 등록한 프로퍼티 Bean 을 java.util.Properties 객체타입으로 DI 해서 참조 가능하다.

아래는 HoneProperty 로 등록한 프로퍼티 Bean 을 로직내에서 DI 해서 사용하는 예제다.

```
@Inject @Qualifier("HoneProperty")
private Properties honeProperty;

public void test() {
    String testValue = honeProperty.getProperty("database.type");
    //생략
}
```

3.1.4.2 Exception 처리

일반적으로 개발 과정에서 예외는 크게 예측 가능한 예외와 예측 불가능한 예외 두 가지 있다.

이중 예측 가능한 예외는 비즈니스 로직과 마찬가지로 로직 내에서 명확하게 처리해야 하며 그 외 예측 불가능한 예외들은 로깅과 별도의 예외 처리 장치 등을 통해 예외를 파악하고 개선이 가능하도록 해야 한다.

예측 가능한 예외의 경우	예측 불가능한 예외의 경우
로그인 실패	버그
대상 파일이 없는 경우	시스템 환경
DB 에 레코드가 없는 경우	

(a) 기본 예외 처리 지침

(1) 절대 예외를 삼키지 말 것

예외는 어떤 형태로든 명확하게 처리가 되어야 한다. 단순히 콘솔에 찍는 것은 예외를 처리했다고 볼 수 없다.

```
// DO Not This!!!
try{
    ...
}catch(SampleException e) {
    //TODO
    e.printStackTrace();
}
```

(2) 예외 정보를 누락하지 말 것

발생한 예외정보를 임의로 누락시키거나 메시지를 변경하는 작업들은 추후 예외 추적의 흔적을 없애는 행위다.

```
// DO Not This!!!
try{
    ...
}catch(SampleException e) {
    throw NewException("error");
}
```

(3) 충분한 예외 정보를 남길 것

예외 처리는 예외 발생시 개선을 전제로 한다.

따라서 예외 발생시에는 항상 개선에 필요한 충분한 정보를 남기도록 해야 한다.

```
// DO Not This!!!
try{
    ...
}catch(SampleException e) {
    logger.error("error");
}
```

(4) 추상적인 예외를 던지지 말 것

예외 클래스는 클래스 이름만으로도 정보를 제공합니다.

Error, Exception, Throwable, RuntimeException과 같은 추상적인 클래스로 예외를 던지는 경우 이러한 최소한의 정보제공도 포기하는 것입니다.

```
// DO Not This!!!
try{
    ...
}catch(Exception e) {
    throw Exception("error", e);
}
```

(b) HONE Smart Platform 예외 클래스

Hone Smart Platform에서는 HoneMobileException이라는 RuntimeException 하위의 예외 클래스를 기본적으로 제공하고 있다.

따라서 Hone Smart Platform에서 발생하는 모든 예외는 HoneMobileException 또는 그 하위의 Exception들로 정의되어 있으며 어플리케이션을 개발하는 개발자들 역시 HoneMobileException 또는 그 하위의 Exception을 정의해서 사용 하면 된다.

- HoneMobileException의 생성하는 방법은 다음과 같다.

```
HoneMobileException()
HoneMobileException(String message)
HoneMobileException(String statusCode, String message)
HoneMobileException(String statusCode, String message, Throwable t)
HoneMobileException(String message, Throwable t)
HoneMobileException(String Throwable t)
```

- 사용 예

```
public AttachFile getAttachFile( AttachFile attachFile ) {
    // 생략
    try {
        queryForObject(sql, new BeanPropertySqlParameterSource(attachFile),
            AttachFile.class);
    } catch (EmptyResultDataAccessException e) {
        throw new HoneMobileException("데이터가 없습니다", e);
    }
    return attachFile;
}
```

다국어 처리나 메시지 관리 등을 위해 메시지를 코드 형태로 사용하는 경우 예외 인자로 완성된 메시지를 전달해주어야 한다.

또한 메시지 완성을 위해 동적 파라미터 등을 필요한 경우에도 MessageFormat클래스등을 이용해 완전하게 완성된 메시지를 전달해줄도록 한다.

- 사용 예

```
public AttachFile getAttachFile( AttachFile attachFile ) {
    // 생략
    try {
        queryForObject(sql, new BeanPropertySqlParameterSource(attachFile),
            AttachFile.class);
    } catch (EmptyResultDataAccessException e) {
        // 완성 예리 메시지 => 메소드:{0}, SQL:{1}, 수행 결과 데이터가 없습니다."
        String[] eParams = {"getAttachFile",sql};
        String eMsg =
            MessageFormat.format(messageService.getMessage("E001"), eParams);
        throw new HoneMobileException(eMsg, e);
    }
    return attachFile;
}
```

3.1.5 트랜잭션 관리

3.1.5.1 기본설명

- 트랜잭션은 작업의 한 단위이다. 특정 작업을 진행할 때 하나의 작업을 시작하여 완료되는 단위를 말한다.
- 트랜잭션의 성질(ACID)
 - 원자성(atomicity): 트랜잭션은 전부, 전무의 실행만이 있지 일부 실행으로 트랜잭션의 기능을 가질 수는 없다.
 - 일관성(consistency): 트랜잭션이 그 실행을 성공적으로 완료하면 언제나 일관된 데이터베이스 상태로 된다. 즉, 이 트랜잭션의 실행으로 일관성이 깨지지 않는다.
 - 격리성(isolation): 연산의 중간결과에 다른 트랜잭션이나 작업이 접근할 수 없다.
 - 영속성(durability): 트랜잭션이 일단 그 실행을 성공적으로 끝내면 그 결과를 어떠한 경우에도 보장받는다.

- HONE Smart Platform 은 Spring 의 트랜잭션 기능을 그대로 지원한다. Spring 은 트랜잭션 관리에 대하여 일관성 있는 추상화된 방법을 제공하며 다음과 같은 장점을 갖는다.
 - JTA, JDBC와 같은 서로 다른 트랜잭션 API에 대해 일관성 있는 프로그래밍 모델을 제공한다.
 - 프로그램적인 트랜잭션 관리에 대한 사용하기 쉬운 API를 제공한다.
 - 선언적인 트랜잭션 관리를 지원한다.
 - Spring JDBC 와 통합이 용이하다.

3.1.5.2 전파방식

전파방식은 트랜잭션이 언제 생성되는지, 언제 현재의 트랜잭션에 결부되어야 하는지를 정의한다

전파방식	상세설명
PROPAGATION_REQUIRED	이미 하나의 트랜잭션이 존재한다면 그 트랜잭션을 지원하고, 트랜잭션이 없다면, 새로운 트랜잭션을 시작한다
PROPAGATION_REQUIRES_NEW	언제나 새로운 트랜잭션을 시작한다. 이미 활성화된 트랜잭션이 있다면, 일시 정지한다.

3.1.5.3 격리레벨

격리레벨은 어떤 트랜잭션이 동시 진행하는 다른 트랜잭션이 영향을 미치는 정도를 결정한다

격리레벨 (Isolation Level)	상세설명
ISOLATION_DEFAULT	개별적인 PlatformTransactionManager 를 위한 디폴트 격리레벨
ISOLATION_READ_UNCOMMITTED	격리 레벨중 가장 낮음. 이것은 다른 커밋 되지 않은 트랜잭션에 의해 변경된 데이터를 볼 수 있기 때문에 거의 트랜잭션의 기능을 수행하지 않는다
ISOLATION_READ_COMMITTED	대개의 데이터베이스에서의 디폴트 레벨. 이것은 다른 트랜잭션에 의해 커밋 되지 않은 데이터는 다른 트랜잭션에서 볼 수 없도록 한다. 다른 트랜잭션에 의해 입력되거나 수정된 데이터를 조회할 수는 있다.
ISOLATION_REPEATABLE_READ	ISOLATION_READ_COMMITTED 보다는 다소 조금 더 엄격함. 당신은 최소한 같은 데이터 세트를 다시 조회할 수 있다. 이것은 만약 다른 트랜잭션이 새로운 데이터를 입력했다면, 당신은 새롭게 입력된 데이터를 조회할 수 있다는 것을 의미한다.
ISOLATION_SERIALIZABLE	가장 비싸고 신뢰할만한 격리레벨. 모든 트랜잭션은 그것들이 하나가 완전히 수행된 후에 다른 것이 수행된다는 것처럼 처리된다

3.1.5.4 기본 트랜잭션 정책

HONE Smart Platform 에서 권고하는 트랜잭션 처리 방식은 기본적으로 Annotation 방식의 트랜잭션을 사용하는 방식이다

(a) 기본 설정

HONE Smart Platform 에서는 기본적으로 Annotation 방식의 트랜잭션 설정을 사용하게 된다.

트랜잭션에 관련된 이러한 기본 설정은 /WEB-INF/classes/META-INF/config/context/applicationContext-transaction.xml 파일을 생성하여 아래와 같이 정의 할 수 있다.

```
<bean id="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource" />
</bean>

<tx:annotation-driven transaction-manager="transactionManager" />
```

(b) 애노테이션 기반 트랜잭션 관리

애노테이션(@Transactional)을 이용해 개발자가 직접 트랜잭션이 필요한 메소드를 선택 적용하는 방식이 된다.

Transaction 적용

트랜잭션이 필요한 메소드는 메소드 선언위에 @Transactional 만 기술해주면 해당 메소드 단위로 트랜잭션이 적용된다.

메소드가 예러없이 정상 처리되는 경우에는 Commit 이 수행되고 에러가 발생한 경우 Rollback 이 자동 수행된다.

이때 적용되는 트랜잭션의 기본 정책은 이미 트랜잭션을 가지고 호출된 경우 해당 트랜잭션을 사용하게 되고 트랜잭션이 생성되지 않은 경우 새로 생성해 진입하게 된다.

```
@Transactional
public void modifyArticle(Article article){
    this.articleDao.updateArticle(article);
    this.articleDao.updateAttachFile(article.getAttachFiles());
}
```

(1) 별도 트랜잭션 적용 시

비지니스 로직 구현 중에 특정 서비스의 경우 별도의 트랜잭션 정책을 적용해야 하는 경우가 있다. 예를 들어 채번이라든가 정보보호 로깅등의 경우 메인 비지니스 로직이 실패해도 해당 단위 로직은 커밋이 되어야 하는 경우 등이다.

이 경우 간단히 Annotation 방식의 트랜잭션 설정을 해당 메소드 단위에 지정함으로써 간단히 해결 가능하다.

기본적으로 Spring내의 트랜잭션정책은 Propagation.REQUIRED이며 이 경우 해당 메소드들은 트랜잭션이 없으면 생성해 사용하지만 타 서비스 등으로부터 트랜잭션이 전달된 경우 해당 트랜잭션을 가지고 진행된다.

그에 비해 Propagation.REQUIRES_NEW 의 경우 무조건 해당 서비스가 새로운 트랜잭션을 생성해 사용하게 되므로 별도의 서브 트랜잭션 역할을 하게 된다.

- 사용 예

```
@Transactional(propagation=Propagation.REQUIRES_NEW)
public BigDecimal getNextValue(String key) {
    return this.genoDao.getNextGeno(key);
}
```

(2) 트랜잭션 제외 적용 시

트랜잭션 적용 범위긴 하지만 특정 메소드의 경우 트랜잭션 적용 자체를 아예 빼야 하는 경우가 있을 수 있다.

이 경우에는 `Propagation.NOT_SUPPORTED` 를 적용하면 해당 메소드는 트랜잭션 범위에서 제외된다.

즉 해당 서비스에서 에러가 발생해도 롤백 마킹이 되거나 하지 않는다.

아래는 이와 같은 트랜잭션 적용을 제외한 예이다.

```
@Transactional(propagation=Propagation.NOT_SUPPORTED)
public void dummyLog(String text) {
    this.LogService.insertLog();
}
```

3.1.6 로깅 처리

3.1.6.1 org.slf4j.Logger

로그 서비스는 시스템 개발, 운영 시 관점에 따라 시스템 상황을 쉽게 파악할 수 있도록 지원하는 기능이다.

프로그램 코드의 변경 없이 개발이나 운영 환경에 따라 로그 기록 수준 및 대상을 설정을 할 수 있어야 하며, 로깅으로 인한 시스템 부하는 최소화되어야 한다.

HONE Smart Platform에서는 log4j의 최신 구현체인 logback를 기본 로깅 라이브러리로 사용하며, 일부 기능을 확장 하였다.

(a) Logger

Logger는 상속 구조로 되어 있으며 최상위에는 root 로거가 존재한다. 그 이후의 로거 이름은 개발자들이 java 패키지 구조와 유사하게 정의하여 사용할 수 있다. 일반적으로는 java 클래스의 FQN(Full Qualified Name)을 그대로 사용하기도 한다.

로거별로 Level을 지정할 수 있으며, 레벨이 명시적으로 지정되지 않은 로거는 가장 가까운 조상에 지정된 레벨을 따른다.

(1) 로거 사용 코드

Hone에서 로거 사용 방법은 다음과 같다.

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class PersonService {
    private static final Logger logger =
        LoggerFactory.getLogger(PersonService.class);
```



```
public Person loadPersonInfo(String personId) {
    logger.debug("personId : %s", personId);
}
}
```

(2) 코드 설명

로깅을 사용하기 위해서는 slf4j에서 제공하는 다음 두 클래스가 import된다.

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
```

Logger 인스턴스는 LoggerFactory.getLogger(로거명)을 통해서 얻어 온다.

로거는 클래스 내에서 여러 곳에서 반복적으로 사용되므로 "private static final"로 선언하기를 추천한다.

```
private static final Logger logger =
    LoggerFactory.getLogger(PersonService.class);
```

로그 기록은 Log클래스의 로그 레벨에 따라 trace(), debug(), info(), warn(), error()메소드를 이용한다.

각 레벨에 따라 다음과 같은 메소드를 제공 (아래는 debug를 예로 설명함)

메소드	설명	사용 예
isDebugEnabled()	로거에 디버그 레벨 기록 가능 여부	if (logger.isDebugEnabled()) { logger.debug(""); }
debug(String message)	디버그 레벨로 message 를 기록함	logger.debug("메시지");
debug(String format, Object... args)	디버그 레벨로 args 항목을 format(String.format())이 적용된 message 로 변환하여 기록함	logger.debug("메시지");
debug(String message, Throwable t)	디버그 레벨로 message 와 Throwable 의 stack trace 를 기록함	catch (Exception ex) { logger.debug("에러", ex); }

(3) Level

5개의 로그 레벨 (trace, debug, info, warn, error) 을 제공 한다.

각 로그 레벨의 우선 순위는 다음과 같다.

error > warn > info > debug > trace

설정된 로거의 로그 레벨보다 우선 순위가 낮은 경우 로그 출력이 되지 않는다.

예를 들어 'SampleLogger'라는 로거의 로그 레벨이 warn로 설정되어 있을 경우 로그 출력여부는 다음과 같다.

```
private static final Logger logger =
    LoggerFactory.getLogger(SampleLogger.class);
logger.trace("logging test"); // 출력 안됨
```

```
logger.debug("logging test"); // 출력 안됨
logger.info("logging test"); // 출력 안됨
logger.warn("logging test"); // 출력
logger.error("logging test"); // 출력
```

warn 로그 레벨의 우선 순위가 debug나 info보다 높기 때문에 debug(), info() 메소드를 이용한 로깅은 출력이 되지 않는다.

(b) 환경설정

HONE Smart Platform 에서 /WEB-INF/classes/logback.xml 파일에 아래와 같이 정의되어 있다.

(1) logback.xml

logback.xml에는 애플리케이션 내부에서 발생하는 로깅에 대한 전체적인 설정을 기술하며 주로 logger와 appender등의 설정을 하게 된다

- 1) logger : 로그의 주체(로그 파일을 작성하는 클래스)로 로깅 메시지를 Appender에 전달
- 2) appender : 전달된 로깅 메시지의 출력 대상(console, file, db...)과 형식등을 지정

```
<configuration>
    <appender name="file"
class="ch.qos.logback.core.rolling.RollingFileAppender">
        <file>../logs/honebatchadmin.log</file>
        <rollingPolicy
class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy"><!-- daily
rollover -->
            <fileNamePattern>../logs/honeadmin.%d{yyyy-MM-
dd}.log</fileNamePattern>
            <!-- keep 30 days' worth of history -->
            <maxHistory>30</maxHistory>
        </rollingPolicy>
        <encoder>
            <pattern>%-4relative [%thread] %-5level %logger{35} -
%msg%n</pattern>
        </encoder>
    </appender>

    <appender name="CONSOLE"
class="ch.qos.logback.core.ConsoleAppender">
        <!-- encoders are assigned the type
ch.qos.logback.classic.encoder.PatternLayoutEncoder by default -->
        <encoder>
            <pattern>%d{HH:mm:ss.SSS} [%thread] %-
5level %logger{36} - %msg%n</pattern>
        </encoder>
    </appender>

    <logger name="org.springframework" level="info" additivity="false">
        <appender-ref ref="CONSOLE" />
    </logger>

    // 생략

    <logger name="org.springframework.jdbc" level="trace"
additivity="false">
```

```

        <appender-ref ref="CONSOLE" />
    </logger>
</configuration>

```

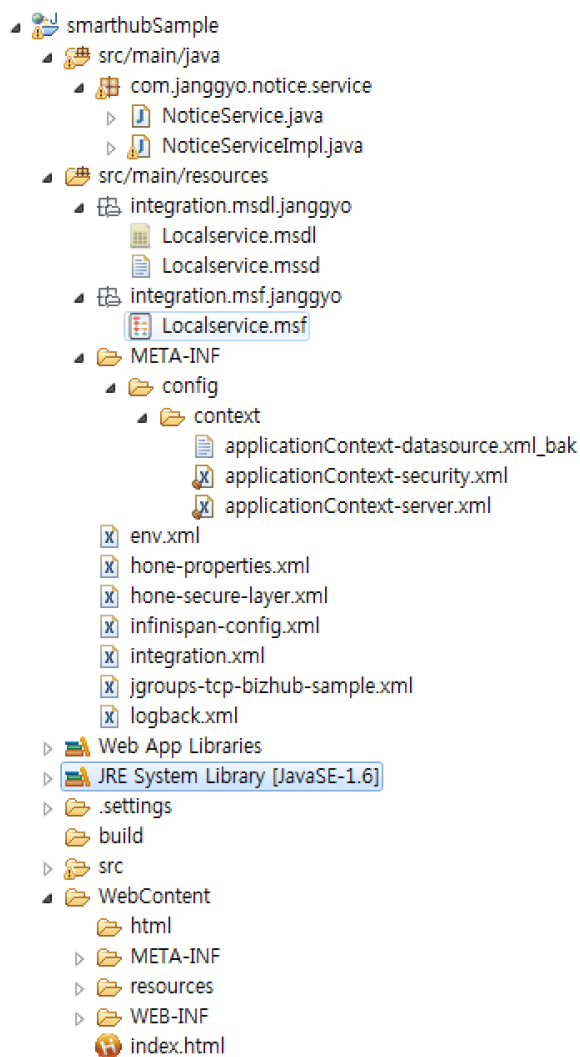
3.2 Sample 소스

Sample 로 제공되는 소스이며 프로젝트 특성에 맞게 변경을 해야 한다.

프로젝트 구성 시 Sample 소스를 복사하거나 이름을 변경하여 프로젝트를 진행해야 한다.

3.2.1 구성

Sample 소스 구성도는 다음과 같다.



3.2.2 env.xml

서버가 구동되면서 가장 먼저 읽어오는 값으로 다른 설정 파일에서 참조하거나 애플리케이션 로직 내에서 접근이 가능하다

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
    <entry key="spring.profiles.active">jdbc,infinispancache.off</entry>
</properties>
```

3.2.3 hone-properties.xml

프로퍼티 값들을 추가 작성할 수 있으며 이곳에 작성된 프로퍼티들은 HONE Smart Platform의 다른 설정 파일에서 참조하거나 애플리케이션 로직내에서 접근이 가능하다

Example

```
<comment>
<![CDATA[
HONEMOBILE 설정파일 및 애플리케이션내에서 사용될 수 있는 여러 설정값들을 관리하는 프로퍼티
파일
- 설정파일내에서는 #{HoneProperty['프로퍼티Key']} 형태로 접근할 수 있으며
- 애플리케이션내에서는 @Value("#{HoneProperty['isSaferMode']}") 형태로 접근가능하다.
]]>
</comment>

<entry key="hubId">janggyo</entry>
<!-- TODO check the property if it is used other places -->
<entry key="bizAppResourcePath">resources</entry>

<!-- Persistence Configuration -->

<entry key="database.driverClassName">oracle.jdbc.OracleDriver</entry>
<entry key="database.url">jdbc:oracle:thin:@10.211.55.xxx:1521:xe</entry>
<entry key="database.username">xxxx</entry>
<entry key="database.password">xxxx</entry>

<entry key="jpa.generate.ddl">>true</entry>
<entry key="jpa.database">ORACLE</entry>
<entry
key="jpa.database.platform">org.hibernate.dialect.Oracle10gDialect</entry>

<entry key="hibernate.hbm2ddl.auto">update</entry>
<entry key="hibernate.show_sql">>false</entry>
<entry key="hibernate.format_sql">>true</entry>
```

3.2.4 applicationContext-server.xml

applicationContext 값을 설정한다. base-package scan 을 세팅하고 Camel 을 사용 하기 때문에 Controller 는 제외 한다.

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:util="http://www.springframework.org/schema/util"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:beans="http://www.springframework.org/schema/beans"
  xmlns:mvc="http://www.springframework.org/schema/mvc"
  xmlns:context="http://www.springframework.org/schema/context"
  xsi:schemaLocation="
    http://www.springframework.org/schema/mvc
    http://www.springframework.org/schema/mvc/spring-mvc-4.1.xsd
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-4.1.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-4.1.xsd
    http://www.springframework.org/schema/util
    http://www.springframework.org/schema/util/spring-util-4.1.xsd">

  <context:component-scan base-package="honemobile.com.janggyo">
    <context:exclude-filter type="annotation"
      expression="org.springframework.stereotype.Controller"/>
  </context:component-scan>

</beans>
```

3.2.5 applicationContext-security.xml

Spring Security 에 대한 세팅을 한다. <https://projects.spring.io/spring-security/> 을 참고한다.

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<beans:beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:beans="http://www.springframework.org/schema/beans"
  xmlns:security="http://www.springframework.org/schema/security"
  xsi:schemaLocation="
    http://www.springframework.org/schema/security/spring-security-4.0.xsd
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-4.1.xsd">

  <security:debug/>
  <!-- Hub infinispan 사용유무에 따른 session세팅 or cach 세팅 방법 -->
```

```

        <beans:beans profile="infinispancache.on" >
            <security:http pattern="/janggyo/**" auto-config="false"
                use-expressions="true" entry-point-
ref="mobileAuthenticationEntryPoint"
                authentication-manager-ref="janggyoAuthenticationManager"
                security-context-repository-
ref="cacheSecurityContextRepository"
                access-decision-manager-ref="mobileAccessDecisionManager">
                <security:custom-filter
position="FORM_LOGIN_FILTER" ref="janggyoAuthenticationFilter" />
                <security:custom-filter position="LOGOUT_FILTER"
ref="janggyoLogoutFilter" />
                <security:access-denied-handler
ref="mobileAccessDeniedHandler" />
                <security:request-cache ref="nullRequestCache" />

                <security:intercept-url pattern="/janggyo/**"
access="isFullyAuthenticated() and hasRole('ROLE_JANGGYO')"/>
                <!-- security 4.0 security:csrf
사용안함으로 처리 -->

                <security:csrf disabled="true"/>
            </security:http>
        </beans:beans>

        <beans:beans profile="infinispancache.off" >
            <security:http pattern="/janggyo/**" auto-config="false"
                use-expressions="true" entry-point-
ref="mobileAuthenticationEntryPoint"
                authentication-manager-ref="janggyoAuthenticationManager"
                access-decision-manager-ref="mobileAccessDecisionManager">

                <security:custom-filter position="FORM_LOGIN_FILTER"
ref="janggyoAuthenticationFilter" />
                <security:custom-filter position="LOGOUT_FILTER"
ref="janggyoLogoutFilter" />
                <security:access-denied-handler
ref="mobileAccessDeniedHandler" />
                <security:request-cache ref="nullRequestCache" />

                <security:intercept-url pattern="/janggyo/**"
access="isFullyAuthenticated() and hasRole('ROLE_JANGGYO')"/>
                <!-- security 4.0 security:csrf
사용안함으로 처리 -->

                <security:csrf disabled="true"/>
            </security:http>

        </beans:beans>

<beans:beans>
    <security:authentication-manager id="janggyoAuthenticationManager">
        <security:authentication-provider user-service-
ref="janggyoUserDetailsService"></security:authentication-provider>
    </security:authentication-manager>
    <security:user-service id="janggyoUserDetailsService">
        <security:user name="jim" password="jim" authorities="ROLE_JANGGYO,

```

```

ROLE_USER, ROLE_ADMIN"/>
    <security:user name="bob" password="bob" authorities="ROLE_JANGGYO,
ROLE_USER"/>
  </security:user-service>

  <bean id="janggyoLogoutFilter"
class="org.springframework.security.web.authentication.logout.LogoutFilter">
    <!-- 첫번째 인자: LogoutSuccessHandler -->
    <constructor-arg index="0" ref="janggyoLogoutSuccessHandler"/>
    <constructor-arg index="1">
      <list>
        <bean
class="honemobile.security.handler.DefaultLogoutHandler"/>
      </list>
    </constructor-arg>
    <property name="filterProcessesUrl"
value="/janggyo/service/default/logout"/>
  </bean>

  <!-- Custom 인증처리 필터 -->
  <bean id="janggyoAuthenticationFilter"
class="honemobile.security.HoneMobileAuthenticationFilter">
    <property name="authenticationManager"
ref="janggyoAuthenticationManager" />
    <property name="authenticationDetailsSource"
ref="mobileAuthenticationDetailsSource" />
    <property name="authenticationSuccessHandler"
ref="janggyoSuccessHandler" />
    <property name="authenticationFailureHandler"
ref="janggyoFailureHandler" />
    <property name="usernameParameter" value="username"/>
    <property name="passwordParameter" value="password"/>
    <!-- 아래 url이면 이 필터가 작동(인증처리)한다. -->
    <property name="filterProcessesUrl"
value="/janggyo/service/default/login" />
  </bean>

  <bean id="janggyoSuccessHandler"
class="honemobile.security.HoneMobileAuthenticationSuccessHandler" >
    <property name="loginSuccessHandler" ref="janggyoLoginSuccessHandler"/>
  </bean>

  <bean id="janggyoFailureHandler"
class="honemobile.security.HoneMobileAuthenticationFailureHandler">
    <property name="loginFailureHandler" ref="janggyoLoginFailureHandler"/>
  </bean>

  <bean id="janggyoLogoutSuccessHandler"
class="honemobile.security.HoneMobileLogoutSuccessHandler">
  </bean>

  <bean id="janggyoLoginSuccessHandler"
class="honemobile.security.handler.DefaultLoginSuccessHandler">
  </bean>

```

```

<bean id="janggyoLoginFailureHandler"
      class="honemobile.security.handler.DefaultLoginFailureHandler">
</bean>

<bean id="janggyoLogoutHandler"
      class="honemobile.security.handler.DefaultLogoutHandler">
</bean>
</beans:beans>

</beans:beans>

```

3.2.6 hone-secure-layer.xml

secureLayerCacheService 설정 한다.

env.xml 에서 infinispance.on 으로 설정할 경우만 사용 가능하다.

entry key	description	example
securelayer.cookie.domain	서버이름 or 서버 ip	hone.hanwha.co.kr
securelayer.cookie.path	쿠키 path	/operations
securelayer.cookie.maxAge	쿠키 max age (초)	600

Example

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
  <comment>
    <![CDATA[
      HONEMOBILE 설정파일 및 애플리케이션내에서 사용될 수 있는 여러 설정값들을 관리하는 프로퍼티
      파일
      - 설정파일내에서는 #{SecureLayerProperty['프로퍼티Key']} 형태로 접근할 수 있으며
      - 애플리케이션내에서는 @Value("#{SecureLayerProperty['isSaferMode']}") 형태로
      접근가능하다.
    ]]>
  </comment>

  <entry key="securelayer.cookie.domain">192.168.0.134</entry>
  <entry key="securelayer.cookie.path">/operations</entry>
  <entry key="securelayer.cookie.maxAge">600</entry>
  <entry key="securelayer.serverKeySize">2048</entry>
</properties>

```


3.2.7 infinispn-config.xml

infinispn 에서 사용할 cachename, cluster 등 값이 들어가게 된다. 아래 키 값에 해당하는 엔트리에 적절한 값을 설정한다.

env.xml 에서 infinispncache.on 으로 설정할 경우만 사용 가능하다.

entry key	description	example
transport	cluster 세팅	jgroups-tcp-bizhub-sample.xml
securityContextCache	로그인 인증관련	
secureLayerContextCache	구간암호화를 위한 키 저장	
location	File cache 위치	/tmp/honemobile/operations/secureLayerContextCache/

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<infinispn xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:infinispn:config:6.0
http://www.infinispn.org/schemas/infinispn-config-6.0.xsd"
  xmlns="urn:infinispn:config:6.0">

  <global>
    <transport clusterName="SampleBizHubCacheCluster">
      <properties>
        <property name="configurationFile" value="jgroups-tcp-
bizhub-sample.xml" />
      </properties>
    </transport>
    <globalJmxStatistics enabled="true" />
  </global>

  <!-- When the bizhub node is more than 2, distribution mode should be
considered. -->
  <default>
    <clustering mode="replication">
      <sync />
    </clustering>
    <jmxStatistics enabled="true" />
  </default>

  <!-- securityContextCache is a repository of Spring Security's security
context -->
  <namedCache name="securityContextCache">
    <!-- maxEntries means the maximum concurrent user number -->
    <eviction strategy="LIRS" maxEntries="10000" />
    <!-- the max idle time is 30 minutes -->
    <expiration maxIdle="1800000" />
    <!-- configuration for the file system store -->
    <persistence passivation="true">
      <store
class="org.infinispn.persistence.file.SingleFileStore"
```

```

shared="false"
    fetchPersistentState="false" preload="false"
    purgeOnStartup="true" ignoreModifications="false">
    <async enabled="true" flushLockTimeout="1"
        shutdownTimeout="25000"
modificationQueueSize="1024" threadPoolSize="1" />
    <properties>
        <property name="location"
value="/tmp/honemobile/bizhub/securityContextCache" />
    </properties>
    </store>
    </persistence>
</namedCache>
</infinispan>

```

3.2.8 jgroups-tcp-bizhub-sample.xml

cluster 관련 세팅이 들어가게 된다. 아래 키 값에 해당하는 엔트리에 적절한 값을 설정한다.

env.xml 에서 infinispancache.on 으로 설정할 경우만 사용 가능하다.

entry key	description	example
bind_addr	자신의 IP 세팅	10.10.10.130
bind_port	자신의 Infinispan Port	7801
thread_pool.max_threads	Thread pool 최대 사용치 초당건수에 따라 높여주어야 한다.	10

Example

```

<!--
Fast configuration for local mode, i.e. all members reside on the same host.
Setting ip_ttl to 0 means that
no multicast packet will make it outside the local host.
Therefore, this configuration will NOT work to cluster members residing on
different hosts !

Author: Bela Ban
Version: $Id: fast-local.xml,v 1.9 2009/12/18 14:50:00 belaban Exp $
-->

<config xmlns="urn:org:jgroups"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="urn:org:jgroups http://www.jgroups.org/schema/JGroups-
3.4.xsd">
    <TCP bind_port="7850" port_range="10"
        recv_buf_size="20000000"
        send_buf_size="640000"
        loopback="false"
        max_bundle_size="64k"
        bundler_type="old"

```

```
enable_diagnostics="true"
thread_naming_pattern="cl"

timer_type="new"
timer.min_threads="4"
timer.max_threads="10"
timer.keep_alive_time="3000"
timer.queue_max_size="1000"
timer.wheel_size="200"
timer.tick_time="50"

thread_pool.enabled="true"
thread_pool.min_threads="2"
thread_pool.max_threads="8"
thread_pool.keep_alive_time="5000"
thread_pool.queue_enabled="true"
thread_pool.queue_max_size="100000"
thread_pool.rejection_policy="discard"

oob_thread_pool.enabled="true"
oob_thread_pool.min_threads="1"
oob_thread_pool.max_threads="8"
oob_thread_pool.keep_alive_time="5000"
oob_thread_pool.queue_enabled="false"
oob_thread_pool.queue_max_size="100"
oob_thread_pool.rejection_policy="discard"/>

<!-- JGroups' default jgroups.mping.mcast_addr is 228.2.4.6 -->
<!-- jgroups.mping.mcast_addr for OperationsMobileCacheCluster: 228.2.4.7 -->
<!-- jgroups.mping.mcast_addr for OperationsAdminCacheCluster: 228.2.4.8 -->
<!-- jgroups.mping.mcast_addr for BizHubCacheCluster: 228.2.4.9... -->
<!--
  <MPING bind_addr="{jgroups.bind_addr:127.0.0.1}" break_on_coord_rsp="true"
    mcast_addr="{jgroups.mping.mcast_addr:228.2.4.9}"
    mcast_port="{jgroups.mping.mcast_port:43366}"
    ip_ttl="{jgroups.udp.ip_ttl:2}"
    num_initial_members="3" timeout="2000"/>
-->

  <TCPPING timeout="2000"

initial_hosts="{jgroups.tcpping.initial_hosts:192.168.56.xxx[7850],192.168.56.xxx
[7850]}"
    port_range="1"
    num_initial_members="2" />

  <MERGE2 max_interval="30000"
    min_interval="10000"/>

  <FD_SOCKET/>
  <FD_ALL interval="2000" timeout="5000" />
  <VERIFY_SUSPECT timeout="500" />
  <BARRIER />
  <pbcast.NAKACK use_mcast_xmit="false"
    retransmit_timeout="100,300,600,1200"
    discard_delivered_msgs="true" />
  <UNICAST3 conn_expiry_timeout="0"/>
```

```

<pbcast.STABLE stability_delay="1000" desired_avg_gossip="50000"
    max_bytes="10m"/>
<pbcast.GMS print_local_addr="true" join_timeout="5000"
    max_bundling_time="30"
    view_bundling="true"/>
<UFC max_credits="2M"
    min_threshold="0.4"/>
<MFC max_credits="2M"
    min_threshold="0.4"/>
<FRAG2 frag_size="60000" />
<pbcast.STATE_TRANSFER />
</config>

```

3.2.9 logback 세팅

로그 형태와 logger 레벨 설정을 한다.

Example

```

<?xml version="1.0" encoding="UTF-8"?>
<configuration debug="false" scan="true" scanPeriod="30 seconds">

    <!--
    <appender name="file"
class="ch.qos.logback.core.rolling.RollingFileAppender">
        <file>../logs/honebatchadmin.log</file>
        <rollingPolicy
class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
            <fileNamePattern>../logs/honeadmin.%d{yyyy-MM-dd}.log</fileNamePattern>
            <maxHistory>30</maxHistory>
        </rollingPolicy>
        <encoder>
            <pattern>%-4relative [%thread] %-5level %logger{35} -
%msg%n</pattern>
        </encoder>
    </appender>
    -->
    <appender name="CONSOLE" class="ch.qos.logback.core.ConsoleAppender">
        <!-- encoders are assigned the type
            ch.qos.logback.classic.encoder.PatternLayoutEncoder by default --
    >

        <encoder>
            <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} -
%msg%n</pattern>
        </encoder>
    </appender>

    <logger name="org.apache.camel" level="DEBUG"/>
    <logger name="org.springframework" level="INFO"/>
    <logger name="org.springframework.jdbc" level="TRACE"/>
    <logger name="org.springframework.security" level="INFO"/>
    <logger name="honemobile.core" level="DEBUG"/>
    <logger name="hone.integration" level="DEBUG"/>

```

```
<logger name="honemobile.security" level="DEBUG"/>
<logger name="honemobile.operations" level="DEBUG"/>

<root level="DEBUG">
    <appender-ref ref="CONSOLE" />
</root>
</configuration>
```

3.2.10 Service

예제용으로 제공되는 Notice 내용을 조회하는 서비스이다.

Interface NoticeService.java 파일과 구현체인 NoticeServiceImpl.java 파일로 구성되어 있다.

3.2.10.1 NoticeService

```
package com.janggyo.notice.service;

import java.util.Map;

public interface NoticeService {

    public Map<String, String> getProductList(Map<String, String> args);

    public Map<String, Object> getNoticeList(Map<String, Object> args);
}
```

3.2.10.2 NoticeServiceImpl

```
package com.janggyo.notice.service;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;
@Named("com.janggyo.notice.service.NoticeService")
public class NoticeServiceImpl implements NoticeService {

    public Map<String, String> getProductList(Map<String, String> args) {
        return args;
    }

    public Map<String, Object> getNoticeList(Map<String, Object> bodyMap) {
        String pageNumber = (String) bodyMap.get("pageNumber");
        String rowaPerPage = (String) bodyMap.get("rowaPerPage");

        Map<String, Object> notice1 = new HashMap<String, Object>();
        notice1.put("id", "1");
        notice1.put("title", "공지사항1");
        notice1.put("link", "http://www.daum.net");
    }
}
```

```

        notice1.put("imageUri",
"http://icon.daumcdn.net/w/c/12/11/10192021148946703.png");
        notice1.put("contents", "첫번째 공지사항입니다.");
        notice1.put("postedAt", "2013-03-22EDT12:56:35");

        Map<String, Object> notice2 = new HashMap<String, Object>();
        notice2.put("id", "2");
        notice2.put("title", "공지사항2");
        notice2.put("link", "http://www.naver.com");
        notice2.put("imageUri",
"http://icon.daumcdn.net/w/c/12/11/10192021148946703.png");
        notice2.put("contents", "두 번째 공지사항입니다.");
        notice2.put("postedAt", "2013-03-23EDT12:56:35");

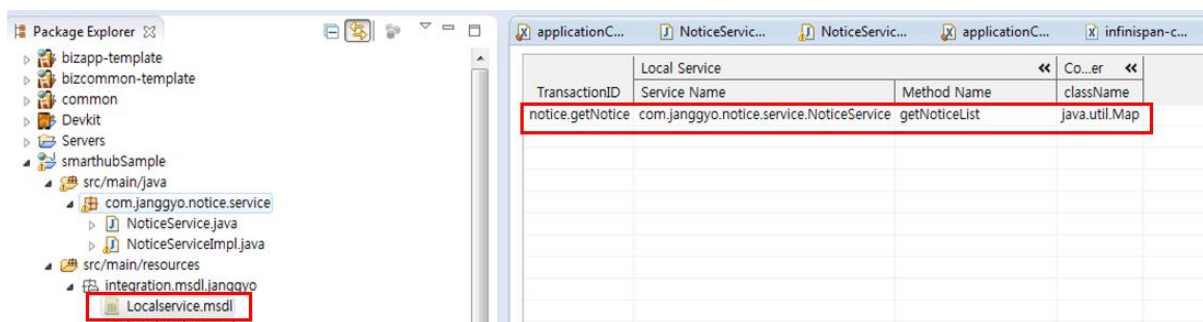
        ArrayList<Map<String, Object>> noticeList = new
ArrayList<Map<String, Object>>();
        noticeList.add(notice1);
        noticeList.add(notice2);

        Map<String, Object> responseMap = new HashMap<String, Object>();
        responseMap.put("totalNumber", "2");
        responseMap.put("currentPage", "1");
        responseMap.put("rows", noticeList);

        return responseMap;
    }
}

```

3.2.10.3 msdl 파일



3.2.10.4 테스트 (Postman 이용함)

(a) 로그인처리

아래의 Request 로 로그인을 확인할 수 있다. Security 를 적용하였기 때문에 모든 서비스를 정상적으로 이용하기 위해서는 로그인이 필요하다.

Example Requests

```
POST /smarthub/janggyo/service/default/login HTTP/1.1
Host: 10.211.55.3:8080
Content-Type: application/json
X-HoneMobile-Header: DeviceId=5089330644789d1de1432837e2655d8d;
LauncherAppVersion=21; LauncherAppId=DEMO; NetworkType=Wi-Fi;
CarrierName=SKTelecom; IsEncrypted=0;
X-HONEMobile-UA: HONEWebClient/2.0.0 (Windows Store App/0.0; Computer.Notebook;
ko; Hone; Hone; x64)
Cache-Control: no-cache
Postman-Token: aa94820c-c170-6782-8782-6f1d36cd6ab2

{"header": {}, "payload": {"username": "jim", "password": "jim"}}
```

Example Response

```
{
  "payload": {
    "authToken":
    "63613864313865622d336265352d343232362d616630612d353536393762623466613432"
  },
  "header": {}
}
```

(b) Notice 조회

아래의 Request 로 Notice 조회가 가능하다. 단 Security 를 적용하였기 때문에 모든 서비스를 정상적으로 이용하기 위해서는 로그인을 진행 후 조회가 가능하다.

Example Requests

```
POST /smarthub/janggyo/service/notice/getNotice HTTP/1.1
Host: 10.211.55.3:8080
Content-Type: application/json
X-HoneMobile-Header: DeviceId=5089330644789d1de1432837e2655d8d;
LauncherAppVersion=21; LauncherAppId=DEMO; NetworkType=Wi-Fi;
CarrierName=SKTelecom; IsEncrypted=0;
X-HONEMobile-UA: HONEWebClient/2.0.0 (Windows Store App/0.0; Computer.Notebook;
ko; Hone; Hone; x64)
Cache-Control: no-cache
Postman-Token: 6afa0703-37f3-3941-d782-ff363e889248
```

```
{"header": {}, "payload": {"pageNumber": "1", "rowsPerPage": "10"}}
```

Example Response

```
{
  "header": {},
  "payload": {
    "totalNumber": "2",
    "currentPage": "1",
    "rows": [
      {
        "imageUri":
"http://icon.daumcdn.net/w/c/12/11/10192021148946703.png",
        "id": "1",
        "title": "공지사항 1",
        "contents": "첫번째 공지사항입니다.",
        "link": "http://www.daum.net",
        "postedAt": "2013-03-22EDT12:56:35"
      },
      {
        "imageUri":
"http://icon.daumcdn.net/w/c/12/11/10192021148946703.png",
        "id": "2",
        "title": "공지사항 2",
        "contents": "두 번째 공지사항입니다.",
        "link": "http://www.naver.com",
        "postedAt": "2013-03-23EDT12:56:35"
      }
    ]
  }
}
```

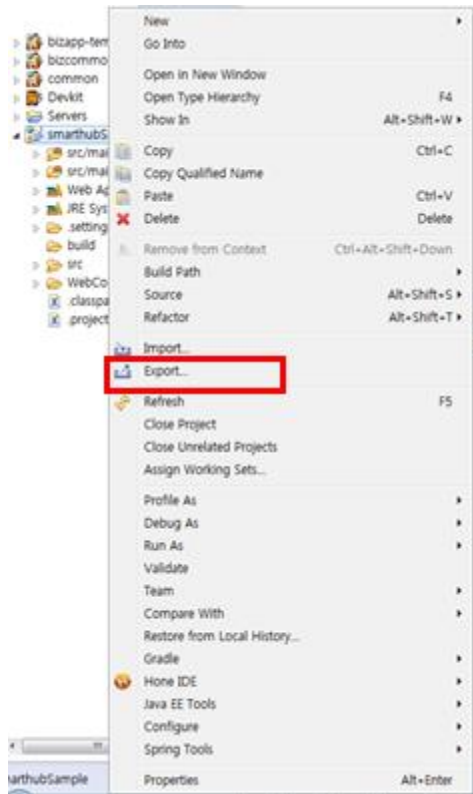
4 서버 배포

로컬 eclipse 상에서 개발한 내용을 개발/운영 서버에 올리고 서버를 구동 한다

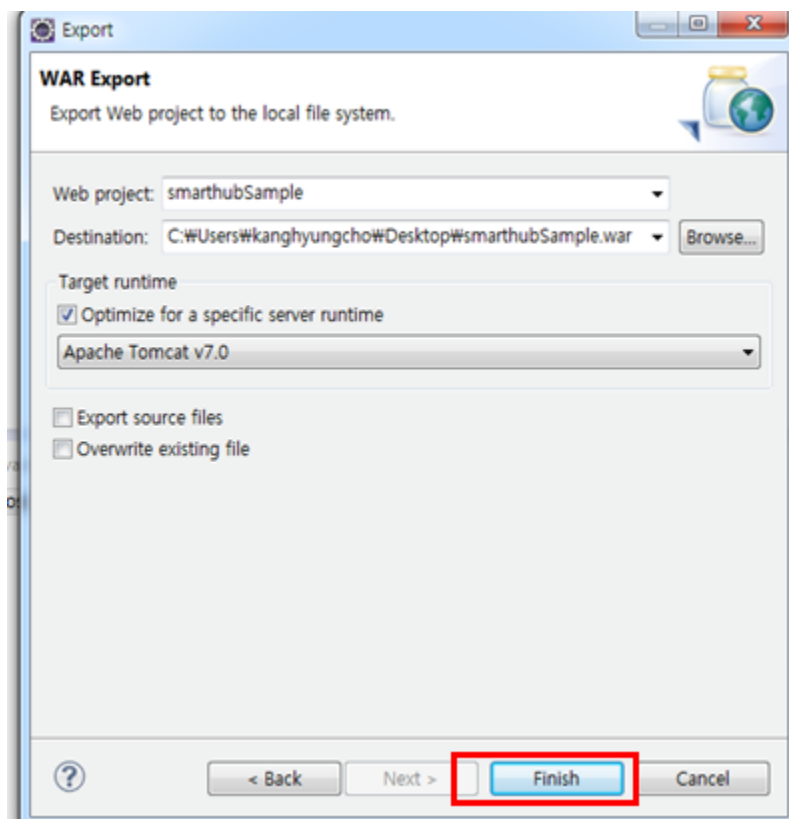
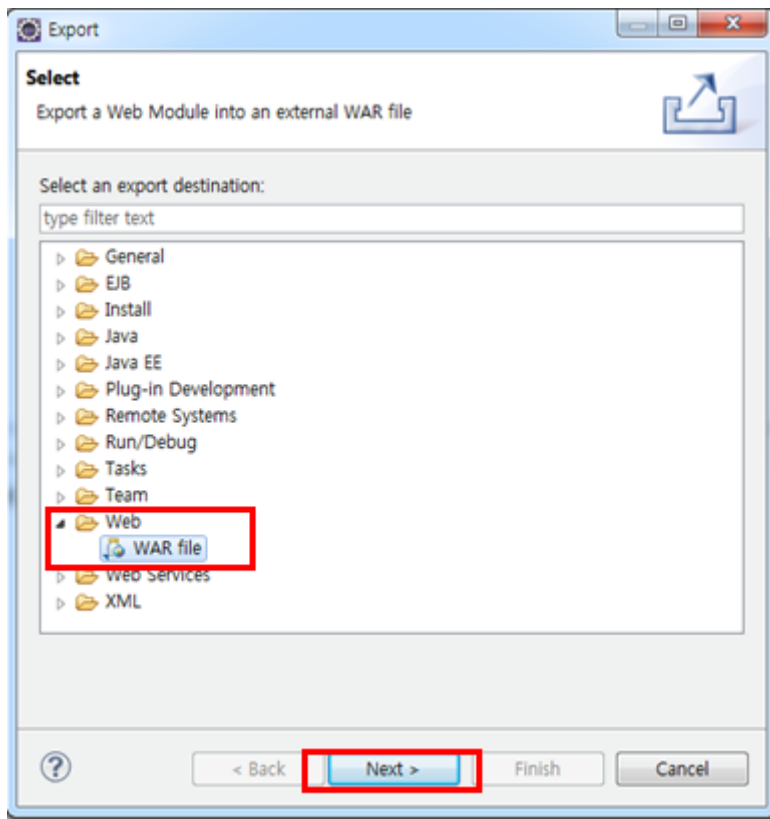
4.1 Eclipse 를 이용한 배포

배포 방법은 여러 가지가 있다. 아래 예제는 이클립스를 이용한 배포 방법이다.

프로젝트 오른쪽 마우스클릭 -> Export 클릭



WAR 파일 클릭



war 파일을 WAS 서버에 배포를 한다.

4.2 기타

Jenkins , nexus 등을 이용한 배포도 가능하다.

5 유의사항

상용 WAS 에 application 을 배포 시 라이브러리 충돌에 인한 서버 구동 실패 시 아래와 같은 방법으로 진행한다.

5.1 Jeus 오류

다음과 같은 오류 발생 시의 원인과 해결 방안을 제시한다.

오류 메시지

```
Caused by: jeus.management.j2ee.DeploymentException: Could not find non-jta-data-source 'jdbc/__default' for the persistence unit transactionMonitor
    at
    jeus.persistence.container.PersistenceUnitInfoImpl.<init>(PersistenceUnitInfoImpl.java:117)
    at
    jeus.persistence.container.PersistenceUnits.load(PersistenceUnits.java:107)
    at
    jeus.management.j2ee.J2EEDeployedObject.loadPersistenceUnits(J2EEDeployedObject.java:807)
        at jeus.management.j2ee.WebModule.initPersistenceUnits(WebModule.java:1161)
        at jeus.servlet.engine.Context.init0(Context.java:641)
        ... 21 more
Caused by: javax.naming.NameNotFoundException: OP_LOOKUP failed: jdbc [Root exception is javax.naming.NameNotFoundException: [jdbc]
    Name not found.]
```

원인: Operations Center 는 JPA v2.0 을 사용하고 스프링을 이용하여 설정하고 dataSource 도 스프링 설정으로 자체적으로 지정. Jeus6 는 JEE5 구현체이므로 JPA v1.0 을 기본적으로 탑재한다. 따라서 Operations 배포의 /META-INF/persistence.xml 을 Jeus 자체의 JPA 가 읽고 데이터 소스를 찾으려고 하고 데이터 소스가 없으니 기본적으로 'jdbc/ __default'라는 data-source 를 찾아발생된 오류이다.

해결방법: persistence.xml 위치를 옮겨서 WAS 가 직접 로드하지 못하게 하고, 클래스로드 순서를 애플리케이션 포함 라이브러리(WEB-INF/lib) 우선으로 설정하여 JPA v2.0 클래스를 적용하게 한다.

상세절차는 다음과 같다..

1. jeus-web-dd.xml 파일 옵션 변경하여 클래스로드 순서 변경
`<webinf-first>true</webinf-first>`
2. 아래 라이브러리를 WEB-INF/lib 로부터 제외시켜 충돌 jar 문제 해결
`stax-api-1.0-2.jar, stax-api-1.0.1.jar, xml-apis-1.0.b2.jar`
3. hibernate-jpa-2.0-api-1.0.1.Final.jar 를 WEB-INF/lib 에 포함시킴
4. persistence.xml 위치 변경하고 스프링이 변경된 위치에서 해당 파일을 읽도록 설정 변경
(honemobile-operations-persistence.jar 수정)

5.2 Weblogic 오류

다음과 같은 오류 발생 시의 원인과 해결 방안을 제시한다.

오류 메시지

```
Caused By: java.lang.NoSuchMethodError:
javax/persistence/spi/PersistenceUnitInfo.getSharedCacheMode()Ljava/persistence/S
haredCacheMode;
    at
org.hibernate.ejb.util.LogHelper.logPersistenceUnitInfo(LogHelper.java:38)
    at
org.hibernate.ejb.Ejb3Configuration.configure(Ejb3Configuration.java:526)
    at
org.hibernate.ejb.HibernatePersistence.createContainerEntityManagerFactory(Hiberna
tePersistence.java:73)
    at
org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean.createNativeEnt
ityManagerFactory(LocalContainerEntityManagerFactoryBean.java:287)
    at
org.springframework.orm.jpa.AbstractEntityManagerFactoryBean.afterPropertiesSet(Ab
stractEntityManagerFactoryBean.java:310)
    Truncated. see log file for complete stacktrace
```

원인: Operations Center 는 JPA v2.0 을 사용하고 스프링을 이용하여 설정하고 dataSource 도 스프링 설정으로 자체적으로 지정하고 있다. Weblogic 11g 도 JEE5 구현체이므로 JPA v1.0 을 기본적으로 탑재하여 Operations 배포의 /META-INF/persistence.xml 을 Weblogic 자체의 JPA 가 읽어 java.lang.NoSuchMethodError 가 발생한 오류이다.

해결방법: persistence.xml 위치를 옮겨서 WAS 가 직접 로드하지 못하게 하고, 클래스로드 순서를 애플리케이션 포함 라이브러리(WEB-INF/lib) 우선으로 설정하여 JPA v2.0 클래스를 적용하게 함.

관련 내용은 다음과 같다.

weblogic.xml 파일에서 prefer-web-inf-classes 을 true 로 조정하여도 별다른 효과가 없음. prefer-application-packages 를 추가함. 이 설정을 통해 weblogic 의 라이브러리가 아닌 어플리케이션의 라이브러리를 사용하게 된다. JPA 뿐만이 아니고 Spring 도 포함시켜야 weblogic 에 포함된 예전 버전의 스프링을 사용하지 않게 된다.

Example

```
<prefer-application-packages>
  <package-name>antlr.*</package-name>
  <package-name>org.apache.commons.*</package-name>
  <package-name>org.apache.xmlbeans.*</package-name>
  <package-name>org.springframework.*</package-name>
  <package-name>org.hibernate.*</package-name>
  <package-name>javax.persistence.*</package-name>
  <package-name>org.joda.*</package-name>
</prefer-application-packages>
```

아래 라이브러리를 WEB-INF/lib 로부터 제외시켜 충돌 jar 문제를 해결한다.

stax-api-1.0-2.jar, stax-api-1.0.1.jar, xml-apis-1.0.b2.jar

hibernate-jpa-2.0-api-1.0.0.Final.jar 파일을 제거하고 대신 javax.persistence_2.1.0.vxxx.jar 파일을 배포하도록 한다. 이 jar 파일은 EclipseLink(Eclipse Persistence Services Project, <http://www.eclipse.org/eclipselink/>) 사이트에서 다운로드 받아서 사용하도록 한다.

EclipseLink 2.x 버전 Installer Zip 파일을 다운로드 받은 후 압축을 풀면 JAR 파일을 확인할 수 있다.

다운로드 : javax.persistence_2.1.0.v201304241213.jar

persistence.xml 위치 변경하고 스프링이 변경된 위치에서 해당 파일을 읽도록 설정을 변경한다. (honemobile-operations-persistence.jar 수정)

6 MyBatis 세팅

DB 관련 툴은 오픈소스를 사용하여 개발 가능 하다. 다음은 가장 많이 사용 하고 있는 mybatis 세팅 방법이며, 다른 툴을 사용하여도 무방하다.

6.1 MyBatis 버전 (권장)

해당 jar 는 maven repository 사이트나, Mybatis 사이트에서 다운 받는다.

mybatis-3.2.2.jar,

mybatis-spring-1.2.0.jar

6.2 상세 설정 예제

아래 내용을 참고 하여 프로젝트에 맞게 수정이 필요하다.

6.2.1 lib 추가 (/WEB-INF/lib)

다음의 jar 파일을 /WEB-INF/lib 에 추가한다.

- mybatis-3.2.2.jar,
- mybatis-spring-1.2.0.jar

6.2.2 applicationContext-persistence.xml 수정 예

해당 xml 파일에서 다음의 정보를 설정한다.

- org.mybatis.spring.SqlSessionFactoryBean : 스프링과 MYBATIS를 연동모듈
- dataSource : 데이터소스
- mapperLocations : mapper 관련된 폴더 ,
- configLocation : 설정파일경로
- org.mybatis.spring.mapper.MapperScannerConfigurer : 컴포넌트 스캔을 통하여 Mapper을 찾기 위한 설정.
- basePackage : Mapper을 찾는 베이스 패키지

Example

```
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="dataSource" />
    <property name="mapperLocations" value="classpath:mappers/*Mapper.xml" />
    <property name="configLocation" value="classpath:META-INF/config/mybatis/mybatis-config.xml" />
</bean>

<bean id="sqlSessionTemplate" class="org.mybatis.spring.SqlSessionTemplate">
    <constructor-arg ref="sqlSessionFactory" />
</bean>

<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
    <property name="basePackage" value="com.janggyo" />
</bean>
```

6.2.3 mybatis-config.xml 설정 예

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-config.dtd">

<configuration>
  <settings>
    <setting name="cacheEnabled" value="false" />
    <setting name="useGeneratedKeys" value="true" />
    <setting name="defaultExecutorType" value="REUSE" />
    <setting name="callSettersOnNulls" value="true"/>
    <setting name="jdbcTypeForNull" value="VARCHAR"/>
  </settings>
</configuration>
```

6.2.4 xxxxMapper.xml 구현 예

```
<?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="com.janggyo.mybatis.dao.LauncherAppDao">
  <resultMap id="LauncherAppMap" type="com.janggyo.mybatis.dto.LauncherApp" >
    <id property="launcherAppId" column="LAUNCHER_APP_ID"
      javaType="String" />
    <result property="launcherAppId" column="LAUNCHER_APP_ID"
      javaType="String" />
    <result property="createdDate" column="CREATED_DATE"
      javaType="Date" />
    <result property="forcedUpdate" column="FORCED_UPDATE"
      javaType="String" />
    <result property="lastForcedUpdateVersion"
      column="LAST_FORCED_UPDATE_VERSION" javaType="int"/>
    <result property="status" column="STATUS" javaType="String" />
    <result property="title" column="TITLE" javaType="String" />
    <result property="launcherAppVersion" column="LAUNCHER_APP_VERSION"
      javaType="int" />
    <result property="config" column="CONFIG" javaType="String" />
  </resultMap>

```

```
        <result property="updatedAt" column="UPDATED_DATE" javaType="Date" />
    </resultMap>
    <select id="getLauncherApp" resultMap="LauncherAppMap"
parameterType="com.janggyo.mybatis.dto.LauncherApp" >
        SELECT
            LAUNCHER_APP_ID
            ,CREATED_DATE
            ,FORCED_UPDATE
            ,LAST_FORCED_UPDATE_VERSION
            ,STATUS
            ,TITLE
            ,STATUS
            ,LAUNCHER_APP_VERSION
            ,CONFIG
            ,UPDATED_DATE
        FROM H2_LAUNCHER_APP
    </select>
</mapper>
```

6.2.5 LauncherAppDao.java 구현 예

인터페이스의 메소드명을 통하여 XML 의 쿼리와 매핑되어 호출되며 자바코드에서는 그냥 인터페이스를 선언 후 호출하기만 하면 된다.

```
package com.janggyo.mybatis.dao;

import java.util.List;

import org.springframework.stereotype.Repository;

import com.janggyo.mybatis.dto.LauncherApp;

@Repository(value="mybatis.dao.LauncherAppMapper")
public interface LauncherAppDao {

    List<LauncherApp> getlauncherApp();

}
```


References

[1]	
[2]	
[3]	

Revision History

Version	Date	Change from Previous	Name	Status