Java 進階應用相關

雖然Java本身已經有提供許多方法，但利用開源工具，可以減少一些邏輯判斷的撰寫，讓程式碼更加精簡。
以下會介紹幾個開源套件，處理的內容包含String、Map、物件映射

StringUtils

StringUtils 官方 API

```
<dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-lang3</artifactId>
</dependency>
```

Apache 提供的一個工具類 StringUtils，總共有兩種類別 `commons-lang` 、 `commons-lang3` ，後者才是目前有在維護更新的版本，StringUtils 比 Java 所提供的方法多出對 null 的處理，可以減少 `NullPointerException` 的產生，以下介紹一些相關 API 。

1. 確認字串是否存在或是否為空字串（ isEmpty 、 isBlank ）
   兩個方法有類似功能，只差在對於空白字串 " " 的處理，isEmpty 視為有內容，isBlank 則否。檢核必填欄位時可使用 isBlank，組成 SQL where 條件時判斷參數是否有值可用 isEmpty

   ```
   String isNull = null;
   String isEmpty = "";
   String isBlank = "    ";
   String isNotNull = "  123456789  ";

   StringUtils.isEmpty(isNull);    // true
   StringUtils.isEmpty(isEmpty);   // true
   StringUtils.isEmpty(isBlank);   // false
   StringUtils.isEmpty(isNotNull); // false

   StringUtils.isBlank(isNull);    // true
   StringUtils.isBlank(isEmpty);   // true
   StringUtils.isBlank(isBlank);   // true
   StringUtils.isBlank(isNotNull); // false
   ```

2. 去除前後指定字串（ trim 、 strip ）
   兩個方法有類似功能，差別在 trim 是直接去除前後空白 " " ， strip 是可以指定去除字串

```
String isNull = null;
String isEmpty = "";
String isBlank = "    ";
String isNotNull = "123456789    ";

StringUtils.trim(isNull);           // null
StringUtils.trim(isEmpty);          // ""
StringUtils.trim(isBlank);          // ""
StringUtils.trim(isNotNull);        // "123456789"

StringUtils.strip(isNull, "5");     // null
StringUtils.strip(isEmpty, "5");    // ""
StringUtils.strip(isBlank, "5");    // ""
StringUtils.strip(isNotNull, "1");  // "23456789    "
```

3. 字串比較（ equals / compare、startsWith / endsWith 、containsOnly / containsNone / containsAny、countMatches ）
   - 兩個字串內容比較：equals 比較是否相同，並回傳boolean值，compare 和 String.compareTo(String) 功能相同

```
StringUtils.equals("123", "1234");  // false
StringUtils.equals("123", "123");   // true
StringUtils.equals("123", null);    // false
StringUtils.compare("123", "1234"); // -1
StringUtils.compare("123", "123");  // 0
StringUtils.compare("123", null);   // 1
```

   - 是否有指定的字串作為開頭 startsWith 或結尾 endsWith

```
StringUtils.startsWith("123456789", "123"); // true
StringUtils.startsWith("123456789", "789"); // false
StringUtils.endsWith("123456789", "123");   // false
StringUtils.endsWith("123456789", "789");   // true
```

   - 是否存在：ContainsOnly 只出現一次，ContainsNone 完全沒出現，ContainsAny 可以放置多個參數，其中一個出現就好

```
StringUtils.contains("123 456 7 456 89", "456");        // true
StringUtils.containsOnly("123 456 7 456 89", "456");    // false
StringUtils.containsNone("123 456 7 456 89", "456");    // false
StringUtils.containsAny("123 456 7 456 89", "456", "7");// true
```

   - 計算字串符合次數

```
StringUtils.countMatches("aSasAsAsa","S");              // 1
```

4. 利用index處理文字（ indexOf / lastIndexOf、indexOfAny / lastIndexOfAny、indexOfAnyBut ）

- 查詢：indexOf 回傳從前面數來第一個遇到的index，lastIndexOf 回傳從後面數來第一個遇到的index；indexOfAny / lastIndexOfAny 可以比較多個字串是否有符合，差異在於從前面查找還是後面；indexOfAnyBut 從頭開始比對，回傳指定字串第一次沒有出現的 index

```
StringUtils.indexOf("1234567892", "2");                 // 1
StringUtils.lastIndexOf("1234567892", "2");             // 9
StringUtils.indexOfAny("1234567892", "345", "9");       // 2
StringUtils.lastIndexOfAny("1234567892", "345", "9");   // 8
StringUtils.indexOfAnyBut("1234567892", "12");          // 2
StringUtils.indexOfAnyBut("1234567892", "123");         // 3
StringUtils.indexOfAnyBut("1234567892", "234");         // 0
```

5. 擷取文字
   - 利用index擷取部分字串，substring 從指定index開始擷取，left 從左邊開始到指定index，right 指定index開始擷取右邊的文字，mid index開始到指定index結束

```
StringUtils.substring("123456789", 5);  // 6789
StringUtils.left("123456789", 5);       // 12345
StringUtils.right("123456789", 5);      // 56789
StringUtils.mid("123456789", 5, 8);     // 6789
```

   - 利用字串擷取部分字串，substringBefore 指定字串以前，substringAfter 指定字串以後，substringBetween 指定字串出現兩次中間的文字

```
StringUtils.substringBefore("123456789", "5");      // 1234
StringUtils.substringAfter("123456789", "5");       // 6789
StringUtils.substringBetween("123456789", "5");     // null
StringUtils.substringBetween("1234567895", "5");    // 6789
```

6. 切割文字

   split 可以分成一個、兩個或三個傳入參數的多載方法，一個參數預設以空白鍵切割字串，兩個或三個參數依據指定字串切割，三個參數還可以設定切割後陣列的元素數量

```
StringUtils.split(" 1234567 8 9 ");         // ["1234567","8","9"]
StringUtils.split("123456789", "5");        // ["1234","6789"]
StringUtils.split("123:456:789", ":", 2);   // ["123","456:789"]
StringUtils.split("1234563789", "3", 0);    // ["12","456","789"]
```

7. 刪除或取代
   - 刪除文字：remove

```
StringUtils.remove("123456789", "5");       // 12346789
```

   - 取代文字：replace 直接取代相對應文字，overlay 取代指定位置區塊的文字

```
StringUtils.replace("123456789", "345", "ZZZZZZ");  // 12ZZZZZZ6789
StringUtils.overlay("123456789", "ZZZZZZ", 3, 5);   // 123ZZZZZZ6789
```

- 刪除最後的字元：chomp 會刪除字串最後一個換行符號 `\n` 、 `\r` 、 `\r\n` ；chop 可刪除最後一個字元，當最後一個字元為換行符號，則刪除換行符號

```
StringUtils.chomp("123456789");         // "123456789"
StringUtils.chomp("123456789 \n \n");   // "123456789 \n "
StringUtils.chomp("123456789 \n \t");   // "123456789 \n \t"

StringUtils.chop("123456789");          // "12345678"
StringUtils.chop("123456789 \n  \n");   // "123456789 \n  "
StringUtils.chop("123456789 \n  \t");   // "123456789 \n  "
```

8. 增加字串
   - 沒有符合指定的前綴或後綴字串，就添加指定增加的字串，appendIfMissing 後綴，prependIfMissing 前綴

```
StringUtils.appendIfMissing("abcdef", "123", "def");  //abcdef
StringUtils.appendIfMissing("abcdef", "123", "xyz");  //abcdef123
StringUtils.prependIfMissing("abcdef", "123", "abc"); //abcdef
StringUtils.prependIfMissing("abcdef", "123", "xyz"); //123abcdef
```

   - 補空白：第二個參數指定字串長度，長度不足補空白，長度超過保持原樣，leftPad/rightPad 補左邊或右邊，center 左右均分

```
StringUtils.leftPad("123", 5);  // "  123"
StringUtils.rightPad("123", 5); // "123  "
StringUtils.center("123", 5);   // " 123 "
```

   - 複製：repeat 指定字串多次複製

```
StringUtils.repeat("123-", 5));  // "123-123-123-123-123-"
```

9. 大小寫切換
   - upperCase（全大寫） / lowerCase（全小寫） / swapCase（全部大小寫反轉） / capitalize（第一個字改大寫） / uncapitalize（第一個字改小寫）

```
StringUtils.upperCase("aSdFgHj");       // "ASDFGHJ"
StringUtils.lowerCase("aSdFgHj");       // "asdfghj"
StringUtils.swapCase("aSdFgHj");        // "AsDfGhJ"
StringUtils.capitalize("aSdFgHj");      // "ASdFgHj"
StringUtils.uncapitalize("ASdFgHj");    // "aSdFgHj"
```

MapUtils

MapUtils 官方 API

```
<dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-collections4</artifactId>
</dependency>
```

和 StringUtils 相同，也是 Apache 提供的一個工具類，主要用於處理 Map，以下主要介紹取得 Map 資料及判斷為空的方法。

1. 取得 Map 內資料
   在 Java 中要取得 Map key 的對應值，通常會寫下以下程式：

   ```
   Map<String, Object> map = new HashMap<>();
   map.put("data1", 1);
   map.put("data2", "2");
   String data3 = (String) map.get("data3");
   data3.trim();
   ```

   從上面的程式碼會發現根本沒有鍵值 `data3` ，那麼就有可能產生轉型或者 `NullPointerException` 錯誤發生，利用 MapUtils 內的方法可以避免類似錯誤。
   取得 Map 內資料相關方法都會先判斷 Map，若 Map 已經是 null ，則會直接返回，若 Map 不為 null ， get 結果為 null ，同樣返回 null 或預設值，整個相關方法可以分成兩大類。
   - 基本型別 (boolean、byte、double、float、int、long、short)
     四種方法：

	有預設值	**沒有預設值**
得到基本型別	getXXX(Map map, K key, XXX defaultValue)	getXXX(Map map, K key) (回傳null)
強轉為基本型別， 強轉失敗則回傳預設值	getXXXValue(Map map, K key, XXX defaultValue)	getXXXValue(Map map, K key) (回傳基本型別預設值)

```
Map<String, Object> map = new HashMap<>();
map.put("data1", 1);
map.put("data2", "2");
map.put("data3", "S");
MapUtils.getInteger(map, "data1");      // 1
MapUtils.getInteger(map, "data4");      // null
MapUtils.getInteger(map, "data2", 0);   // 2
MapUtils.getInteger(map, "data3", 0);   // 0
MapUtils.getIntValue(map, "data2"));    // 2
MapUtils.getIntValue(map, "data4");     // 0
MapUtils.getIntValue(map, "data3");     // 0
MapUtils.getIntValue(map, "data3", 0);  // 0
```

- 物件類別（String、Map、Object）
  兩種方法：

有預設值	沒有預設值
getXXX(Map map, K key, XXX defaultValue)	getXXX(Map map, K key) (回傳null)

```
Map<String, Object> map = new HashMap<>();
map.put("data1", 1);
map.put("data2", "2");
System.err.println(MapUtils.getString(map, "data1"));          // 1
System.err.println(MapUtils.getString(map, "data2"));          // 2
System.err.println(MapUtils.getString(map, "data3"));          // null
System.err.println(MapUtils.getString(map, "data1", "XXX"));   // 1
System.err.println(MapUtils.getString(map, "data2", "XXX"));   // 2
System.err.println(MapUtils.getString(map, "data3", "XXX"));   // XXX
```

2. 判定 Map 是否為空
   在 Java 中要判斷 Map 是否有建立或者是否為空，通常會寫下以下程式：
   必須要在前面做一個是否為 null 的判斷，否則有機會發生 `NullPointerException`

```
Map<String, Object> map = null;
if(map == null || map.isEmpty()){
    // Map為空
}
```

   但如果使用 MapUtils ，可以直接避免掉問題，直接幫我們判斷，另外還有判斷是否有值的方法可以使用。

```
Map<String, Object> map = null;
MapUtils.isEmpty(map);      //true
MapUtils.isNotEmpty(map);   //false
```

ObjectMapper

JSON（JavaScript Object Notation）是一個資料交換的格式並非程式語言，在此之前，工程師們如果要交換資料，都是以檔案為單位，使用起來沒有那麼方便。
JSON 儲存資料的方式與 JavaScript 物件相同，內容可以包含字串、數字、布林值、陣列、物件等資料格式。

JSON 格式的流行也讓各個程式語言對於處理 JSON 資料成為重要議題， Java 本身並沒有處理 JSON 的方法，必須要倚賴第三方套件，常見的套件有三種：Gson（Google）、Jackson（社群活躍度最高）、Json-lib（最早的第三方套件）。

github - FasterXML/jackson
ObjectMapper 官方 API

```
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
</dependency>
```

Jackson 是當前使用比較廣泛的，用來序列化和反序列化 JSON 的 Java 開源框架。Spring 的預設 JSON 解析器就是 Jackson。
優點：

- 所依賴的 jar 包較少，簡單易用
- 解析大的 json 檔案速度比較快
- 執行時佔用記憶體比較低，效能比較好
- 有靈活的 API，可以很容易進行擴充套件和定制

Jackson 包含了三包套件，jackson-core（核心）、jackson-annotations（註解）、jackson-databind（資料繫結）。

本次要介紹的 ObjectMapper 便是屬於 jackson-databind，ObjectMapper 提供了從基本 POJO 或 JsonNode 讀取和寫入 JSON ，以及用於執行轉換的相關功能。
以下介紹三種常見的轉換：

1. JSON 和字串
   在轉換前，除了需要 import 套件，還要準備 POJO ，範例準備了三種 POJO

```
// POJO
@Data
class BasicInfo {

    @JsonProperty("BranchId")
    private String branchId;

    @JsonProperty("BranchName")
    private String branchName;

}
//  POJO 多出一種屬性
@Data
class BasicInfo2 {

    @JsonProperty("BranchId")
    private String branchId;

    @JsonProperty("BranchName")
    private String branchName;

    @JsonProperty("CreateTimestamp")
    private Timestamp createTimestamp;

}
//  POJO 少一種屬性
@Data
class BasicInfo3 {

    @JsonProperty("BranchId")
    private String branchId;

}


ObjectMapper oMapper = new ObjectMapper();
try {
    String jsonString = "{\"BranchId\": \"00083\" ,\"BranchName\": \"XXXXX\" }";
    BasicInfo basicInfo = oMapper.readValue(jsonString, BasicInfo.class);
    // 映射結果：BasicInfo(branchId=00083, branchName=XXXXX)

    BasicInfo2 basicInfo2 = oMapper.readValue(jsonString, BasicInfo2.class);
    // 映射結果：BasicInfo2(branchId=00083, branchName=XXXXX, createTimestamp=null)

    BasicInfo3 basicInfo3 = oMapper.readValue(jsonString, BasicInfo3.class);
    // 轉換失敗：com.fasterxml.jackson.databind.exc.UnrecognizedPropertyException

} catch (JsonProcessingException e) {
    e.printStackTrace();
}
```

可以看到第三種在轉換時失敗了，因為 POJO 缺少了對應屬性，除了必須確保映射的物件不需要有對應屬性外，還可以使用兩個方法來解決這個問題，

- @JsonIgnoreProperties(ignoreUnknown = true)
  在 POJO 加上 `@JsonIgnoreProperties(ignoreUnknown = true)` ，在 Jackson 反序列話的時候，會忽略不存在的屬性。

```
//  POJO 少一種屬性
@Data
@JsonIgnoreProperties(ignoreUnknown = true)
class BasicInfo3 {

    @JsonProperty("BranchId")
    private String branchId;

}


ObjectMapper oMapper = new ObjectMapper();
try {
    String jsonString = "{\"BranchId\": \"00083\" ,\"BranchName\": \"XXXXX\" }";
    BasicInfo3 basicInfo3 = oMapper.readValue(jsonString, BasicInfo3.class);
    // 映射結果：BasicInfo3(branchId=00083)

} catch (JsonProcessingException e) {
    e.printStackTrace();
}
```

- ObjectMapper.configure(DeserializationFeature.FAIL_ON_UNKNOWN_PROPERTIES,false);
  在建立 objectMapper 後，設定相關資訊， `DeserializationFeature.FAIL_ON_UNKNOWN_PROPERTIES,false` 代表忽略沒有的屬性，只要使用該實例出來的 objectMapper

```
ObjectMapper oMapper = new ObjectMapper();
oMapper.configure(DeserializationFeature.FAIL_ON_UNKNOWN_PROPERTIES,false);
```

2. JSON 和集合、Map
   ObjectMapper 可以直接處理集合及 Map ，直接看範例：

```
try {
    List<BasicInfo> basicInfos = new ArrayList<>();
    basicInfos.add(new BasicInfo("1","1"));
    basicInfos.add(new BasicInfo("2","2"));
    basicInfos.add(new BasicInfo("3","3"));
    basicInfos.add(new BasicInfo("4","4"));

    Map<String, BasicInfo> map = new HashedMap<>();
    map.put("1", new BasicInfo("1","1"));

    ObjectMapper oMapper = new ObjectMapper();

    // 陣列
    String basicInfosString = oMapper.writeValueAsString(basicInfos);
    System.err.println(basicInfosString);
    // 輸出結果：[{"BranchId":"1","BranchName":"1"}, {"BranchId":"2","BranchName":"2"},
    //{"BranchId":"3","BranchName":"3"}, {"BranchId":"4","BranchName":"4"}]

    List<BasicInfo> basicInfosJSON = oMapper.readValue(basicInfosString, List.class);
    System.err.println(basicInfosJSON.toString());
    // 輸出結果：[{BranchId=1, BranchName=1}, {BranchId=2, BranchName=2},
    //{BranchId=3, BranchName=3}, {BranchId=4, BranchName=4}]

    // Map
    basicInfosString = oMapper.writeValueAsString(map);
    System.err.println(basicInfosString);
    // 輸出結果：{"1":{"BranchId":"1","BranchName":"1"}}

    Map<String,BasicInfo> map2 = oMapper.readValue(basicInfosString, Map.class);
    System.err.println(map2.toString());
    // 輸出結果：{1={BranchId=1, BranchName=1}}

} catch (JsonProcessingException e) {
    e.printStackTrace();
}
```

3. JSON 和日期

   ObjectMapper 還提供日期格式化的方式，有兩種設定方法：

   - 在建立 ObjectMapper 後，設定相關資訊， `oMapper.setDateFormat( SimpleDateFormat simpleDateFormat);`

```
ObjectMapper oMapper = new ObjectMapper();
oMapper.setDateFormat(new SimpleDateFormat("yyyy-mm"));
try {
    Timestamp timestamp = new Timestamp(System.currentTimeMillis());
    BasicInfo2 basicInfos = new BasicInfo2("1", "1", timestamp);
    String basicInfosString = oMapper.writeValueAsString(basicInfos);
    System.err.println(basicInfosString);
    // 輸出結果：{"BranchId":"1","BranchName":"1","CreateTimestamp":"2022-05"}

} catch (JsonProcessingException e) {
    e.printStackTrace();
}
```

- 利用 @JsonFormat

```
@Data
class BasicInfo {
    BasicInfo(String branchId, String branchName, Timestamp createTimestamp) {
        setBranchId(branchId);
        setBranchName(branchName);
        setCreateTimestamp(createTimestamp);
    }

    @JsonProperty("BranchId")
    private String branchId;

    @JsonProperty("BranchName")
    private String branchName;

    @JsonFormat(pattern = "yyyy-MM-dd", timezone = "UTC")
    @JsonProperty("CreateTimestamp")
    private Timestamp createTimestamp;

}
```

OrikaMapper

在 ObjectMapper 介紹時，介紹了 JSON 和物件間的轉換，但並沒有介紹到物件與物件之間對應的轉換，其實 ObjectMapper 中 convertValue(Object2Object) 也可以做到這一點，但ObjectMapper轉換時會先把物件變成字串，再變回物件，當轉換需求量過大時會影響效能，Spring 本身的 BeanUtils 內 BeanUtils.copyProperties(Object source, Object target) 也有效能的問題存在，為了解決這樣的問題，建議轉換數量大時，使用 OrikaMapper 作為轉換器。

Spring - Mapping with Orika

github - orika-mapper

orika-mapper

```
<dependency>
    <groupId>ma.glasnost.orika</groupId>
    <artifactId>orika-core</artifactId>
</dependency>
```

0. 說明之前
   先準備兩個 POJO，POJO 建立有幾個重點：

- 必須是 public class
- 必須實作 Serializable
- 必須有 getter / setter 方法，可以使用 lombok 的 @Data ，但要注意變數命名不能是 pName ，這樣在映射時，getPName/setPName會找不到，並報錯 “pName does not belong to DataDTO”
- 映射名稱是依照屬性變數名稱，如果有使用 Jackson ，要注意 @JsonProperty 和 OrikaMapper 映射完全沒有關係。

```
@Data
public class BasicInfoData implements Serializable {

    private static final long serialVersionUID = 1L;

    public BasicInfoData() {
    }

    public BasicInfoData(String branchId, String branchName) {
        setBranchId(branchId);
        setBranchName(branchName);
    }

    @JsonProperty("BranchId")
    private String branchId;

    @JsonProperty("BranchName")
    private String branchName;

    @Override
    public String toString() {
        return "BasicInfo [branchId=" + branchId + ", branchName=" + branchName + "]";
    }

}
```

```
@Data
public class BasicInfoData2 implements Serializable {

    private static final long serialVersionUID = 1L;

    public BasicInfoData2() {
    }

    public BasicInfoData2(String branchId, String branchName, String branchType) {
        setBranch(branchId);
        setBranchName(branchName);
        setBranchType(branchType);
    }

    @JsonProperty("BranchId")
    private String branch;

    @JsonProperty("BranchName")
    private String branchName;

    @JsonProperty("BranchType")
    private String branchType;

    @Override
    public String toString() {
        return "BasicInfo2 [branchId=" + branch + ", branchName="
             + branchName + ", branchType=" + branchType + "]";
    }

}
```

1. MapperFacade 和 MapperFactory
   簡單了解 MapperFacade 和 MapperFactory ，這兩個介面是 OrikaMapper 核心介面：

- MapperFactory 介面
  簡單來說，MapperFactory 像是一個轉換器，OrikaMapper 提供我們自定義映射方式，利用 Orika 實作類別 DefaultMapperFactory 。

```
MapperFactory mapperFactory = new DefaultMapperFactory.Builder().build();
// mapperFactory 設定
```

  舉一個簡單的例子，有時候可能兩個 POJO 的屬性名稱不相同，或者有想要忽略的，就可以在這個時候設定，例如：

```
// BasicInfoData 和 BasicInfoData2 在映射時不同屬性名稱的映射、排除屬性不做映射
mapperFactory.classMap(BasicInfoData.class, BasicInfoData2.class)
    .field("branchId", "branch")    // 名稱不一致屬性映射
    .exclude("branchName")          // 排除複製屬性
    .byDefault().register();
```

再複雜一點，也可以針對各種型別的轉換去做設定，這邊只提供簡單寫法：

```
MapperFactory mapperFactory = new DefaultMapperFactory.Builder().build();
ConverterFactory conveterFactory = mapperFactory.getConverterFactory();
conveterFactory.registerConverter( CustomConverter<S, D> customConverter); // S型別轉換成D型
```

- MapperFacade 介面
  MapperFacade 等同於前一個介紹的 ObjectMapper 或 Spring 的 BeanUtils，目的是去執行映射這件事，本身是介面，不能直接 new 出物件，只能透過 MapperFactory 建立：

```
MapperFacade mapper = mapperFactory.getMapperFacade();
```

  MapperFacade 建立完後，即可進行物件之間的映射：

```
BasicInfoData2 newDataBasicInfo2 = mapper.map(new BasicInfoData("1", "2"), BasicInfoData
```

- 整體程式碼：

```
public static void main(String[] args) {
    MapperFactory mapperFactory = new DefaultMapperFactory.Builder().build();
    mapperFactory.classMap(BasicInfoData.class, BasicInfoData2.class)
            .field("branchId", "branch")
            .exclude("branchName")
            .byDefault().register();
    MapperFacade mapper = mapperFactory.getMapperFacade();
    BasicInfoData2 newDataBasicInfo2 = mapper.map(new BasicInfoData("1", "2"), BasicInfoData
    System.err.println(newDataBasicInfo2);
    // 輸出結果：BasicInfo2 [branchId=1, branchName=null, branchType=null]
}
```

2. PropertyResolverStrategy
   除了上面提到的 MapperFactory 設定的方式，還可以指定映射時的策略，同樣在建立 MapperFactory 時去做設定：

```
MapperFactory factory = new DefaultMapperFactory.Builder()
    .propertyResolverStrategy(PropertyResolverStrategy propertyResolverStrategy).build();
```

`propertyResolverStrategy()` 內參數為指定策略實作的內容，必須要實作 PropertyResolverStrategy 介面。

3. OrikaMapperUtil

   最後是說明行內實作出的公用類別 `OrikaMapperUtil` ，除了專案 pom.xml 內加入 OrikaMapper，另外還有兩支檔案 OrikaMapperUtil.java 及 CapitalizePropertyResolver.java 放置在 `cub.cathaybk.util.mapper` 下，才能使用實作好的方法。

   方法介紹：首先 @Autowired OrikaMapperUtil

   ```
   @Autowired
   private OrikaMapperUtil mapper;
   ```

   - map：

     Object2Object

     ```
     List<BasicInfoData> dataList = BasicInfoData.findByEformId(eformId);
     List<BasicInfoData2> rtnList = new ArrayList<>();
     for (BasicInfoData data : dataList) {
         rtnList.add(mapper.map(data, BasicInfoData2.class));
     }
     ```

     Object2 小寫駝峰 Map

     ```
     Map<String, Object> fieldMap = mapper.map(req.getTranrq(), Map.class);
     ```

   - mapAsList：

     Collection2List，這樣的寫法和 Object2Object 的範例有同等效果。

     ```
     List<BasicInfoData> dataList = basicInfoDataRepository.findByEformId(eformId);
     List<BasicInfoData2>rtnList = mapper.mapAsList(dataList ,BasicInfoData2.class);
     ```

   - mapAsCapitalizeMap：

     Object2 大寫駝峰 Map

     傳入的 DTO/Entity/Object 中不可有基本型別(如：int, long 等)，且僅能轉換單層，第二層則會維持原先 Object 的型別，若有需要請自行轉換

     ```
     Map<String, Object> tranrqMap = mapper.mapAsCapitalizeMap(tranrq);
     ```

   - 補充：List<Map<String, Object>>轉為 Object，仍使用原先的 MapReflectUtil