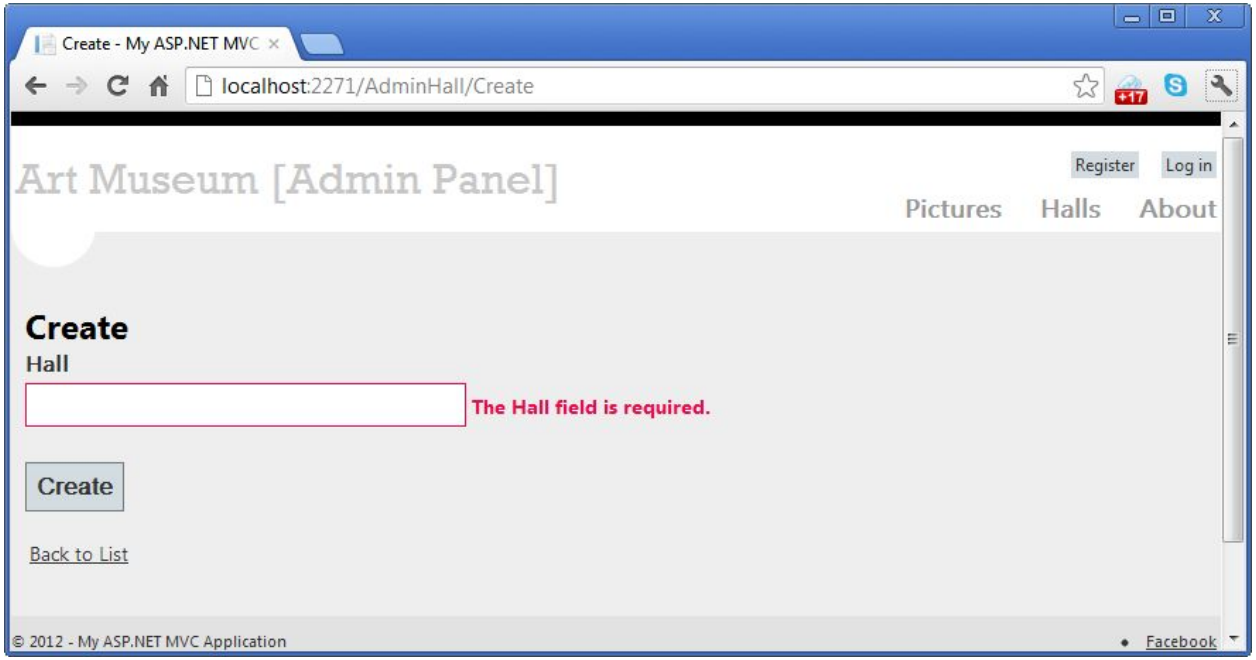
Валидация пользовательского ввода

ASP.NET MVC 4.0

2013

Валидация в MVC

Цель проверки ввода – не дать ошибкам пользователя далеко распространиться по приложению.
Если пользователь ошибся, в результате проверки он должен получить понятное сообщение об этом.



Проверка ввода обязательно выполняется на стороне сервера и опционально на стороне клиента.

Проверка, встроенная в модель

Чтобы модель могла сама себя валидировать, она должна реализовать интерфейс IValidatableObject.

```
// Модель проверяет, что название зала может быть одним из семи цветов радуги.
//
public class Hall : IValidatableObject
{
    [Key]
    public int HallId { set; get; }

    [Column]
    [Required]
    [Display(Name = "Hall")]
    public string Name { set; get; }
    public virtual List<Picture> Pictures { set; get; }

    public IEnumerable<ValidationResult> Validate(ValidationContext validationContext)
    {
        const string colors = "Red Orange Yellow Green Blue Indigo Violet";
        if (Name != null && !colors.Contains(Name))
            yield return new ValidationResult(
               "The hall's name must be one of the rainbow colors.", new[] { "Name" });
    }
}
```

Мероприятия в контроллере и в представлении

В контроллере:

```
[HttpPost]
public ActionResult Index(RegisterForm f)
{
    if (ModelState.IsValid)
    {
        return RedirectToAction("Ok");
    }
    return View();
}
```

В представлении:

```
@using (Html.BeginForm()) {
    @Html.ValidationSummary(true)

    <fieldset>
        <legend>RegisterForm</legend>

        <div class="editor-label">
            @Html.LabelFor(model => model.Login)
        </div>
        <div class="editor-field">
            @Html.EditorFor(model => model.Login)
            @Html.ValidationMessageFor(model => model.Login)
        </div>
```

Декорирование свойств модели

Другой способ встроить валидацию в модель – пометить свойства модели атрибутами.

Атрибуты DataAnnotations

Атрибуты валидации объявлены в пространствах System.Web.Mvc и System.ComponentModel.DataAnnotations

Required

```
[Required(ErrorMessage="Требуется название картины")]
public string PictureName { get; set; }
```

StringLength

```
[StringLength(160, MinimumLength=3)]
public string ArtistName { get; set; }
```

RegularExpression

```
[RegularExpression(@"[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,4}")]
public string Email { get; set; }
```

Range

```
[Range(35,44)]
public int Age { get; set; }
```

Атрибут Compare

Находится в пространстве System.Web.Mvc

Позволяет сравнить значения двух свойств модели.

```
[RegularExpression(
@"[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,4}")]
public string Email { get; set; }

[Compare("Email")]
public string EmailConfirm { get; set;}
```


7

Атрибут Remote

Находится в пространстве System.Web.Mvc.
Позволяет выполнить клиентскую проверку с помощью ajax-запроса к серверу

Атрибут

```
[Remote("CheckUserName", "Account")]
public string UserName { get; set; }
```

Метод контроллера

```
public JsonResult CheckUserName(string username)
{
   var result = Membership.FindUsersByName(username).Count == 0;
   return Json(result, JsonRequestBehavior.AllowGet);
}
```

CheckUserName – имя метода-действия, Account – имя контроллера.
Имя параметра метода должно совпадать с именем проверяемого свойства.

Локализация сообщений об ошибках

Каждый атрибут валидации имеет три именованных параметра

```
[Required(ErrorMessage = "Your {0} is required.")]
[StringLength(160, ErrorMessage = "{0} is too long.")]
public string LastName { get; set; }
```

ErrorMessage, - это строка, в которой содержится сообщение об ошибке валидации, адресованное пользователю программы. Строка может содержать символ шаблона {0}, который будет замещен именем валидируемого свойства.

```
[Required(ErrorMessageResourceType = typeof(ErrorMessages),
          ErrorMessageResourceName = "LastNameRequired")]

[StringLength(160, ErrorMessageResourceType = typeof(ErrorMessages),
                   ErrorMessageResourceName = "LastNameTooLong")]

public string LastName { get; set; }
```

ErrorMessageResourceType и ErrorMessageResourceName задают ресурс, в котором находится локализованное сообщение об ошибке.
Предполагается, что в приложении имеется файл ресурса ErrorMessages.resx, в котором есть два элемента с именами "LastNameRequired" и "LastNameTooLong".

Привязка модели и валидация

- Привязка модели – это процесс определения значений аргументов методов-действий.
- Привязка происходит неявно, но ее можно вызвать и явно при помощи методов UpdateModel() или TryUpdateModel().

```
[HttpPost]
 public ActionResult Create(Hall hall)
 {
     Hall h = new Hall();
     UpdateModel(h);
     …
 }
```

- Валидаторы, объявленные в атрибутах, срабатывают в процессе привязки модели.
- Результатом привязки является объект ModelSatate - состояние модели

Состояние модели – ModelState

- MS содержит все значения, введенные пользователем в поля формы.
- MS содержит ошибки, ассоциированные с отдельными свойствами модели и с моделью в целом.

Например, пользователь сделал ошибку при вводе в поле LastName. Тогда:

```
ModelState.IsValid == false
ModelState.IsValidField("LastName") == false
ModelState["LastName"].Errors.Count > 0
```

Сообщения об ошибках также находится в состоянии модели.

```
var lastNameErrorMessage =  ModelState["LastName"].Errors[0].ErrorMessage;
```

Пользовательский атрибут валидации

Для реализации пользовательской валидации есть два способа:

a) проверка, встроенная в модель (IValidatableObject);
b) пользовательский атрибут аннотации.

Последний выбирают, кода хотят использовать проверку более, чем в одной модели.

Пример атрибута валидации

Предположим, мы хотим ограничить год написания картины не константой, а текущим годом.

```
// Атрибут проверяет, что значение не превышает текущего года.
//
public class UpToNowAttribute: ValidationAttribute
{
     public UpToNowAttribute()
        :base("{0} must be less than the current year.") {}

     protected override ValidationResult IsValid(object value,
         ValidationContext validationContext)
     {
        if (value != null)
        {
            int year = (int)value;
            var errorMessage = FormatErrorMessage(validationContext.DisplayName);
            if (year > DateTime.Now.Year)
               return new ValidationResult(errorMessage);
        }
        return ValidationResult.Success;
     }
}
```

Самостоятельно

Создать форму для регистрации пользователя и провалидировать ее.

- 1. Создать пустое MVC-приложение.
- 2. Добавить модель.

```
public class RegisterForm
    {
        public string Login { set; get; }  // не должен совпадать со логинами, находящимися в Application["loglist"] =
"aaa,bbb,ccc,ddd,eee,...,ggg,";
        public string Pass { set; get; }   // должен содержать буквы и цифры, длина от 6 до 100 символов
        public string Pass2 { set; get; }  // совпадает с Pass
        public string Email { set; get; }  // удовлетворяет регулярному выражению для emai
        public string Phone { set; get; }  // 9 или более цифр, может предшествовать +
        public double Stage { set; get; }  // от 2 до 25 включительно
        public int BirthYear { set; get; } // исходя из года рождения, текущий возраст - не должен превышать 100 лет
    }
```

- 3.Проставить атрибуты
- 4. Для свойства Login использовать [Remote]
- 5. Для свойства BirthYear разработать пользовательский атрибут валидации.