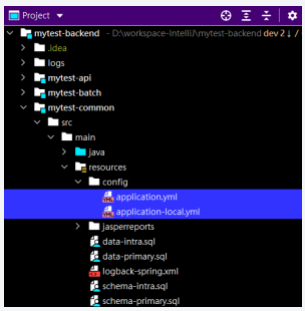
Spring Data JPA

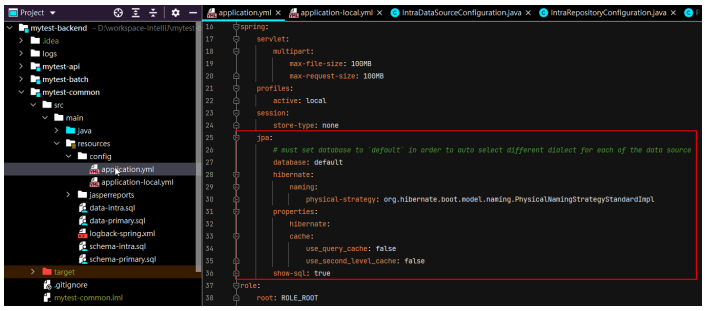
Spring Data JPA 是基於 ORM 框架、JPA規範而封裝的一套應用框架，能讓開發者用「制式的程式碼」對資料庫進行CRUD操作，也易於進行功能擴充

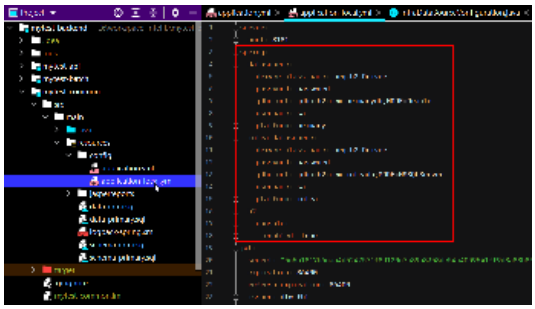
> *Spring Boot 對 JPA 的基本配置，請透過鍵盤快捷鍵<Ctrl><N>，尋找 「**JpaBaseConfiguration**」*
>
> - *大致上看看就好....因為接下來的擴展才是認真看的重點 !!!*

連線設定

application*.yml







Java Config

IntraDataSourceConfiguration .java

```
package com.fubonlife.mytest.
common.config;

import org.springframework.beans.
factory.annotation.Autowired;
import org.springframework.beans.
factory.annotation.Qualifier;
import org.springframework.beans.
factory.annotation.Value;
import org.springframework.boot.
context.properties.
ConfigurationProperties;
import org.springframework.boot.
jdbc.DataSourceBuilder;
import org.springframework.context.
annotation.Bean;
import org.springframework.context.
annotation.Configuration;
import org.springframework.core.io.
ResourceLoader;
import org.springframework.jdbc.
datasource.init.
DataSourceInitializer;
import org.springframework.jdbc.
datasource.init.
ResourceDatabasePopulator;
```

PrimaryDataSourceConfiguration.java

```
package com.fubonlife.mytest.common.
config;

import org.springframework.boot.context.
properties.ConfigurationProperties;
import org.springframework.boot.jdbc.
DataSourceBuilder;
import org.springframework.context.
annotation.Bean;
import org.springframework.context.
annotation.Configuration;
import org.springframework.context.
annotation.Primary;

import javax.sql.DataSource;

@Configuration
public class
PrimaryDataSourceConfiguration {

    @Bean("primaryDataSource")
    @Primary
    @ConfigurationProperties(prefix =
"spring.datasource")
    public DataSource primaryDataSource()
{
        return DataSourceBuilder.create().
build();
    }
}
```

PrimaryRepositoryConfiguration.java

```
package com.fubonlife.mytest.common.
config;

import org.springframework.beans.factory.
annotation.Autowired;
import org.springframework.beans.factory.
annotation.Qualifier;
import org.springframework.boot.
autoconfigure.orm.jpa.HibernateProperties;
import org.springframework.boot.
autoconfigure.orm.jpa.HibernateSettings;
import org.springframework.boot.
autoconfigure.orm.jpa.JpaProperties;
import org.springframework.boot.orm.jpa.
EntityManagerFactoryBuilder;
import org.springframework.context.
annotation.Bean;
import org.springframework.context.
annotation.Configuration;
import org.springframework.context.
annotation.Primary;
import org.springframework.data.jpa.
repository.config.EnableJpaAuditing;
import org.springframework.data.jpa.
repository.config.EnableJpaRepositories;
import org.springframework.orm.jpa.
JpaTransactionManager;
import org.springframework.orm.jpa.
LocalContainerEntityManagerFactoryBean;
import org.springframework.transaction.
PlatformTransactionManager;
import org.springframework.transaction.
annotation.EnableTransactionManagement;
import javax.sql.DataSource;

@Configuration
public class
IntraDataSourceConfiguration {

    @Value("${spring.intra-
datasource.platform}")
    private String platform;

    @Autowired
    ResourceLoader resourceLoader;

    @Bean("intraDataSource")
    @ConfigurationProperties
(prefix = "spring.intra-
datasource")
    public DataSource
intraDataSource() {
        return DataSourceBuilder.
create().build();
    }

    /*  H2  DBIntra  SQL  H2
DataSourceInitializer*/
    @Bean
    public DataSourceInitializer
dataSourceInitializer(@Qualifier
("intraDataSource") DataSource
datasource) {
        ResourceDatabasePopulator
resourceDatabasePopulator = new
ResourceDatabasePopulator();
        resourceDatabasePopulator.
addScript(resourceLoader.
getResource("classpath:/schema-" +
platform + ".sql"));
        resourceDatabasePopulator.
addScript(resourceLoader.
getResource("classpath:/data-" +
platform + ".sql"));

        DataSourceInitializer
dataSourceInitializer = new
DataSourceInitializer();
        dataSourceInitializer.
setDataSource(datasource);
        dataSourceInitializer.
setDatabasePopulator
(resourceDatabasePopulator);
        return
dataSourceInitializer;
    }
}
```

IntraRepositoryConfiguration.java

```
package com.fubonlife.mytest.
common.config;

import org.springframework.beans.
factory.annotation.Autowired;
import org.springframework.beans.
factory.annotation.Qualifier;
import org.springframework.boot.
autoconfigure.orm.jpa.
HibernateProperties;
import org.springframework.boot.
autoconfigure.orm.jpa.
```

```
import javax.persistence.EntityManager;
import javax.sql.DataSource;
import java.util.Map;

@EnableJpaRepositories(
        entityManagerFactoryRef =
"primaryEntityManagerFactory",
        transactionManagerRef =
"primaryTransactionManager",
        basePackages = {"com.fubonlife.
mytest.common.repository.primary"})
@EnableTransactionManagement
@EnableJpaAuditing
@Configuration
public class
PrimaryRepositoryConfiguration {

    @Autowired
    @Qualifier("primaryDataSource")
    private DataSource dataSource;

    @Autowired
    private JpaProperties jpaProperties;

    @Autowired
    private HibernateProperties
hibernateProperties;


    @Primary
    @Bean(name =
"primaryEntityManagerFactory")
    public
LocalContainerEntityManagerFactoryBean
entityManagerFactory
(EntityManagerFactoryBuilder builder) {
        return builder
                .dataSource(dataSource)
                .properties
(getVendorProperties())
                .packages("com.fubonlife.
mytest.common.entity.primary")
                .persistenceUnit
("primaryPersistenceUnit")

                .build();
    }


    private Map<String, Object>
getVendorProperties() {
        Map<String, Object> retVal =
hibernateProperties.
determineHibernateProperties
(jpaProperties.getProperties(), new
HibernateSettings());
        retVal.put("hibernate.dialect",
"org.hibernate.dialect.Oracle12cDialect");
        return retVal;
    }

    @Primary
    @Bean(name =
"primaryTransactionManager")
    public PlatformTransactionManager
transactionManagerPrimary
(EntityManagerFactoryBuilder builder) {
        return new JpaTransactionManager
(entityManagerFactory(builder).
getObject());
```

```
HibernateSettings;
import org.springframework.boot.
autoconfigure.orm.jpa.
JpaProperties;
import org.springframework.boot.
orm.jpa.
EntityManagerFactoryBuilder;
import org.springframework.context.
annotation.Bean;
import org.springframework.context.
annotation.Configuration;
import org.springframework.data.
jpa.repository.config.
EnableJpaAuditing;
import org.springframework.data.
jpa.repository.config.
EnableJpaRepositories;
import org.springframework.orm.jpa.
JpaTransactionManager;
import org.springframework.orm.jpa.
LocalContainerEntityManagerFactoryB
ean;
import org.springframework.
transaction.
PlatformTransactionManager;
import org.springframework.
transaction.annotation.
EnableTransactionManagement;

import javax.persistence.
EntityManager;
import javax.sql.DataSource;
import java.util.Map;

@EnableJpaRepositories(
        entityManagerFactoryRef =
"intraEntityManagerFactory",
        transactionManagerRef =
"intraTransactionManager",
        basePackages = {"com.
fubonlife.mytest.common.repository.
intra"})
@EnableTransactionManagement
@Configuration
public class
IntraRepositoryConfiguration {

    @Autowired
    @Qualifier("intraDataSource")
    private DataSource dataSource;

    @Autowired
    private JpaProperties
jpaProperties;

    @Autowired
    private HibernateProperties
hibernateProperties;

    @Bean("intraEntityManager")
    public EntityManager
entityManager
(EntityManagerFactoryBuilder
builder) {
        return entityManagerFactory
(builder).getObject().
createEntityManager();
    }


    @Bean(name =
```

```
    }
}
```

```
"intraEntityManagerFactory")
    public
LocalContainerEntityManagerFactoryB
ean entityManagerFactory
(EntityManagerFactoryBuilder
builder) {
        return builder
                .dataSource
(dataSource)
                .properties
(getVendorProperties())
                .packages("com.
fubonlife.mytest.common.entity.
intra")
                .persistenceUnit
("intraPersistenceUnit")
                .build();
    }


    private Map<String, Object>
getVendorProperties() {
        return hibernateProperties.
determineHibernateProperties
(jpaProperties.getProperties(),
new HibernateSettings());
    }


    @Bean(name =
"intraTransactionManager")
    public
PlatformTransactionManager
transactionManagerPrimary
(EntityManagerFactoryBuilder
builder) {
        return new
JpaTransactionManager
(entityManagerFactory(builder).
getObject());
    }
}
```

透過 JNDI 獲取 DataSource

> *jndi-name ?? 就一個字串，大概就像 【 java:comp/env/datasource/primaryMyTest 】*

PrimaryDataSourceConfiguration.java

```
package com.fubonlife.mytest.common.config;

import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.boot.jdbc.DataSourceBuilder;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Primary;
import org.springframework.core.env.Environment;
import org.springframework.core.env.Profiles;
import org.springframework.jdbc.datasource.lookup.JndiDataSourceLookup;

import javax.sql.DataSource;

@Configuration
@Slf4j
public class PrimaryDataSourceConfiguration {

    @Autowired
    Environment environment;

    @Value("${spring.datasource.jndi-name:}")
    private String jndiName;

    @Bean("primaryDataSource")
    @Primary
    @ConfigurationProperties(prefix = "spring.datasource")
    public DataSource primaryDataSource() {
        DataSource ds = null;

        try {
            ds = new JndiDataSourceLookup().getDataSource(jndiName);
            log.info("primaryDataSource jndiName={}", jndiName);
        } catch (Exception e) {
            if (environment.acceptsProfiles(Profiles.of("prod"))) {
                log.error("primaryDataSource exception occurred", e);
            } else {
                log.warn("primaryDataSource exception occurred: {}", e.getMessage());
                ds = DataSourceBuilder.create().build();
                log.warn("primaryDataSource in yml");
            }
        }

        return ds;
    }
}
```

客製化 hibernate.dialect

> *若有需要升級 dialect，務必確認測試機 JBOSS 預設使用的資料庫 Driver 是否支援 !!*
>
> *Oracle*
>
> - *retVal.put("hibernate.dialect", "org.hibernate.dialect.Oracle12cDialect");*
>
> *MSSQL*
>
> - *retVal.put("hibernate.dialect", "org.hibernate.dialect.SQLServer2008Dialect");*

官方資料來源：Hibernate User Guide

PrimaryRepositoryConfiguration.java

```
import org.springframework.core.env.Environment;


@EnableJpaRepositories(
        entityManagerFactoryRef = "primaryEntityManagerFactory",
        transactionManagerRef = "primaryTransactionManager",
        basePackages = {"com.fubonlife.mytest.common.repository.primary"})
@EnableTransactionManagement
@EnableJpaAuditing
@Configuration
public class PrimaryRepositoryConfiguration {

    // ...

    @Autowired
    private JpaProperties jpaProperties;

    @Autowired
    private HibernateProperties hibernateProperties;

    @Autowired
    private Environment environment;

    // ...

    private Map<String, Object> getVendorProperties() {
        Map<String, Object> retVal = hibernateProperties.determineHibernateProperties(jpaProperties.
getProperties(), new HibernateSettings());

                if(!this.environment.acceptsProfiles(Profiles.of("local"))) {
            //  Oracle
            retVal.put("hibernate.dialect", "org.hibernate.dialect.Oracle12cDialect");
        }

                return retVal;
    }

    // ...
}
```

⚠ 若 Hibernate 被配置不合適的 dialect ，會發生什麼事？

```
[2022-10-25 15:46:52.282][      main][INFO ][org.hibernate.dialect.Dialect ][ 175] - HHH000400: Using dialect: org.hibernate.dialect.Oracle12cDial
[2022-10-25 15:46:52.339][      main][ERROR][.h.e.j.e.i.JdbcEnvironmentImpl][ 420] - Could not fetch the SequenceInformation from the database
```

撰寫 Repository

> *Spring Data JPA 為降低存取資料層（Data Access Object, DAO）的開發工作量，讓開發人員只需寫出 Repository 的【介面】，由 Spring's SimpleJpaRepository 自動實作其功能。*
>
> - *JpaRepository 是PagingAndSortingRepository的子介面*
> - *PagingAndSortingRepository 是CrudRepository的子介面，添加分頁和排序的功能*
> - *CrudRepository 是Repository的子介面，提供CRUD的功能*
> - *Repository 最頂層的介面*

```
import com.fubonlife.mytest.common.entity.primary.LogTrace;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.JpaSpecificationExecutor;
import org.springframework.stereotype.Repository;

@Repository 可省略，但建議保留
public interface LogTraceRepository extends JpaRepository<LogTrace, String>, JpaSpecificationExecutor<LogTrace> {

}
```

Entity　Primary Key　Entity

固定繼承　固定繼承

- 舉例說明一

```
@Repository
public interface SysCodeMainRepository extends JpaRepository<SysCodeMain, String>, JpaSpecificationExecutor<SysCodeMain>
```

- 舉例說明二

```
@Repository
public interface SysCodeDetailRepository extends JpaRepository<SysCodeDetail, SysCodeDetailPK>, JpaSpecificationExecutor<SysCodeDetail>
```

- 舉例說明三

```
@Repository
public interface SysCodeGroupRepository extends JpaRepository<SysCodeGroup, SysCodeGroupPK>, JpaSpecificationExecutor<SysCodeGroup>
```

認識 JpaRepository 內建的 CRUD 操作

> *Spring Data JPA 將「基本的 CRUD 操作」封裝在此*
>
> - *提高 CRUD 程式碼的可讀性、可移植性 !!!*
>
> *Spring Data JPA 對「基本的 CRUD 之 Read / Delete 操作」，提供命名式擴充法之設計*

1. 新增

```
@NoRepositoryBean
public interface JpaRepository<T, ID> extends PagingAndSortingRepository<T, ID>, QueryByExampleExecutor<T> {
    List<T> findAll();

    List<T> findAll(Sort sort);

    List<T> findAllById(Iterable<ID> ids);

    <S extends T> List<S> saveAll(Iterable<S> entities);

    void flush();

    <S extends T> S saveAndFlush(S entity);

    <S extends T> List<S> saveAllAndFlush(Iterable<S> entities);
```

```
@NoRepositoryBean
public interface CrudRepository<T, ID> extends Repository<T, ID> {
    <S extends T> S save(S entity);

    <S extends T> Iterable<S> saveAll(Iterable<S> entities);
```

2. 單表查詢

```
@NoRepositoryBean
public interface JpaRepository<T, ID> extends PagingAndSortingRepository<T, ID>, QueryByExampleExecutor<T> {
    List<T> findAll();

    List<T> findAll(Sort sort);

    List<T> findAllById(Iterable<ID> ids);
```

```java
@NoRepositoryBean
public interface PagingAndSortingRepository<T, ID> extends CrudRepository<T, ID> {
    Iterable<T> findAll(Sort sort);

    Page<T> findAll(Pageable pageable);
}
```

```java
@NoRepositoryBean
public interface CrudRepository<T, ID> extends Repository<T, ID> {
    <S extends T> S save(S entity);

    <S extends T> Iterable<S> saveAll(Iterable<S> entities);

    Optional<T> findById(ID id);

    boolean existsById(ID id);

    Iterable<T> findAll();

    Iterable<T> findAllById(Iterable<ID> ids);

    long count();

    void deleteById(ID id);
```

3. 修改

> *步驟一：在 Service 層撰寫查詢的操作方法，來篩選出指定的待修改資料*
>
> *步驟二：透過【Setter】方法來修改上一步驟的查詢結果，此過程即為對資料庫的實際資料進行修改 !!!*

- 舉例說明

Service 層

```java
@Override
@Transactional // ()
public boolean modify(ModifyMenuItem modifyMenuItem) {
    if (modifyMenuItem.getId() == null) {
        log.warn("{}", modifyMenuItem);
```

```
        return false;
    } else {
        boolean isOK = false;

        Menu menu = menuRepository.findById(modifyMenuItem.getId()).orElse(null);
        if (menu != null) {

            // (NULL)
            if (modifyMenuItem.getIsLeaf() != null) {
                menu.setIsLeaf(modifyMenuItem.getIsLeaf());
                isOK = true;
            }

            // (NULL)
            if (StringUtils.isNotBlank(modifyMenuItem.getTitle())) {
                menu.setTitle(modifyMenuItem.getTitle());
                isOK = true;
            }

            // (NULL)
            if (modifyMenuItem.getRoute() != null) {
                menu.setRoute(StringUtils.isWhitespace(modifyMenuItem.getRoute()) ? null : modifyMenuItem.
getRoute());
                isOK = true;
            }

            // (NULL)
            if (modifyMenuItem.getUri() != null) {
                menu.setUri(StringUtils.isWhitespace(modifyMenuItem.getUri()) ? null : modifyMenuItem.getUri());
                isOK = true;
            }

            // (NULL)
            if (modifyMenuItem.getEnabled() != null) {
                menu.setEnabled(modifyMenuItem.getEnabled());
                isOK = true;
            }

            // (0)
            if (modifyMenuItem.getSequence() != null && modifyMenuItem.getSequence() > 0) {
                menu.setSequence(modifyMenuItem.getSequence());
                isOK = true;
            }

            //  ID(NULL)
            if (modifyMenuItem.getParentMenuId() != null) {
                if (StringUtils.isWhitespace(modifyMenuItem.getParentMenuId())) {
                    menu.setParentMenuId(null);
                    isOK = true;
                } else {
                    Menu parentMenu = menuRepository.findById(modifyMenuItem.getParentMenuId()).orElse(null);
                    if (parentMenu != null) {
                        menu.setParentMenuId(modifyMenuItem.getParentMenuId());
                        isOK = true;
                    }
                }
            }
        }

        if (!isOK) {
            log.warn("{}", modifyMenuItem);
        }
        // else {
        //    menuRepository.save(menu); <---
        //}

        return isOK;
    }
}
```

Controller 層

```
@PostMapping("/modify")
@ApiOperation("")
@PreAuthorize("hasAnyRole('ROOT')")
public String modifyMenuItem(@RequestBody ModifyMenuItem modifyMenuItem) {
    boolean isOK = menuService.modify(modifyMenuItem);
    return isOK ? "" : "";
}
```

4. 刪除

```
@NoRepositoryBean
public interface CrudRepository<T, ID> extends Repository<T, ID> {
    <S extends T> S save(S entity);

    <S extends T> Iterable<S> saveAll(Iterable<S> entities);

    Optional<T> findById(ID id);

    boolean existsById(ID id);

    Iterable<T> findAll();

    Iterable<T> findAllById(Iterable<ID> ids);

    long count();

    void deleteById(ID id);

    void delete(T entity);

    void deleteAllById(Iterable<? extends ID> ids);

    void deleteAll(Iterable<? extends T> entities);

    void deleteAll();
}
```

5. 命名式擴充法的 RD 操作

> *方法主題關鍵字*

- 原文連結：Supported query method subject keywords

Keyword	Description
`find…By` , `read…By` , `get…By` , `query…By` , `search…By` , `stream…By`	General query method returning typically the repository type, a `Collection` or `Streamable` subtype or a result wrapper such as `Page` , `GeoResults` or any other store-specific result wrapper. Can be used as `findBy…` , `findMyDomainTypeBy…` or in combination with additional keywords.
`exists…By`	Exists projection, returning typically a `boolean` result.
`count…By`	Count projection returning a numeric result.
`delete…By` , `remove…By`	Delete query method returning either no result (`void`) or the delete count.
`…First<number>…` , `…Top<number>…`	Limit the query results to the first `<number>` of results. This keyword can occur in any place of the subject between `find` (and the other keywords) and `by` .
`…Distinct…`	Use a distinct query to return only unique results. Consult the store-specific documentation whether that feature is supported. This keyword can occur in any place of the subject between `find` (and the other keywords) and `by` .

Keyword	Description
`IgnoreCase` , `IgnoringCase`	Used with a predicate keyword for case-insensitive comparison.
`AllIgnoreCase` , `AllIgnoringCase`	Ignore case for all suitable properties. Used somewhere in the query method predicate.
`OrderBy…`	Specify a static sorting order followed by the property path and direction (e. g. `OrderByFirstnameAscLastnameDesc`).

- 舉例說明（原文連結：JPA Query Methods）

Keyword	Sample	JPQL snippet
`Distinct`	`findDistinctByLastnameAndFirstname`	`select distinct … where x.lastname = ?1 and x.firstname = ?2`
`And`	`findByLastnameAndFirstname`	`… where x.lastname = ?1 and x.firstname = ?2`
`Or`	`findByLastnameOrFirstname`	`… where x.lastname = ?1 or x.firstname = ?2`
`Is`, `Equals`	`findByFirstname`, `findByFirstnameIs`, `findByFirstnameEquals`	`… where x.firstname = ?1`
`Between`	`findByStartDateBetween`	`… where x.startDate between ?1 and ?2`
`LessThan`	`findByAgeLessThan`	`… where x.age < ?1`
`LessThanEqual`	`findByAgeLessThanEqual`	`… where x.age <= ?1`
`GreaterThan`	`findByAgeGreaterThan`	`… where x.age > ?1`
`GreaterThanEqual`	`findByAgeGreaterThanEqual`	`… where x.age >= ?1`
`After`	`findByStartDateAfter`	`… where x.startDate > ?1`
`Before`	`findByStartDateBefore`	`… where x.startDate < ?1`
`IsNull`, `Null`	`findByAge(Is)Null`	`… where x.age is null`
`IsNotNull`, `NotNull`	`findByAge(Is)NotNull`	`… where x.age not null`
`Like`	`findByFirstnameLike`	`… where x.firstname like ?1`
`NotLike`	`findByFirstnameNotLike`	`… where x.firstname not like ?1`
`StartingWith`	`findByFirstnameStartingWith`	`… where x.firstname like ?1` (parameter bound with appended `%`)
`EndingWith`	`findByFirstnameEndingWith`	`… where x.firstname like ?1` (parameter bound with prepended `%`)
`Containing`	`findByFirstnameContaining`	`… where x.firstname like ?1` (parameter bound wrapped in `%`)
`OrderBy`	`findByAgeOrderByLastnameDesc`	`… where x.age = ?1 order by x.lastname desc`
`Not`	`findByLastnameNot`	`… where x.lastname <> ?1`
`In`	`findByAgeIn(Collection<Age> ages)`	`… where x.age in ?1`
`NotIn`	`findByAgeNotIn(Collection<Age> ages)`	`… where x.age not in ?1`
`True`	`findByActiveTrue()`	`… where x.active = true`
`False`	`findByActiveFalse()`	`… where x.active = false`
`IgnoreCase`	`findByFirstnameIgnoreCase`	`… where UPPER(x.firstname) = UPPER(?1)`

> *方法返回值類型*

- 原文連結：Supported Query Return Types
- 舉例說明（原文連結：Defining Query Methods）

聲明式 @Query 的 CRUD 操作

1. JPQL

> *1. JPQL 的全名是 Java Persistence Query Language，是一種與使用 DB 無關的的物件導向 SQL*
>
> - *JPQL 在基於 Entity 的規範下，Entity 的物件名稱就是資料表名稱（大寫駝峰式），以及物件內的變數名稱就是欄位名稱（小寫駝峰式）*
>
> *2. JPA 則會根據使用 DB 的類型，決定組成 SQL 指令時候的保留字或 SQL 組成規範*
>
> - *這讓開發人員不需要針對不同的 DB 做太大幅度的調整，盡量讓 JPA 幫忙處理瑣碎的事情即可，因此大大增加了開發的彈性*

JPQL 僅支援標準的子查詢指令，因此無法像 Native SQL 一次性的使用多階層、多 Table 的子查詢 SQL 指令，意即無法在 JPQL 中做太多複雜的變化

2. Native SQL

> *1. Native SQL 會將參數、條件帶入後，就直接執行*
>
> *2. 當撰寫的原生 SQL 長度特長時，建議改寫在 Service 層（同時建議搭配具有一致性的方法名稱規範）*

3. 應用方式

	JPQL 或 Native SQL

新增	**** JPQL 不支援 Insert 語法，所以只能用 Native SQL 來進行 ******
	 SQL
	 `@Modifying  //`
	 `@Query(value = "insert into User (name, status, email) values (:name, :status, :email)", nativeQuery = true)`
	 `void insertUserUsingSQL(String name, Integer status, String email);`

查詢	JPQL
	 `@Query("SELECT u FROM User u WHERE u.deadStatus = false")`
	 `Collection<User> selectAllActiveUsersUsingJPQL();`
	
	 JPQL
	 `@Query("SELECT u FROM User u WHERE u.deadStatus = :deadStatus")`
	 `Collection<User> selectAllActiveUsersUsingJPQL(boolean deadStatus);`
	
	 Native SQL
	 `@Query(value = "SELECT * FROM USER u WHERE u.DEAD_STATUS = 0", nativeQuery = true)`
	 `Collection<User> selectAllActiveUsersUsingSQL();`
	
	 Native SQL
	 `@Query(value = "SELECT * FROM USER u WHERE u.DEAD_STATUS = :deadStatusCode", nativeQuery = true)`
	 `Collection<User> selectAllActiveUsersUsingSQL(int deadStatusCode);`

修改 OR 刪除	JPQL
	`@Modifying  //`
	`@Query("update Department d set d.deleted = true where d.id in :ids")`
	`void deleteByIdsUsingJPQL(Set<String> ids);`
	
	SQL
	`@Modifying  //`
	`@Query(value = "DELETE FROM APPENDIX_CONTENT WHERE APPENDIX_ID IN ( " +`
	`  "SELECT APPENDIX_ID FROM APPENDIX WHERE CRAWLER_DATA_ID IN :crawlerData " +`
	`  ") ", nativeQuery = true)`
	`void deleteAppendixContentDataUsingSQL(List<String> crawlerData);`

排序	JPQL
	`@Query(value = "SELECT u FROM User u ORDER BY u.name")`
	`List<User> selectAllUsersSortUsingJPQL();`
	
	`// UserName`
	`userRepository.selectAllUsersSortUsingJPQL(Sort.by(Sort.Direction.ASC, "name"));`
	
	`// UserName`
	`userRepository.selectAllUsersSortUsingJPQL(JpaSort.unsafe("LENGTH(name)")); //  JpaSort.unsafe`
	
	`// XXX()`
	`@Query(value = "SELECT u FROM User u")`
	`List<User> selectAllUsersSortUsingJPQL(Sort sort);`
	
	SQL
	`@Query(value = "SELECT * FROM USER u WHERE u.DEAD_STATUS = :deadStatusCode ORDER BY NAME", nativeQuery = true)`
	`Collection<User> selectAllActiveUsersUsingSQL(int deadStatusCode);`

分頁	JPQL
	`@Query(value = "SELECT u FROM User u ORDER BY u.id")`
	`Page<User> findAllUsersWithPaginationUsingJPQL(Pageable pageable); // PageableInterfacePageRequest`
	
	SQL
	`@Query(value = "SELECT * FROM Users ORDER BY id",`
	`  countQuery = "SELECT count(*) FROM Users",  // countQuery`
	`  nativeQuery = true)`
	`Page<User> findAllUsersWithPaginationUsingSQL(Pageable pageable);`

撰寫 Service

> *首先設計 Interface，在設計其實現的類，這樣我們就可以在應用中調用Service接口來進行業務處理*
>
> - *負責業務模塊的邏輯應用設計，有利於通用的業務邏輯的獨立性和重複利用性*
>
> *原則上，業務實現會調用到已定義的DAO層的接口，來進行數據業務的處理*

- 舉例說明

```java
@Service
public class SysCodeMainServiceImpl implements SysCodeMainService {

    @Autowired
    SysCodeMainRepository sysCodeMainRepository;

    @Autowired
    ModelMapper modelMapper;
```

```java
@Service
public class SysCodeDetailServiceImpl implements SysCodeDetailService {

    @Autowired
    SysCodeDetailRepository sysCodeDetailRepository;

    @Autowired
    SysCodeMainService sysCodeMainService;

    @Autowired
    ModelMapper modelMapper;
```

```java
@Service
public class SysCodeGroupServiceImpl implements SysCodeGroupService {

    @Autowired
    SysCodeGroupRepository sysCodeGroupRepository;

    @Autowired
    SysCodeDetailService sysCodeDetailService;

    @Autowired
    ModelMapper modelMapper;
```

交易管理

- 通常會宣告在 Service 層的實現類
- 只要在方法上面加上標記【 @Transactional 】、【 @Transactional(readOnly=true)】

動態查詢

> *模式：JpaSpecification*

單表+動態查詢條件

```
@Override
public List<IntraDepartment> getAll() {
    Map<String, String> filter = sysCodeDetailService.getMap("DB_INTRA");
    String delimiter = ",";

    Specification<IntraDepartment> dynamicQueryRule = (root, query, cb) -> {
        //
        query.orderBy(cb.asc(root.get("id")));

        //
        Predicate p = cb.conjunction();
        if (ObjectUtils.isNotEmpty(filter) && !StringUtils.equalsIgnoreCase("local", springProfile)) {

            // AND-GRP_CD not in ('___')
            p = cb.and(p, root.get("grpCd").in(Arrays.asList("XXX", "")).not());

            // AND-DPT_TYPE not in ('___')
            p = cb.and(p, root.get("dptType").in(Arrays.asList("00", "01")).not());

            // AND-DPT_NAME not like '%___'
            if (filter.containsKey("dpt_not_end_with")) {
                for (String keyword : filter.get("dpt_not_end_with").split(delimiter)) {
                    if (StringUtils.isNotBlank(keyword)) {
                        p = cb.and(p, cb.like(root.get("name"), "%" + keyword).not());
                    }
                }
            }

            // AND-DPT_NAME not like '%___%'
            if (filter.containsKey("dpt_not_like")) {
                for (String keyword : filter.get("dpt_not_like").split(delimiter)) {
                    if (StringUtils.isNotBlank(keyword)) {
                        p = cb.and(p, cb.like(root.get("name"), "%" + keyword + "%").not());
                    }
                }
            }

            // AND-DPT_NAME not like '___%'
            if (filter.containsKey("dpt_not_start_with")) {
                for (String keyword : filter.get("dpt_not_start_with").split(delimiter)) {
                    if (StringUtils.isNotBlank(keyword)) {
                        p = cb.and(p, cb.like(root.get("name"), keyword + "%").not());
                    }
                }
            }

            // OR-DPT_NAME like '%___%'
            if (filter.containsKey("dpt_like")) {
                for (String keyword : filter.get("dpt_like").split(delimiter)) {
                    if (StringUtils.isNotBlank(keyword)) {
                        p = cb.or(p, cb.like(root.get("name"), "%" + keyword + "%"));
                    }
                }
            }
        }

        return p;
    };

    return intraDepartmentRepository.findAll(dynamicQueryRule);
}
```

單表+動態查詢條件+分頁

```
@Override
public Page<AnnouncementDto> getRecentList(QueryAnnouncement search) {
    Specification<Announcement> dynamicQueryRule = (root, query, cb) -> {
        //
        query.orderBy(cb.desc(root.get("startDatetime")), cb.desc(root.get("endDatetime")));

        //
        Predicate p = cb.conjunction();

        //
        LocalDateTime start, end, now = LocalDateTime.now();

        if (StringUtils.isBlank(search.getStartYYYYMMDD()) || StringUtils.isBlank(search.getEndYYYYMMDD())) {
            long months = 18;
            end = now;
            start = end.minusMonths(months);
            log.warn("({}{}){}({}{})", search.getStartYYYYMMDD(), search.getEndYYYYMMDD(), months, start, end);
        } else {
            start = dateTimeUtil.parseLocalDateTime(search.getStartYYYYMMDD(), "00:00:00");
            end = dateTimeUtil.parseLocalDateTime(search.getEndYYYYMMDD(), "23:59:59");
            //
            if (end.isAfter(now)) {
                end = now;
            }
        }

        // AND-START_DATETIME
        p = cb.and(p, cb.between(root.get("startDatetime"), start, end));

        // AND-CONTENT
        if (StringUtils.isNotBlank(search.getKeyword())) {
            p = cb.and(p, cb.like(root.get("content"), "%" + search.getKeyword() + "%"));
        }

        // FBLFBL
        Map<String, String> profile = accountService.getProfile();
        List<String> departmentIds = Arrays.asList(profile.get("FBL"), profile.get("FBL"));

        //
        List<String> unAuthorizedAnnouncementId = getAnnouncementIdForOtherUnit(start, end, departmentIds);

        // AND-ANNOUNCEMENT_ID
        if (!unAuthorizedAnnouncementId.isEmpty()) {
            p = cb.and(p, root.get("announcementId").in(unAuthorizedAnnouncementId).not());
        }

        return p;
    };

    //
    Map<String, String> categoryMap = getAnnouncementCategoryMap();

    if (search.getPageSize() == 0) {
        search.setPageSize(100);
    }

    Page<AnnouncementDto> result = announcementRepository.findAll(dynamicQueryRule, PageRequest.of(search.
getPageIndex(), search.getPageSize()))
        .map(x -> modelMapper.map(x, AnnouncementDto.class).toBuilder()
            .categoryName(MapUtils.getString(categoryMap, x.getCategory(), x.getCategory() + "?"))
            .startTimestamp(x.getStartDatetime())
            .endTimestamp(x.getEndDatetime())
            .build());

    return result;
}
```

> *模式：QueryDSL*

多表+動態查詢條件+分頁

```
@PersistenceContext
EntityManager em;


@Override
public Page<SysCodeDto> get(SearchSysCode search) {
    PageRequest pr = PageRequest.of(search.getPageIndex(), (search.getPageSize() == 0) ? 100 : search.
getPageSize());

    QSysCodeMain t_main = QSysCodeMain.sysCodeMain;
    QSysCodeDetail t_detail = QSysCodeDetail.sysCodeDetail;
    QSysCodeGroup t_group = QSysCodeGroup.sysCodeGroup;

    JPAQuery<SysCodeDto> query = new JPAQueryFactory(em).select(
        Projections.fields(SysCodeDto.class,
            // ()
            t_main.sysCodeMainId, t_main.defaultText.as("defaultTextInSysCodeMain"), t_main.doubleCheck, t_main.
isEditable.as("isEditableInSysCodeMain"), t_main.remark.as("remarkInSysCodeMain"),
            // ()
            t_detail.sysCodeDetailId, t_detail.defaultText.as("defaultTextInSysCodeDetail"), t_detail.sequence.
as("sequenceInSysCodeDetail"), t_detail.enabled.as("enabledInDetail"), t_detail.isEditable.as
("isEditableInSysCodeDetail"), t_detail.remark.as("remarkInSysCodeDetail"),
            // ()
            t_group.sysCodeGroupId, t_group.defaultText.as("defaultTextInSysCodeGroup"), t_group.sequence.as
("sequenceInSysCodeGroup"), t_group.enabled.as("enabledInGroup"), t_group.isEditable.as
("isEditableInSysCodeGroup"), t_group.remark.as("remarkInSysCodeGroup"))
    ).from(t_main)
        .leftJoin(t_main.sysCodeDetailList, t_detail)
        .leftJoin(t_detail.sysCodeGroupList, t_group)
        .where(t_main.sysCodeMainId.eq(search.getSysCodeMainId()), getWhereClause(search));

    List<SysCodeDto> resultPaginated = query
        .limit(pr.getPageSize()).offset(pr.getOffset()) //
        .orderBy(t_main.sysCodeMainId.asc(), t_detail.sysCodeDetailId.asc(), t_group.sequence.asc()) //
        .fetch();

    return new PageImpl<>(resultPaginated, pr, query.fetchCount());
}
```