

# Spring MVC Web Form

# Table of Contents

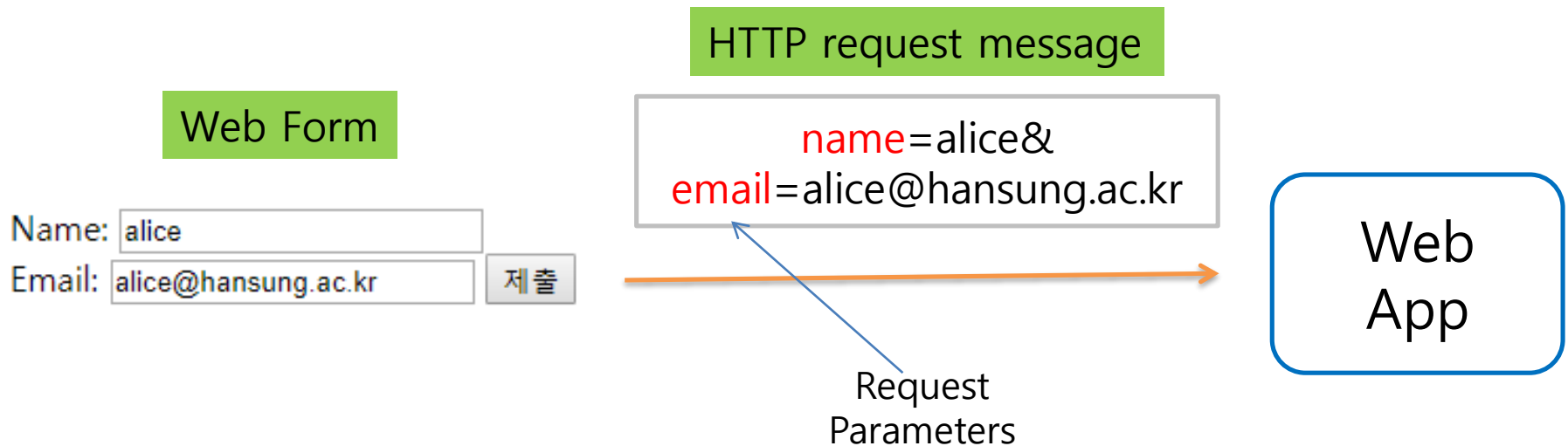
1. Web Basics
2. Data Binding
3. Data Validation
4. Data Buffering

# Web Basics

# Request Parameters

사용자가 입력한 값이 서버로 전달됨

- Request parameters are sent as part of the HTTP request message issued by the client



# Request Parameters

- Request parameters can be transferred as part of

2가지 방식이 있다

- The query string (in GET Method)
- The HTTP entity body (in POST Method)

# 1) Query String

GET OK!

- Request parameters are appended to the **query string**

HTTP request

GET /helloWeb/docreate?name=Alice&email=alice.hansung.ac.kr

Host: myserver.com

User-Agent: ...

Accept-Encoding: ...

query string

## 2) HTTP Entity body

POST 예시

- Request parameters are appended to the **body** of HTTP request message

HTTP request

```
POST /helloWeb/docreate HTTP/1.1
```

```
Host: myserver.com
```

```
User-Agent: ...
```

```
Accept-Encoding:
```

Body



```
name=Alice&email=alice@hansung.ac.kr
```

# Data Binding

사용자가 입력한 값을 객체에 넣는 것



# Problem

- How to move from the **request parameters** to the corresponding **object**?

POST 방식으로 전달받은 값을  
클래스에 넣고자 한다

```
POST /helloWeb/docreate HTTP/1.1
```

```
Host: myserver.com
```

```
User-Agent: ...
```

```
Accept-Encoding:
```

```
name=Alice&email=alice@hansung.ac.kr
```

```
public class Offer {
```

```
private String name;
```

```
private String email;
```

```
//getters and setters
```

```
}
```

# 1) Naive solution

클래스에 값을 수동적으로 넣는 방식

- The `@RequestParam` annotation binds request parameters to method parameters

Controller

```
@RequestMapping("/docreate")
public String doCreate(@RequestParam("name") String name,
                      @RequestParam("email") String email,
                      Model model) {
```

```
// we manually populate the Offer object with
// the data coming from the user
```

```
Offer offer = new Offer();
```

```
(offer.setName(name);
 offer.setEmail(email);
```

```
...
```

```
}
```

name 값을 name에 넣고 ...  
파싱 할 필요가 없어짐

## 2) Spring Data Binding

자동으로 넣는 방식

- Data Binding
  - The process of binding the request parameters to a "form bean" (also called command object) *form에서 오는 값을 자동으로 객체에...*
  - Data coming from the form could be automatically bound to an object

# Spring Data Binding

- All we need to do is to declare an object as a method parameter

```
@RequestMapping(value="/docreate", method=RequestMethod.POST)
```

```
    public String doCreate(Offer offer) {
```

*form bean* ← 주소 넣는다

```
    // offer object will be automatically populated
```

```
    // with request parameters 사용자 입력값
```

```
    }
```

우리는 객체 선언만 하면 된다. 스프링은 객체를 만들고,

사용자 입력값을 객체에 채우고, 주소 넣는다

---

아파라는 모델안에 추가되고, 이러면 리퀘스트 파라미터 값이 자동으로 전달된다

# Spring Data Binding

12p 정리

- The following sequence of operations occurs

- A new form bean is **instantiated**
- The form bean is **populated** from the request parameters
- The form bean is **added** to the model

1. form bean 이 인스턴스화 됨

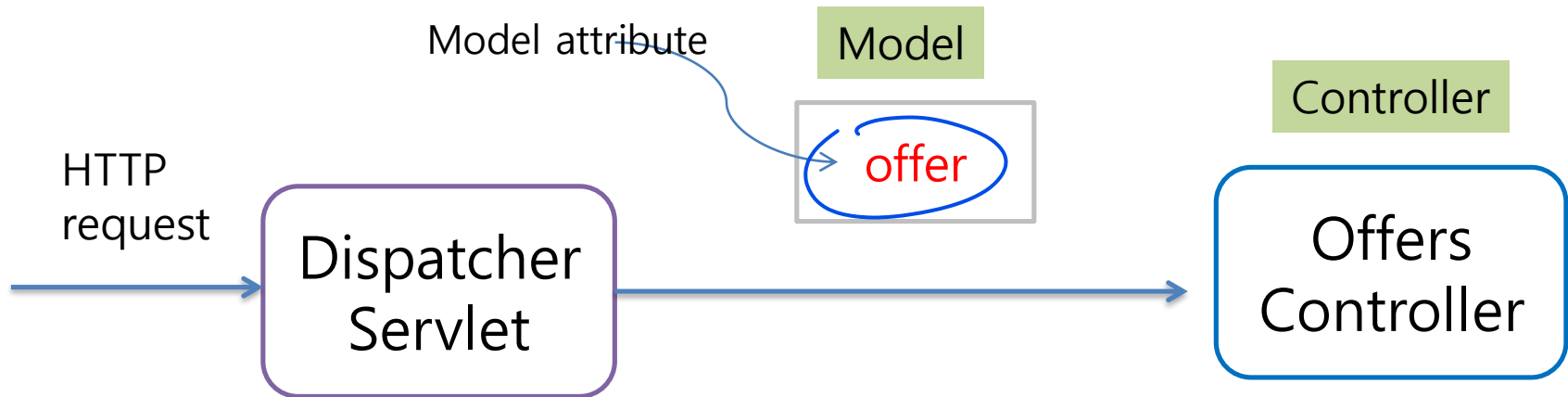
2. form bean 이 채워짐

3. form bean 이 모델에 추가됨

# Spring Data Binding

12p 그림

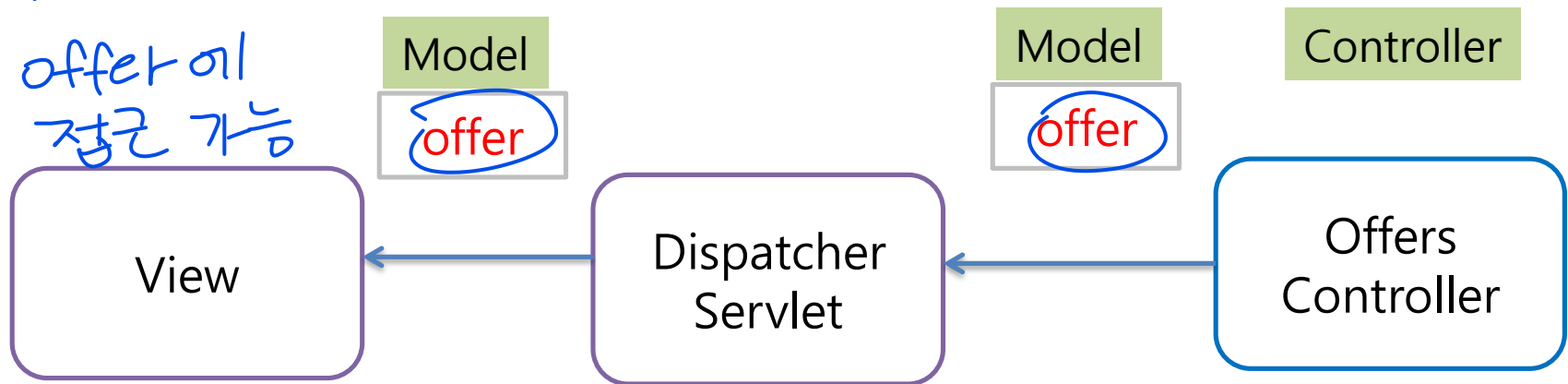
- The 'offer' form bean will be **automatically added** to the model
- Form bean is a model attribute



# Spring Data Binding

- Views can access and render the form bean content

View 도  
offer의  
접근 가능



# Spring Data Binding

## View

```
<html>
<head>
    <title>Thanks</title>
</head>

<body> Hi, ${offer.name}.
        You have successfully registered. <br/>
</body>

</html>
```



# Data Validation

스프링의 데이터 검증,  
보안에 필요하다.

# 0. Users make mistakes

- Users will accidentally fill it out with invalid information
  - e.g., invalid email format

Name: stacy

Email: stacyhansung.ac.kr

Your offer: I will teach you how to play golf

Create advert

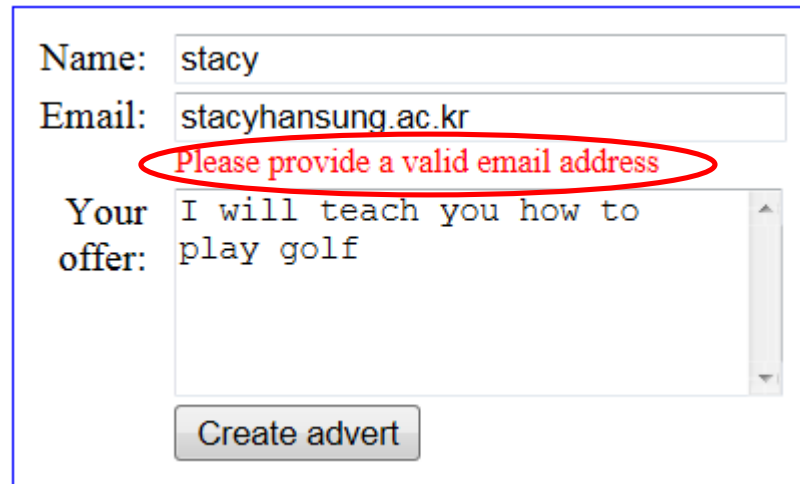
여기

invalid

올바르지 않은  
이메일 주소

# Users make mistakes

- When this happens, we want to explain the error and help the users to overcome it



The image shows a web form for creating an advertisement. It contains three input fields: 'Name' with the value 'stacy', 'Email' with the value 'stacyhansung.ac.kr', and 'Your offer' with the value 'I will teach you how to play golf'. A red oval highlights the error message 'Please provide a valid email address' which is displayed below the email field. At the bottom of the form is a 'Create advert' button.

Name: stacy

Email: stacyhansung.ac.kr

Please provide a valid email address

Your offer: I will teach you how to play golf

Create advert

# 1. Hibernate Validator


검증 라이브러리

- To detect user's errors, we need to **validate** the form data that are encapsulated in the form bean
- The Bean Validation API (**JSR-303**) is a specification that defines an API for JavaBean validation
  - It is possible to annotate bean properties with declarative validation constraints
  - @NotNull, @Pattern, @Size 이런걸로 constraints를 지정해주는 거임

<https://docs.oracle.com/javaee/7/api/javax/validation/constraints/package-summary.html>

# Hibernate Validator

- We are going to use **Hibernate Validator**, which is the reference implementation of the JSR-303 specification
- Hibernate Validator provides some custom annotations (e.g., @Email)
- Library 추가-해줘야 함



```
<dependency>  
    <groupId>org.hibernate</groupId>  
    <artifactId>hibernate-validator</artifactId>  
    <version>5.2.4.Final</version>  
</dependency>
```

# Hibernate Validator

```
public class Offer {
```

```
    private int id;
```

```
    @Size(min=5, max=100)
```

```
    @Pattern(regexp="^[A-Z]{1}[a-z]+$")
```

```
    private String name;
```

```
    @NotEmpty
```

```
    @Email
```

```
    private String email;
```

```
    ...
```

```
}
```

Start with a capital letter and have at least one additional lowercase letter

^는 라인의 시작,

\$는 라인의 끝,

문자열의 시작은 대문자,

하나이상의 소문자 필요

{1}는 한 글자

+는 여러 글자

# (참고) Regular Expression

- Standard and well-documented way of matching text

<code>[abc]</code>	A single character of: a, b, or c
<code>[^abc]</code>	Any single character except: a, b, or c
<code>[a-z]</code>	Any single character in the range a-z
<code>[a-zA-Z]</code>	Any single character in the range a-z or A-Z
<code>^</code>	Start of line
<code>\$</code>	End of line
<code>\A</code>	Start of string
<code>\z</code>	End of string

<code>(...)</code>	Capture everything enclosed
<code>(a b)</code>	a or b
<code>a?</code>	Zero or one of a
<code>a*</code>	Zero or more of a
<code>a+</code>	One or more of a
<code>a{3}</code>	Exactly 3 of a
<code>a{3,}</code>	3 or more of a
<code>a{3,6}</code>	Between 3 and 6 of a

<code>.</code>	Any single character
<code>\s</code>	Any whitespace character
<code>\S</code>	Any non-whitespace character
<code>\d</code>	Any digit
<code>\D</code>	Any non-digit
<code>\w</code>	Any word character (letter, number, underscore)
<code>\W</code>	Any non-word character
<code>\b</code>	Any word boundary

Helpful tool for editing regular expressions and making sure that the expression is correct

<http://www.rubular.com/>



## 2. Message Interpolation

- Message interpolation is the process of creating **error messages** for violated Bean Validation constraints

짧은 경고문을  
표시하고자 함

Name:

Name must be between 5 and 100 characters

Email:

the email address cannot be empty

Your offer:

Text must be between 5 and 100 characters

Create advert

# Message Interpolation

- We can define the **message descriptor** of each property through the message attribute

```
public class Offer {
```

```
    private int id;
```

여기에 쓰여진 문  
↓

```
@Size(min=5, max=100, message="Name must be between 5 and 100 characters")  
    private String name;
```

```
@Email(message="please provide a valid email address")
```

```
@NotEmpty(message="the email address cannot be empty")
```

```
    private String email;
```

```
@Size(min=5, max=100, message="Text must be between 5 and 100 characters")  
    private String text;
```

```
}
```

### 3. Validating Object

- Validation is achieved through the **@Valid** annotation
- The @Valid annotation causes the object to be first validated and then added to the model

```
@RequestMapping(...)
```

```
public String doCreate(@Valid Offer offer { ... }
```

↑  
이것만 붙이면 스프링에게  
검증을 해달라고 요구함

# Validating Object

- The handler method may ask for a **BindingResult** object, which represent the result of the validation process

```
@RequestMapping(...)
```

```
public String doCreate(@Valid Offer offer, BindingResult result)  
{ ... }
```

검증 결과가  
result에 들어간다  
↓

# Validating Object

- We can then **inspect** the BindingResult object for possible validation errors

```
@RequestMapping(...)
```

```
public String doCreate(@Valid Offer offer, BindingResult result) {  
    if(result.hasErrors()) {  
        List<ObjectError> errors = result.getAllErrors();  
  
        for(ObjectError error:errors) {  
            System.out.println(error.getDefaultMessage());  
        }  
        return "createoffer";  
    }  
    ...  
}
```

result 가 들어가서 만약 에러 있으면  
에러를 출력

# Data Buffering

# Why Data Buffering?

- The user forgets to provide a mandatory field? Does she have to **re-type** everything from scratch?

실수로 비번 확인 안했는데, 그것 때문에 에러나서  
다 날아가면 안되니까, 이것을 해결해보고자 한다

# 1. Spring form tag library

- We need to bind its content to the Web form
- To deal with prepopulated form beans, Spring provides a set of **data binding-aware tags**

입력 받은 값들이 있는 객체를 다시 표시

Client: Web Form

Server: Controller

Name:

Email:

Your offer:

Create advert

```
public class Offer {
```

```
    private int id;
```

```
    private String name;
```

```
    private String email;
```

```
    private String text;
```

```
    ...  
}
```



# Spring form tag library

- To use the tags from the 'spring form tag library', the following directive needs to be added at the top of the JSP page:

sf는 나 다음대로 써드 된  
↓

```
<%@ taglib prefix="sf"
            uri="http://www.springframework.org/tags/form"%>
```

32p 를 하기 위해서는 새로운 태그를 써야 한디-

# Spring form tags

- The `<sf:input>`, `<sf:password>` and `<sf:checkbox>` tags are **data-binding versions** of the corresponding HTML elements

form 이 아니라 sf: form 처럼 바뀌어서 써야함

Spring form tag lib	HTML
<code>&lt;sf:form&gt;</code>	<code>&lt;form&gt;</code>
<code>&lt;sf:input&gt;</code>	<code>&lt;input type="text"&gt;</code>
<code>&lt;sf:password&gt;</code>	<code>&lt;input type="password"&gt;</code>
<code>&lt;sf:checkbox&gt;</code>	<code>&lt;input type="checkbox"&gt;</code>

코드 예시

## 2. Revised JSP (createoffer.jsp)

```
<%@ taglib prefix="sf" uri="http://www.springframework.org/tags/form"%>
...
<body>

<sf:form method="post"
action="${pageContext.request.contextPath}/docreate" modelAttribute="offer">

    <table class="formtable">
    <tr>
        <td class="label">Name:</td>
        <td><sf:input class="control" path="name"/></td>
    </tr>
    ...
</sf:form>
</table>

</body>
</html>
```

① Object

② Object Property

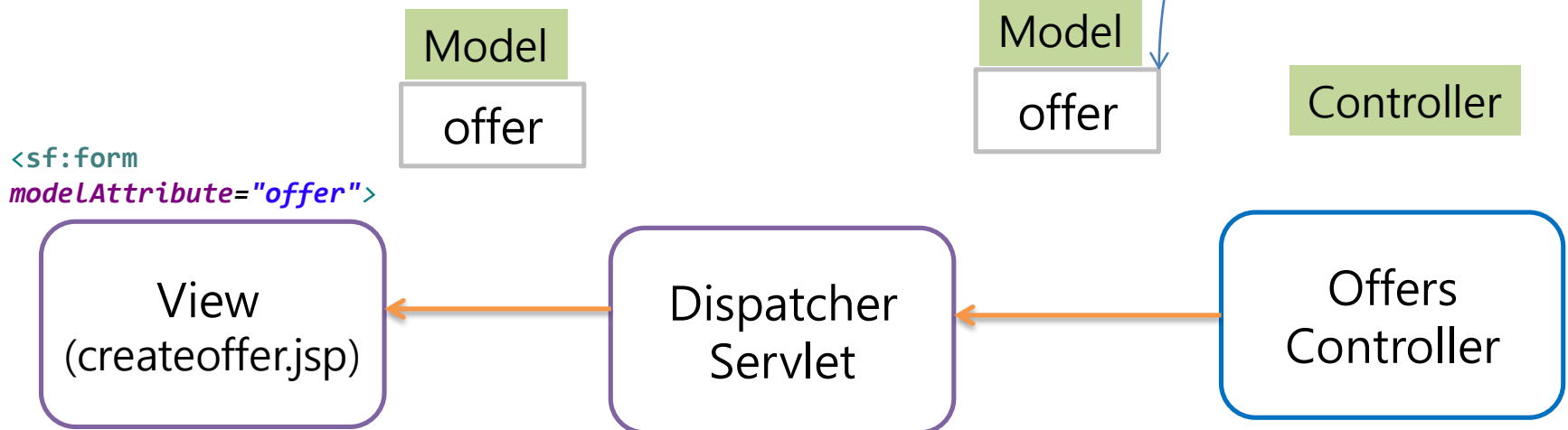
뷰를 새롭게 생성해서 사용자가 볼 수 있게 함

# 3. Revised Controller

## ① Initial Web Form

```
@RequestMapping("/createoffer")  
public String createOffer(Model model) {  
    model.addAttribute("offer", new Offer());  
    return "createoffer";  
}
```

bean 객체를 생성



# Revised Controller

## ② Web Form on Error

```
@RequestMapping(value="/docreate", method=RequestMethod.POST)
```

```
public String doCreate(@Valid Offer offer, BindingResult result) {
```

```
    if(result.hasErrors()) {
```

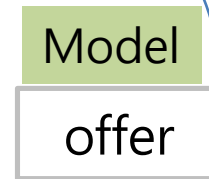
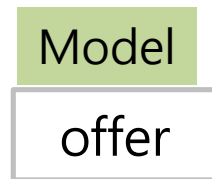
```
        return "createoffer";
```

```
    }
```

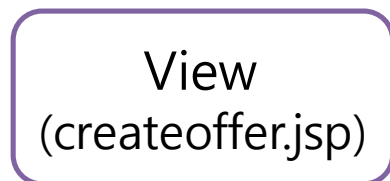
```
    ...
```

```
}
```

```
<sf:form  
modelAttribute="offer">
```



Controller



Note that the offer object will be automatically instantiated and added to the model

## 4. Error Messages

- We still have to inform the user on the **reason why** the data have been rejected

Name:

Name must be between 5 and 100 characters

Email:

This does not appear to be a valid email address

Your offer:

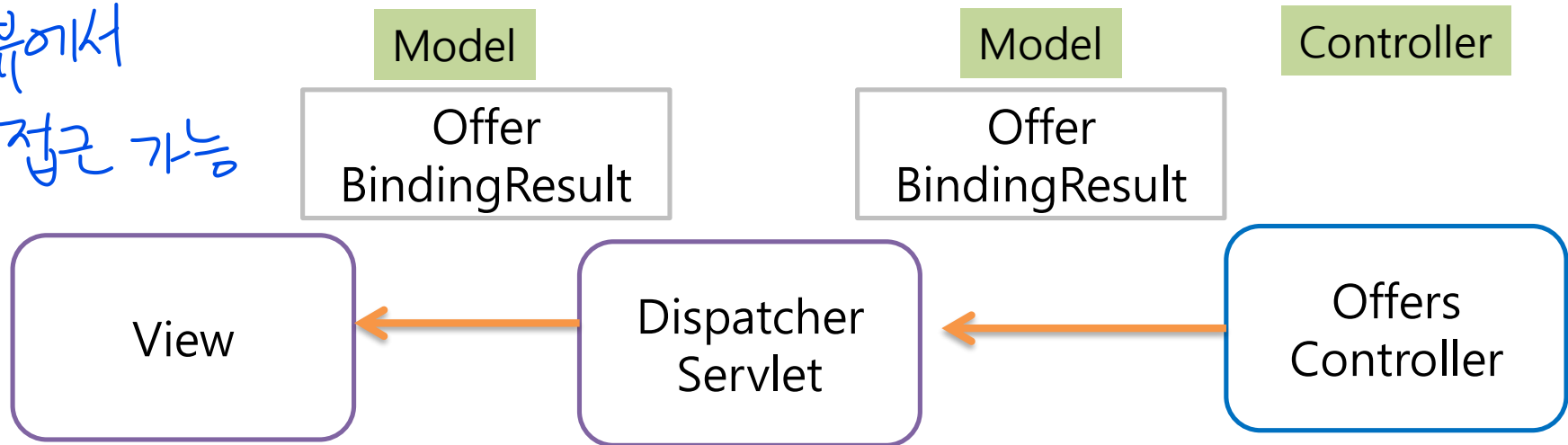
Text must be between 5 and 100 characters

Create advert

# Error Messages

- To this end, the BindingResult object is **automatically inserted** into the model and sent back to the view

여기서나  
뷰에서  
접근 가능



# Error Messages

- Spring MVC provides the `<sf:errors>` tag as part of the Spring's form tag library
- The `<sf:errors>` tag renders in HTML error messages taken from the `BindingResult` object

```
<sf:errors path="name"/>
```



# Error Messages

```
<sf:form modelAttribute="offer">
```

*Name:*

```
<sf:input path="name" />
```

```
<sf:errors path="name" />
```

name에 대한 오류

*Email:*

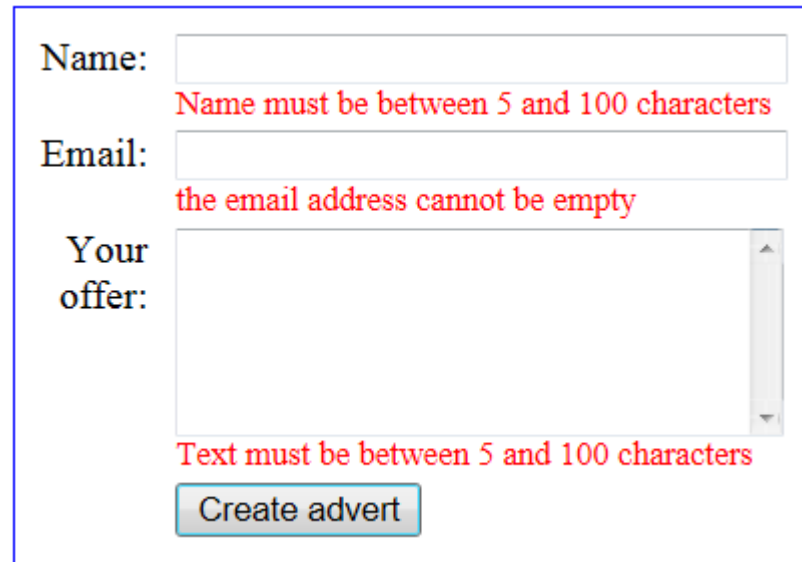
```
<sf:input path="email" />
```

```
<sf:errors path="email" />
```

...

```
</sf:form>
```

# Error Messages



Name:   
Name must be between 5 and 100 characters

Email:   
the email address cannot be empty

Your offer:   
Text must be between 5 and 100 characters

Create advert

```
public class Offer {
```

```
    private int id;
```

```
    @Size(min=5, max=100, message="Name must be between 5 and 100 characters")  
    private String name;
```

```
    @Email(message="please provide a valid email address")  
    @NotEmpty(message="the email address cannot be empty")  
    private String email;
```

```
    @Size(min=5, max=100, message="Text must be between 5 and 100 characters")  
    private String text;
```

```
}
```

# Summary

- Form beans are versatile objects, as they play **different roles** at the same time

- Data binder
- Data validator
- Data buffer

form bean은 다목적 객체,  
동시에 다른 역할을 수행함