

MapStruct

報告人：蔡東翰

大綱



大綱



- MapStruct是一款Java映射器工具，它可以自動生成Java類之間的映射代碼。
- 使用MapStruct可以減少手動編寫重複、冗長的映射代碼，提高開發效率和代碼質量。
- MapStruct支持基於注解的配置方式，使其易於使用和配置。
- MapStruct還提供了許多高級功能，例如條件映射、集合映射和注入等。
- 適用於各種Java應用程序，包括Spring和Java EE應用程序。

大綱



1. MapStruct的主要功能

- 自動化Java類之間的映射，從而減少手動編寫重複、冗長的映射代碼
- 支持基於注解的配置方式，使其易於使用和配置
- 提供了許多高級功能，例如條件映射、集合映射和注入等

2. MapStruct的核心概念

- **映射器**（ Mapper ）：定義Java類之間的映射關係
- 映射方法（ Mapping Method ）：定義具體的映射邏輯
- 映射選項（ Mapping Options ）：定義映射時的配置選項，例如忽略某些字段或使用自定義轉換器等

3. MapStruct的使用流程

- 定義映射器接口
- 在接口上添加@Mapper注解，配置映射器的選項
- 在接口中定義映射方法，使用@Mapping注解定義映射邏輯
- 編譯代碼，MapStruct會自動生成映射器實現類

4. MapStruct的優點

- 減少手動編寫映射代碼，提高開發效率和代碼質量
- 易於使用和配置，支持基於注解的配置方式
- 提供了許多高級功能，例如條件映射、集合映射和注入等
- 支持多種複雜映射場景，例如日期、枚舉和集合等

大綱



MapStruct的一些常用註釋及其用法(一)：

1. @Mapper

- 用於標記映射器接口，指示MapStruct自動生成映射器實現類
- 可以添加配置選項，例如映射策略、日期格式和命名規則等

2. @Mapping

- 用於定義映射方法中的具體映射關係
- 可以指定源字段和目標字段的名稱、類型和轉換器等
- 可以添加ignore屬性，忽略某些字段的映射

3. @Mappings

- 用於定義多個映射關係
- 可以用@Mapping元素指定具體映射關係
- 可以用@MappingConstants元素定義映射常量

MapStruct的一些常用註釋及其用法(二)：

4. @ValueMapping

- 用於定義枚舉類型的值映射
- 可以定義枚舉類型的值和對應的目標值

5. @InheritInverseConfiguration

- 用於定義反向映射
- 可以繼承目標類到源類的映射配置，從而自動實現反向映射

6. @Named

- 用於定義自定義轉換器的名稱
- 可以在映射方法中使用@Named來指定使用哪個自定義轉換器

大綱



映射

■ 在@Mapping中，可以以下參數指定條件

- source：指定源屬性名稱或表達式。
- target：指定目標屬性名稱。
- dateFormat：指定日期格式。
- numberFormat：指定數字格式。
- qualifiedBy：指定一個或多個限定符的類，以選擇與註釋一起使用的轉換器。
- nullValueCheckStrategy：指定如何檢查null值。
- defaultValue：指定默認值。
- expression：指定一個EL表達式來計算屬性值。
- condition：主要用於控制是否進行映射。

參數指定條件 範例

```
@Mapper
public interface EmployeeMapper {
    EmployeeMapper INSTANCE = Mappers.getMapper(EmployeeMapper.class);

    @Mapping(source = "salary", target = "salary", numberFormat = "$#.##")
    @Mapping(target = "isAdult", expression = "java(source.getAge() >= 18)")
    @Mapping(source = "address.street", target = "street" )
    @Mapping(source = "address.zipCode", target = "zipCode" defaultValue = "123")
    Employee personToEmployee(Person person);
}
```

- 1.指定數字格式
- 2.指定映射結果
- 3.無設定條件
- 4.預設映射結果

映射

定義源對象和目標對象的類型

```
java

public class Car {
    private String make;
    private int numberOfSeats;
    // getters and setters
}

public class CarDto {
    private String make;
    private int seatCount;
    // getters and setters
}
```

定義一個映射接口

```
java


@Mapper
public interface CarMapper {
    CarMapper INSTANCE = Mappers.getMapper(CarMapper.class);

    @Mapping(source = "numberOfSeats", target = "seatCount")
    CarDto carToCarDto(Car car);
}
```

在上述程式碼中，`@Mapper`註釋表示這是一個使用MapStruct進行對象映射的接口。CarMapper接口中定義了一個carToCarDto方法，該方法用於將Car對象映射為CarDto對象。在方法上使用@Mapping註釋指定源對象的屬性名稱和目標對象的屬性名稱之間的映射關係。

深度映射 (Deep Mapping)

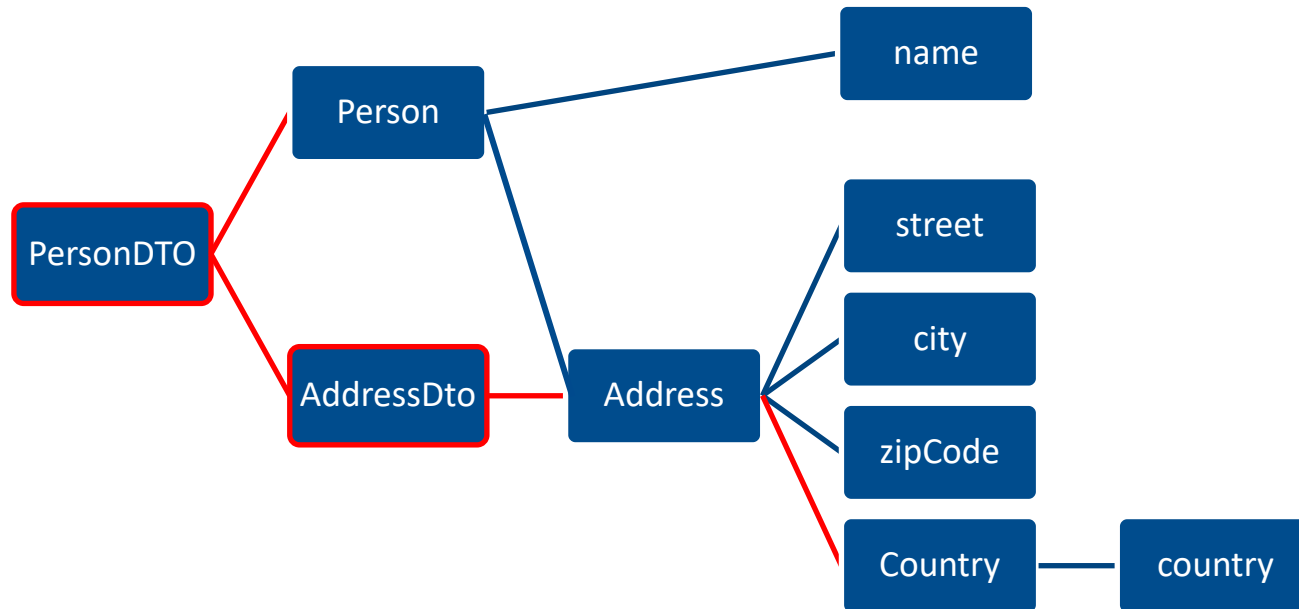
java

 Copy code

```
public class Person {  
    private String name;  
    private Address address;  
    // getter and setter methods  
}  
  
public class Address {  
    private String street;  
    private String city;  
    private String zipCode;  
    private Country country;  
    // getter and setter methods  
}  
  
public class Country {  
    private String name;  
    // getter and setter methods  
}
```

定義源對象

目標




藍色 原有資料

紅色 MapStruct 映射資料

開頭大寫 : DTO Class Entity

開頭小寫 : 資料內容

java


 Copy code

```
public class PersonDto {  
    private String name;  
    private AddressDto address;  
    // getter and setter methods  
}
```

定義目標對象的類型

```
public class AddressDto {  
    private String street;  
    private String city;  
    private String zipCode;  
    private String countryName;  
    // getter and setter methods  
}
```

java

 Copy code

定義一個映射介面

```
@Mapper
public interface PersonMapper {
    @Mapping(source = "address", target = "addressDto")
    PersonDto personToPersonDto(Person person);
}

@Mapper
public interface AddressMapper {
    @Mapping(source = "country.name", target = "countryName")
    AddressDto addressToAddressDto(Address address);
}
```

```
PersonMapper personMapper = Mappers.getMapper(PersonMapper.class);  
AddressMapper addressMapper = Mappers.getMapper(AddressMapper.class);
```

```
Address address = new Address();  
address.setStreet("123 Main St");  
address.setCity("Anytown");  
address.setZipCode("12345");  
Country country = new Country();  
country.setName("USA");  
address.setCountry(country);
```

```
Person person = new Person();  
person.setName("John Doe");  
person.setAddress(address);
```

```
PersonDto personDto = personMapper.personToPersonDto(person);
```

實際在程式中應用

條件映射

條件映射是指在執行對象映射時，可以根據某些條件來決定是否映射某些屬性，我們要將Person映射到PersonDto類，但只有當PersonDto的city屬性為空時，才映射Person的地址屬性。

```
@Mapper
public interface PersonMapper {
    @Mapping(target = "address", source = "address", condition = " (personDto.getCity() != null)")
    PersonDto personToPersonDto(Person person);
}
```

如何自定義映射選項

```
@Data
public class Person {
    private String name;
    private int age;
    private Address address;
}
```

```
@Data
public class PersonDto {
    private String name;
    private int age;
    private String city;
}
```

如何自定義映射選項

```
@Mapper
public interface PersonMapper {
    @Mapping(source = "address.city", target = "city", qualifiedByName = "toUpperCase")
    PersonDto personToPersonDto(Person person);

    @Named("toUpperCase")
    default String toUpperCase(String value) {
        return value.toUpperCase();
    }
}
```

自訂義映射過程中如何的特殊處裡(例:文字轉為大寫)

大綱



優點

1. 非常快速
2. 易于使用
3. 可配置性强
4. 可擴展性强
5. 支持構造函數注入

缺點

- 生成的代碼可能較長
- 學習曲線較陡峭
- 不支持複雜的轉換邏輯

Thank you for listening.