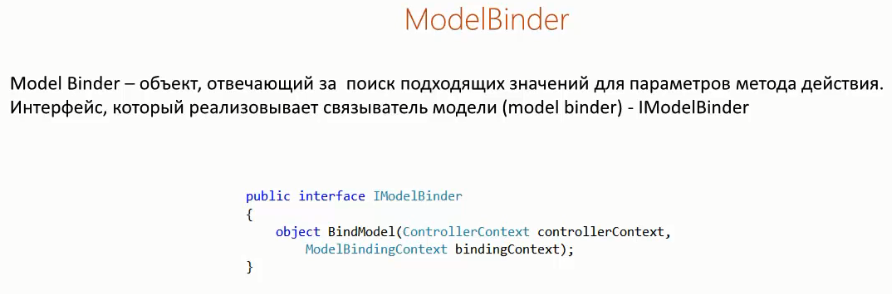


Пользовательский связной



Связыватель модели - объект который производит поиск значений в запросе

в соответствии с именами параметров метода действия.

Связыватель модели по умолчанию производит поиск значений в следующем порядке:

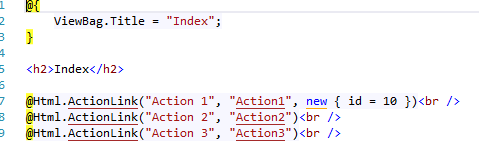
1) Request.Form

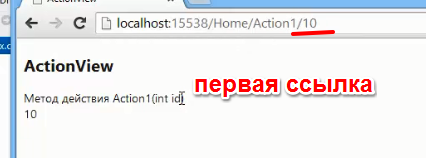
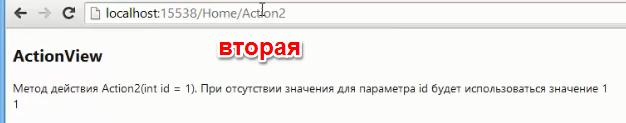
2) RouteData.Values

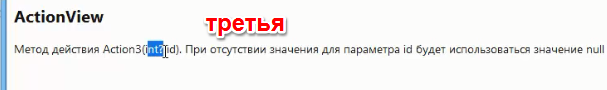
3) Request.QueryString

4) Request.Files

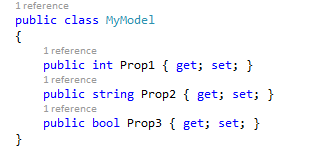


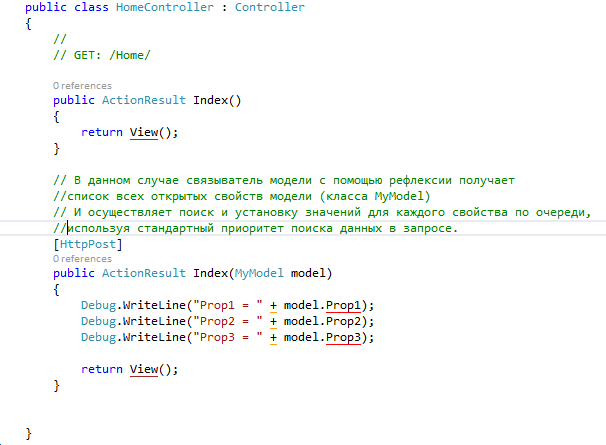


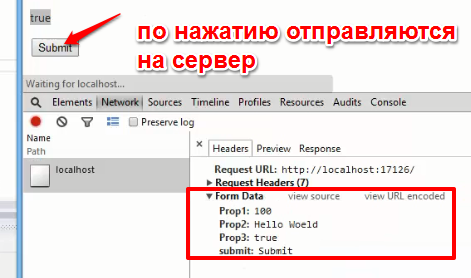


Связь с пользовательским классом

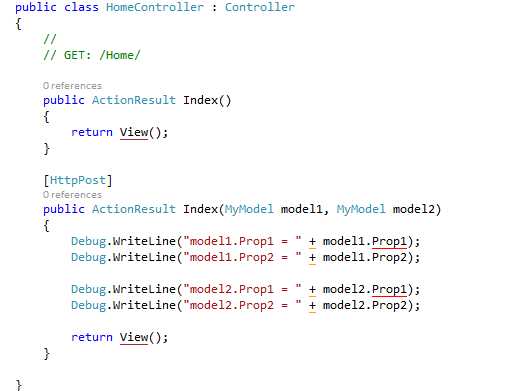


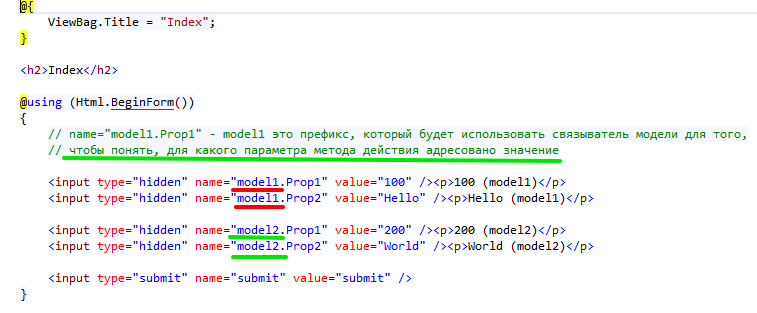
,



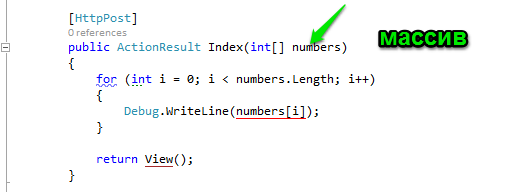


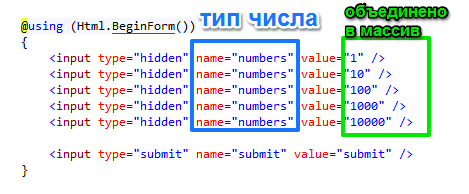
Привязка с помощью префиксов

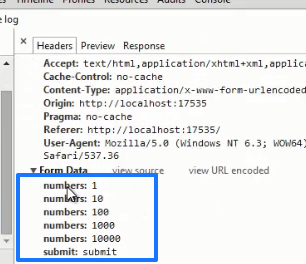




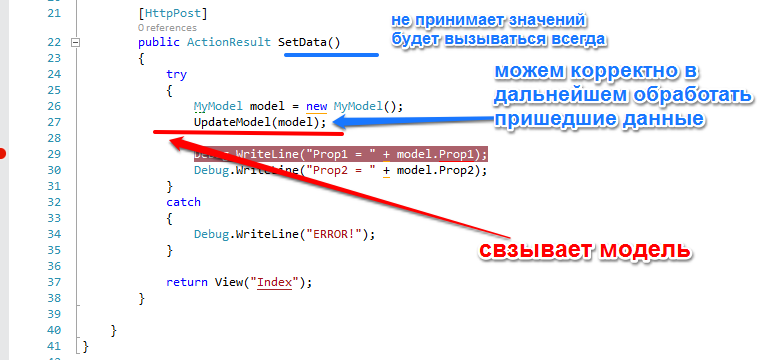
Привязка к массиву

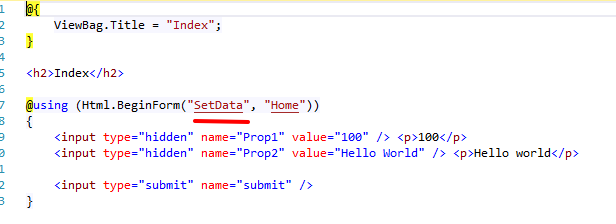






Пользовательская привязка



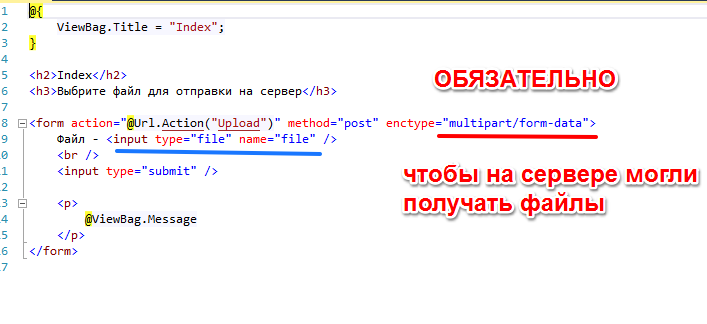


Отсылать файлы пользователю на сервер

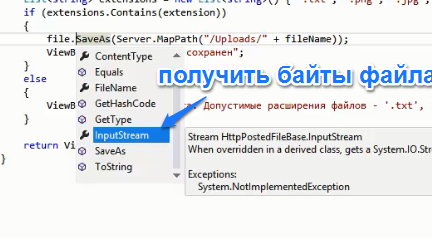
Указывать путь и сохранять

HttpPostedFileBase file

file.SaveAs(Server.MapPath("/Uploads/" + fileName));

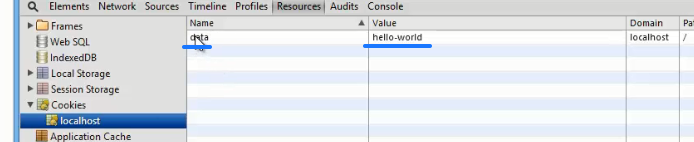






Поставщик данных который читает COOKIE







IValueProvider - класс (поставщик значений), который отвечает за поиск данных в запросе и инициализацию значений в методе действия.

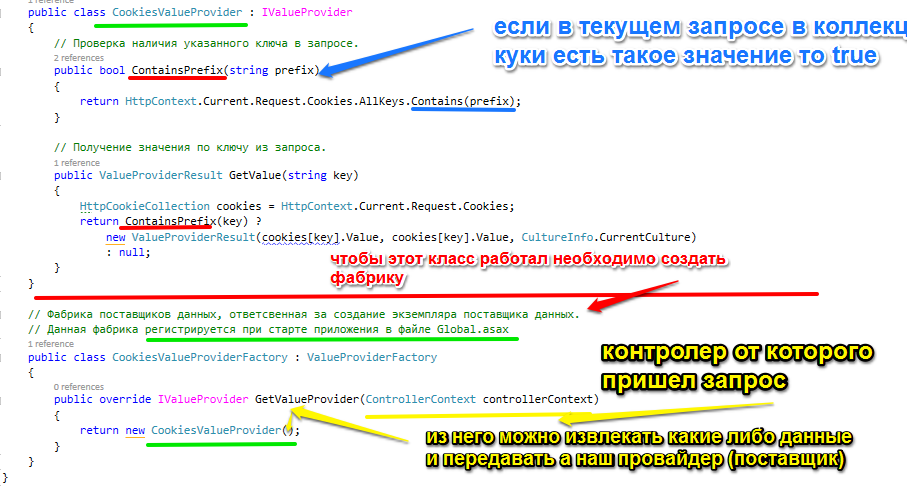
Системные классы реализующие данный интерфейс:

1) FormValueProvider

2) RouteDataValueProvider

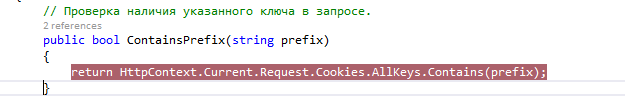
3) QueryStringValueProvider

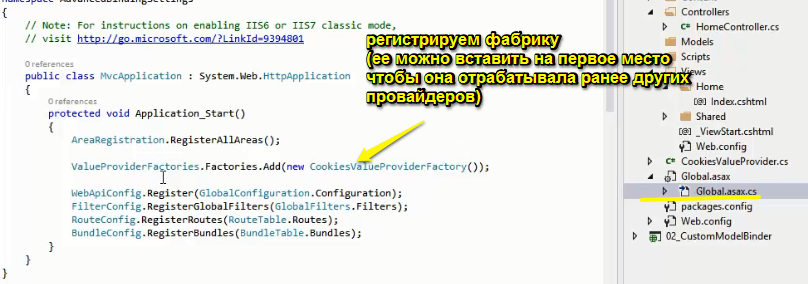
4) HttpFileCollectionValueProvider



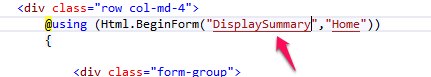
В метод public bool ContainsPrefix(string prefix) а потом если true то вызовется public ValueProviderResult GetValue(string key) заходит переменая с именем data, зависит от имени в контролере public ActionResult Index(string data)

Изначально зайдет вот сюда!!!!!!

 Регистрация фабрики



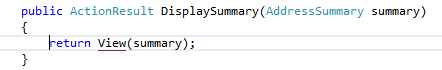
Передаем данные одного класса Person

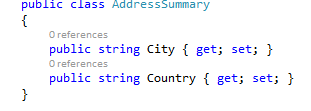




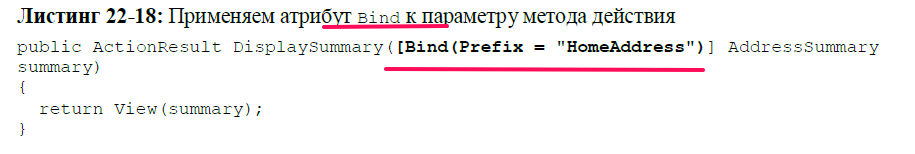
Но обрабатывать будет другой класс, но с такими же имена свойств

(в таком случае связать не сможет)

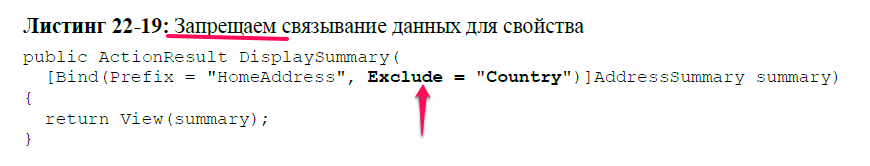




Нужно применить атрибут Bind



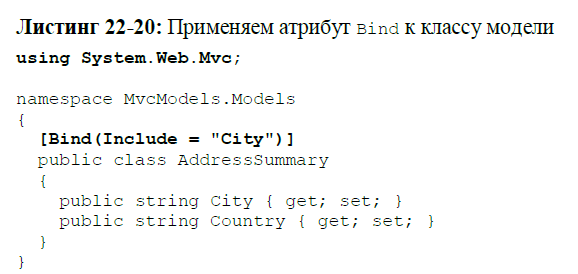
Если запретить связывание Exclude, для какого либо из свойств



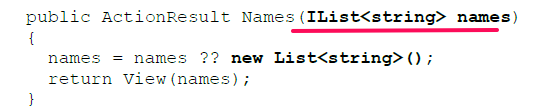
Include

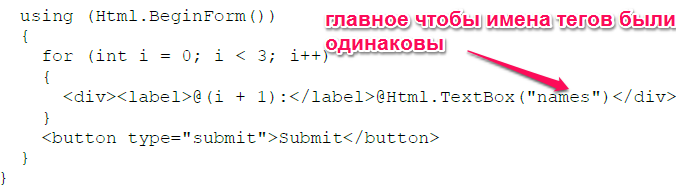
(В качестве альтернативы, вы можете использовать свойство Include, чтобы указать только те свойства, которые должны быть связаны для этой модели; все остальные свойства будут игнорироваться).

Можно применить к модели

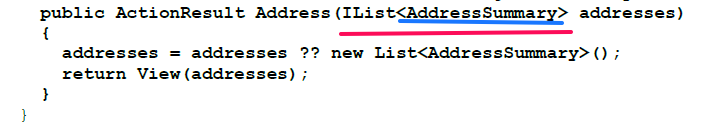


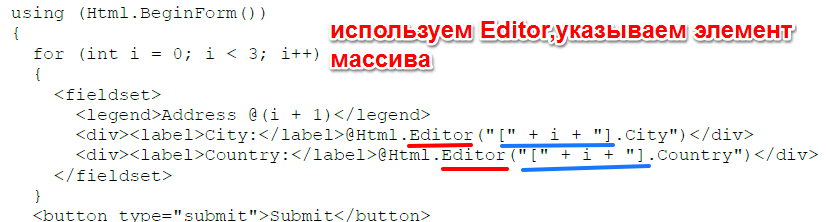
Принимает коллекцию IList (можно массив указать)





Связывание с коллекцией пользовательских типов



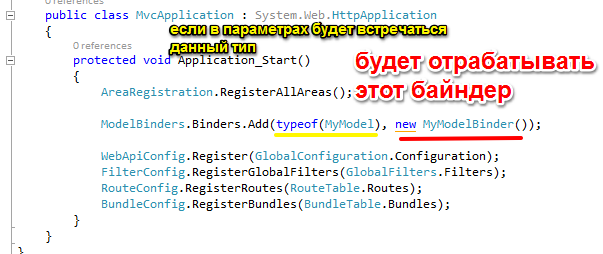


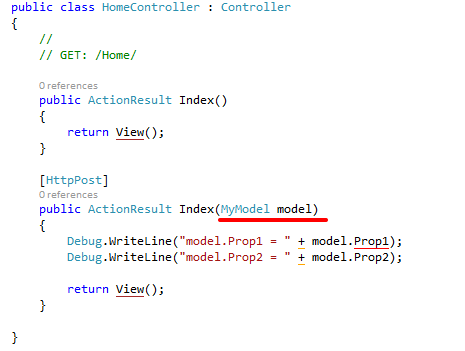
Результат

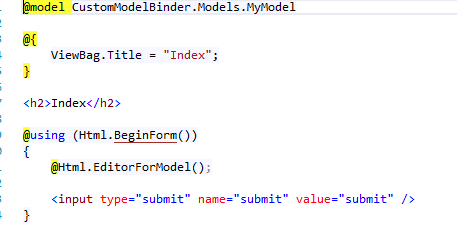


Собственная привязка модели

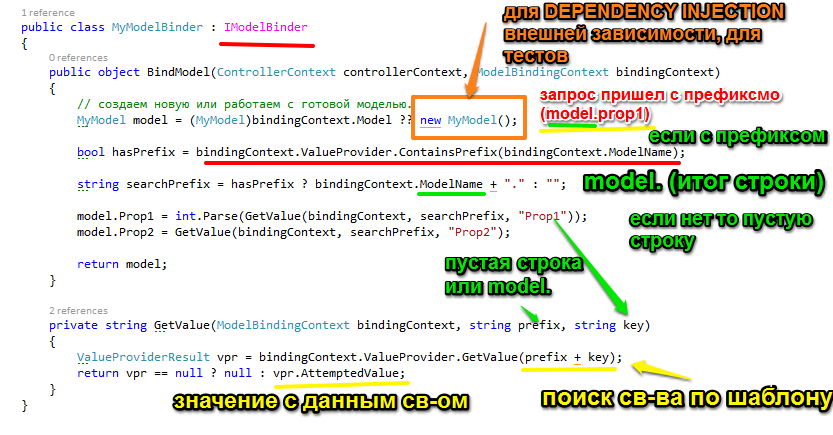
где будет встречаться данный тип, то будет изначально заходить в байндер



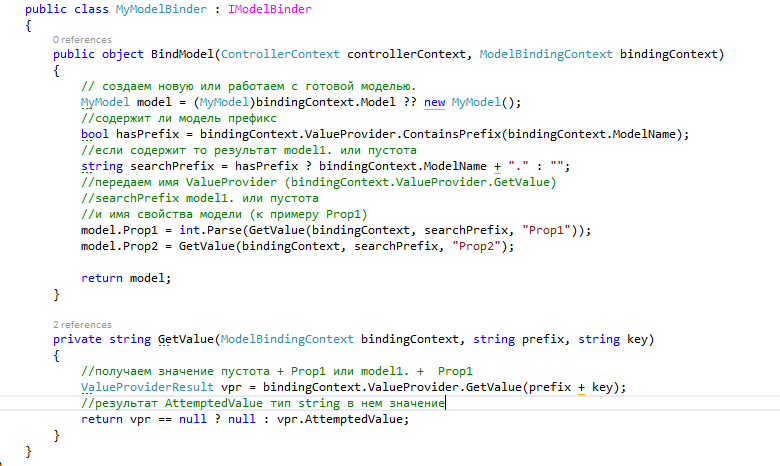


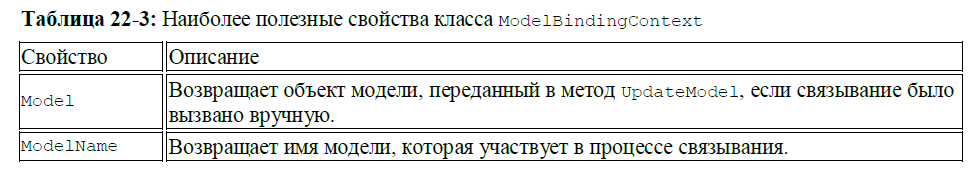


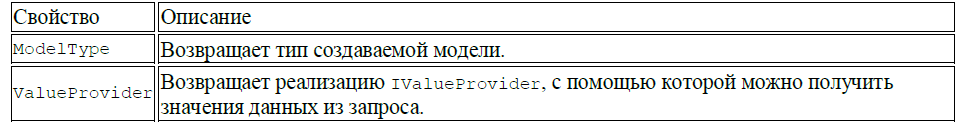
Наш класс

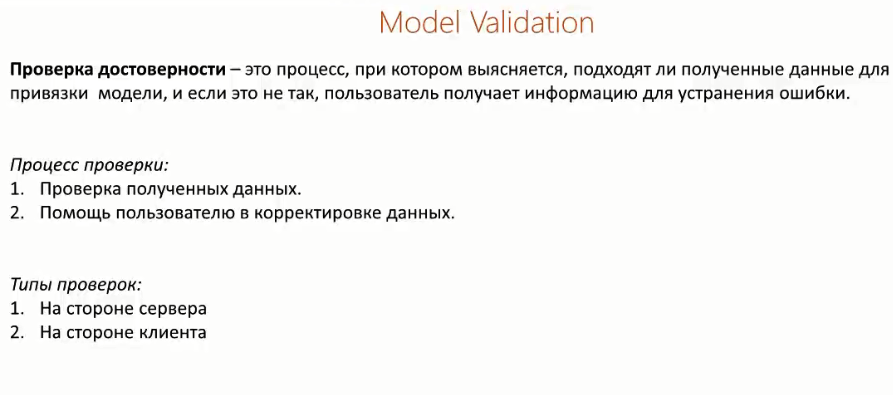


Подробное описание каждого действия

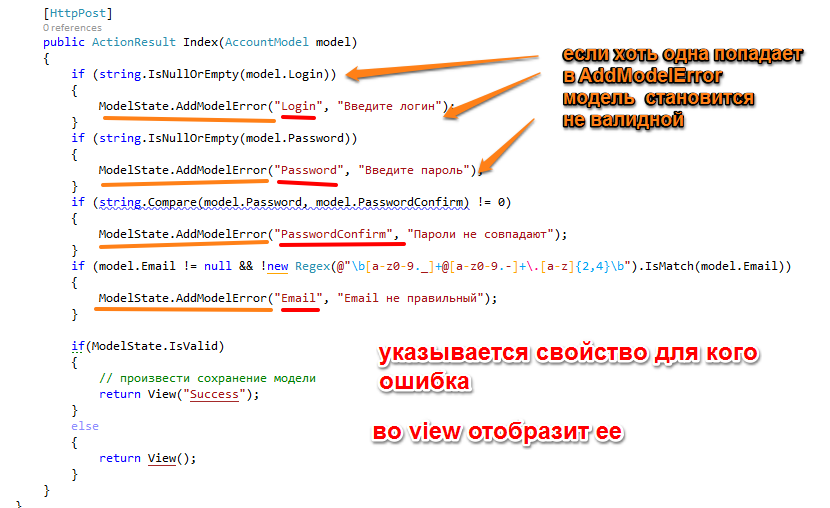




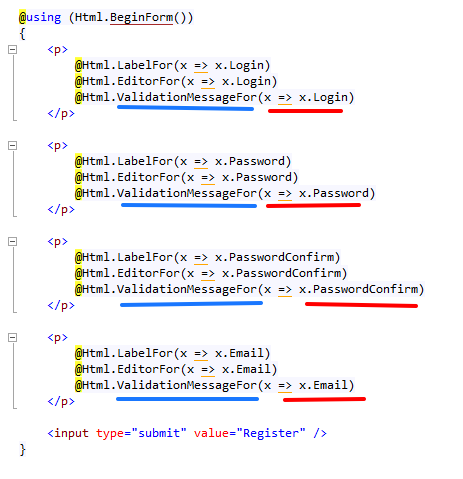


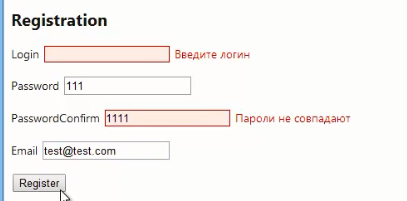


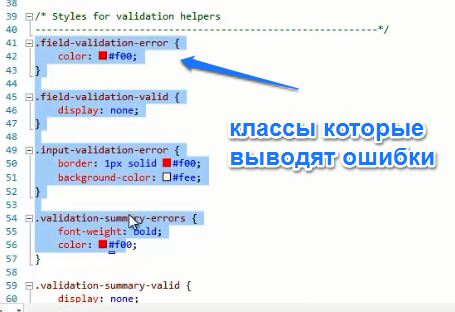
ModelState.AddModelError("PasswordConfirm", "Пароли не совпадают");



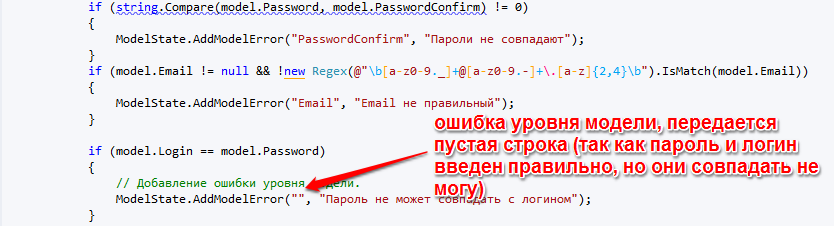
@Html.ValidationMessageFor(x => x.Password) – выводится текст ошибки



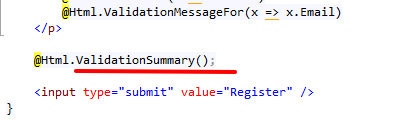


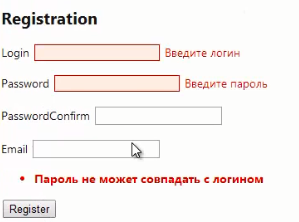


Ошибка уровня модели



@Html.ValidationSummary(); - если передать true то только на странице будут отображены ошибки уровня модели (ошибки уровня св-тв отображаться не будут)





Валидация модели с помощью атрибутов

Required - поле обязательное для заполнения

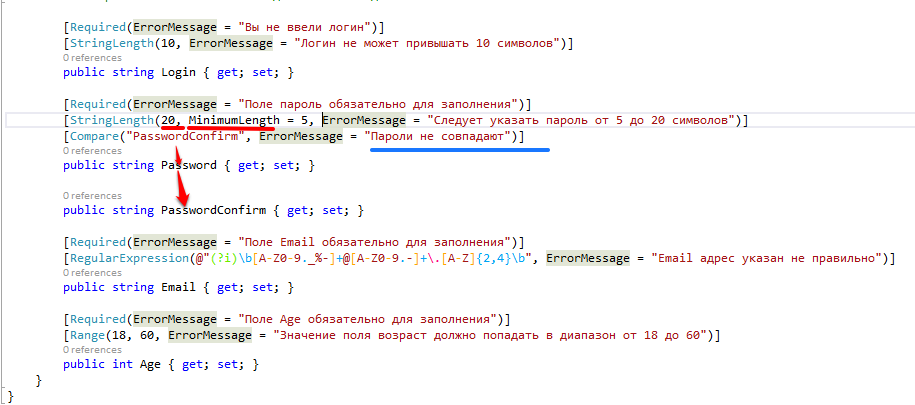
StringLength - допустимая длина слова для поля ввода

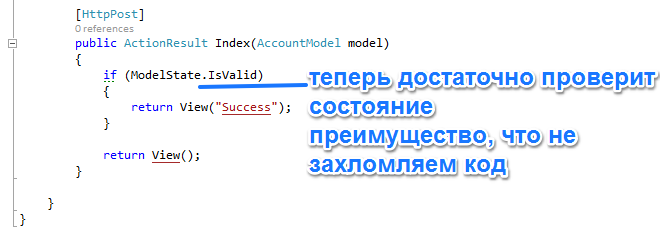
RegularExpression - регулярное выражение для проверки введенных данных

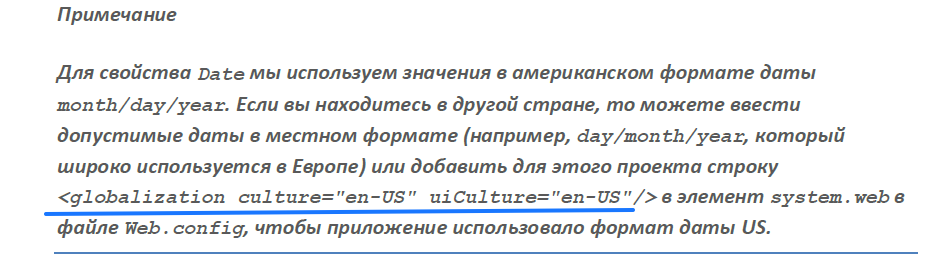
Range - диапазон допустимых значений для поля ввода

Compare - значения свойств должны иметь одинаковые значения

using System.ComponentModel.DataAnnotations;

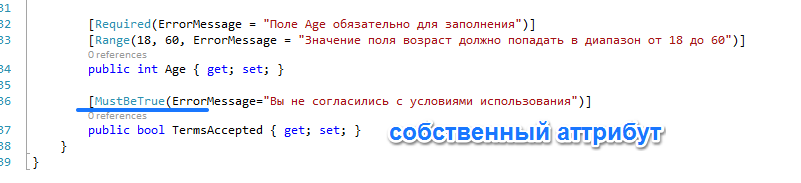




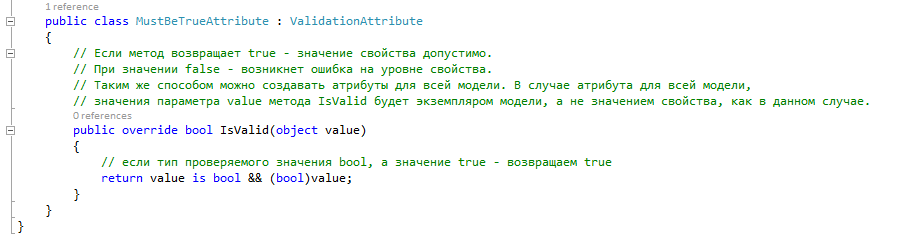


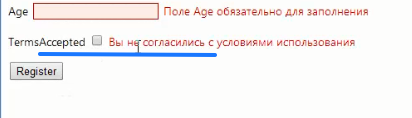
Собственные атрибуты на валидацию

Наследуем от ValidationAttribute

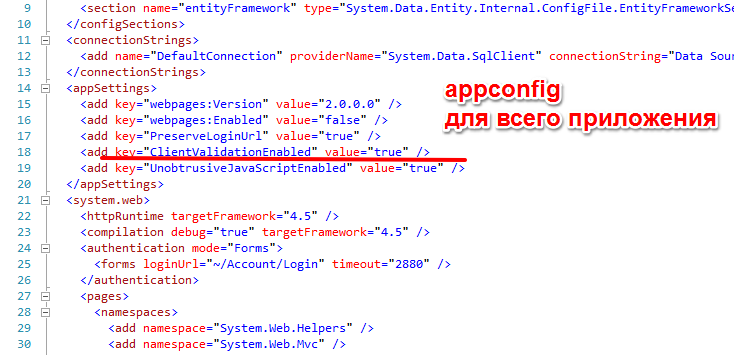


Таким же способом можно создавать атрибуты для всей модели.

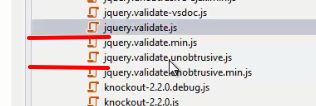




Валидация на стороне клиента

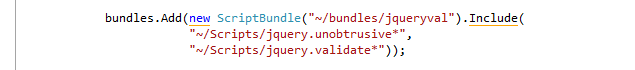


Библиотеки проверки на стороне клиента



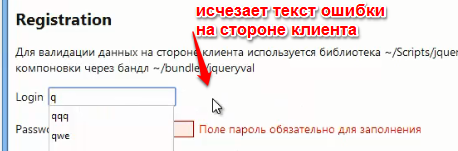
Для валидации данных на стороне клиента используется библиотека ~/Scripts/jquery.unobtrusive и ~/Scripts/jquery.validate

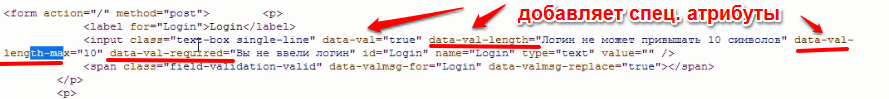
Данные библиотеки добавлены в файле компоновки через бандл ~/bundles/jqueryval

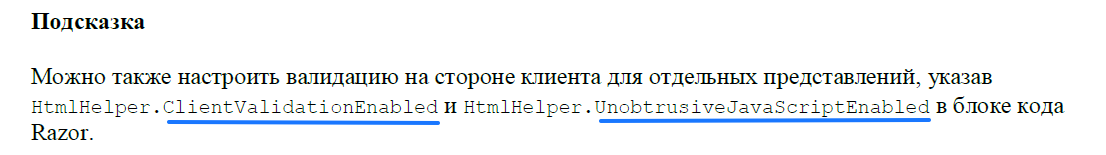


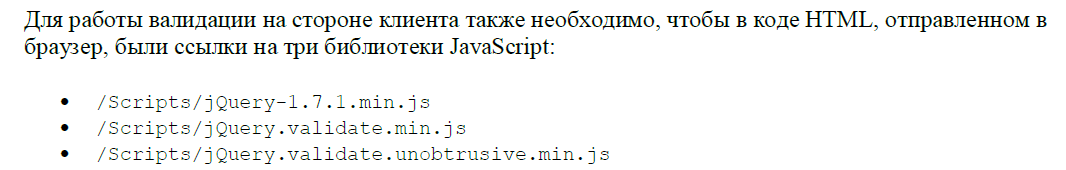


При вводе текста



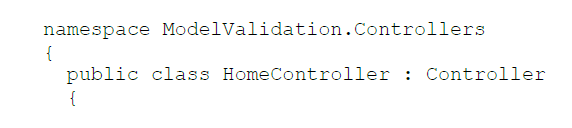


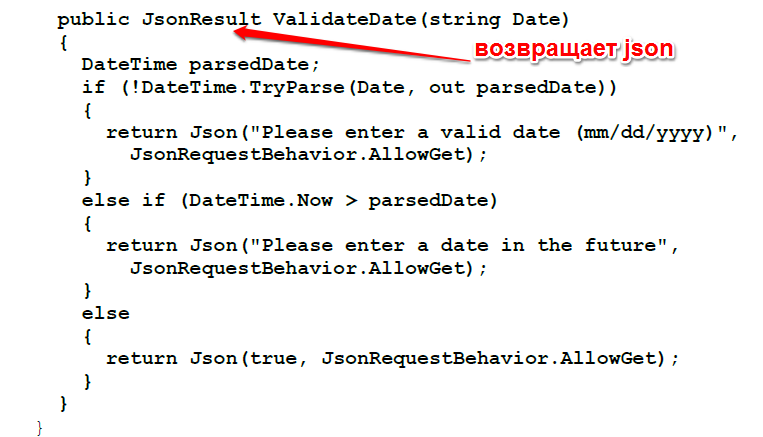




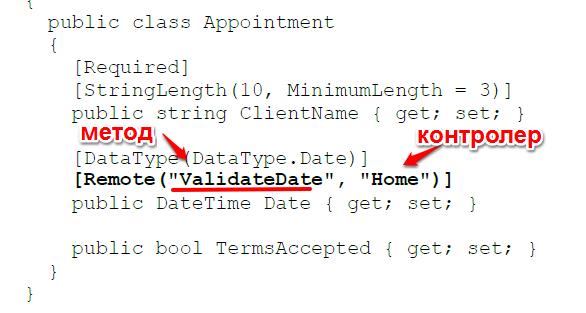
Логика валидации при помощи кода

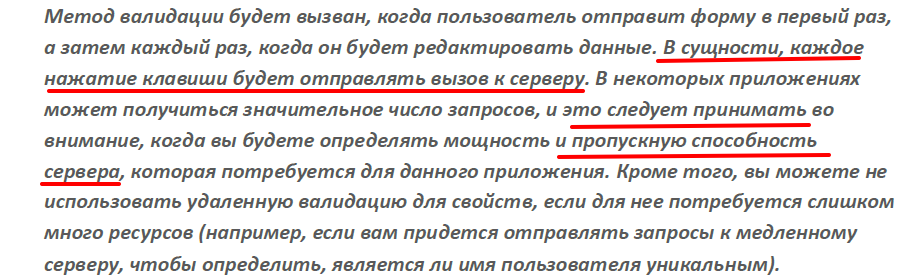
В контроллере объявляем метод логики





Модель





Провайдер валидации

//[Column(TypeName = "date")]

[DisplayFormat(DataFormatString = "{0:dd/MM/yyyy}", ApplyFormatInEditMode = true)]

//[DataType(DataType.Date,ErrorMessage = "Значение должно соответствовать дате")]

[Required(ErrorMessage = "Дата обязательна!")]

[DateValidationAttribute(Name = "DateBirthday",ErrorMessage = "DateValidationAttribute")]

public **DateTime** DateBirthday { get; set; }

[Required]

[StringLength(24)]

public string INN { get; set; }

public virtual ICollection<EmpPromotion> EmpPromotion { get; set; }

}

Атрибут для класса

public class DateValidationAttribute : ValidationAttribute

{

public string Name { get; set; } имя элемента в разметки

public override bool IsValid(object value)

{

try

{

return value.ToString().IsDateTime();

}

catch (Exception)

{

return false;

}

}

}

Провайдер

public class DateValidator : DataAnnotationsModelValidator<DateValidationAttribute>

{

public DateValidator(ModelMetadata metadata, ControllerContext context, DateValidationAttribute attribute)

: base(metadata, context, attribute)

{

if (!attribute.IsValid(context.HttpContext.Request.Form[attribute.Name]))

{

var propertyName = metadata.PropertyName;

if (context.Controller.ViewData.ModelState[propertyName]!=null)

{

context.Controller.ViewData.ModelState[propertyName].Errors.Clear();

context.Controller.ViewData.ModelState[propertyName].Errors.Add(attribute.ErrorMessage);

}

}

}

}

Global asax

DataAnnotationsModelValidatorProvider.RegisterAdapter(typeof(DateValidationAttribute), typeof(DateValidator));

Валидация сообщений

