Models trong ASP.NET MVC

- Models là gì?
- Tạo model
- Chuyển model dữ liệu từ controller đến view

Models là gì?

- Là lớp chứa các thuộc tính biểu diễn dữ liệu
- Tượng trưng cho dữ liệu kết nối với ứng dụng
- MVC định nghĩa 3 kiểu models:
  - Data model: lớp tương tác với dữ liệu. Là tập các lớp hoặc theo phương pháp database-first hoặc theo phương pháp code-first
  - Business model: lớp tượng trưng thực hiện các chức năng
  - View model: lớp tương tác giữa controller và view

Tạo model

- Để tạo model MVC
  - Tạo lớp public
  - Khai báo thuộc tính public cho mỗi thông tin của model
- VD: Khai báo một lớp model có tên User

```
public class User
{
    public long Id { get; set; }
    public string name { get; set; }
    public string address { get; set; }
    public string email { get; set; }
}
```

3

3

Truy cập model từ controller

- Mọi yêu cầu được trả về bởi action
- Action được sử dụng để truy cập các model chứa dữ liệu
- Để truy cập dữ liệu, cần tạo một đối tượng của lớp model nhằm lấy hoặc gán giá trị
- VD:

```
public ActionResult Index()
{
    var user = new MVCModelDemo.Models.User();
    user.name = "John Smith";
    user.address = "Park Street";
    user.email = "john@mvcexample.com";
    return View();
}
```

4

4

Chuyển dữ liệu model từ controller đến view

- Khi truy cập dữ liệu trong controller, cần chuyển model cho view khi gọi view, như vậy view mới có thể hiển thị dữ liệu cho người dùng
- Có hai mô hình
  - Đối tượng đơn lẻ
  - Tập đối tượng
- Trong action method, tạo một đối tượng model sau đó chuyển qua view bằng cách sử dụng `ViewBag`

Chuyển dữ liệu model từ controller đến view

- VD

```
public ActionResult Index()
{
    var user = new MVCModelDemo.Models.User();
    user.name = "John Smith";
    user.address = "Park Street";
    user.email = "john@mvcexample.com";
    ViewBag.user = user;
    return View();
}
```

Chuyển dữ liệu model từ controller đến view

- Truy cập dữ liệu lưu trong ViewBag

```
<!DOCTYPE html>
<html> <body>
    <p>User Name: @ViewBag.user.name</p>
    <p>Address: @ViewBag.user.address</p>
    <p>Email: @ViewBag.user.email</p>
</body> </html
```



```
public ActionResult Index(){
      var lstuser = new List<User>();
      var user1 = new User();
      user1.name = "Mark Smith";
      user1.address = "Park Street";
      user1.email = "Mark@mvcexample.com";
      var user2 = new User();
      user2.name = "John Parker";
      user2.address = "New Park";
      user2.email = "John@mvcexample.com";
      var user3 = new User();
      user3.name = "Steave Edward ";
      user3.address = "Melbourne Street";
      user3.email = "steave@mvcexample.com";
      lstuser.Add(user1);
      lstuser.Add(user2);
      lstuser.Add(user3);
      ViewBag.lstuser = lstuser;
      return View();
}
```

Chuyển dữ liệu model từ controller đến view

```
<!DOCTYPE html>
<html> <body>
   <h3>User Details</h3>
   @{ var lstuser = ViewBag.lstuser; }
   @foreach (var p in lstuser){
     @p.name<br />
     @p.address<br />
     @p.email<br />
     <br /> }
</body> </html>
```


Chuyển dữ liệu model từ controller đến view

```
<!DOCTYPE html>
<html> <body>
   <h3>User Details</h3>
   @{ var user = ViewBag.user; }
   @foreach (var p in user) {
      @p.name<br />
      @p.address<br />
      @p.email<br /><br /> }
</body> </html>
public ActionResult Index() {
  var user = new List<User>(); var user1 = new User();
   user1.name="Mark Smith"; user1.address="Park Street";
   user1.email = "Mark@mvcexample.com";
  var user2 = new User(); user2.name = "John Parker";
   user2.address = "New Park";
   user2.email = "John@mvcexample.com";
  var user3 = new User(); user3.name = "Steave Edward ";
   user3.address = "Melbourn Street";
   user3.email = "steave@mvcexample.com";
  user.Add(user1); user.Add(user2); user.Add(user3);
return View(user); }
```

Sử dụng strong typing

- Trong trường hợp view không nhận chính xác kiểu dữ liệu, gõ chính xác

```
<html> <body>
 <h3>User Details</h3>
 @{var user=Model as MVCModelDemo.Models.User;}
 @user.name <br/>
 @user.address<br/>
 @user.email<br/>
</body> </html
```

- Sử dụng strong typing
- Cú pháp: `@model <model_name>`
- Khai báo @model có thể truy cập các thuộc tính của model trong view

Sử dụng strong typing

```
@model MVCModelDemo.Models.User
<html><body>
    <h3>User Details</h3>
    @Model.name <br/>
    @Model.address<br/>
    @Model.email<br/>
</body> </html>
```

- Chuyển một tập các đối tượng cho view
  - VD: `@model IEnumerable<MVCModelDemo.Models.User`

Sử dụng strong typing

```
@model IEnumerable<MVCModelDemo.Models.User>
<html><body>
   <h3>User Details</h3>
   @{var user = Model;}
   @foreach(var u in user){
     @u.name <br/>
     @u.address<br/>
     @u.email<br/><br/>}
</body></html>
```


Các phương thức HTML Helper trong Strongly Types

- MVC cho phép
  - Kết nối trực tiếp với các thuộc tính của model chỉ ở dạng strongly types
    - Html.LabelFor()
    - Html.DisplayNameFor()
    - Html.DisplayFor()
    - Html.TextBoxFor()
    - Html.TextAreaFor()
    - Html.EditorFor()
    - Html.PasswordFor()
    - Html.DropDownListFor()

```
@model MVCModelDemo.Models.User
@{ViewBag.Title = "User Form";}
<h2>User Form</h2>
@using (Html.BeginForm()) {
   @Html.ValidationSummary(true)
   <div>@Html.LabelFor(model => model.name)</div>
   <div>@Html.EditorFor(model => model.name)</div>
   <div>@Html.LabelFor(model =>model.address)</div>
   @Html.EditorFor(model =>model.address)
   <div>@Html.LabelFor(model =>model.email)</div>
   <div>@Html.EditorFor(model =>model.email) </div>
   <p><input type="submit" value="Create" /></p>
}
```

15

15

Model Binder

- Khi người dùng submit thông tin trên form trong strongly typed view, MVC tự động kiểm tra HttpRequest và ánh xạ thông tin gửi đến trường trong model
- Tiến trình ánh xạ này gọi là model binding
- Một số lợi ích của model binding:
  - Tự động trích dữ liệu từ HttpRequest
  - Tự động chuyển kiểu dữ liệu
  - Tạo dữ liệu hợp lệ dễ dàng

16

16

Model Binder

- DefaultModelBinder là lớp model binder của MVC
- Model binder:
  - Yêu cầu giá trị nguyên thủy
  - Yêu cầu đối tượng

17

Yêu cầu giá trị nguyên thủy

- Tạo lớp
- Tạo view kết nối đến lớp
- VD: tạo lớp login

```
public class Login{
   public string userName { get; set; }
   [DataType(DataType.Password)]
   public string password { get; set; }
}
```


18

Yêu cầu giá trị nguyên thủy

- Tạo view index.cshtml

```
@model ModelDemo.Models.Login
@{ ViewBag.Title = "Index";}
<h2>User Details</h2>
@using (Html.BeginForm()) {
@Html.ValidationSummary(true)
<div>@Html.LabelFor(model =>model.userName)</div>
<div>@Html.EditorFor(model=>model.userName)</div>
<div>@Html.LabelFor(model =>model.password)</div>
<div>@Html.EditorFor(model=>model.password)</div>
<div><input type="submit" value="Submit" /></div>
}
```


19

- Sau đó tạo controller chứa action method: index() để hiển thị view

```
public class HomeController : Controller  {
public ActionResult Index(){return View();}
[HttpPost]
public ActionResult Index(string userName,
string password) {
if (userName == "Peter" && password ==
"pass@123")  {
     string msg = "Welcome " + userName;
     return Content(msg);}
     else  {return View();}
     }
}
```


20

Yêu cầu đối tượng

```
public class HomeController : Controller {
public ActionResult Index() {return View();}
[HttpPost]
  public ActionResult Index(Login login) {
  if (login.userName == "Peter" &&login.password ==
  "pass@123") {
  String msg = "Welcome " + login.userName;
  return Content(msg);  }
  else  {
  return View(); }  }
  }
```

- Index() đầu tiên sinh view để hiển thị login form
- Index() thứ 2 chuyển dữ liệu tự HttpRequest và đẩy vào Login object

21

Yêu cầu đối tượng

- Khi người dùng submit dữ liệu login, phương thức Index() kiểm tra username và password được chuyển trong đối tượng login
- Nếu thành công, view hiển thị thông điệp chào
- Khi truy cập ứng dụng từ trình duyệt, Index.cshtml hiển thị login form
- Gõ Peter và pass@123 trong login form
- Nhấn Submit hiển thị thông điệp chào “Welcome Peter”

22