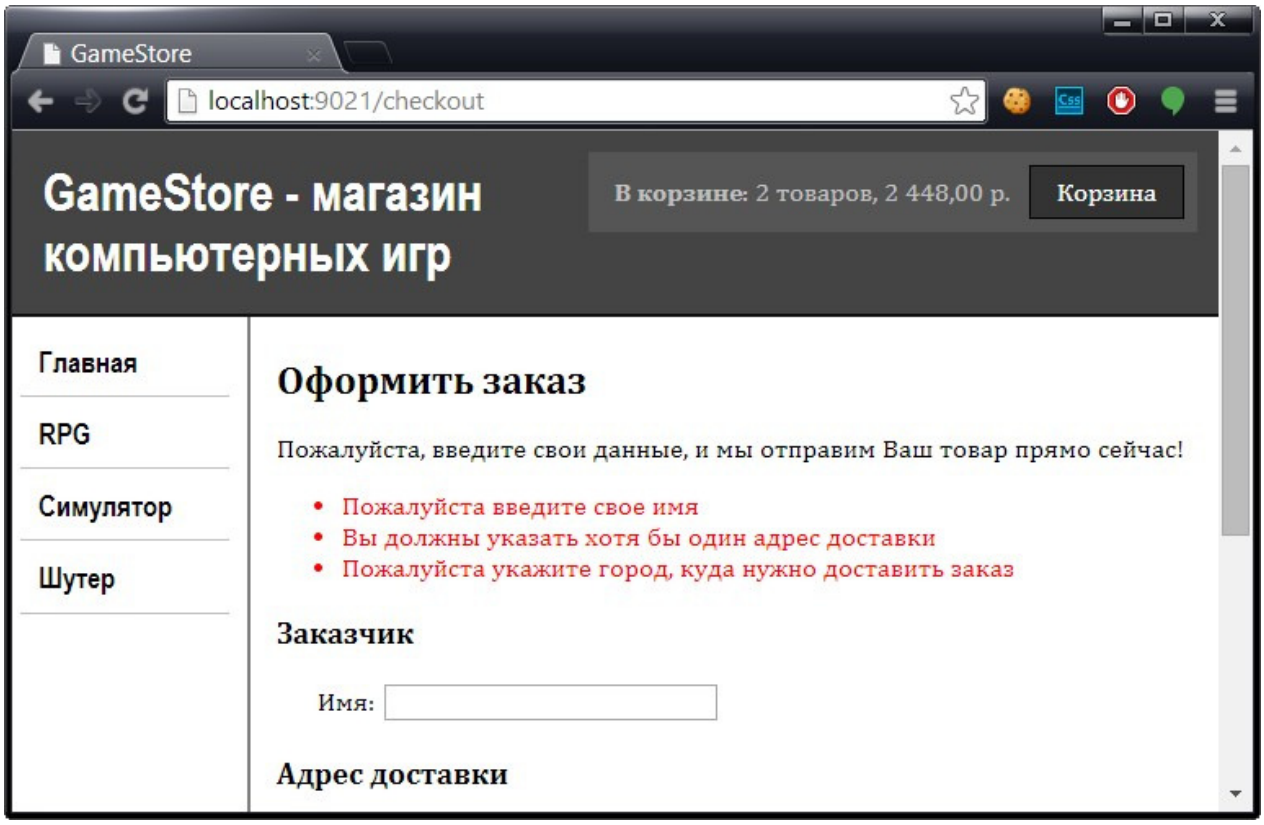
Добавление проверки достоверности

Исходный код проекта

Базовый процесс оплаты реализован в предыдущей статье, однако нужно еще добавить средства проверки достоверности, гарантирующие, что пользователь не сможет завершить процесс, не предоставив необходимой информации. В примере ниже демонстрируется применение атрибута `Required` к некоторым свойствам в классе `Order`:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.ComponentModel.DataAnnotations;

namespace GameStore.Models
{
    public class Order
    {
        public int OrderId { get; set; }

        [Required(ErrorMessage="Пожалуйста введите свое имя")]
        public string Name { get; set; }

        [Required(ErrorMessage="Вы должны указать хотя бы один адрес достав
        public string Line1 { get; set; }
        public string Line2 { get; set; }
        public string Line3 { get; set; }

        [Required(ErrorMessage="Пожалуйста укажите город, куда нужно достав
        public string City { get; set; }
        public bool GiftWrap { get; set; }
        public bool Dispatched { get; set; }
        public virtual List<OrderLine> OrderLines { get; set; }
    }

    public class OrderLine
    {
        public int OrderLineId { get; set; }
        public Order Order { get; set; }
        public Game Game { get; set; }
        public int Quantity { get; set; }
    }
}
```

Здесь в свойстве `ErrorMessage` указываются специальные сообщения об ошибках, отображаемые пользователю, если он не предоставил значение. Элемент управления `ValidationSummary` был добавлен к веб-форме `Checkout.aspx` при ее определении. Этот элемент управления будет отображать сообщения об ошибках в ситуации, когда значения в полях формы, помеченных как `Required`, не были введены:



Этот процесс известен как *проверка достоверности серверной стороны* и предполагает, что браузер посылает данные формы серверу для проверки, после чего сервер отправляет в качестве ответа завершенную HTML-страницу, включающую сообщения об ошибках.

Описанный процесс не идеален, т.к. он приводит к задержке между моментом отправки пользователем формы и моментом получения отклика о проблемах с введенными данными. Такая задержка вполне может достигать нескольких секунд, если серверы заняты или доступная ширина полосы пропускания ограничена.

Решение заключается в дополнении проверки достоверности серверной стороны проверкой достоверности клиентской стороны, при которой с помощью кода JavaScript выясняется, предоставил ли пользователь подходящие данные, еще до отправки данных формы на сервер.

Проверка достоверности клиентской стороны является дополнением такой

проверки на стороне сервера, а не ее заменой. Проверка достоверности клиентской стороны не будет выполняться в браузерах с отключенной поддержкой JavaScript (на удивление распространенная конфигурация). Проверка достоверности серверной стороны — это важная защита от злонамеренных пользователей, пытающихся вставить бессмысленные данные в базу. Проверка достоверности клиентской стороны должна применяться для улучшения пользовательского интерфейса, а проверка достоверности серверной стороны для защиты приложения.

Платформа ASP.NET Framework имеет поддержку проверки достоверности клиентской стороны, но работает она довольно плохо. Вы добавляете в свои веб-формы специальные элементы управления проверкой достоверности, а сервер генерирует код JavaScript, необходимый для выполнения проверки, когда пользователь пытается отправить форму.

С таким подходом связаны серьезные недостатки. Во-первых, не существует способа использования атрибутов проверки (таких как Required), примененных к классам модели данных, для управления процессом проверки достоверности. Это означает дублирование настроек проверки в классе модели (поэтому можно использовать привязку модели в классе отделенного кода) и внутри веб-формы.

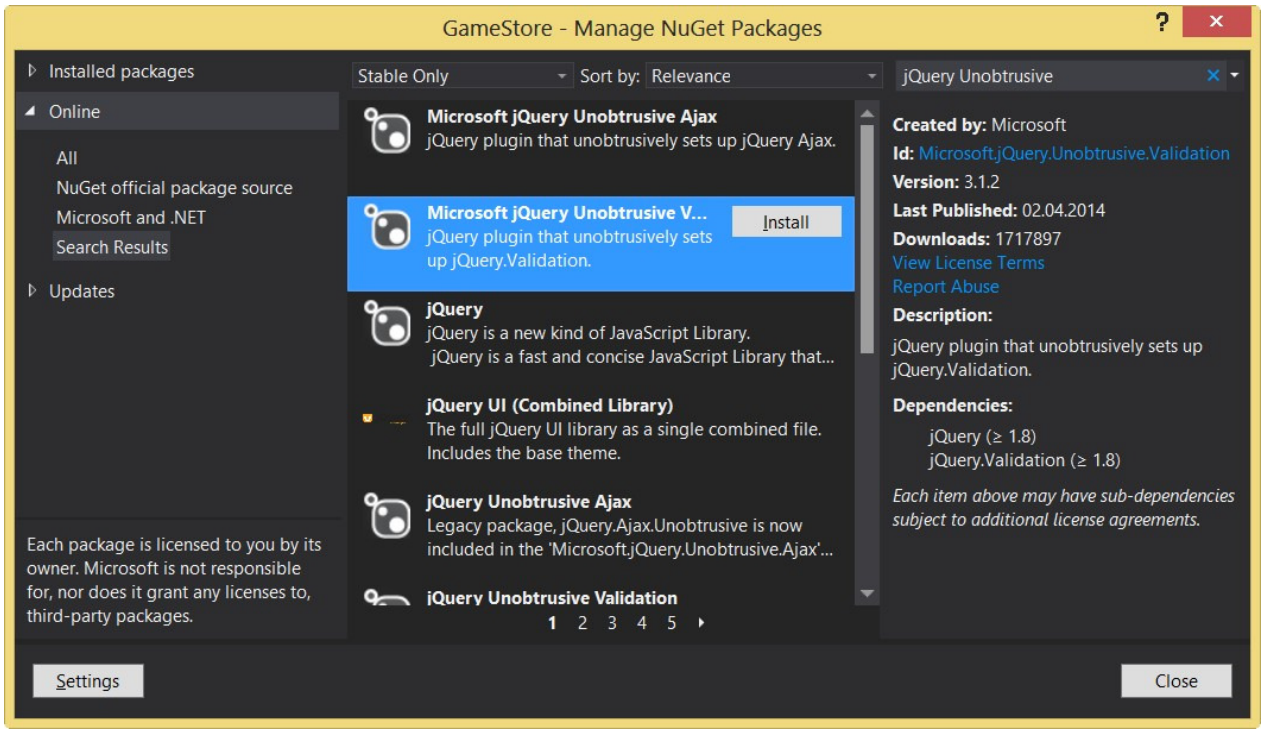
Во-вторых, элементы управления проверкой достоверности ASP.NET Framework не работают, когда в классе отделенного кода с помощью привязки модели создаются объекты данных на основе существующих данных, как это делалось в файле Checkout.aspx.cs. Вызов метода TryUpdateModel() приводит к прекращению работы элементов управления проверкой достоверности, т.е. придется выбирать какое-то одно из этих двух средств ASP.NET Framework.

Мы отдаем предпочтение средству привязки модели, а это значит, что мы должны найти альтернативный способ выполнения проверки достоверности на стороне клиента. Применяемый нами подход будет описан в последующих разделах.

Добавление пакетов NuGet

Мы планируем использовать библиотеку jQuery Validation, которая построена на основе базовой функциональности jQuery и поддерживает довольно широкий спектр вариантов проверки достоверности форм. Мы не собираемся работать с библиотекой jQuery Validation напрямую. Вместо этого мы перепрофилируем JavaScript-библиотеку, которую в Microsoft изначально написали для инфраструктуры MVC Framework.

Выберите пункт Manage Nuget Packages (Управлять пакетами NuGet) в меню Project (Проект) среды Visual Studio. В открывшемся окне Manage NuGet Packages (Управление пакетами NuGet) выберите категорию Online (Онлайновые) в левой панели и введите строку поиска "unobstrusive validation" в поле, расположенном в правом верхнем углу. Найдите в списке найденных вариантов пакет Microsoft jQuery Unobtrusive Validation (Ненавязчивая проверка достоверности с помощью jQuery от Microsoft) и щелкните на кнопке Install (Установить), как показано на рисунке ниже:



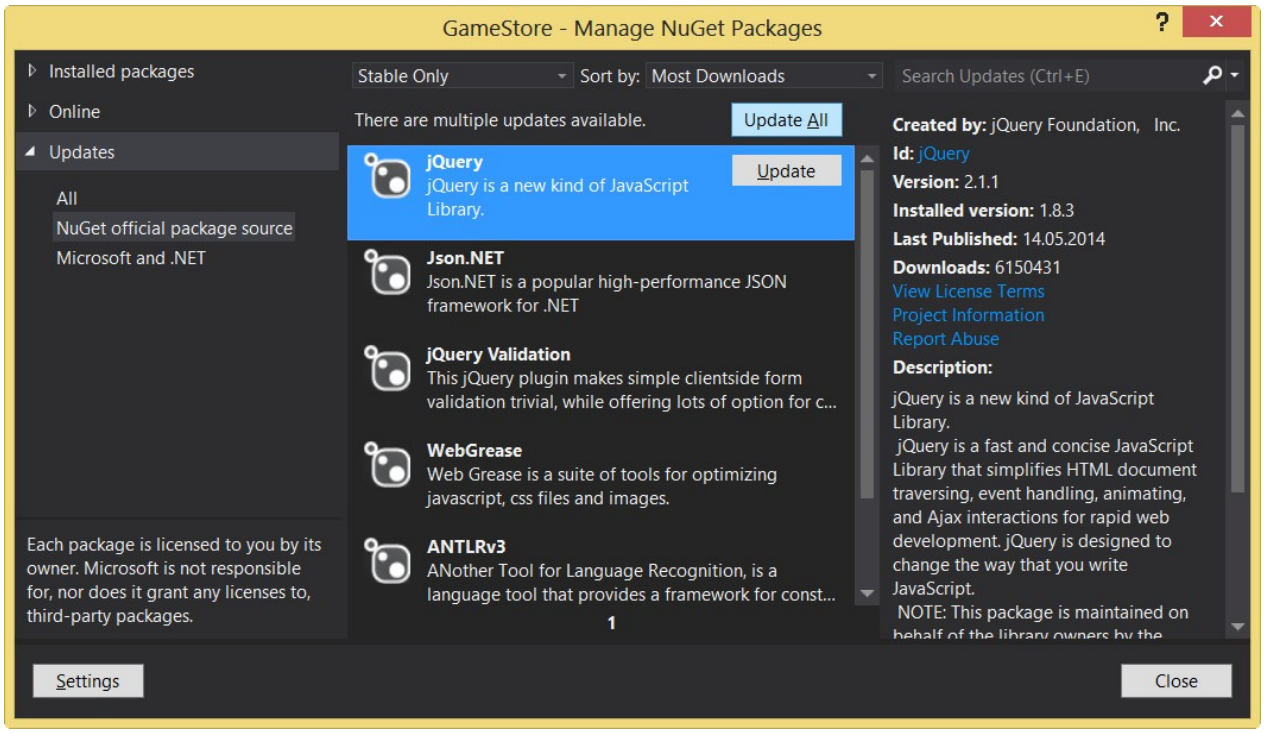
Ненавязчивый JavaScript — это неточно определенный термин, которым в общем случае обозначается код JavaScript, содержащийся в элементе script или внешнем файле, а не являющийся частью HTML-элементов. В случае проверки достоверности форм это также означает указание желаемого поведения проверки путем добавления атрибутов data к элементам input.

Диспетчер пакетов NuGet загрузит и установит нужные пакеты, включая jQuery и jQuery Validation. Установленные JavaScript-файлы находятся в папке \Scripts.

Нам также необходим пакет Microsoft ASP.NET Web Optimization Framework (Инфраструктура веб-оптимизации Microsoft ASP.NET), который содержит удобные средства для управления JavaScript-файлами. В его состав входит средство упаковки, используемое в следующем разделе. Найдите указанный пакет и добавьте его в проект.

Обновление пакетов

Из-за особенностей упаковки кода мы в конечном итоге получаем старые версии библиотек jQuery и jQuery Validation. Чтобы извлечь последние версии, выберите категорию Updates (Обновления) в левой панели и щелкните на кнопке Update (Обновить) для каждого ранее установленного пакета, как показано на рисунке ниже:



Создание и использование пакета сценариев

Средство пакетов позволяет более просто управлять файлами JavaScript и CSS за счет определения групп связанных файлов и трактовки их как единого модуля. По соглашению относительно настройки пакетов начинать следует с создания нового файла класса по имени BundleConfig.cs в папке App_Start. Код, помещаемый в этот файл, показан в примере ниже:

```
using System.Web.Optimization;

namespace GameStore
{
    public class BundleConfig
    {
        public static void RegisterBundles(BundleCollection bundles)
        {
            bundles.Add(new ScriptBundle("~/bundles/validation").Include(
                "~/Scripts/jquery-{version}.js",
                "~/Scripts/jquery.validate.js",
                "~/Scripts/jquery.validate.unobtrusive.js"));
        }
    }
}
```

Если Visual Studio не может распознать пространство System.web.Optimization, используемое в примере, то вероятнее всего вы не установили один из пакетов

NuGet, как было описано в предыдущем разделе.

Код в методе RegisterBundles() позволяет ссылаться на три файла сценариев, необходимые для проведения проверки достоверности клиентской стороны, как на единый модуль с помощью имени ~/bundles/validation.

Обратите внимание на изменение пространства имен, которое Visual Studio по умолчанию добавляет в файл класса, как это уже делалось для файла RouteConfig.cs.

Наш пакет файлов не является зарегистрированным до тех пор, пока не будет вызван статический метод RegisterBundles(), что делается в файле отделенного кода глобального класса приложения Global.asax.cs, как показано ниже. Это обеспечивает установку конфигурации пакетов при запуске приложения GameStore.

```
using System;
using System.Web.Routing;
using System.Web.Optimization;

namespace GameStore
{
    public class Global : System.Web.HttpApplication
    {

        protected void Application_Start(object sender, EventArgs e)
        {
            RouteConfig.RegisterRoutes(RouteTable.Routes);
            BundleConfig.RegisterBundles(BundleTable.Bundles);
        }
    }
}
```

Далее необходимо применить пакет к мастер-странице \Pages\Store.Master, чтобы среда ASP.NET Framework включила элементы script для трех JavaScript-файлов в HTML-разметку, отправляемую браузеру. В примере ниже показано, как это делается:

```
<%@ Master Language="C#" AutoEventWireup="true" CodeBehind="Store.master.cs"
   Inherits="GameStore.Pages.Store" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title>GameStore</title>
    <link rel="stylesheet" href="/Content/Styles.css" />
    <%: System.Web.Optimization.Scripts.Render("~/bundles/validation") %>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <div id="header">
                <GS:cartsummary runat="server" />
                <div class="title">GameStore - магазин компьютерных игр</div
            </div>
            <div id="categories">
                <GS:CategoryLinks runat="server" />
            </div>
            <div>
                <asp:ContentPlaceHolder ID="bodyContent" runat="server" />
            </div>
        </div>
    </form>
</body>
```

Запустив приложение и просмотрев HTML-разметку, отправленную браузеру, вы заметите, что в раздел head были добавлены следующие элементы:

```
<script src="/Scripts/jquery-2.1.1.js"></script>
<script src="/Scripts/jquery.validate.js"></script>
<script src="/Scripts/jquery.validate.unobtrusive.js"></script>
```

Возможность ссылки на пакеты сценариев обеспечивает получение требуемых файлов в нужном порядке. Вдобавок определение файлов, ассоциированных с пакетом, в файле BundleConfig.cs позволяет изменять используемые файлы в одном месте, избегая просмотра множества файлов веб-форм и мастер-страниц для нахождения и редактирования отдельных ссылок.

Обратите внимание, что при конфигурировании пакета мы не указывали желаемую версию файла jQuery, но среда ASP.NET Framework корректно нашла и воспользовалась файлом jquery-2.1.1.js.

Настройка проверки достоверности клиентской стороны

Проверка достоверности клиентской стороны настраивается за счет добавления атрибутов data к элементам input — какое-то время атрибуты data были неофициальным соглашением при определении специальных атрибутов для HTML-элементов, а сейчас они сделаны частью спецификации HTML5. В примере ниже можно видеть атрибуты data, добавленные в файл \Pages\Checkout.aspx:

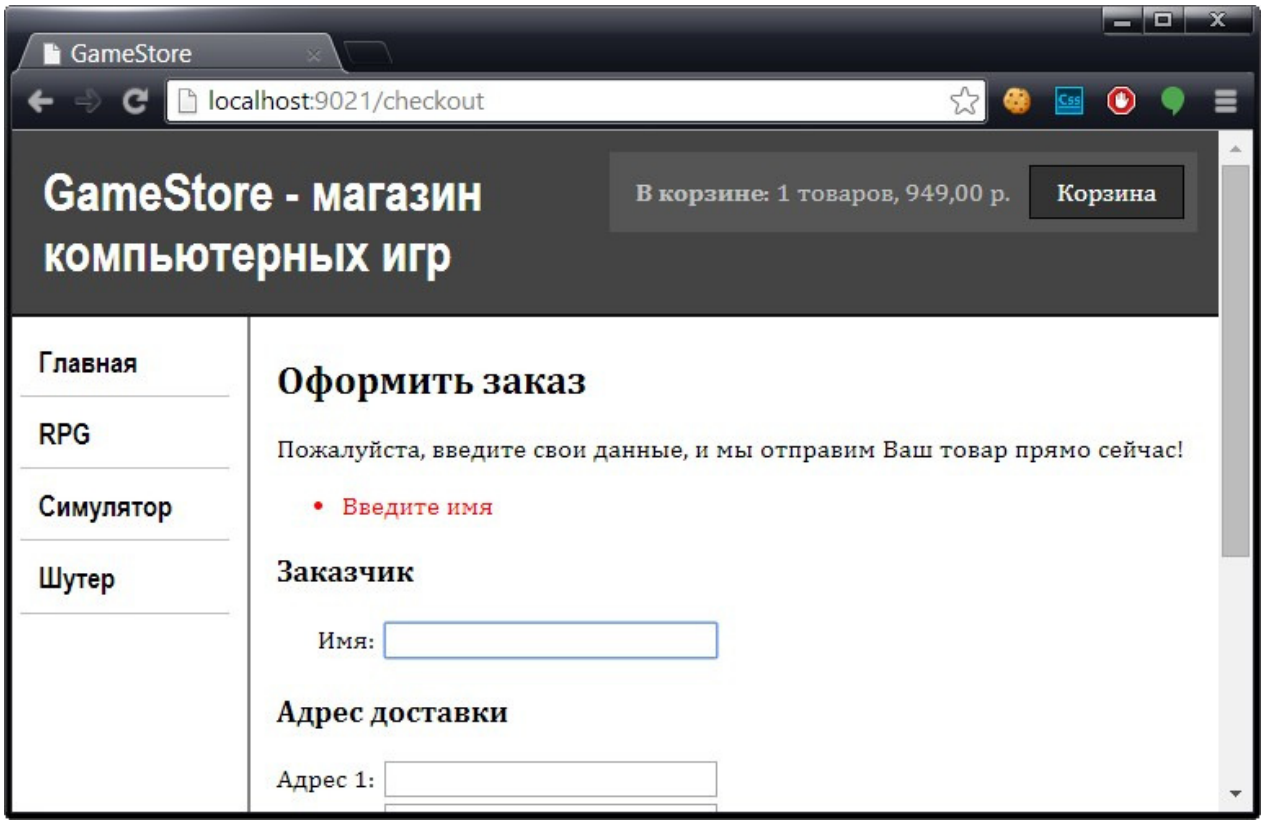
```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Checkout.aspx.cs" In
    MasterPageFile="~/Pages/Store.Master" %>

<asp:Content ID="Content1" ContentPlaceHolderID="bodyContent" runat="server">
    <div id="content">

        <div id="checkoutForm" class="checkout" runat="server">
            <h2>Оформить заказ</h2>
            Пожалуйста, введите свои данные, и мы отправим Ваш товар прям

        <div id="errors" data-valmsg-summary="true">
            <ul>
                <li style="display:none"></li>
            </ul>
            <asp:ValidationSummary ID="ValidationSummary1" runat="server" />
        </div>

            <h3>Заказчик</h3>
            <div>
                <label for="name">Имя:</label>
                <input id="name" name="name" data-val="true" data-val-required

            </div>

            <h3>Адрес доставки</h3>
            <div>
                <label for="line1">Адрес 1:</label>
                <input id="line1" name="line1" />
            </div>
            <div>
                <label for="line2">Адрес 2:</label>
                <input id="line2" name="line2" />
            </div>
            <div>
                <label for="line3">Адрес 3:</label>
                <input id="line3" name="line3" />
            </div>
            <div>
                <label for="city">Город:</label>
                <input id="city" name="city" />
            </div>

            <h3>Детали заказа</h3>
```

Библиотека ненавязчивой проверки достоверности от Microsoft, добавленная в проект, ищет специфичные атрибуты data и применяет их для конфигурирования библиотеки jQuery Validation, когда HTML-документ загружен браузером. Это означает, что добавлять какой-либо код JavaScript к веб-форме не понадобится, а конфигурация маршрутизации создается автоматически с учетом упомянутых атрибутов.

Достаточно знать, что атрибут data-valmsg-summary указывает элемент, который должен использоваться для отображения итоговой информации по ошибкам проверки — код проверки достоверности находит добавленные списковые элементы и применяет их для размещения сообщений проверки. Из-за повторного использования того же самого элемента, содержащего элемент управления Validation Summary, сообщения об ошибках проверки достоверности клиентской и серверной стороны отображаются в одном месте (с теми же стилями CSS).

Для выполнения проверки достоверности необходимо добавить к элементам input два атрибута data. Применение атрибута data-val со значением true указывает, что для элемента должна быть выполнена проверка достоверности. Атрибут data-val-required определяет тип проверки достоверности (в этом случае — обязательное предоставление значения) и сообщение, которое будет отображаться при наличии ошибки проверки. Доступны разные атрибуты data, позволяющие задавать различные типы проверки достоверности, но далее будет использоваться только data-val-required.

В результате добавления новых атрибутов мы настраиваем проверку достоверности клиентской стороны для поля name. Чтобы взглянуть, как это работает, запустите приложение, добавьте несколько товаров в корзину и перейдите к оплате. Отправьте форму с информацией о доставке, не указав имя; вы увидите сообщение проверки достоверности, показанное на рисунке ниже:



Проверка достоверности была выполнена без отправки запроса серверу, и

пользователь не сможет отправить форму до тех пор, пока не будут исправлены все ошибки проверки достоверности клиентской стороны. Это сокращает количество запросов, которые сервер должен обрабатывать, и предоставляет более быстрый отклик пользователю.

Создание серверного элемента управления

С помощью JavaScript-библиотек проверки достоверности решается только одна из проблем. Мы можем смешивать проверку достоверности клиентской стороны с привязкой модели, но по-прежнему дублируем логику проверки в двух местах — в атрибутах data внутри веб-формы и в атрибутах Required, применяемых к классу Order.

Создав элемент управления, генерирующий элемент input, который имеет атрибуты data, полученные от класса модели данных, мы решим вторую проблему. Для этого используется серверный элемент управления, поэтому при построении приложения GameStore мы продемонстрируем множество новых приемов, хотя можно было бы легко достигнуть аналогичного результата с помощью пользовательского элемента управления.

Щелкните правой кнопкой мыши на папке Controls в окне Solution Explorer и выберите в контекстном меню пункт Add New Item (Добавить новый элемент). Выберите в списке элемент ASP.NET Server Control (Серверный элемент управления ASP.NET), укажите VInput в качестве имени и щелкните на кнопке Add (Добавить), чтобы создать новый элемент. Серверный элемент управления состоит из единственного класса C# и вы увидите, что Visual Studio создаст в папке Controls новый файл по имени VInput.cs. Приведите содержимое этого файла в соответствие со следующим кодом:

```
using System;
using System.ComponentModel.DataAnnotations;
using System.Reflection;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace GameStore.Controls
{
    public class VInput : WebControl
    {
        private string nspace = "GameStore.Models";
        private string model = "Order";

        public string Namespace
        {
            get { return nspace; }
            set { nspace = value; }
        }
        public string Model
        {
            get { return model; }
            set { model = value; }
        }

        public string Property { get; set; }

        protected override void RenderContents(HtmlTextWriter output)
        {

            output.AddAttribute(HtmlTextWriterAttribute.Id, Property.ToLower()
            output.AddAttribute(HtmlTextWriterAttribute.Name, Property.ToLower

            Type modelType = Type.GetType(string.Format("{0}.{1}", Namespace,
            PropertyInfo propInfo = modelType.GetProperty(Property);
            var attr = propInfo.GetCustomAttribute<RequiredAttribute>(false);
            if (attr != null)
            {
                output.AddAttribute("data-val", "true");
                output.AddAttribute("data-val-required", attr.ErrorMessage);
            }
            output.RenderBeginTag("input");
            output.RenderEndTag();
```

Мы не собираемся здесь вникать в детали работы серверных элементов управления (более подробно вы можете прочитать в статье "Специальные элементы управления"), однако сообщим, что элемент управления VInput находит атрибут Required в свойстве класса модели данных и генерирует элемент input, который содержит требуемые атрибуты data. Мы сохранили элемент управления VInput простым, поэтому он не обрабатывает других атрибутов проверки достоверности кроме Required, и по умолчанию он сконфигурирован на использование класса GameStore.Models.Order.

Этот элемент управления должен быть зарегистрирован с помощью ASP.NET Framework, и это делается за счет модификации файла Web.config, как показано в примере ниже:

```
<system.web>
    ...
    <pages controlRenderingCompatibilityVersion="4.0">
      <controls>
        <add tagPrefix="GS" tagName="CategoryLinks" src="~/Controls/CategoryLi
        <add tagPrefix="GS" tagName="CartSummary" src="~/Controls/CartSummary.
        <add tagPrefix="SX" namespace="GameStore.Controls" assembly="GameStore
      </controls>
    </pages>
</system.web>
```

Для серверных элементов управления указывается пространство имен и сборка, поэтому среда ASP.NET Framework найдет все элементы управления автоматически. Серверные и пользовательские элементы управления не могут разделять один и тот же префикс, так что был выбран префикс SX.

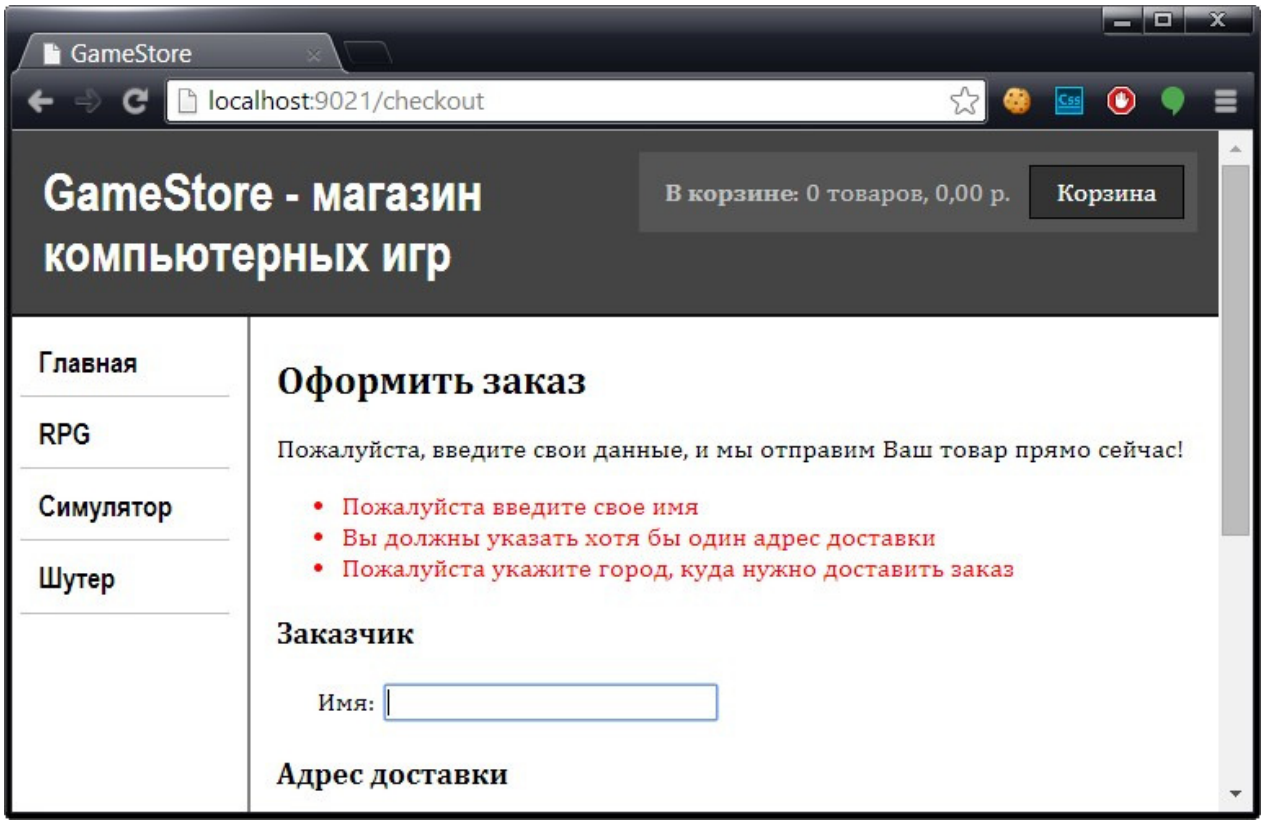
Применение серверного элемента управления

Осталось только применить серверный элемент управления VInput в файле веб-формы \Pages\Checkout.aspx, как показано в примере ниже:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Checkout.aspx.cs" In
    MasterPageFile="~/Pages/Store.Master" %>

<asp:Content ID="Content1" ContentPlaceHolderID="bodyContent" runat="server">
    <div id="content">

        <div id="checkoutForm" class="checkout" runat="server">
            <h2>Оформить заказ</h2>
            Пожалуйста, введите свои данные, и мы отправим Ваш товар прям

        <div id="errors" data-valmsg-summary="true">
            <ul>
                <li style="display:none"></li>
            </ul>
            <asp:ValidationSummary ID="ValidationSummary1" runat="server" />
        </div>

            <h3>Заказчик</h3>
            <div>
                <label for="name">Имя:</label>
                <SX:VInput Property="Name" runat="server" />
            </div>

            <h3>Адрес доставки</h3>
            <div>
                <label for="line1">Адрес 1:</label>
                <SX:VInput Property="Line1" runat="server" />
            </div>
            <div>
                <label for="line2">Адрес 2:</label>
                <SX:VInput Property="Line2" runat="server" />
            </div>
            <div>
                <label for="line3">Адрес 3:</label>
                <SX:VInput Property="Line3" runat="server" />
            </div>
            <div>
                <label for="city">Город:</label>
                <SX:VInput Property="City" runat="server" />
            </div>

            <h3>Детали заказа</h3>
```

Элемент управления VInput используется даже для свойств, к которым атрибут required не применялся — как обычно, не следует забывать о сопровождении приложения, поэтому мы хотим, чтобы генерируемая HTML-разметка автоматически отражала любые новые атрибуты, которые будут добавлены в класс модели данных.

В результате мы используем код JavaScript для выполнения проверки достоверности клиентской стороны с применением атрибутов проверки, добавляемых в класс модели данных. На рисунке ниже продемонстрирован эффект от попытки отправки пустой формы Checkout; обратите внимание на сообщения проверки, которые соответствуют значениям ErrorMessage, установленным для атрибутов Required ранее:



В этой статье мы сохранили серверный элемент управления простым, но его функциональность легко расширить для поддержки других атрибутов проверки достоверности и типов HTML-элементов.

С одной стороны, жаль, что встроенная проверка достоверности клиентской стороны работает плохо, однако с другой стороны, это позволяет нам продемонстрировать возможность настройки или замены практически любого средства, предлагаемого платформой ASP.NET Framework.