Bintang Glenn J

1506757535

Ringkasan Materi

Pada tutorial kali ini saya belajar lebih jauh mengenai cara berinteraksi dengan basis data menggunakan program serta cara melakukan debugging yang baik dengan menggunakan *library* Slf4j. Berinteraksi dengan basis data ternyata cukup sederhana dengan bantuan dari *library* MyBatis, yakni dengan menaruh *query* langsung pada anotasi yang diletakkan di atas *method.* Pada tutorial kali ini saya juga belajar untuk menggunakan objek sebagai parameter yang sangat berguna untuk menghindari penulisan parameter dengan RequestParam yang cukup melelahkan jika *fieldnya* banyak.

Jawaban Pertanyaan

1. Untuk melakukan validasi input yang opsional dan yang diperlukan tanpa menggunakan atribut *required* pada form dengan parameter berupa objek pada form POST, dapat digunakan javax.validation.

   Pertama, tambahkan *constraint* pada field yang diinginkan pada model. Contoh, misalkan field name tidak boleh null, maka dilakukan:

```
import javax.validation.constraints.NotNull;
```

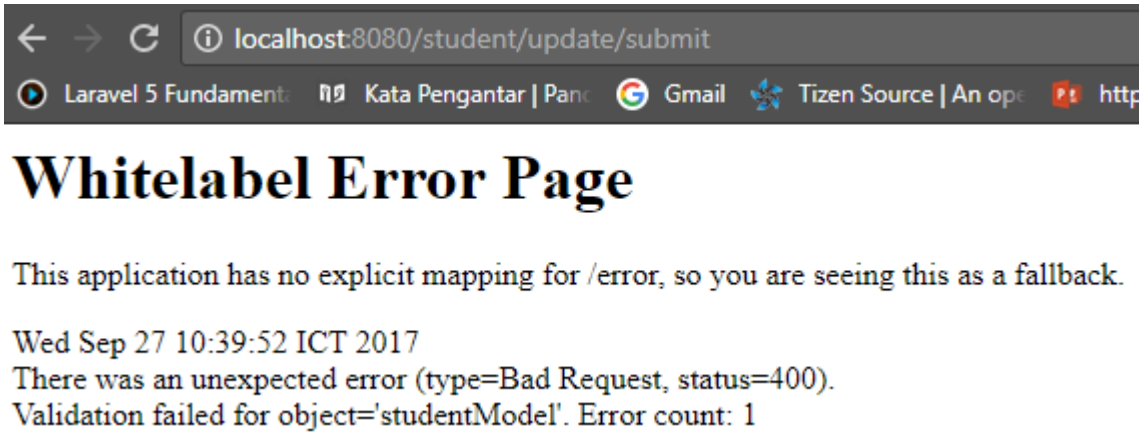
```
@NotNull
private String name;
```

   Kedua, tambahkan anotasi @Valid pada objek yang ingin dicek apakah sesuai *constraint* atau tidak. Contohnya, misalkan pada updateSubmit yang ada di tutorial:

```
@RequestMapping(value = "/student/update/submit", method = RequestMethod.POST)
public String updateSubmit (@Valid StudentModel student)
{
    studentDAO.updateStudent (student);

    return "success-update";
}
```

   Jika objek yang diterima tidak sesuai dengan ketentuan, maka program akan error. Contohnya, misalkan nama yang diupdate bernilai null, maka:



Error tersebut nantinya dapat dihandle lebih lanjut sesuai kebutuhan.

2. Form submit biasanya menggunakan POST method karena beberapa hal. Data yang disubmit biasanya mengandung data-data sensitif, sehingga akan lebih aman menggunakan POST karena data tidak ditempel pada URL. Selain itu, bila data yang dikirimkan besar maka jika menggunakan GET tidak akan cukup karena terdapat limitasi pada panjang URL. Dengan menggunakan POST, data juga tidak bisa dikirimkan langsung dengan mengetikkan URL, melainkan harus melalui form yang telah dibuat. Tidak ada perbedaan pada header atau body method jika menggunakan POST atau GET, hanya perlu menambahkan “method = RequestMethod.POST” pada RequestMapping.
3. Ya, mungkin. Cukup sebutkan request method apa saja yang ingin diterima pada RequestMapping. Contohnya bila ingin menerima POST dan GET, maka tambahkan “method = {RequestMethod.POST, RequestMethod.GET}”

Latihan Menambahkan Delete

Pertama-tama saya menambahkan baris yang diminta pada viewall.html:

```
<a th:href="'/student/delete/' + ${student.npm}">Delete Data</a>
```

Kemudian saya menambahkan method deleteStudent pada class StudentMapper:

```
@Delete("DELETE FROM student WHERE npm = #{npm}")
void deleteStudent(@Param("npm") String npm);
```

Query yang ada cukup jelas karena hanya mencari satu student kemudian dihapus dari basis data. Selanjutnya method tersebut akan dipanggil oleh method deleteStudent pada class StudentServiceDatabase:

```java
@Override
public void deleteStudent (String npm)
{
    log.info("student " + npm + " deleted");
    studentMapper.deleteStudent(npm);
}
```

Sebelum menuju ke method tersebut, dilakukan validasi terlebih dahulu di *controller*:

```java
@RequestMapping("/student/delete/{npm}")
public String delete (Model model, @PathVariable(value = "npm") String npm)
{
    StudentModel student = studentDAO.selectStudent (npm);

    if (student != null) {
        studentDAO.deleteStudent (npm);
        return "delete";
    } else {
        model.addAttribute ("npm", npm);
        return "not-found";
    }
}
```

Student terlebih dahulu dicek menggunakan method selectStudent, jika tidak ada maka akan menuju view not-found, sementara jika ada akan memanggil method sebelumnya kemudian menuju view delete.

Latihan Menambahkan Delete

Pertama-tama saya menambahkan method updateStudent pada class StudentMapper:

```java
@Update("UPDATE student SET name = #{name}, gpa = #{gpa} WHERE npm = #{npm}")
void updateStudent(StudentModel student);
```

Karena NPM tidak diganti, saya tidak perlu set nilai dari NPM kembali. Kemudian saya menambahkan method updateStudent pada interface StudentService:

```java
void updateStudent(StudentModel student);
```

Selanjutnya, method updateStudent pada class StudentMapper akan dipanggil pada method updateStudent yang ada pada class StudentServiceDatabase:

```java
@Override
public void updateStudent(StudentModel student)
{
    log.info("student " + student.getNpm() + " updated");
    studentMapper.updateStudent(student);
}
```

Lalu saya menambahkan link pada viewall.html agar dapat diakses melalui halaman viewall:

```
<a th:href="'/student/update/' + ${student.npm}">Update Data</a>
```

Saya kemudian membuat form-update.html yang hanya tinggal mengganti beberapa hal dari form-add.html:

```
<!DOCTYPE HTML>
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:th="http://www.thymeleaf.org">
<head>

<title>Update student</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>

<body>

    <h1 class="page-header">Problem Editor</h1>

    <form action="/student/update/submit" method="post">
        <div>
            <label for="npm">NPM</label> <input type="text" name="npm" readonly="true" th:value="${student.npm}" />
        </div>
        <div>
            <label for="name">Name</label> <input type="text" name="name" th:value="${student.name}" />
        </div>
        <div>
            <label for="gpa">GPA</label> <input type="text" name="gpa" th:value="${student.gpa}" />
        </div>

        <div>
            <button type="submit" name="action" value="save">Save</button>
        </div>
    </form>

</body>

</html>
```

Tambahan yang penting adalah penambahan readonly pada npm agar tidak bisa diedit. Saya juga menambahkan halaman success-update.html:

```
<html>
    <head>
        <title>Update</title>
    </head>
    <body>
        <h2>Data berhasil diupdate</h2>
    </body>
</html>
```

Sebelum menuju ke method tersebut, dilakukan validasi terlebih dahulu di *controller*:

```
@RequestMapping("/student/update/{npm}")
public String update (Model model, @PathVariable(value = "npm") String npm)
{
    StudentModel student = studentDAO.selectStudent (npm);

    if (student != null) {
        model.addAttribute("student", student);
        return "form-update";
    } else {
        model.addAttribute ("npm", npm);
        return "not-found";
    }
}
```

Student terlebih dahulu dicek menggunakan method selectStudent, jika tidak ada maka akan menuju view not-found, sementara jika ada akan memanggil method sebelumnya kemudian menuju view form-update. Kemudian saya membuat method updateSubmit pada class StudentController:

```java
@RequestMapping(value = "/student/update/submit", method = RequestMethod.POST)
public String updateSubmit (
        @RequestParam(value = "npm", required = false) String npm,
        @RequestParam(value = "name", required = false) String name,
        @RequestParam(value = "gpa", required = false) double gpa)
{
    StudentModel student = new StudentModel (npm, name, gpa);
    studentDAO.updateStudent (student);

    return "success-update";
}
```

Latihan Menambahkan Object sebagai Parameter

Pada latihan ini, saya hanya mengubah view form-update dan method updateSubmit pada controller. Pertama, saya menambahkan th:object pada tag form. Kemudian, saya mengubah th:value pada view menjadi th:field beserta dengan nilainya:

```html
<form action="/student/update/submit" method="post" th:object="${student}">
    <div>
        <label for="npm">NPM</label> <input type="text" name="npm" readonly="true" th:field="*{npm}" />
    </div>
    <div>
        <label for="name">Name</label> <input type="text" name="name" th:field="*{name}" />
    </div>
    <div>
        <label for="gpa">GPA</label> <input type="text" name="gpa" th:field="*{gpa}" />
    </div>

    <div>
        <button type="submit" name="action" value="save">Save</button>
    </div>
</form>
```

Setelahnya, saya hanya mengubah parameter dari method updateSubmit pada controller hanya menjadi StudentModel:

```java
@RequestMapping(value = "/student/update/submit", method = RequestMethod.POST)
public String updateSubmit (StudentModel student)
{
    studentDAO.updateStudent (student);

    return "success-update";
}
```

Data yang diterima otomatis merupakan objek dari StudentModel sehingga bisa langsung digunakan sebagai parameter updateStudent.