主題 5：認識程式設計的參數化思維

建置專案的預設參數

即便是同一個專案，有時候我們也會希望可以傳遞一些不同的變數來當做此次的建置參數

這時就可修改 Spring Boot 原生的參數化設定值，來幫我們增加彈性

> *專案啟動時，指定套用哪一種配置文件*
>
> *application-**local**.yml 代表在自己的電腦內模擬 IT 的測試環境，然而此時搭配【 H2 記憶體型資料庫 】*
>
> *application-**dev**.yml 代表專案被啟動在 IT 的測試環境，或者在自己的電腦內模擬 IT 的測試環境，然而此時搭配公司資料庫*
>
> *application-**uat**.yml 代表專案被啟動在 USER 的測試環境，或者在自己的電腦內模擬 USER 的測試環境，然而此時搭配公司資料庫*
>
> *application-**prod**.yml 代表專案被啟動在 USER 的正式環境，然而此時嚴禁在自己的電腦內模擬 USER 的正式環境，然而此時搭配公司資料庫*

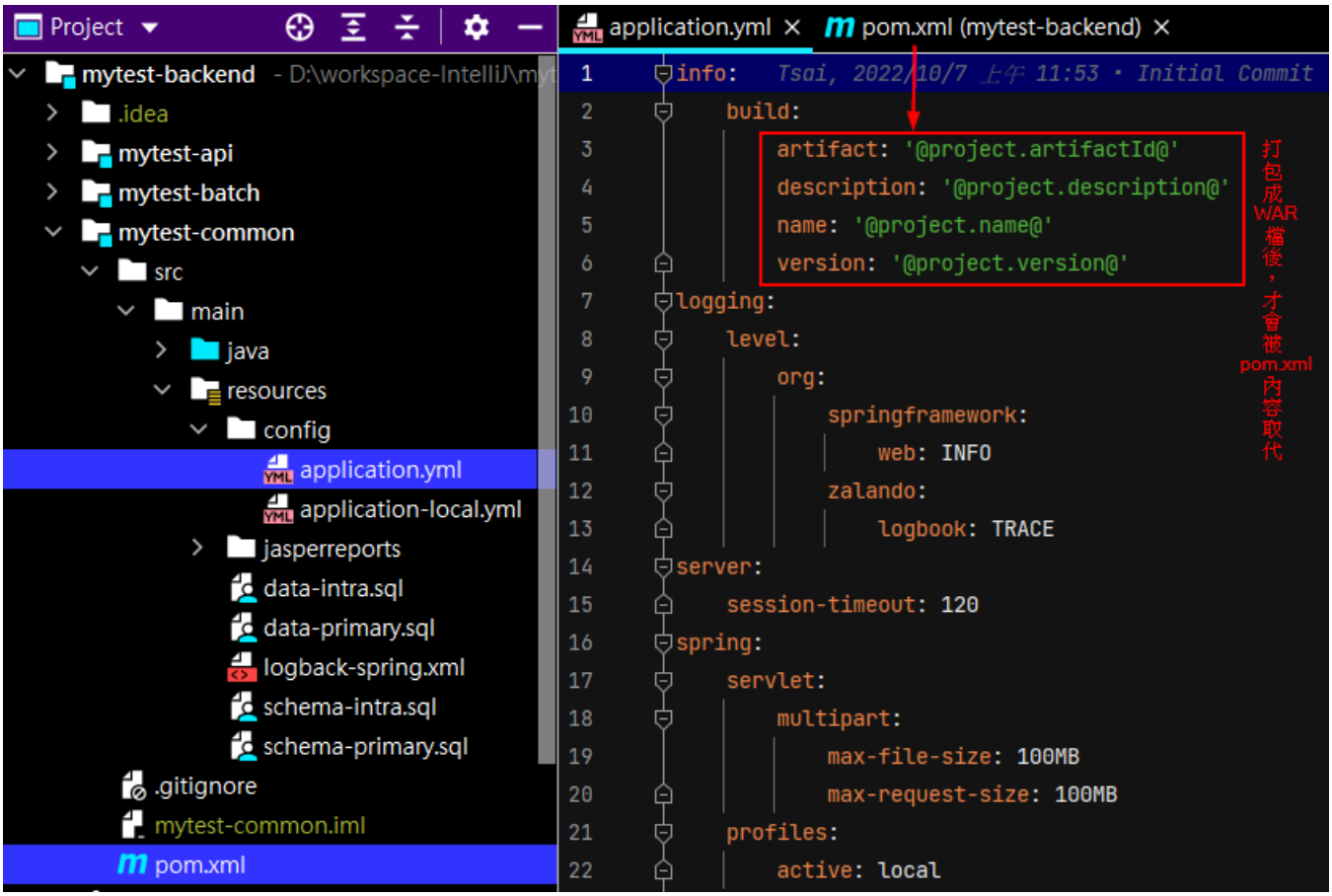
```yaml
# 方法 1
spring.profiles.active: uat

# 方法 2(建議)
spring:
  profiles:
    active: uat
```

application.yml

> *檔案位置 D:\workspace-IntelliJ\mytest-backend\mytest-common\src\main\resources\config\application.yml*

- 後端 API 包版資訊



- 上傳容量限制
- 指定套用哪一種配置文件
- 是否顯示資料庫 CRUD 的 SQL

application.yml

```yaml
spring:
    servlet:
        multipart:
            max-file-size: 100MB      修改單一檔案最大上傳容量限制
            max-request-size: 100MB  修改總上傳檔案，連同文字訊息，最多上傳容量限制
    profiles:
        active: local
    session:
        store-type: none
    jpa:
        # must set database to `default` in order to auto select different dialect for each of the data source
        database: default
        hibernate:
            naming:
                physical-strategy: org.hibernate.boot.model.naming.PhysicalNamingStrategyStandardImpl
        properties:
            hibernate:
            cache:
                use_query_cache: false
                use_second_level_cache: false
        show-sql: true
```

- 本機預設用 Spring Boot 內嵌式 Tomcat 來啟動專案所以此參數會影響 Tomcat 的預設值
- 在 SIT 或 UAT 則是用 JBoss 內 Undertow 來啟動專案然而此參數異動並不會影響 Undertow 的預設值

- Open API (Swagger) 文件自動生成
    - 預設不啟用 Swagger 來當作線上版 API 規格說明文件（文件位置 http://<hostname or ip>:<port>/swagger-ui.html）

application.yml

```yaml
swagger:
    # For security reason, it's better to explicitly enable on each non-production profile
    enabled: false
    forCodeGeneration: true
    hide-error-endpoint: true
    api-info:
        version: latest
    ui:
        deep-linking: true
        display-operation-id: false
        default-models-expand-depth: 1
        default-model-expand-depth: 1
        default-Model-rendering: EXAMPLE
        display-request-duration: true
        doc-expansion: NONE
        filter: false
        operations-sorter: ALPHA
        show-extensions: false
        tags-sorter: ALPHA
        supported-submit-methods: ["get", "put", "post", "delete", "options", "head", "patch", "trace"]
```

- HTTP 請求與回應日誌
    - 使用第三方套件 Logbook 攔截 Request、Response，後續會在由 RequestLoggingHandler 將收集到的資訊紀錄在資料表 MYTEST_LOG_TRACE

application.yml

```yaml
logbook:
    include:
      - /api/**
    filter:
        enabled: true
        form-request-mode: body
    strategy: default
    format:
        style: json
```

application-local.yml

> *檔案位置 D:\workspace-IntelliJ\mytest-backend\mytest-common\src\main\resources\config\application-local.yml*

- 在自己的電腦內模擬測試環境的 PORT

application-local.yml

```yaml
server:
  port: 8181
```

僅在本機端才會發揮作用

- 資料庫連線資訊（預設連線 2 個資料庫）

application-local.yml

```yaml
spring:
  datasource:
    driver-class-name: org.h2.Driver
    password: password
    jdbc-url: jdbc:h2:mem:primarydb;MODE=Oracle
    username: sa
    platform: primary
  intra-datasource:
    driver-class-name: org.h2.Driver
    password: password
    jdbc-url: jdbc:h2:mem:intradb;MODE=MSSQLServer
    username: sa
    platform: intra
  h2: 僅在 H2 資料庫才會發揮作用
    console:
      enabled: true  允許直接透過瀏覽器對資料庫資料進行CRUD操作
      settings.web-allow-others: true  允許其他人透過瀏覽器進行CRUD操作
```

- JSON Web Token (JWT) 身份驗證
  - JWT 內容的驗證金鑰
  - JWT 過期時間
  - JWT 簽發者

application-local.yml

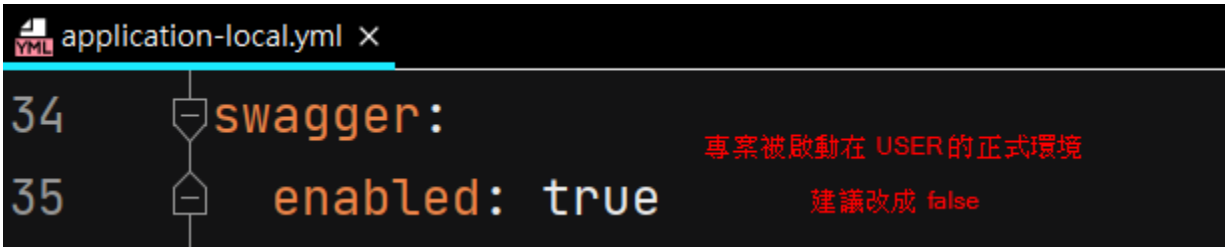
```yaml
jwt:
  secret: "5c5a1817b6caa4d5247571f8d120c7c89a8358de54c4039541ff9d8a88b80362"
  expiration: 86400
  refresh-expiration: 86400
  issuer: dtw.fbl
```

- Crowd SSO 身份驗證

application-local.yml

```yaml
crowd:
  hostname: ssouat.dtw.intranet
  port: 8095
  scheme: http
  appname: YOUR_CROWD_APP_NAME
  password: YOUR_CROWD_APP_PASSWORD
  is-mock: false
  cookie-sso-enabled: false
```

- 當spring.profiles.active 為【 local 】，啟用 Swagger 來當作線上版 API 規格說明文件（文件位置 http://<hostname or ip>:<port>/swagger-ui.html）

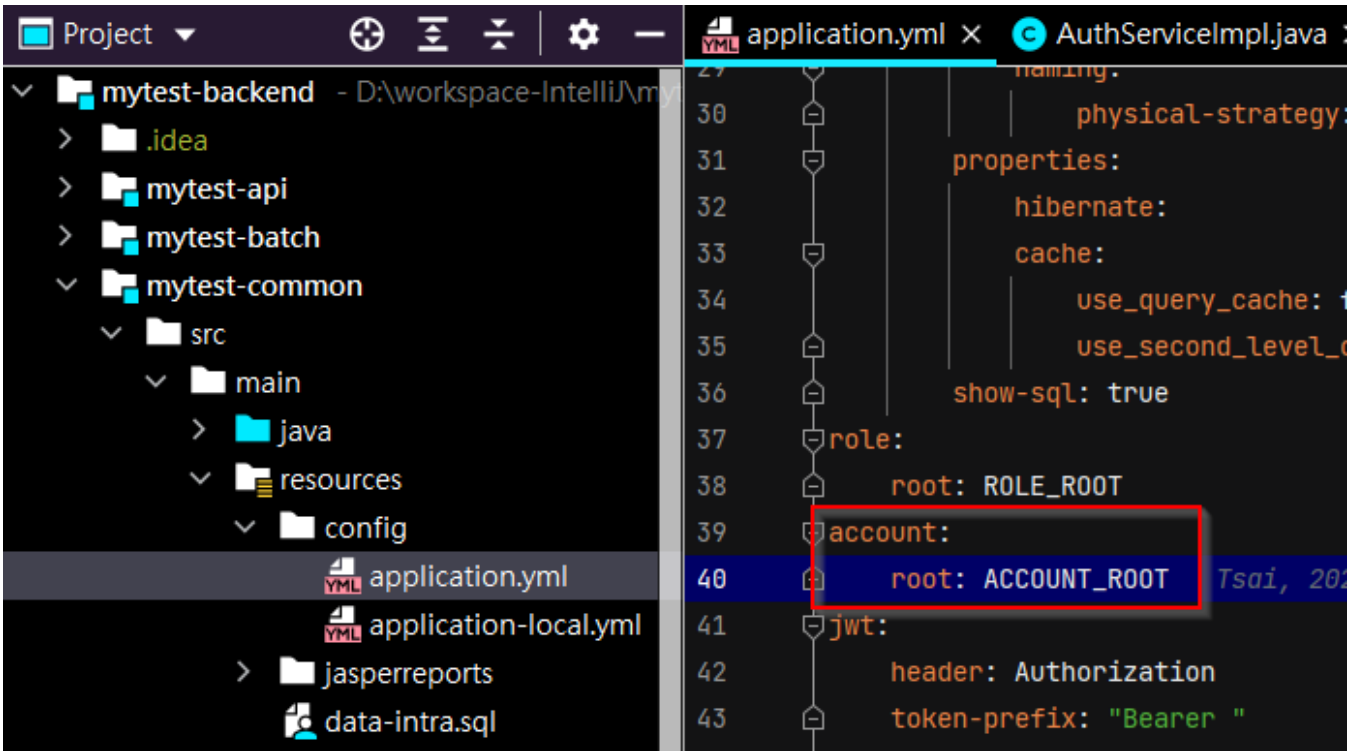


讀取來自配置文件內的參數設定值

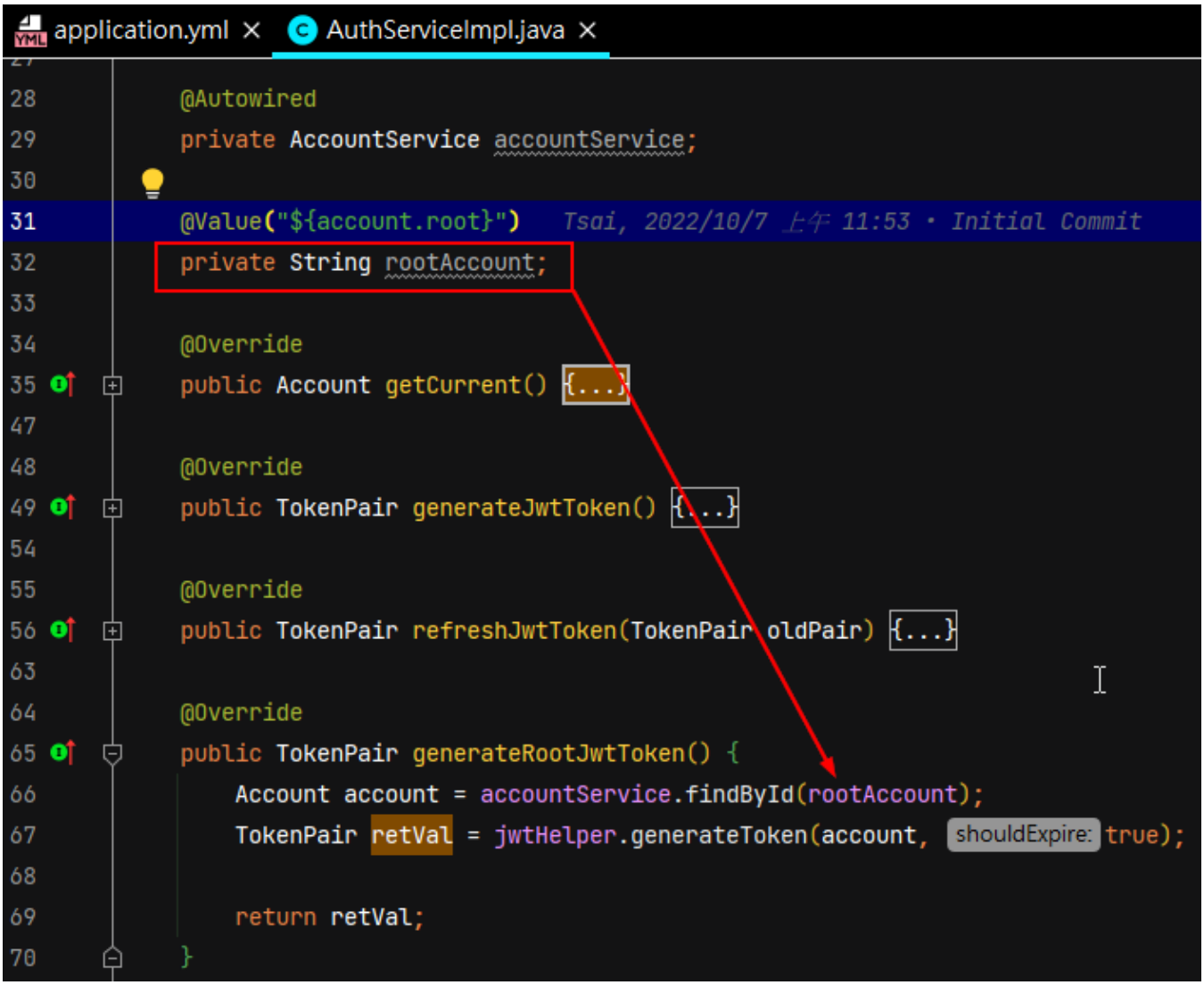
方法一：@Value

> *舉例說明*

- 配置文件


Project
mytest-backend - D:\workspace-IntelliJ\my
.idea
mytest-api
mytest-batch
mytest-common
src
main
java
resources
config
application.yml
application-local.yml
jasperreports
data-intra.sql
application.yml
AuthServiceImpl.java
physical-strategy:
properties:
hibernate:
cache:
use_query_cache:
use_second_level_
show-sql: true
role:
root: ROLE_ROOT
account:
root: ACCOUNT_ROOT
jwt:
header: Authorization
token-prefix: "Bearer "

- 讀取並使用

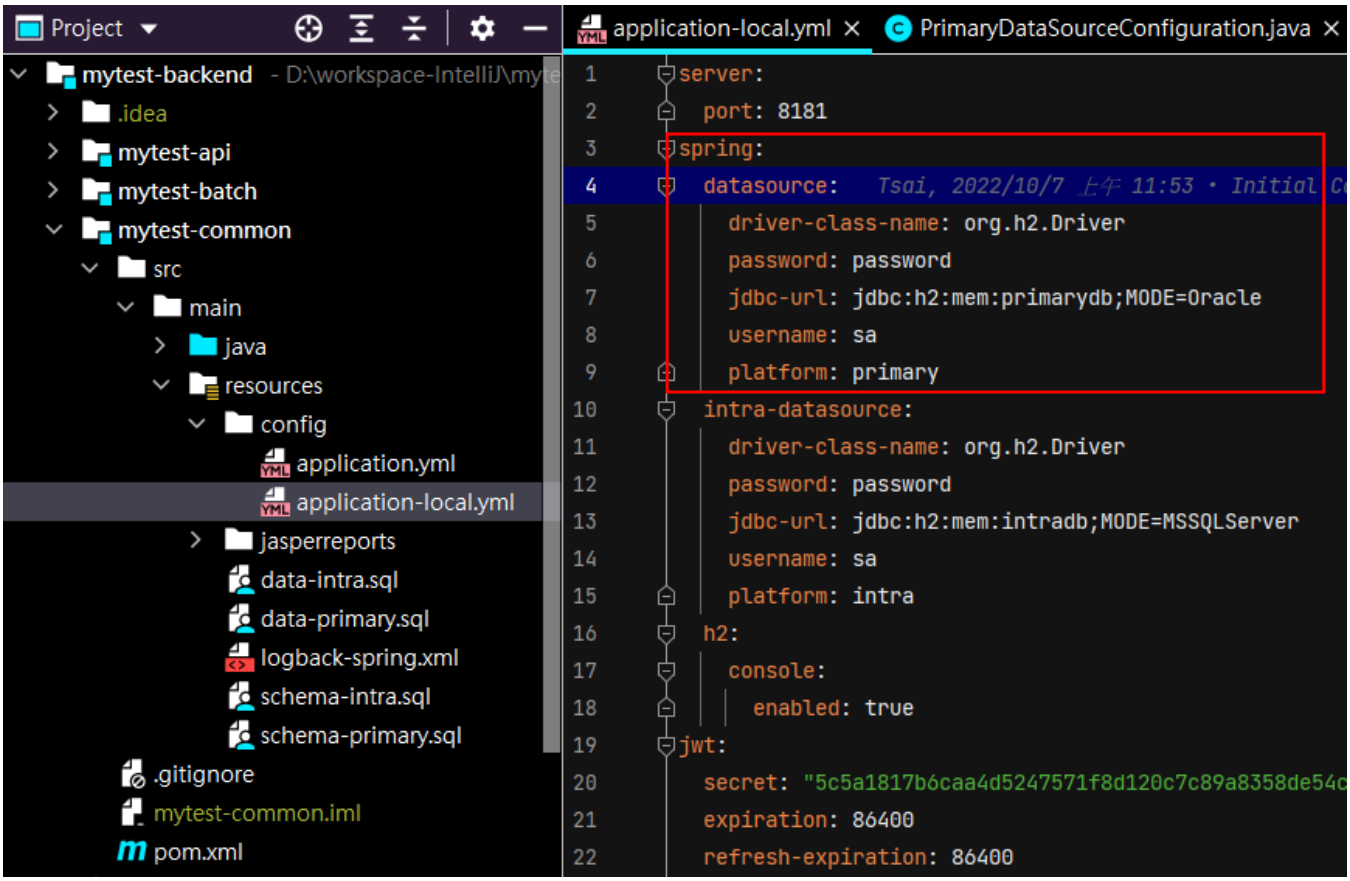


方法二：@ConfigurationProperties

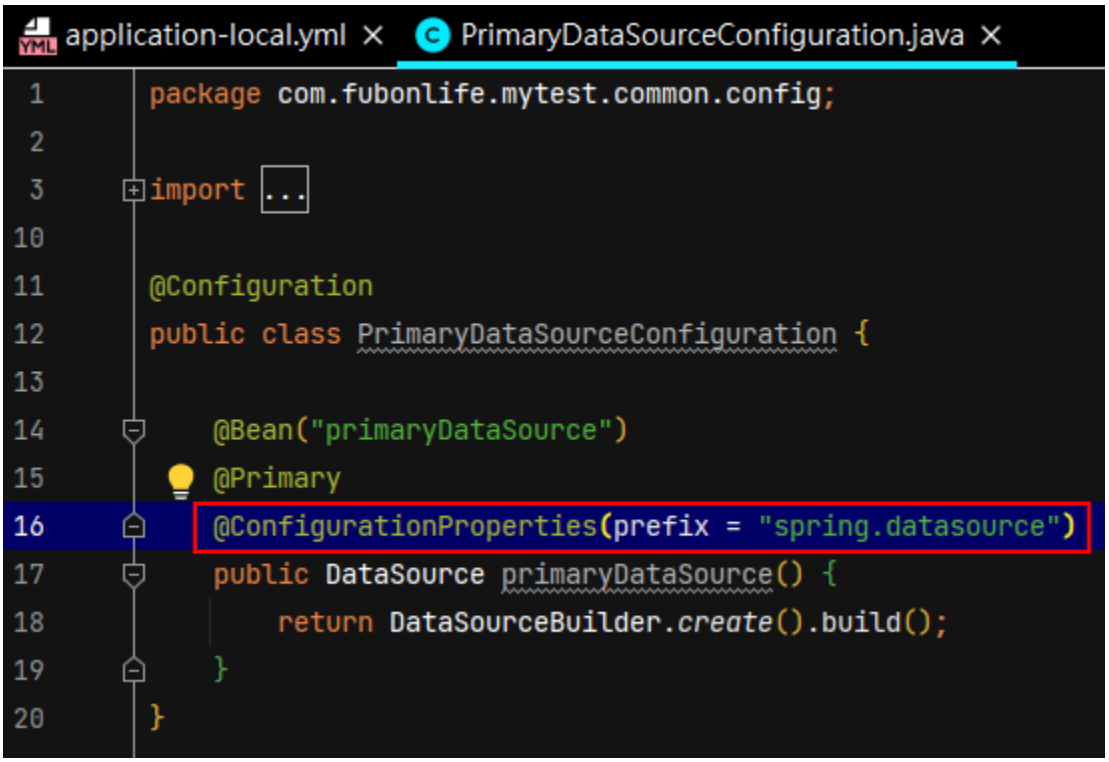
Configuration檔案建議放在子專案 xxx-common\src\main\java\com\fubonlife\mytest\common\config\OOOConfiguration.java

> *舉例說明－以【主要資料庫】連線設定值為例*

- 配置文件

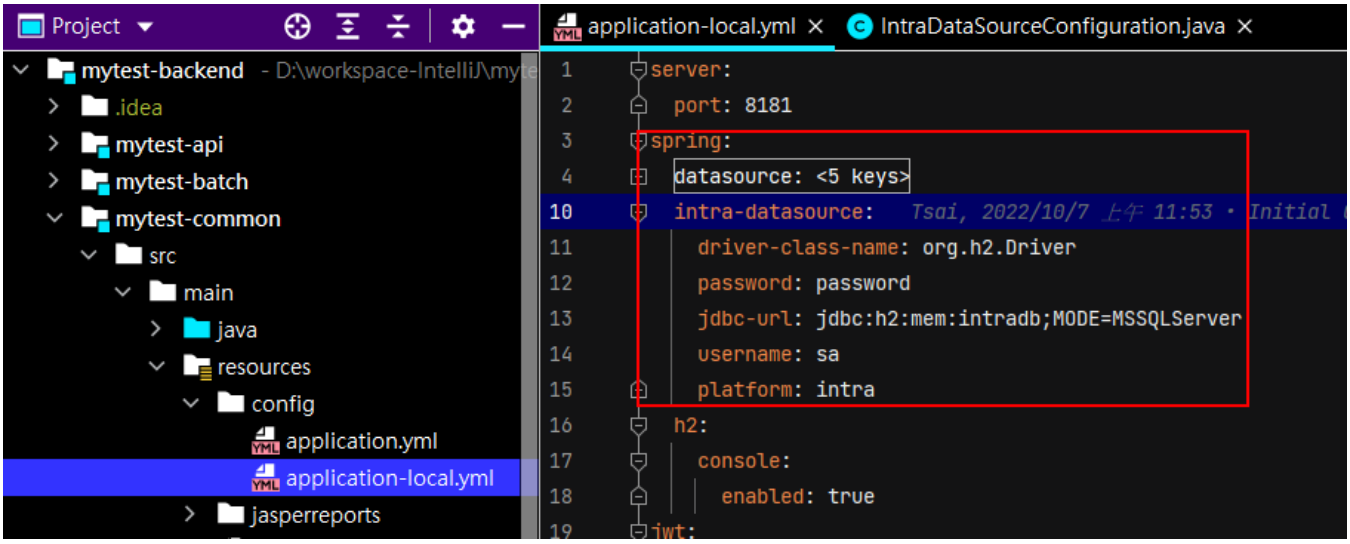


- 讀取並使用

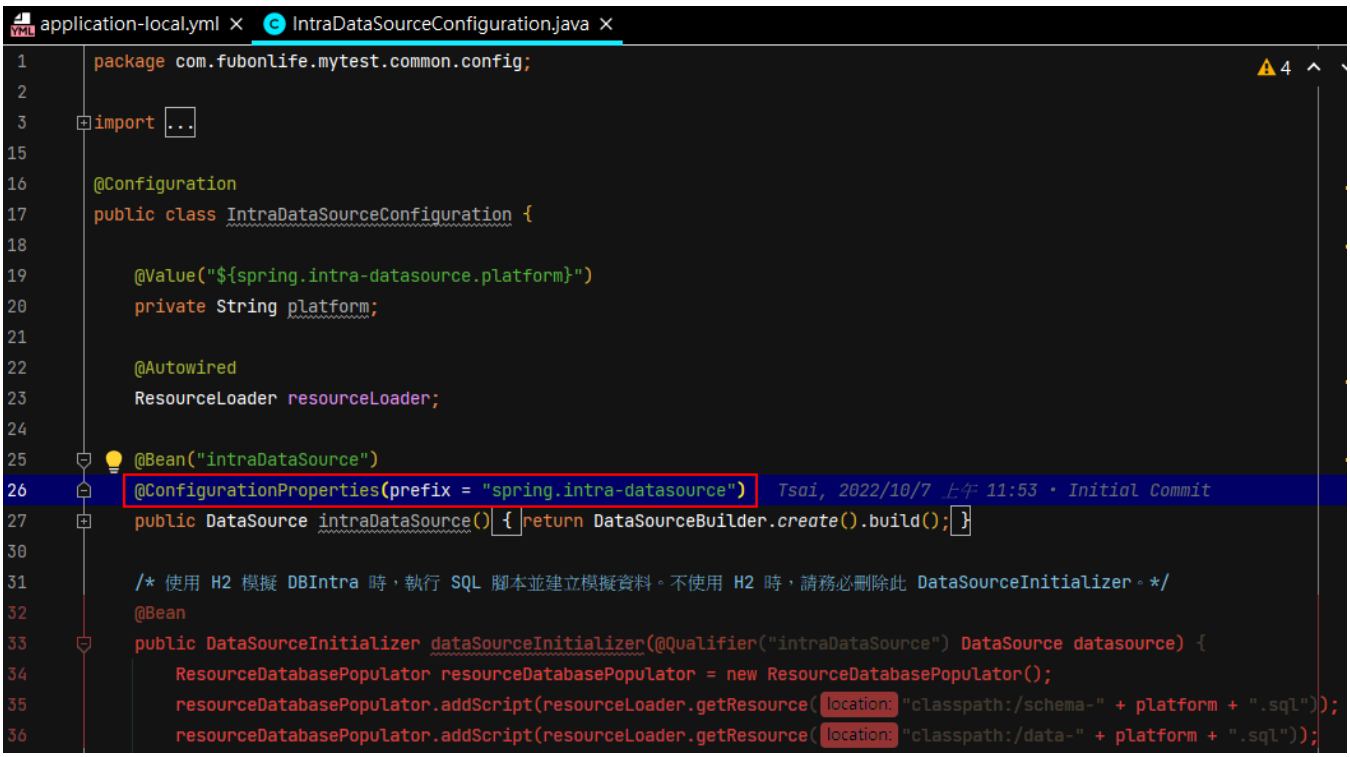


> *舉例說明－以【次要資料庫】連線設定值為例*

- 配置文件



- 讀取並使用



> 使用 H2 模擬 DBIntra 時，執行 SQL 腳本並建立模擬資料。
>
> 不使用 H2 時，請務必刪除此 DataSourceInitializer !!

> *舉例說明－客製化一個 DatasourceConnectionConfiguration 並註冊成 Bean*

- 配置文件

配置文件

```
spring.datasource.url=jdbc:mysql://127.0.0.1:7108/test?
useUnicode=false&autoReconnect=true&characterEncoding=utf-8
spring.datasource.username=sa
spring.datasource.password=b1384
spring.datasource.driver-class-name=com.microsoft.sqlserver.jdbc.SQLServerDriver
```

- 讀取

讀取

```
@Configuration
@ConfigurationProperties(prefix = "spring.datasource")
public class DatasourceConnectionConfiguration {

    private String url;
    private String username;
    private String password;
    private String driverClassName; //  driver-class-name,

    public String getUrl() {
        return url;
    }

    public void setUrl(String url) {
        this.url = url;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getDriverClassName() {
        return driverClassName;
    }

    public void setDriverClassName(String driverClassName) {
        this.driverClassName = driverClassName;
    }
}
```

- 透過 Spring 容器取得指定的 Configuration Bean 後，就能用了

使用

```
@RestController
@RequestMapping(value = "/api/b1384")
public class B1384Controller {

    @Autowired
    DatasourceConnectionConfiguration datasourceConnectionConfiguration;

    @PostMapping("/datasource-config/list")
    public Map<String, Object> listDatasourceConfigInB1384(){

        Map<String, Object> map = new HashMap<>();
        map.put("url", datasourceConnectionConfiguration.getUrl());
        map.put("userName", datasourceConnectionConfiguration.getUsername());
        map.put("password", datasourceConnectionConfiguration.getPassword());
        map.put("className", datasourceConnectionConfiguration.getDriverClassName());

        return map;
    }
}
```

方法三：@ConditionalOnProperty

> *舉例說明*

- 配置文件

配置文件

```
demo.enabled=false
```

- 讀取並使用

讀取並使用

```
@Configuration
@ConditionalOnProperty(
    prefix = "demo",        //
    name = {"enabled"},     //
    havingValue = "true",   //  havingValue TestConfigurationBean
    matchIfMissing = true   // TestConfigurationBean
)
public class TestConfiguration {

    // ....
}
```