
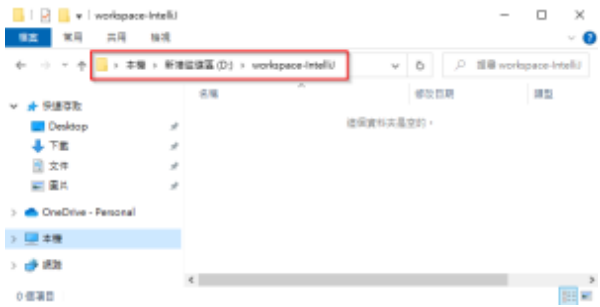
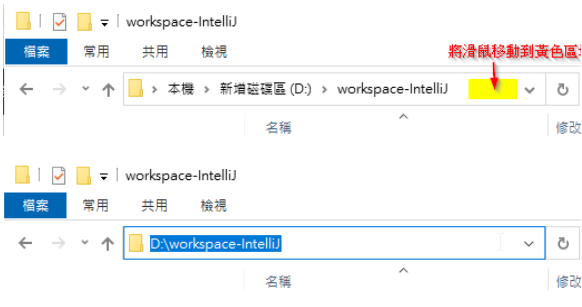

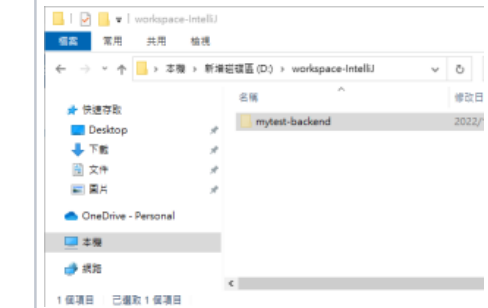
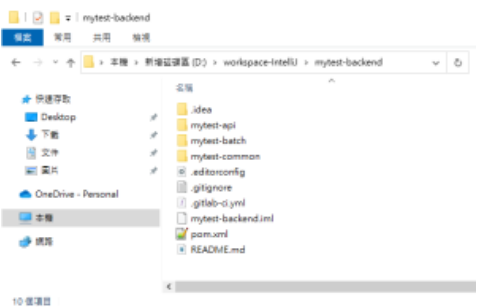


# 主題 4：使用後端框架產生器建立專案

使用 spring-boot-starter-fbl-archetype 產生器建立後端專案

步驟一	步驟二	步驟三
		
使用 [搜尋] 來尋找檔案總管	進入D槽的 workspace-IntelliJ 資料夾	1. 將滑鼠移動到黃色區域，接著用滑鼠左鍵點一下 2. 刪除藍底白字，接著在原處手動輸入 cmd 之後，按下鍵盤 [enter]

## 後端專案結構說明

步驟一	步驟二	步驟三	步驟四
			<p>02_後端框架說明 - S037 資訊架構管理科</p> <p>透過上述連結，瀏覽內文標題</p> <ul style="list-style-type: none"><li>專案結構</li><li>XXX-api 專案目錄</li><li>XXX-common 專案目錄</li><li>XXX-batch 專案目錄</li></ul>
使用 [搜尋] 來尋找檔案總管	進入D槽的 workspace-IntelliJ 資料夾	進入workspace-IntelliJ 的下一層資料夾 mytest-backend	主專案 mytest-backend，底下包含三個子專案

## 主專案結構

```
xxx-backend
├── mytest-api/      # 提供前端呼叫的Rest API實作的專案
├── mytest-batch/    # 提供批次功能實作的專案
├── mytest-common/   # 提供服務處理，資料庫存取，共用功能的專案
├── .editorconfig
├── .gitignore
├── .gitlab-ci.yml
├── pom.xml          # 主專案MAVEN設定檔
└── README.md        # README說明檔
```

子專案：mytest-api



```

1  src/                                     #
2      main/                               # maven
3      java/                               # java
4          com.fubonlife.mytest.common/    # Packagemaven generate(groupId.code.api)
5
6          config/                         # @Configuration
7
8          entity/                         # ORM(generator)
9
10         error/                          # Error
11
12         external/                       # API
13
14         model/                          # DTOAPI
15
16         pia/                            # PIA log
17
18         reqlog/                         # Request log
19
20         repository/                    # Jpa Repository
21
22         service/                        # @Service InterfaceimplementAPI
23             impl/                      # Service implement
24
25         util/                          # StaticService
26
27     resources/                          #
28
29         config/                        # Springboot
30
31         jasperreports/                 # jasperreports
32
33         logback-spring.xml             # springlogback
34
35         schema-intra.sql               # DBschema sql
36         schema-primary.sql
37
38         data-intra.sql                 # DBdata sql
39         data-primary.sql
40
41     test/                               # maven
42         java/
43             common/
44                 AbcTest.java           # Junit
45 pom.xml                               # MAVEN
46
47

```

## 認識第一支程式

### @SpringBootApplication

是Spring Boot專案的核心註解，目的是開啟自動配置



#### 鍵盤快捷鍵

同時按下 <Ctrl> <Shift> <F> 後，在輸入框內填入關鍵字「@SpringBootApplication」

檔案位置 *D:\workspace-IntelliJ\mytest-backend\mytest-api\src\main\java\com\fubonlife\mytest\api\app\Application.java*

```
17
18 @ComponentScan(basePackages = {"com.fubonlife.mytest.common", "com.fubonlife.mytest.api"})
19 @EnableFbldataBind
20 @EnableFbldataAsync
21 @EnableFbldataSwagger(basePackages = "com.fubonlife.mytest.api.controller")
22 @EnableFbldataCrowd
23 @EnableFbldataJwt
24 @EnableFbldataCors
25 @EnableFbldataSecurity
26 @EnableFbldataErrorHandle
27 @EnableFbldataPia
28 @EnableFbldataRequestLogging
29 @SpringBootApplication
30 @Slf4j
31 public class Application extends SpringBootServletInitializer {
32
33     @Override
34     protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {...}
35
36     public static void main(String[] args) {...}
37
38     Run 'Application.main()' Ctrl+Shift+F10
39     Debug 'Application.main()'
40 }
```

檔案位置 D:\workspace-IntelliJ\mytest-backend\mytest-batch\src\main\java\com\fubonlife\mytest\batch\app\Application.java

```
19 @EnableScheduling
20 @EnableFbldataBind
21 @EnableFbldataAsync
22 @EnableFbldataSecurity
23 @EnableFbldataJwt
24 @ComponentScan(basePackages = {"com.fubonlife.mytest.common", "com.fubonlife.mytest.batch"})
25 @SpringBootApplication
26 public class Application extends SpringBootServletInitializer {
27
28     @Override
29     protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {
30         return application.sources(Application.class);
31     }
32
33     public static void main(String[] args) {...}
34
35     Run 'Application.main()' Ctrl+Shift+F10
36     Debug 'Application.main()'
37 }
```

## @ComponentScan

透過掃描 package，也包含 child package，去檢查有什麼 class 需要被註冊為 Spring 的 Bean

1. 註冊為 Bean 的意思是 class 的實例化
2. Bean 生命週期及注入等過程，交由Spring容器(Spring IoC Container)管理

## Spring容器裡面的Bean

### 實例化

當 class 有包含以下標記時，才會被實例化

- @Configuration (通常會跟 @Bean 搭配使用)
- @Service
- @RestController (通常會跟 @RequestMapping 搭配使用)
- @Repository
- @Component

如何因應有多個相同 Class 類型的 Bean 要被實例化的情境

- 為每一個 Bean 指定一個獨一無二的名子
- 搭配使用標記 @Primary

## 注入

標記 `@Autowired`

- 讓 Spring 自動依照所需的 Class 類型，把需要的 Bean 從容器中找到出來，並注入給該變數。
- 必須要能找到 Bean，不然就要改成 `@Autowired(required=false)` 來避免錯誤發生。
- 若找到多個相同 Class 類型的 Bean，可能會發生 `NoUniqueBeanDefinitionException`，不然就要搭配 `@Qualifier( "Bean的名子" )` 來指定一個。

同一個 Spring 容器內，每一個 Bean 預設只存在一個實例

- 這對單執行緒的程式來說並不會有什麼問題，但對於多執行緒的程式，就必須注意到執行緒安全
- 程式在設計時，必須確保 Bean 是 Stateless 的

好文推薦

- [Bean 不是 Stateless 的窘境](#)
- [Are Spring objects thread safe ?](#)

## 初始化

標記 `@PostConstruct`

在這個Bean所有必要的屬性設定完成後才執行初始化的工作

```
@PostConstruct
private void init() {
    logger.info("Init method after properties are set : " + this.message);
}
```

## 銷毀前

標記 `@PreDestroy`

在這個Bean所在的容器被銷毀時執行

```
@PreDestroy
private void cleanUp() {
    logger.info("Spring Container is destroy! Calling clean up");
}
```