ВЫСШЕЕ ОБРАЗОВАНИЕ

JAVASCRIPT
Визуальные редакторы

В. В. Янцев

ЛАНЬ

E.LANBOOK.COM

В. В. ЯНЦЕВ

JAVASCRIPT ВИЗУАЛЬНЫЕ РЕДАКТОРЫ

Учебное пособие

ЛАНЬ

· САНКТ-ПЕТЕРБУРГ · МОСКВА · КРАСНОДАР ·
· 2022 ·

УДК 004
ББК 32.97я723

Я 65 **Янцев В. В.** JavaScript. Визуальные редакторы : учебное пособие для СПО / В. В. Янцев. — Санкт-Петербург : Лань, 2022. — 168 с. : ил. — Текст : непосредственный.

ISBN 978-5-8114-8943-5

Подавляющее большинство ныне действующих сайтов сделано при помощи CMS. А такие системы невозможно представить без визуальных редакторов страниц. WYSIWYG-редакторы позволяют администратору управлять содержимым ресурса, не будучи знакомым даже с азами программирования.

Многие разработчики используют не готовые CMS, а пишут собственные. Данная книга призвана помочь им в этом непростом деле. На ее страницах описаны четыре WYSIWYG-редактора — на разный вкус и для разных задач. Все системы являются оригинальными и не содержат каких-либо заимствований кода. Функциональные возможности приведенных разработок имеют исчерпывающие описания и разъяснения.

Разобранные в книге визуальные редакторы могут быть интегрированы в уже существующую CMS или работать напрямую с сайтом без сопутствующих инструментов. Особо надо отметить, что эти редакторы позволяют администратору сайта видеть, как будет выглядеть страница еще до того, как он запишет внесенные изменения в файл или базу данных.

Книга имеет сайт поддержки, где читатель может ознакомиться с редакторами и опробовать их в действии. Кроме того, вы можете скачать zip-архив со всеми файлами и запустить их на своем компьютере. Для этого книга содержит подробные инструкции по созданию локального хостинга на ПК.

Соответствует современным требованиям Федерального государственного образовательного стандарта среднего профессионального образования и профессиональным квалификационным требованиям.

Рекомендовано в качестве дополнительной литературы для студентов, обучающихся в средних профессиональных учебных заведениях по направлению «Информатика и вычислительная техника».

УДК 004
ББК 32.97я723

Обложка
П. И. ПОЛЯКОВА

ОГЛАВЛЕНИЕ

1. Введение

Автор начал заниматься программированием в 2003 году. С 2004 года за довольно короткий промежуток времени успел поработать в нескольких студиях web-дизайна и вот на что обратил внимание: в те годы почти каждая такая студия, большая или маленькая, старалась написать собственные «движки» для сайтов, или, иначе говоря, CMS. Получались они очень разными — от весьма простеньких и неказистых до серьезных профессиональных разработок.

Не остался в стороне от этой моды и автор, только написал такую систему не во взаимодействии с группой программистов и дизайнеров, а полностью самостоятельно. Называлась она непритязательно — СОУС (Система оперативного управления сайтом). Естественно, автор постоянно ее совершенствовал, наращивал функциональные возможности и т. д. До 2016 года я и многие мои клиенты вполне успешно пользовались данной разработкой.

Однако в это время происходили два важных процесса:

– зарождение HTML 5;

– появление на рынке большого числа мощных CMS.

Наступил момент, когда автор уже не мог в одиночку конкурировать с такими проектами, как WordPress, Joomla, Drupal, 1С-Битрикс и другими. Кроме того, замена ингредиентов СОУСа с HTML 4 на HTML 5 требовала немалых усилий.

В итоге СОУС был отправлен в архив, после чего автор занялся другими разработками, но по-прежнему интересовался тем, что происходит в индустрии создания «движков». А самое главное, продолжал заниматься написанием WYSIWYG-редакторов, которые являются составными частями любых CMS.

Таким образом, на сегодняшний день у меня накопился немалый опыт создания самых разных визуальных редакторов для самых разных web-проектов. Этим опытом я и хочу поделиться на страницах данной книги.

1.1. О чем эта книга

Книг о синтаксисе языка программирования JavaScript много. Я бы даже сказал, очень много. Конечно, каждая из них по-своему хороша, интересна и полезна, но в конечном счете все они об одном и том же.

Есть ряд книг, в которых излагаются не только основы языка, но и даются примеры сценариев. Они могут объяснить начинающему разработчику, каким образом создаются реальные проекты.

Но изданий, рассказывающих о такой популярной среди программистов теме, как создание CMS, и, более конкретно, о создании WYSIWYG-редакторов, практически нет. Из известных мне книг по JavaScript лишь в двух упоминается о существовании таких редакторов и лишь в одной (изданной много лет тому назад) рассматривается довольно элементарный пример. Частично я попытался осветить этот вопрос в своем учебнике «JavaScript. Готовые программы». Но и

там, несмотря на описание нескольких очень простых разработок, информация по теме была крайне незначительна.

Как уже понятно из сказанного выше, эта книга познакомит читателя с процессом создания профессиональных визуальных редакторов. Причем не одного, а сразу нескольких. Если быть точнее — четырех. Вы узнаете, как разработать сайт с контентом, который может редактироваться онлайн. Узнаете, как выглядит структура типичного WYSIWYG-редактора, как с помощью такого редактора можно написать web-страницу и по мере необходимости вносить изменения в текст, добавлять изображения, таблицы, линии, символы и многое другое.

Но это будут не просто теоретические рассуждения. Если в книге представлены лишь фрагменты кода, то, скачав по ссылке **https://editjs.ru/EDIT.zip** zip-архив, вы получите в свое распоряжение весь проект, включая демонстрационный сайт, 4 редактора, файлы сценариев и таблиц стилей, а также весь контент, использованный в данной системе.

И это еще не все. Вы можете заглянуть на демонстрационный сайт проекта **https://editjs.ru/** и проверить в действии и сайт, и редакторы. Единственное, на что необходимо обратить внимание: в демонстрационных редакторах функции записи изменений в контенте отключены. Сделано это для того, чтобы избавить особо пытливые умы от соблазна оставить после себя на сайте что-нибудь непотребное.

Наконец, еще один важный момент. Как понятно из названия книги, главное действующее лицо в ней — язык программирования JavaScript. 90% кода любого из редакторов — на JavaScript. Поэтому было бы неплохо, если бы читатели оказались знакомы с основами и синтаксисом этого языка.

1.2. Особенности изложения материала

В книге четырнадцать глав.

Первая — «Введение». Вы сейчас читаете ее.

Во второй мы познакомимся с «фундаментом», на котором построены «здания» редакторов, а также подробнее разберем, что же все-таки нам предстоит написать.

Третья глава посвящена созданию среды разработки на персональном компьютере. Если вы хотите скачать zip-архив и экспериментировать на своем ПК, необходимо создать на нем локальный хостинг, установив несколько программ.

Из главы 4 вы узнаете, как проверялись и тестировались файлы нашего проекта.

Глава 5 познакомит вас со структурой проекта и составом zip-архива.

В шестой мы подробно разберем, как создать демонстрационный сайт и выделить в контенте редактируемую область.

В седьмой рассмотрим структуру редакторов.

Название восьмой главы говорит само за себя — «Базовые компоненты редакторов». Мы увидим, как они устроены и какими панелями снабжены.

В главе 9 будет много кода. Мы рассмотрим общие фрагменты сценариев, написанные на JavaScript.

Главы 10–13 непосредственно познакомят вас с кодом, который отвечает за создание и редактирование контента.

Глава 14 рассказывает о записи редактируемых данных в соответствующие файлы.

И в самой короткой главе 15 подведем итоги нашей работы.

На что еще хотел бы обратить внимание читателя: во избежание терминологических споров автор ориентировался на определения, данные на страницах сайта **https://developer.mozilla.org/ru/**.

1.3. Оформление сценариев

Сценарии имеют различное типографское оформление в соответствии с их размещением в тексте.

Если код в книге выделен в отдельный блок, то он оформлен моноширинным шрифтом, например так:

```
function del()
   {
   view.style.visibility="hidden";
   bas.visibility="hidden";
   pict.src="img/net.jpg";
   }
```

Если фрагменты сценария внедрены непосредственно в текст, то в этом случае части кода выделены полужирным шрифтом, например так:

присваиваем переменной **idc** новое значение

Обратите внимание: в некоторых блоках программ сделан перенос части кода на вторую строку (из-за недостатка ширины страницы в книге). В реальном сценарии код записывается одной строкой. Запомните правило: все переносы строк кода существуют только в их типографском воспроизведении. Если вы в дальнейшем столкнетесь с подобной ситуацией, учитывайте данный аспект.

Еще один момент. Многие фрагменты кода в нашем проекте написаны на PHP. В сети циркулирует придуманная группой программистов концепция так называемой стандартизации кода — PHP Standards Recommendations (PSR). Автор считает некоторые рекомендации из данной концепции как минимум усложняющими чтение программ. Поэтому при создании проекта автор руководствовался следующими принципами:

– код PHP должен соответствовать официальным стандартам самого языка, а не рекомендациям каких-либо, пусть даже авторитетных, групп;

– код должен быть оформлен таким образом, чтобы даже самые сложные его фрагменты легко читались: это в первую очередь касается отступов, распределения кода по строкам и расположения фигурных скобок.

2. Прежде чем начать

До того как мы приступим к написанию редакторов, автор посчитал необходимым дать некоторую предварительную информацию:
– пояснить смысл терминов, которые нам придется использовать неоднократно;
– рассказать, какие технологии и методы необходимо будет применить в нашем проекте;
– осветить вопрос, устарели или нет приемы редактирования контента, которые еще некоторое время назад составляли фундамент почти любого визуального редактора;
– четко определить, что же мы с вами в конце концов напишем.

Поэтому данный материал я решил выделить пусть и в небольшую, но самостоятельную главу.

2.1. CMS

Представим такую ситуацию: вы разработали сайт, состоящий из нескольких HTML-страниц, и разместили его в сети. Пройдет немного времени, и вы обнаружите, что информацию на одной или нескольких страницах необходимо отредактировать, а может, и кардинально изменить. Даже самые стабильные интернет-ресурсы нуждаются в периодическом обновлении, а что говорить про сайты, которые рассказывают о быстро меняющихся вещах, например о строительстве жилого микрорайона или о новинках в мире компьютеров. В таких случаях обновление информации происходит чуть ли не каждый день.

И вот вы стоите перед задачей ежедневно редактировать одну или несколько страниц и ставить их вместо устаревших версий. Выглядит это примерно так: открыли на своем компьютере папку с файлами сайта, добавили новые рисунки, схемы или фотографии, отредактировали страницу, подключились к хостингу, закачали новые изображения, заменили страницу, запустили браузер, проверили полученный результат. Если что-то не понравилось, вновь правите страницу и закачиваете ее новую версию на хостинг. Уже чувствуется, что это довольно хлопотное занятие. Между тем, если бы ваш сайт был снабжен CMS, все оказалось бы гораздо проще: запустили в браузере систему управления и выполнили все манипуляции через нее. Не надо подключаться к хостингу, не надо переносить страницу на свой компьютер, не надо закачивать отредактированную страницу через файловый менеджер. Еще раз повторюсь: все делается через браузер, без предварительной обработки данных на своем компьютере.

Обычно CMS — это симбиоз двух продуктов: конструктора сайта и системы управления сайтом. В сети рекламируется много различных CMS, например уже упоминавшиеся во введении WordPress, Joomla, Drupal, 1С-Битрикс и другие.

Аббревиатура CMS происходит от английского Content management system, что переводится как «система управления содержимым». Многие

современные сайты (если не большинство) управляются такими системами. CMS позволяют администратору вносить любые изменения на страницы сайта в онлайн-режиме. Для этого системы управления снабжают WYSIWYG-редакторами.

2.2. WYSIWYG

Сайты пишутся на языках программирования HTML и JavaScript, поэтому редактирование web-страницы — по сути, процесс изменения ее кода. Для профессионального разработчика это привычное занятие. Но ведь большинство сайтов программисты создают не для себя, а для клиентов. Много ли среди них специалистов, досконально знающих HTML и JavaScript? Ответ — единицы. Значит, для большинства клиентов необходим механизм, который бы позволял создавать на сайте новые страницы и редактировать существующие без специальных знаний.

Поэтому креативные программисты придумали не простой редактор, а WYSIWYG (или, как еще говорят, визуальный редактор). Он отображает редактируемые материалы практически в том же самом виде, как и на странице сайта. То есть нажали кнопку BOLD — фрагмент текста действительно выделился жирным. Нажали кнопку IMG — и появилась настоящая фотография, а не ее HTML-код.

Термин WYSIWYG — это аббревиатура, произошедшая от английского выражения «What You See Is What You Get», что в переводе означает «что вы видите, то вы и получаете». В нашем случае смысл этого выражения таков: что вы видите в редакторе, то и будет на странице сайта. Очень удобно для создания и редактирования контента!

Отступление от темы
Вообще, термин WYSIWYG относится не только к разработке визуальных редакторов для web-проектов. Так называют программы, в которых содержание выглядит точно так же, как и реальный документ. Например, всем известный Word — это WYSIWYG-редактор.

2.3. Атрибут contentEditable

Данный атрибут, если он указан в элементе, сообщает браузеру, что этот элемент — редактируемый.

В чем польза **contentEditable**? Допустим, есть слой **div**, в который мы загружаем предназначенный для редактирования фрагмент web-страницы. Если у **div** установлен данный атрибут, мы можем вставлять в слой курсор, выделять, удалять, добавлять текст, в том числе вводя его с клавиатуры. То есть у нас уже получился простейший редактор. А если к нему добавить кнопки для импорта в **div** HTML-кода рисунков, таблиц, линий и т. д., то у нас выйдет уже полноценный редактор! Естественно, написанное с нуля или исправленное

содержимое слоя можно будет передать специальной программе для записи в файл или базу данных.

Мы воспользуемся таким подходом при написании второго, третьего и четвертого редакторов.

Некоторые программисты указывают атрибут **contentEditable** без значения. Это не совсем корректно, хотя и будет работать. Как утверждает **https://developer.mozilla.org/ru/**, правильно писать

```
contentEditable="true"
```

Атрибут можно указать непосредственно в теге, например так:

```
<div id="edit" contenteditable="true">
```

или присвоить его в JavaScript-сценарии:

```
document.getElementById("edit").
           setAttribute("contenteditable", true);
```

Обратите внимание! Приведенный выше фрагмент — это пример переноса строки, сделанный в книге из-за недостатка ширины страницы. В реальном файле данный код записывается одной строкой! Автор уже предупреждал о таких переносах во введении. Думаю, читателям все ясно и в дальнейшем не нужно повторять это правило.

Добавлю, что операция присвоения элементу атрибута в JavaScript-сценарии должна производиться после загрузки страницы:

```
addEventListener("load", function()
{
document.getElementById("edit").
           setAttribute("contenteditable", true);
});
```

2.4. Свойство designMode

Свойство **designMode** имеет много общего с **contentEditable**. Отличие в том, что **designMode** устанавливается не для отдельного элемента, а для всего документа в целом. Когда свойство включено (**document.designMode="on"**), можно редактировать любую HTML-страницу полностью. Эта особенность натолкнула программистов на идею создания визуальных редакторов, используя загрузку целой страницы или ее отдельного фрагмента в **iframe**.

Процедура включения режима редактирования может выглядеть, например, так. Пусть на странице редактора есть фрейм, в который загружается другая HTML-страница, предназначенная для обработки. Тогда в JavaScript-коде редактора необходимо записать всего одну строку:

```
frames[0].document.designMode="on";
```

Естественно, данная операция, как и в предыдущем случае, должна производиться после загрузки страницы:

```
addEventListener("load", function()
{
frames[0].document.designMode="on";
});
```

Мы используем свойство **designMode** в первом редакторе, но будем обрабатывать не всю страницу сайта, а только ее часть — область контента.

2.5. Метод getSelection

Теперь мы переходим к методам, которые предоставляет в наше распоряжение JavaScript.

Первым упомянем **getSelection**. Тут все очень просто: метод возвращает объект **Selection**, который содержит выделенный пользователем текст. Разработчик имеет возможность на программном уровне манипулировать выделенным фрагментом: форматировать его, копировать, удалять, изменять.

Как выглядит применение метода на практике? Например, у вас есть редактируемая область контента. Вы выделили в этой области одно или несколько слов и нажали кнопку BOLD. К тексту, содержащемуся в объекте **Selection**, будет добавлена пара тегов **b** — открывающий и закрывающий. После этого текущее выделение будет преобразовано к полужирному начертанию.

Аналогично происходит, если вы просто вставили курсор в какое-либо место редактируемой области. Объект **Selection** в этом случае хранит точку вставки курсора. В эту точку можно поместить рисунок, таблицу, линию, символ и т. д.

2.6. Отменен ли execCommand?

Метод **execCommand** служит механизмом внесения изменений в содержимое web-страницы. Метод получает в качестве аргумента имя команды. При простом форматировании обычно достаточно одного аргумента, например **execCommand("Bold")**. Однако, в некоторых случаях требуется передать три аргумента: имя команды; значение, указывающее, отображать или нет пользовательский интерфейс; адрес или текст. Второй аргумент обычно указывают **false**. Это означает, что третий аргумент передается в метод непосредственно из программы. Если указать второй аргумент **true**, то метод будет ожидать ввода третьего аргумента пользователем. Более аргументоемкие варианты **execCommand** применяются, например, для вставки ссылки или изображения в редактируемую страницу. В этих случаях сначала указывают имя, затем **false**, затем адрес ссылки или картинки.

Метод **execCommand** создан давно. Многие годы на нем строился фундамент большинства редакторов. Но теперь с ним не все так просто. В связи с появлением HTML5 некоторые из ранее существовавших команд на сегодняшний день потеряли свою актуальность (например, fontSize или fontName).

Более того, на уже упоминавшемся сайте **https://developer.mozilla.org/ru/**, на странице, посвященной этому методу, можно прочитать следующее:
«Document.execCommand()
Вышла из употребления
Эта возможность вышла из употребления. Хотя она может продолжать работать в некоторых браузерах, её использование не рекомендуется, поскольку она может быть удалена в любое время. Старайтесь избегать её использования».

Однако равнозначной замены этому методу на сегодняшний день нет. Насколько мне известно, альтернативы пока существуют только в виде не очень далеко продвинувшихся проектов.

Тем не менее при создании последнего, четвертого, редактора мы воспользуемся этим методом. Редактор у нас так и будет называться — ретро.

Почему же, невзирая на предупреждения Mozilla, автор тем не менее предложил редактор на основе такой технологии? Причин четыре.

Первая. Метод продолжает работать в требуемых нам объемах во всех браузерах, с которыми мне приходилось иметь дело.

Вторая. **execCommand** до сих пор используется во многих визуальных редакторах, рекламируемых в сети: например, в CKEditor или в NicEdit.

Третья. Благодаря тому, что многие команды метода унифицированы, редакторы на основе данного метода получаются самыми «легкими». Достаточно сравнить сценарии из zip-архива. Файл последнего редактора **editor4.js** самый короткий и занимает наименьшее место на диске.

Четвертая, самая важная. Автор не уверен, что на этом методе окончательно поставлено клеймо устаревшего. Не случиться ли с **execCommand** то же самое, что и с HTML? Для тех, кто не в курсе, короткая справка. Был период, когда Консорциум Всемирной Паутины — World Wide Web Consortium (W3C) — прекратил дальнейшую разработку языка HTML и попытался заменить его на XHTML. Из этого ничего не вышло, и тогда HTML возродился в новой, усовершенствованной и «очищенной» версии — в пятой. И весьма успешно применяется на протяжении уже нескольких лет. Так вот, не случится ли подобная история и с **execCommand**? Для возрождения метода надо совсем немного: удалить вышедшие из употребления команды и оставить актуальные. Ну, может быть, добавить еще несколько команд. И у метода начнется вторая жизнь.

Поэтому четвертый редактор был создан из соображений, что еще рано ставить окончательную точку в деле **execCommand**. Прекратят браузеры поддержку этого метода — вы просто не станете пользоваться последним редактором. Поддержка сохранится — значит редактор останется «на плаву».

У читателя наверняка возникнет вопрос: если альтернативы методу нет, то как же сделаны первые три редактора? На самом деле есть некоторые другие — современные — технологии, которые позволяют получить необхо-

димые результаты, но, правда, с большими затратами кода. Они не аналогичны **execCommand**, но вполне работоспособны. С ними мы сейчас и познакомимся.

2.7. Методы createElement и surroundContents

Первый метод позволяет создать новый элемент документа, а второй — добавить его к выделению, полученному методом **getSelection**. Например, у нас в редактируемой области есть слово «Важно», которое мы хотим выделить полужирным шрифтом. Для этого можно написать вот такой сценарий из 4 строк:

```
let sel=document.getSelection();
let rng=sel.getRangeAt(0);
let elm=document.createElement("b");
rng.surroundContents(elm);
```

Разберем, что происходит в каждой строке кода.

1. Создаем объект, содержащий выделенный фрагмент текста.

2. Используя метод **getRangeAt**, создаем объект **Range** с диапазоном, равным выделению. Индекс **0** указывает, что выбранный диапазон у нас единственный.

3. В теле документа методом **createElement** создаем открывающий и закрывающий теги элемента **b** (тег для полужирного начертания шрифта).

4. Методом **surroundContents** добавляем эти теги перед началом выделения и после конца выделения.

Теперь редактируемое слово выглядит так: **Важно**.

Показанный выше пример служит основой первых трех редакторов. Этот способ позволяет добавлять теги не только к выделенному тексту, но и вставлять элементы в положение курсора в редактируемой области, например рисунки, таблицы, линии и т. д.

2.8. Что и как мы напишем

Собственно говоря, все уже написано. Наша задача — разобраться в принципах создания редакторов и понять технологию их применения.

Итак, у нас есть четыре редактора:

– традиционный;
– с фиксированными стилями;
– креативный;
– ретро.

В этом порядке они и будут разобраны в книге. Откуда взялись такие названия, выясним в процессе подробного описания каждого из редакторов. Подчеркну, что особенно выделяется своей необычностью третий редактор, получивший ярлык «креативный». Но не станем раскрывать секреты раньше времени. В нужный момент все разъяснится.

Кстати, в креативном редакторе предусмотрена процедура входа для администратора. В остальных — нет. Сейчас «проникнуть» в остальные

редакторы без пароля может любой желающий. Сделано это вот по какой причине. Особенность «конструкции» третьего редактора такова, что без системы проверки администратора показывать его просто нет смысла. Почему — поясню в свое время. Остальные редакторы не обременены проверками, так как автор не хотел связывать читателей какими-то уже готовыми решениями. Тем, кто возьмется их воспроизводить, просто хочу напомнить, что вам придется добавить код, который будет отвечать за «пропускной» режим.

Запись отредактированного контента у всех редакторов выполняет одна и та же программа на PHP. В ней тоже не предусмотрена проверка, в «законном» ли порядке поступают данные или от недоброжелателя. Поэтому и в данный файл нужно будет добавить код проверки.

Подводя черту под темой безопасности, можем констатировать следующее: проверка входных данных при записи отредактированного контента и при входе в редактор зависит от способа аутентификации и авторизации администратора. Каковы будут эти способы — решать вам.

Пойдем дальше. Если есть редакторы, значит, они должны что-то редактировать. В качестве ресурса для экспериментов у нас есть демонстрационный сайт, посвященный космической тематике. В нем пять страниц. Одна уже наполнена контентом, остальные — пустые. Таким образом, тот, кто возьмется экспериментировать с редакторами, может как изменять существующие данные, так и создавать новые на пустых страницах. При этом еще раз обращаю внимание: в демонстрационных редакторах функции записи изменений в контенте отключены.

При создании проекта не использовалась база данных. Вся информация, размещенная на сайте, хранится в простых текстовых файлах. Сделано это по двум причинам:

– сайту нечего скрывать, так как никакой секретной информации он не содержит;

– так проект получается проще и доступнее для повторения.

Естественно, при желании вы можете переписать соответствующие фрагменты кода для взаимодействия с базами данных.

Создавая файлы управления редакторами, автор старался соблюсти баланс между оптимизацией и читабельностью кода. При этом все такие файлы написаны на чистом, или, как еще говорят, ванильном, JavaScript, то есть без использования библиотек и фреймворков типа jQuery.

Отступление от темы

*Vanilla JS — **http://vanilla-js.com/** — шуточный сайт кроссплатформенного фреймворка Vanilla JS, под которым подразумевается чистый JavaScript. Его автор ставит целью напомнить разработчикам, что довольно часто при создании кода можно обойтись без библиотек и фреймворков. В очень многих случаях писать программы на чистом JavaScript гораздо продуктивнее и рациональнее. От имени сайта и образовалось это выражение — «ванильный JavaScript», то есть JavaScript в его первозданном виде.*

Описанные в книге программы могут быть использованы как составная часть какой-либо самодельной CMS (понимаю, что не один я занимаюсь подобным творчеством) или как ориентир при написании разработчиками собственных редакторов.

Обращу ваше внимание на еще один важный момент. Если редакторы применяются самостоятельно, необходимо предусмотреть дополнительную программу, которая позволяла бы закачивать и удалять изображения, создавать новые и удалять старые страницы, редактировать меню и блок рекламы. Конечно, это можно делать вручную, но очень неудобно. Гораздо производительнее доверить эти процессы специальному web-приложению к редактору. Для выполнения перечисленных задач вполне подойдет одностраничное приложение.

3. Среда разработки

Для написания и тестирования сайта и редакторов нам в обязательном порядке необходимо создать на своем компьютере локальный хостинг.

Вообще, хостинг — это, упрощенно говоря, компьютер, предназначенный для размещения на нем сайтов и обеспечивающий к ним доступ из Интернета. Локальный хостинг — набор программ на вашем персональном компьютере, которые позволяют имитировать реальный хостинг и проводить тестирование ваших сайтов (включая HTML-страницы и серверные скрипты) перед их размещением в сети. Кстати, обратите внимание: локальный хостинг работает без подключения к Интернету!

Первая ошибка, которую допускают многие начинающие программисты, состоит в попытках создать локальный хостинг, следуя описаниям из Интернета, где предлагаются «навороченные» методы. Их авторы выделяют на диске С отдельные сектора, создают кучу дополнительных директорий, вносят большое количество изменений в конфигурационные файлы. В результате неопытный человек начинает путаться в этих сложных описаниях, ошибаться в настройках программного обеспечения и лишь путем многочисленных проб и консультаций с энной попытки наконец получает требуемый результат (а иногда так и не получает, из-за чего начинает искать другие описания на данную тему). Между тем создание хостинга на своем ПК — дело несложное.

Чтобы получить результаты, описанные в данной книге, вам понадобиться установить на компьютер распространяемый пакет компонентов **Visual C++** от Microsoft, сервер **Apache** и интерпретатор **PHP**. Описание технологии их установки — в следующих разделах.

Полноценную среду разработки невозможно представить без установки на ваш компьютер настоящего текстового редактора, специально предназначенного для создания программ. Конечно, весь код можно писать в обычном «Блокноте». Но гораздо лучше и рациональнее пользоваться специализированными редакторами. Такие приложения намного удобнее: они подсвечивают код, предлагают синтаксические подсказки, позволяют менять кодировку документов, сохраняют файлы в разных форматах.

В нашем случае необходим редактор, который позволит:

– выполнять качественную разметку;

– создавать файлы с таблицами стилей;

– писать сценарии на JavaScript;

– делать на языке программирования PHP компоновщики HTML-страниц.

Этим принципам удовлетворяют многие редакторы. Но лучше всего, на мой взгляд, **Notepad++**. Где его взять и как установить, рассказано в пятом разделе этой главы.

Итак, вводная часть завершена — приступим к созданию локального хостинга. А начнем мы с выяснения разрядности вашей ОС.

Обратите внимание: все процедуры описаны для компьютера с операционной системой Windows 10.

3.1. Выясняем разрядность ОС

Перед тем как мы создадим на компьютере среду разработки, необходимо выяснить разрядность вашей операционной системы, чтобы знать, какие файлы скачивать для локального хостинга.

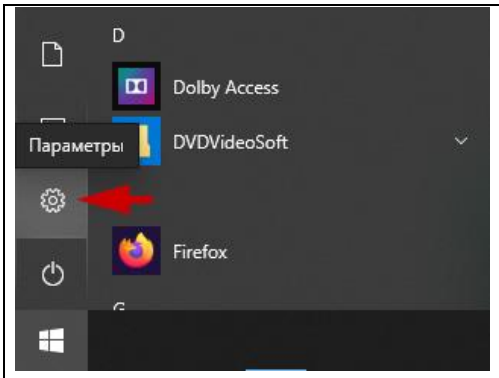

Рис. 3.1.1. Кнопка «Параметры» в меню

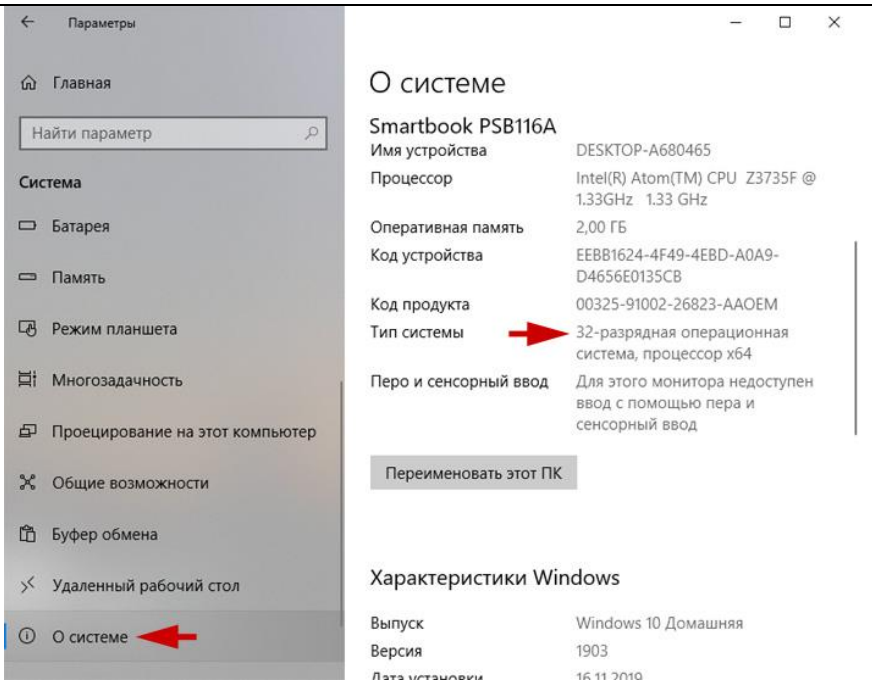

Рис. 3.1.2. Разрядность или тип операционной системы

Включите компьютер, дождитесь полной загрузки операционной системы. Щелкните на кнопке «Пуск», а затем на кнопке «Параметры» (рис. 3.1.1).

В открывшемся окне выберите пункт меню «Система». Откроется следующая вкладка, в которой необходимо кликнуть на строке «О системе». После этого на вкладке «Характеристики устройства» посмотрите строку «Тип системы» (рис. 3.1.2).

Типов ОС Windows существует два — 32-разрядная и 64-разрядная. Запомните разрядность вашей. Это число понадобится трижды: при скачивании пакета **Visual C++**, а также zip-архивов сервера Apache и интерпретатора PHP.

3.2. Установка пакета Visual C++

Откройте браузер и зайдите на страницу **https://www.apachelounge.com/download/**. Это сайт, с которого мы будем скачивать архив с сервером Apache. Но сначала надо установить на ваш компьютер компоненты Visual C++, без которых сервер не заработает.

Для этого найдите на указанной странице текст «**Be sure** you installed latest 14.29.30037.0 Visual C++ Redistributable for Visual Studio 2015-2019: **vc_redist_x64** or **vc_redist_x86** see **Redistributable**» (на момент подготовки книги данные строки были последними в описательной части страницы; со временем текст может несколько измениться). Для нас важны две ссылки, расположенные в тексте: **vc_redist_x64** и **vc_redist_x86**. Если у вас 64-разрядная ОС, нажмите ссылку **vc_redist_x64**. Если 32-разрядная, необходимо щелкнуть на ссылке **vc_redist_x86** (рис. 3.2.1).

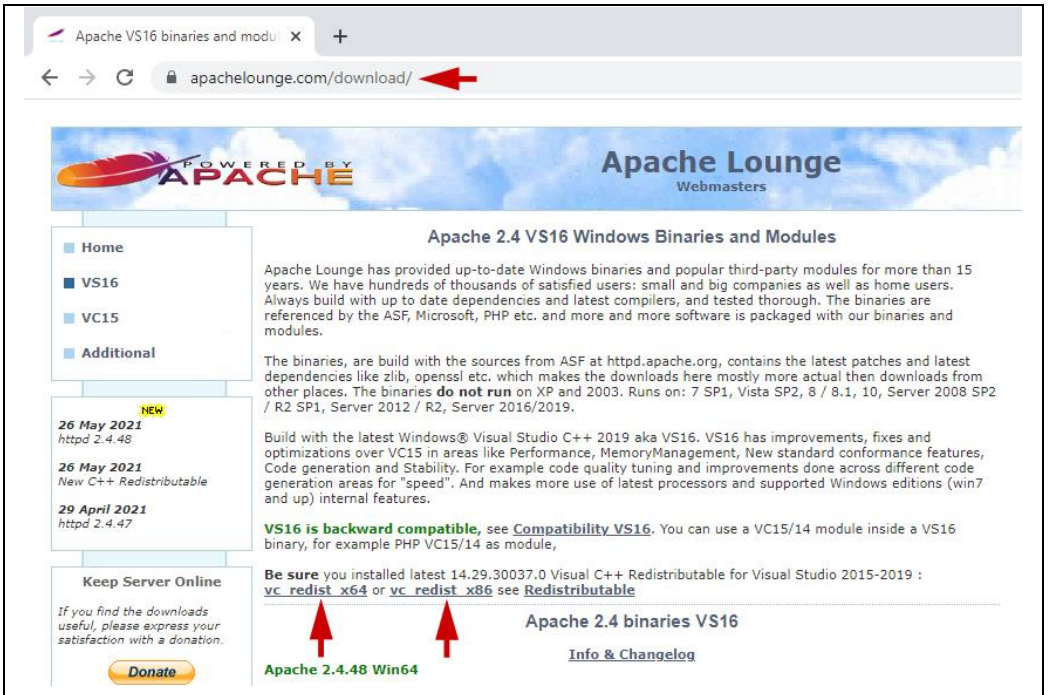

Рис. 3.2.1. Ссылки для скачивания пакетов Visual C++

Запустите скачанный файл (рис. 3.2.2). Примите условия лицензионного соглашения и нажмите кнопку «Установить». Дождитесь завершения процесса установки (рис. 3.2.3). Нажмите кнопку «Закрыть» (рис. 3.2.4).

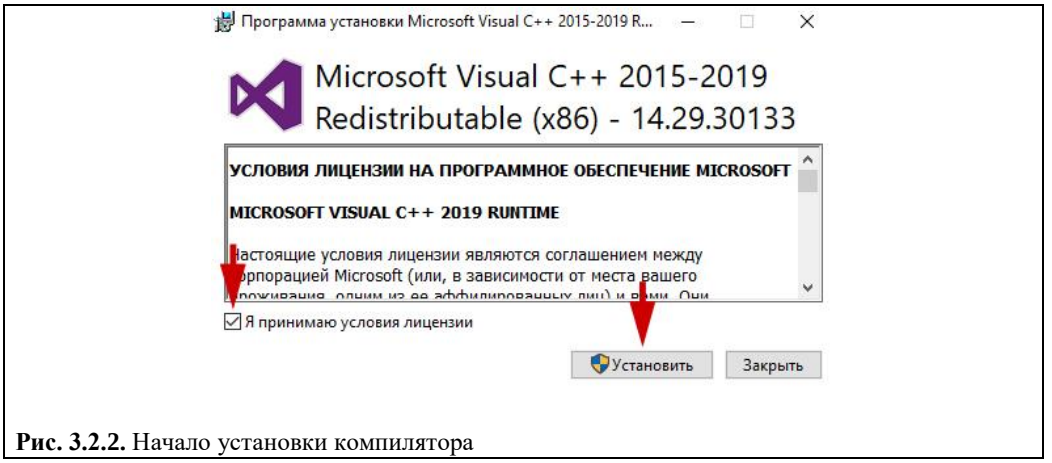

Рис. 3.2.2. Начало установки компилятора

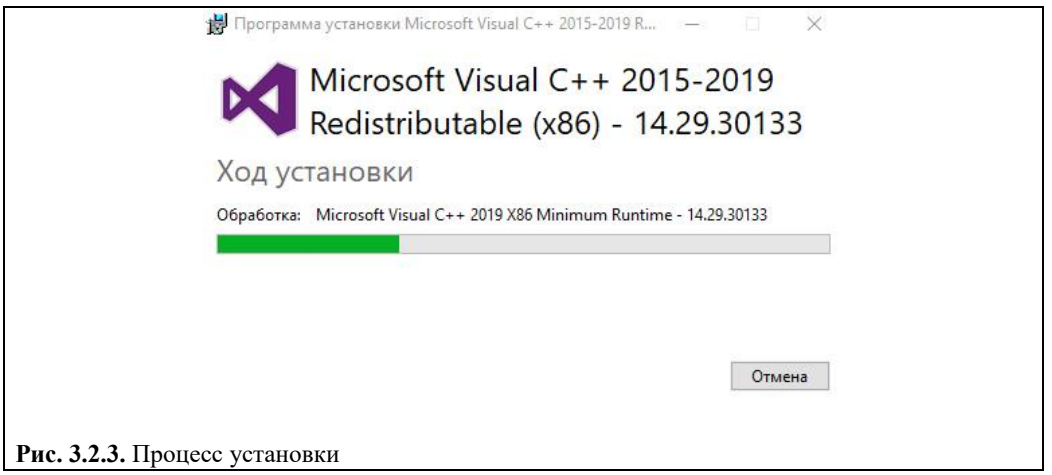

Рис. 3.2.3. Процесс установки

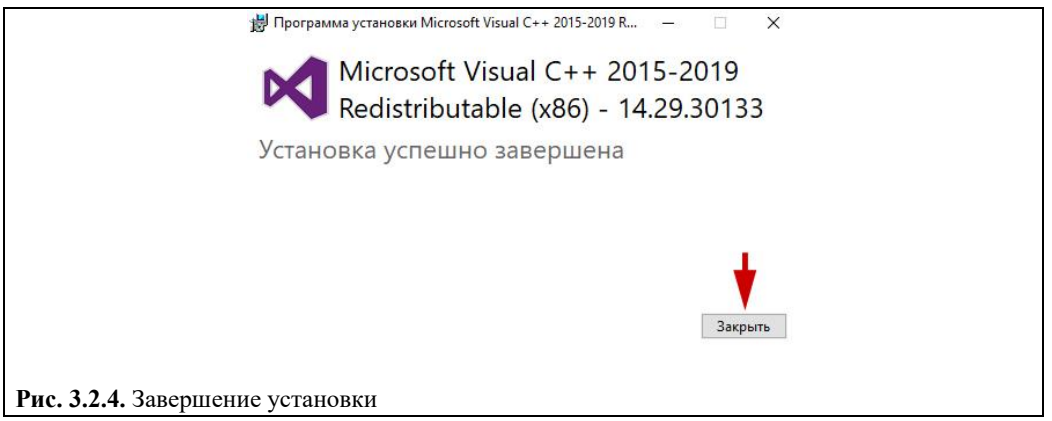

Рис. 3.2.4. Завершение установки

3.3. Установка сервера Apache 2.4

Теперь скачайте на рабочий стол компьютера с сайта **https://www.apachelounge.com/download/** zip-архив сервера, соответствующий разрядности вашей ОС (рис. 3.3.1). Для 64-разрядной системы — архив с пометкой **Win64** (файл httpd-2.4.48-win64-VS16.zip), для 32-разрядной — с пометкой **Win32** (файл httpd-2.4.48-win32-VS16.zip). Распакуйте архив. Скопируйте папку **Apache24** (только ее) непосредственно на диск **C**, чтобы ее адрес был **C:\Apache24** (рис. 3.3.2).

Рис. 3.3.1. Сайт с zip-архивом сервера

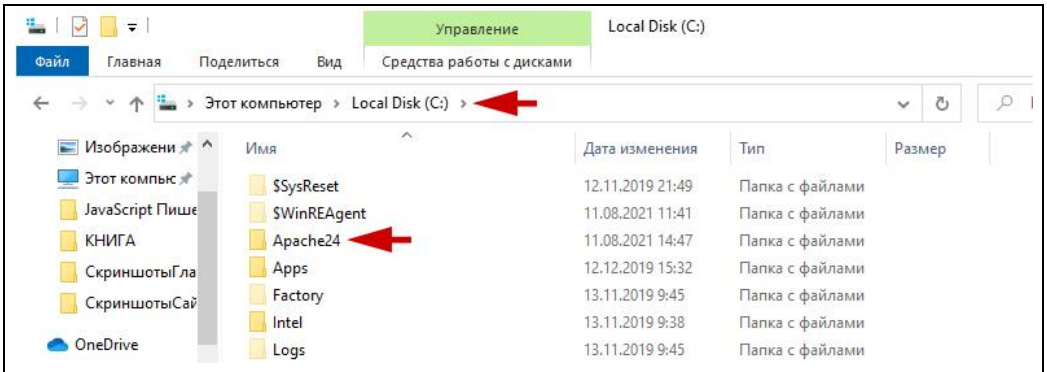

Рис. 3.3.2. Копируем папку Apache24 на диск C

Откройте приложение «Командная строка» от имени администратора. Для этого нажмите кнопку «Пуск», в меню выберите «Служебные — Windows», найдите пункт «Командная строка» и щелкните на нем правой (правой!) кнопкой мыши. В выпадающем списке выберите «Дополнительно», а затем «Запуск от имени администратора» (рис. 3.3.3). Откроется окно программы.

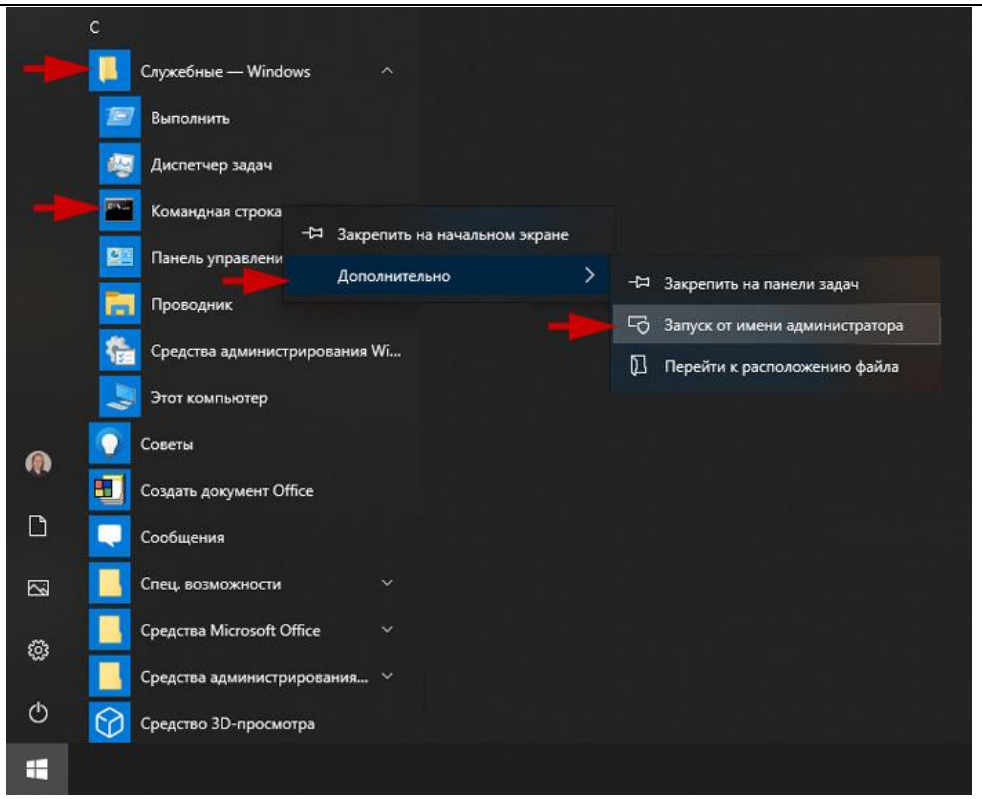

Рис. 3.3.3. Запуск командной строки

Теперь надо инсталлировать, а затем запустить сервер. Для этого в окне командной строки сразу после

```
C:\WINDOWS\System32>
```

наберите

```
C:\Apache24\bin\httpd.exe -k install
```

Должно получиться

```
C:\WINDOWS\System32>C:\Apache24\bin\httpd.exe -k install
```

Нажмите «Enter». Когда процесс инсталляции закончится (рис. 3.3.4), в окне программы вновь появится строка

```
C:\WINDOWS\System32>
```

Закройте окно командной строки и перезагрузите компьютер.

ВНИМАНИЕ! Может открыться окно брандмауэра с запросом на разрешение работы «Apache». Разрешите доступ во всех сетях.

Нам надо убедиться, что сервер заработал. Для этого откройте ваш браузер и введите в строке адреса **http://localhost/**. Если появилось сообщение «It works!», значит, все в порядке — сервер функционирует как положено (рис. 3.3.5).

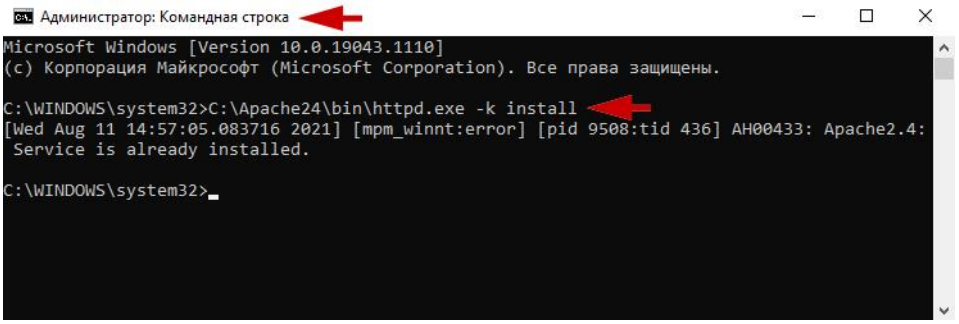

Рис. 3.3.4. Инсталляция сервера из приложения «Командная строка»

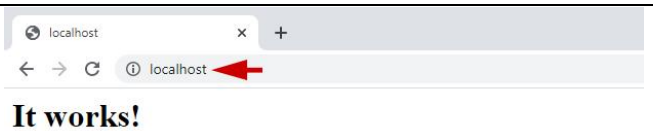

Рис. 3.3.5. Сервер подтверждает, что он работает

Теперь обратите внимание на один важный момент. Метод, которым мы установили сервер, позволяет запускать его автоматически при включении компьютера и завершать его работу при выключении компьютера. То есть Apache продолжает функционировать, даже если в данное время он вам не нужен. Эта информация вызывает у вас беспокойство по поводу теоретической уязвимости ПК? Тогда переведите сервер в режим ручного запуска. Для этого выполните следующие действия:

– введите в строке поиска слово «службы» (рис. 3.3.6) и нажмите на соответствующую иконку, которая появится в списке результатов поиска;

– в открывшемся окне найдите строку «Apache2.4» и щелкните по ней правой (правой!) кнопкой мыши — появится меню управления службой (рис. 3.3.7);

– кликните на строке «Свойства» и в окне на вкладке «Общие» найдите список «Тип запуска»;

– выберите пункт «Вручную», затем последовательно нажмите кнопки «Применить» и «ОК» (рис. 3.3.8);

– теперь остановите службу, нажав соответствующую ссылку в левой стороне окна (рис. 3.3.9);

– дождитесь окончания процесса (рис. 3.3.10);
– закройте вкладку «Службы».

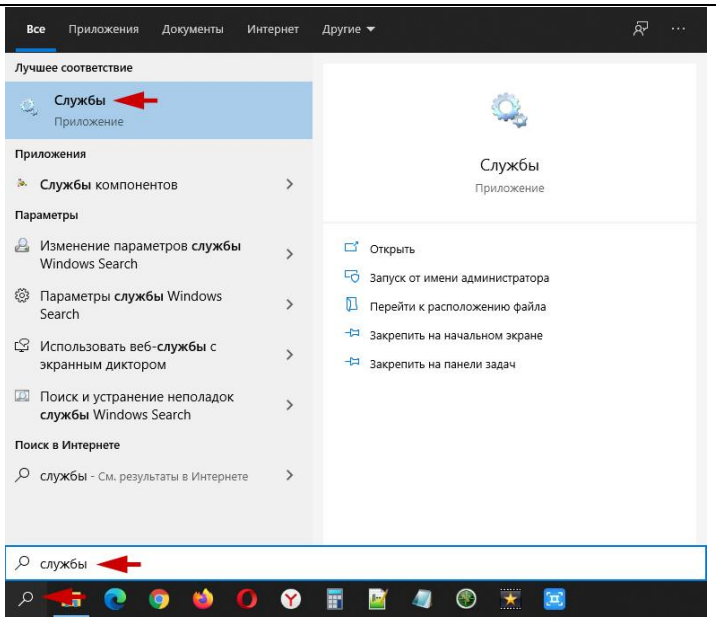

Рис. 3.3.6. Находим приложение «Службы»

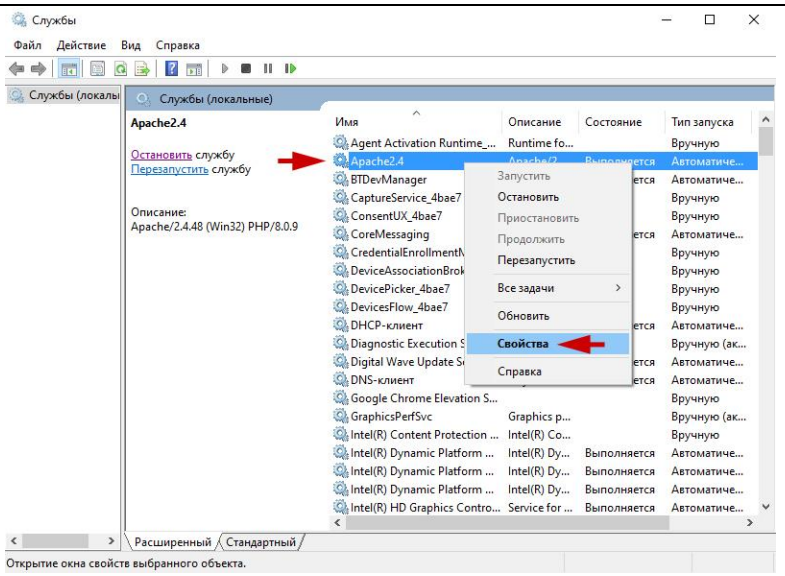

Рис. 3.3.7. Открываем меню управления службой

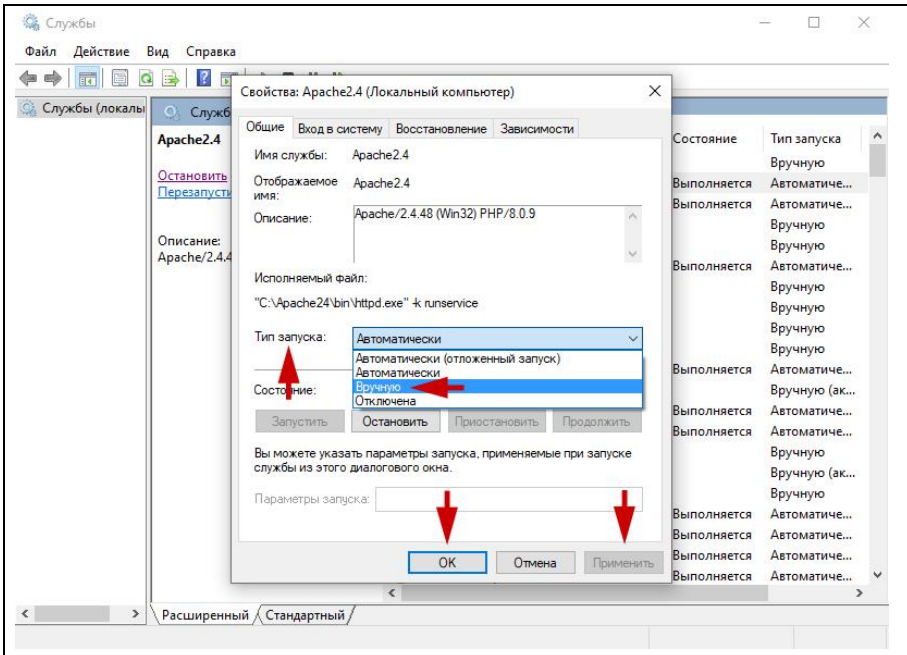

Рис. 3.3.8. Переводим службу в режим ручного запуска

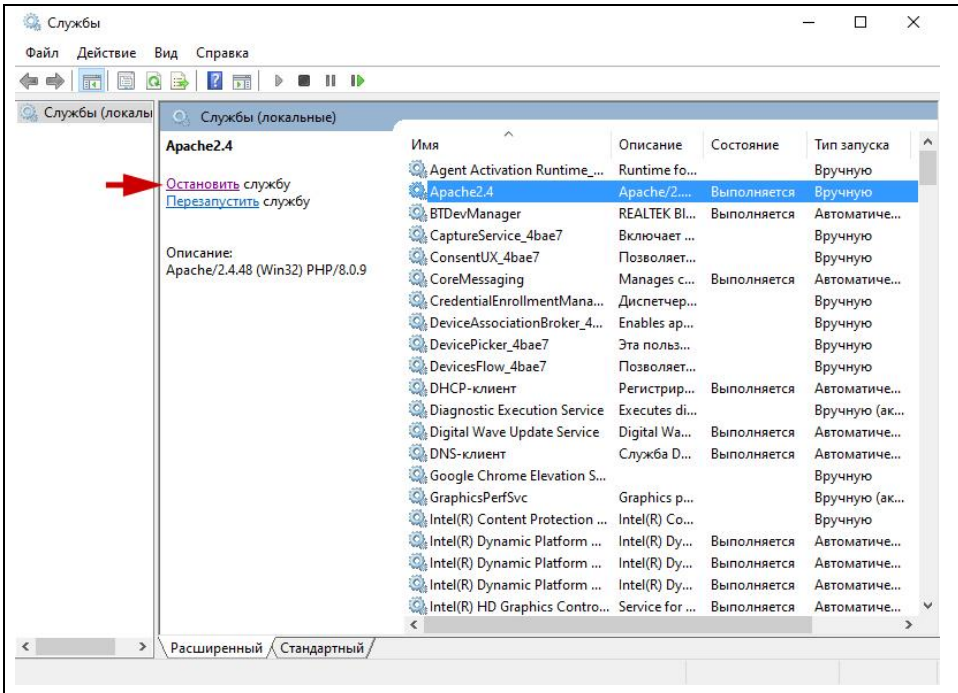

Рис. 3.3.9. Нажимаем ссылку «Остановить службу»

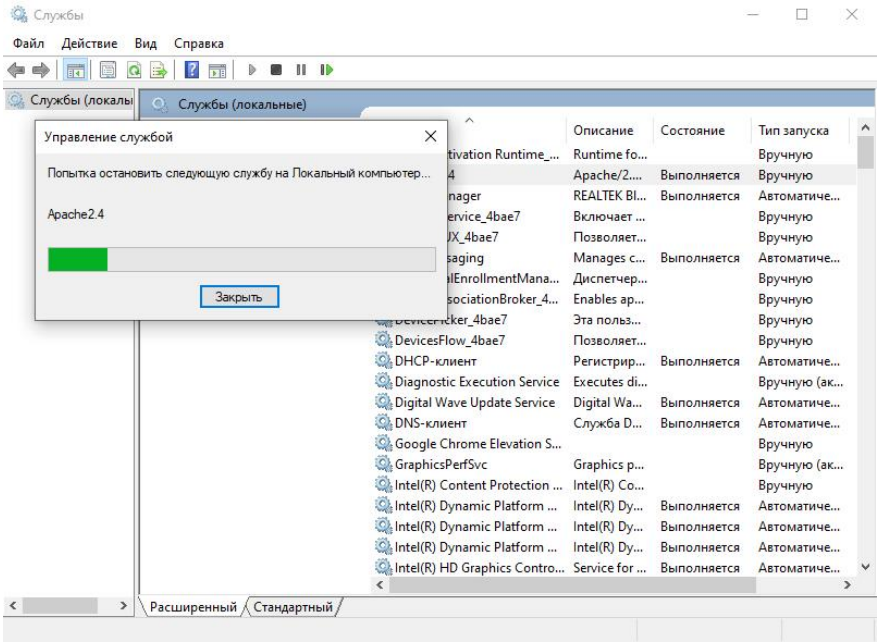

Рис. 3.3.10. Идет процесс остановки Apache

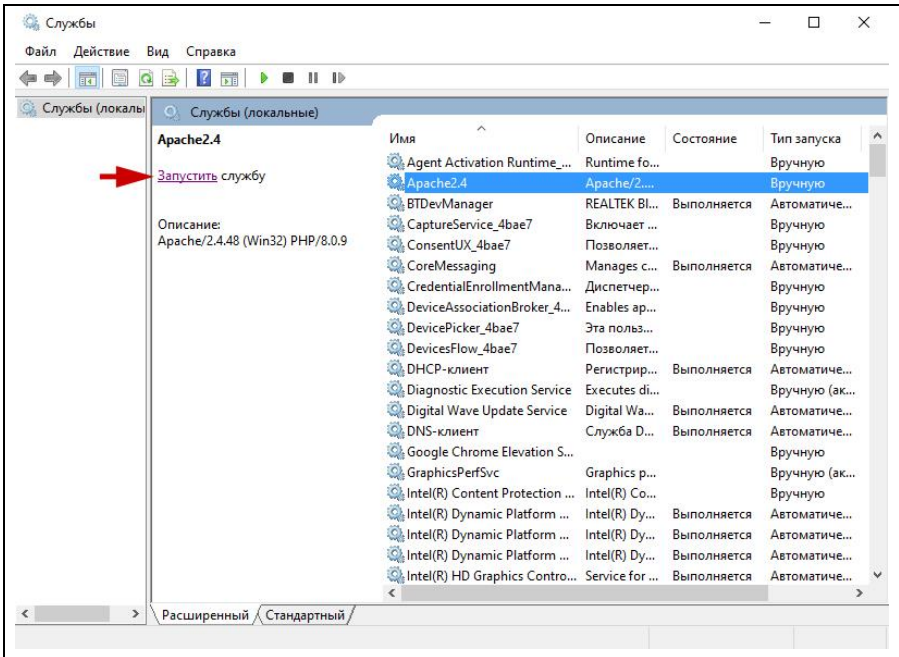

Рис. 3.3.11. Для запуска Apache нажмите ссылку «Запустить службу»

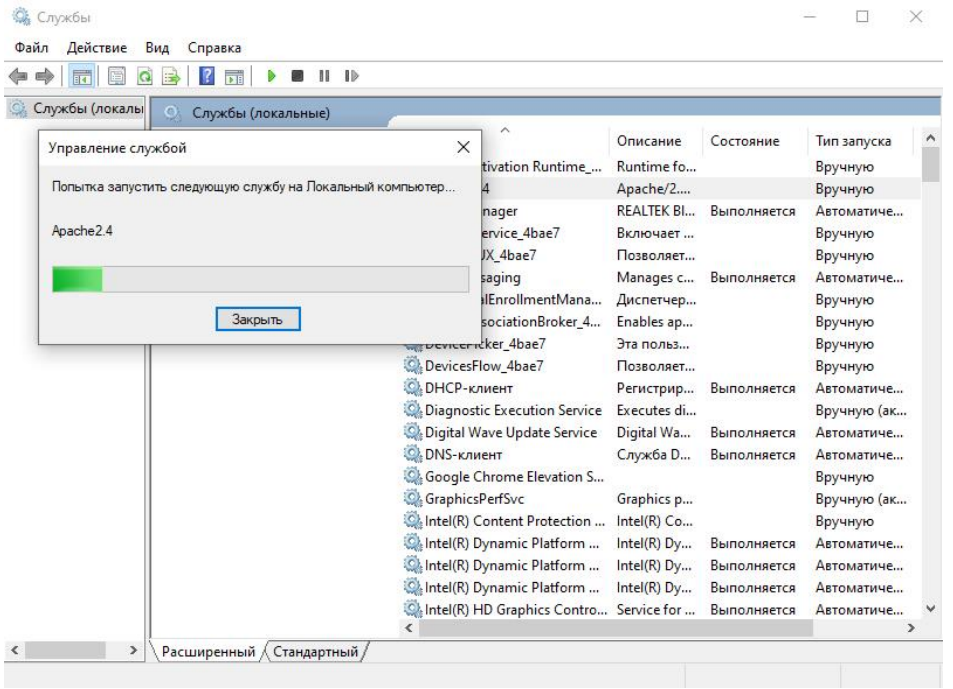

Рис. 3.3.12. Идет процесс запуска сервера

Теперь Apache остановлен и переведен в режим ручного запуска. При необходимости воспользоваться сервером снова зайдите в раздел «Службы», выделите строку «Apache2.4» и кликните на ссылке «Запустить службу» (рис. 3.3.11). Дождитесь окончания процесса (3.3.12).

Продолжаете беспокоиться о безопасности? Можете на время работы с сервером отключать свой ПК от Интернета. Напомню — локальный хостинг работает без сети.

3.4. Установка PHP 8

Скачайте на рабочий стол компьютера с сайта **https://windows.php.net/download/** zip-архив PHP 8, соответствующий разрядности вашей ОС. Для 64-разрядной системы — архив с пометкой **x64**, для 32-разрядной — с пометкой **x86**. Обратите внимание: так как мы будем устанавливать PHP в качестве модуля сервера Apache, скачивать надо дистрибутив **Thread Safe** (рис. 3.4.1).

Распакуйте архив. Переименуйте распакованную папку на **php** и переместите ее непосредственно на диск **C**, чтобы ее адрес был **C:\php** (рис. 3.4.2).

В папке **C:\Apache24\conf** с помощью текстового редактора «Блокнот» откройте файл **httpd.conf** (рис. 3.4.3), найдите в нем строку

```
DirectoryIndex index.html
```

и добавьте к ней

```
index.php
```

Должно получиться

```
DirectoryIndex index.html index.php
```

Теперь в самый конец файла **httpd.conf** добавьте 2 строки:

```
AddHandler application/x-httpd-php .php
LoadModule php_module "C:/php/php8apache2_4.dll"
```

Сохраните изменения, закройте файл и перезагрузите компьютер.

Убедимся, что PHP заработал. Для этого в папке **C:\Apache24\htdocs** создайте текстовый файл и запишите в него следующий код :

```
<?php
echo "Good !";
```

Сохраните этот файл под именем, например, **test.php**. Откройте ваш браузер и введите в строке адреса **http://localhost/test.php**. Если появилось сообщение «Good !», значит, все в порядке — PHP работает (рис. 3.4.4).

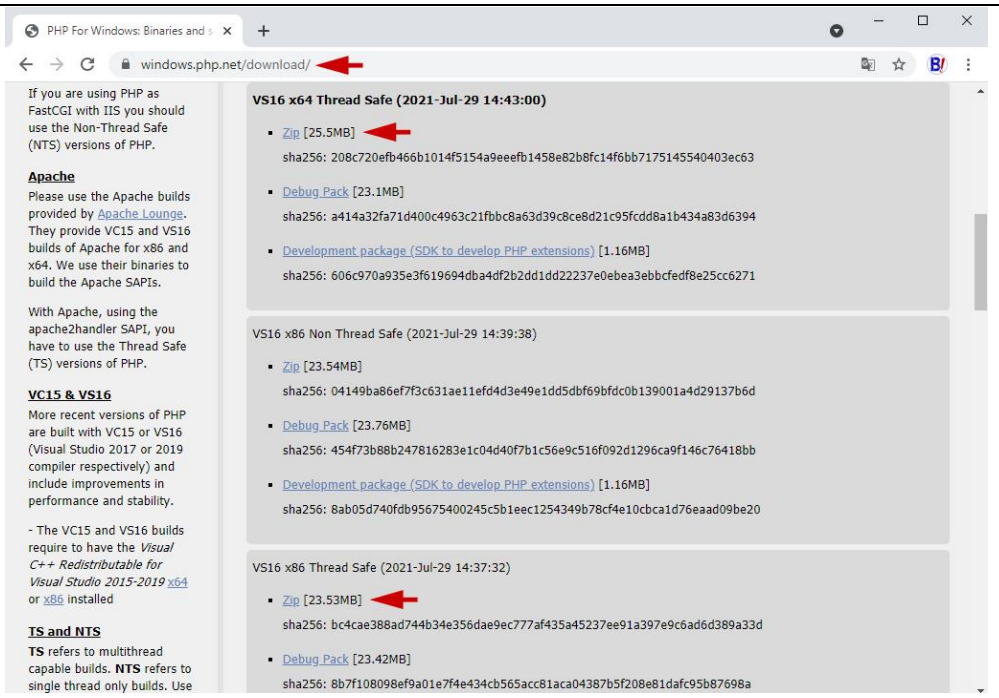

Рис. 3.4.1. Сайт с zip-архивом PHP

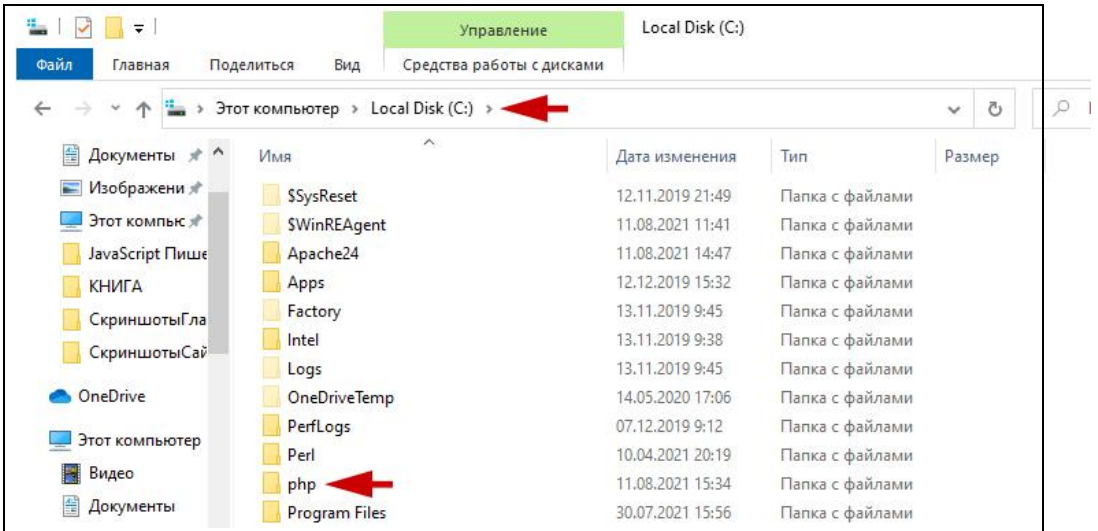

Рис. 3.4.2. Перемещаем папку php на диск C

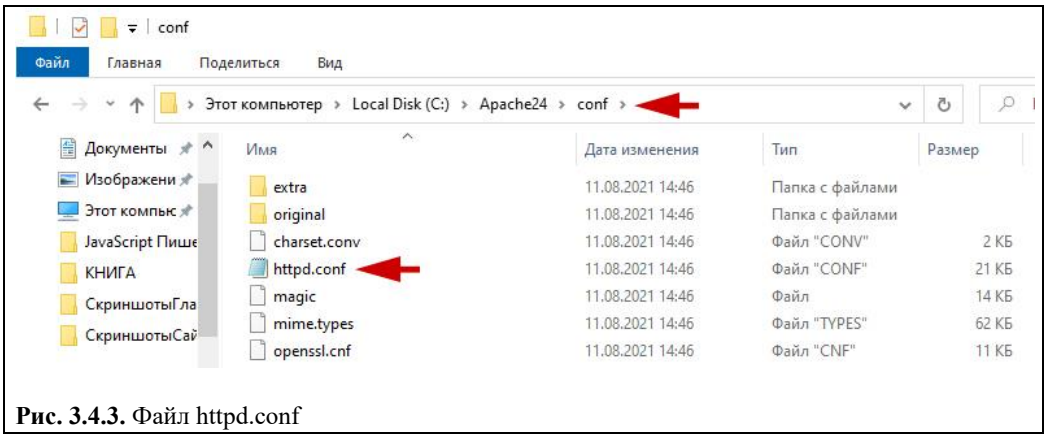

Рис. 3.4.3. Файл httpd.conf

Рис. 3.4.4. Проверяем работу PHP

3.5. Установка редактора Notepad++ 8

Скачать Notepad++ можно по адресу **https://notepad-plus-plus.org/downloads/** (рис. 3.5.1).

Устанавливается редактор следующим образом. Запустите скачанный файл. В первую очередь появится окно выбора языка программы. Оставляем русский (рис. 3.5.2) и нажимаем кнопку «ОК».

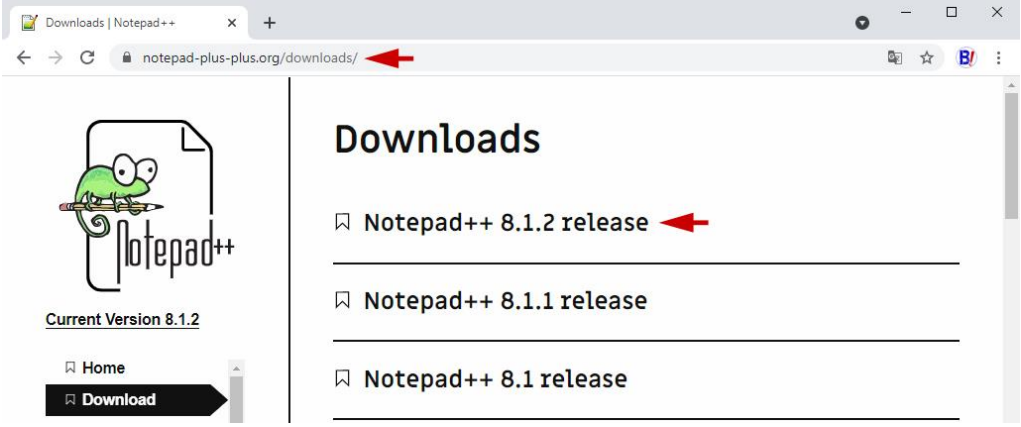

Рис. 3.5.1. Сайт с файлами редактора Notepad++

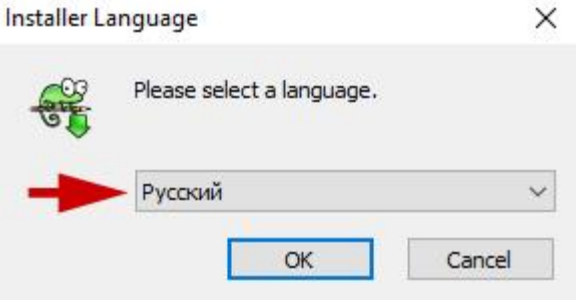

Рис. 3.5.2. Выбираем язык ПО

В следующем окне нажимаем кнопку «Далее» (рис. 3.5.3). Затем нажатием кнопки «Принимаю» соглашаемся с условиями лицензии (рис. 3.5.4).

Теперь нам необходимо выбрать папку для установки программы. По умолчанию предлагается **C:\Program Files\Notepad++**. Думаю, такой вариант установки подходит всем, поэтому данную папку оставляем без изменений (рис. 3.5.5).

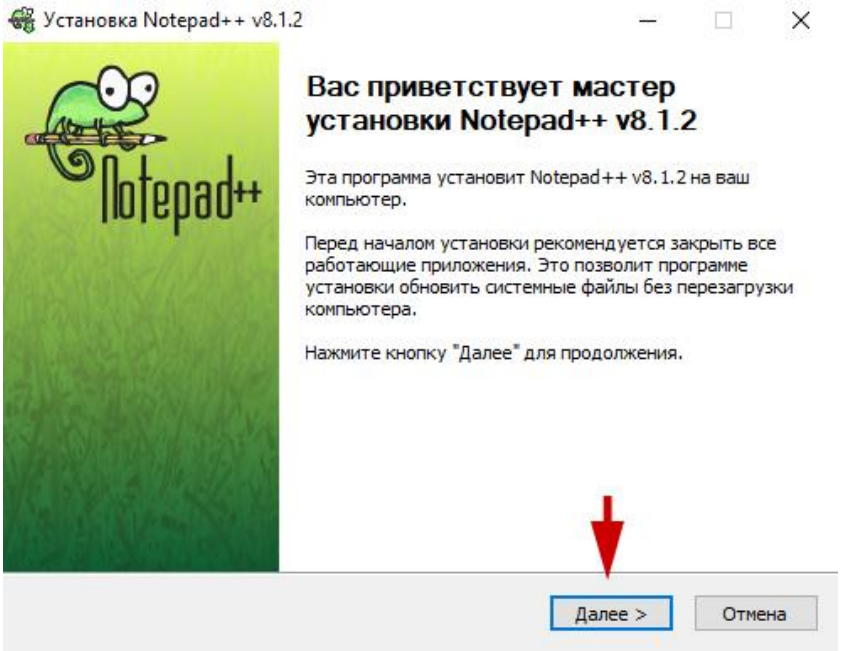

Рис. 3.5.3. Нажимаем кнопку «Далее»

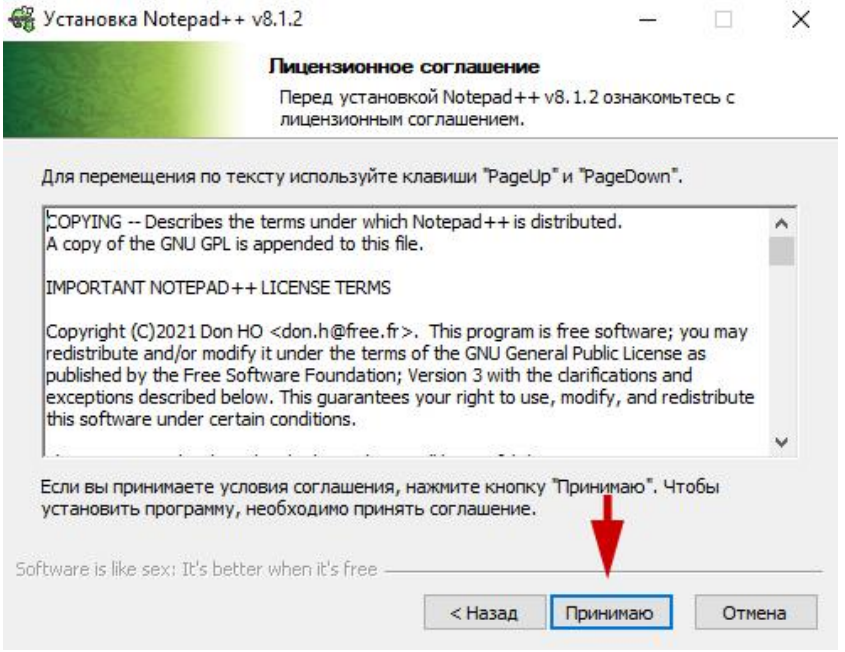

Рис. 3.5.4. Соглашаемся с условиями лицензии

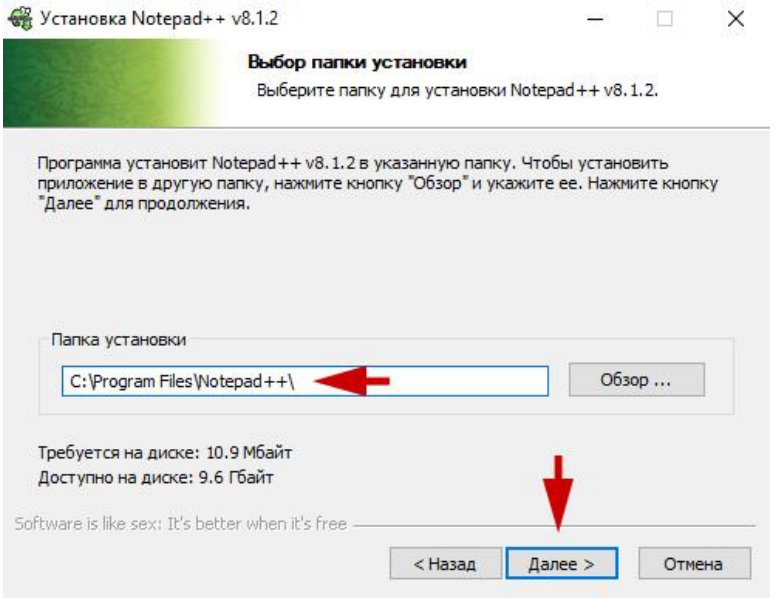

Рис. 3.5.5. Выбираем папку для установки программы

Дальше будет окно с выбором компонентов, которые можно установить вместе с программой. Этот раздел предназначен в первую очередь для опытных программистов, каковыми мы пока не являемся. Поэтому здесь ничего не меняем, жмем кнопку «Далее» (рис. 3.5.6).

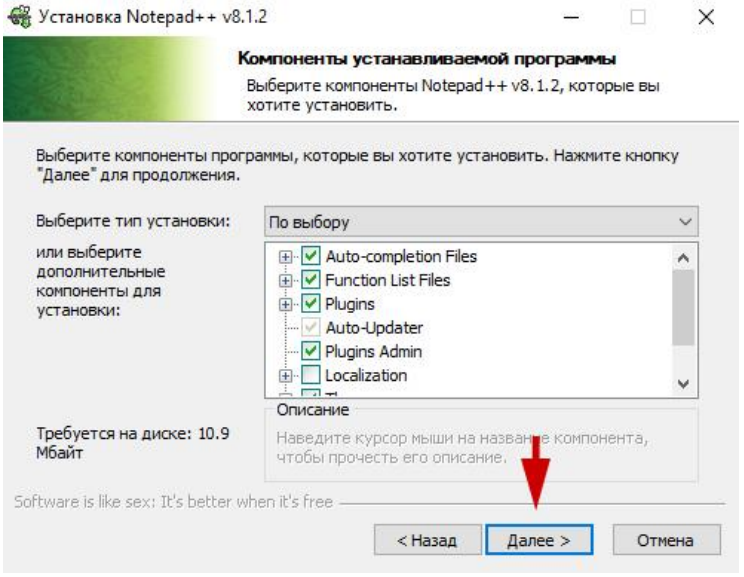

Рис. 3.5.6. Здесь ничего не меняем, жмем кнопку «Далее»

На последнем этапе перед началом непосредственной установки на вкладке появится пункт «Create Shortcut on Desktop» (рис. 3.5.7), то есть нам предлагают создать ярлык программы на рабочем столе. Делать это или нет — зависит от вашего желания. Замечу, что после установки редактора ссылка на него добавится в контекстное меню. Достаточно будет навести указатель мыши на файл, щелкнуть правой клавишей, и в списке возможных действий вы увидите строку «Edit with Notepad++». Кликните по ней — и файл будет открыт в редакторе.

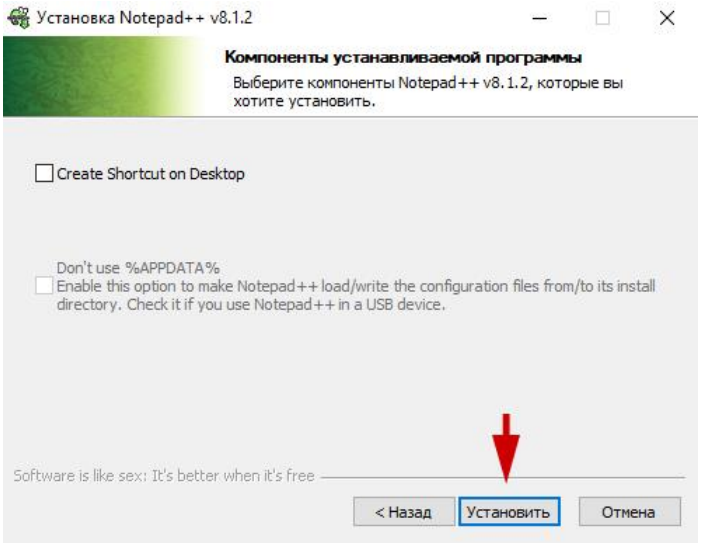

Рис. 3.5.7. Нажимаем кнопку «Установить»

Разобравшись с вопросом, необходим ли ярлык на рабочем столе или нет, нажмите кнопку «Установить». Дождитесь окончания процесса (рис. 3.5.8).

После завершения установки вам нужно будет принять еще одно решение: сразу запустить программу или отложить это дело на потом. Для этого оставьте или снимите «галочку» в пункте «Запустить Notepad++» (рис. 3.5.9). Теперь нажмите кнопку «Готово». Поздравляю — отныне на вашем компьютере установлен профессиональный текстовый редактор!

Познакомимся с ним поближе. Как выглядит редактор, вы можете видеть на рисунке 3.5.10.

Notepad++ позволяет:

– выполнять поиск по файлу в разных направлениях;

– производить замену по шаблону;

– устанавливать кодировки, в том числе необходимую для нашего проекта UTF-8;

– менять страницы, форматируя их по стандартам разных операционных систем — Windows, Unix или macOS;

– подсвечивать код, выделяя разные по назначению фрагменты, операторы, функции, переменные, комментарии и т. д.;

– выбирать синтаксис документа (благодаря чему меняются варианты подсветки кода), например HTML, CSS, JavaScript или PHP;

– сохранять файлы с разным расширением;

– менять стили оформления окна программы — вариантов достаточно, найдется любой по желанию;

– выполнять еще многие и многие операции.

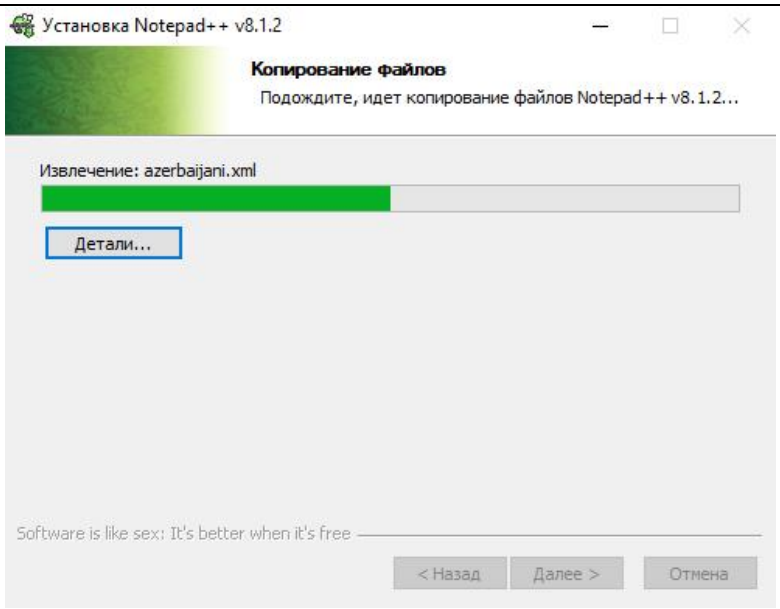

Рис. 3.5.8. Ждем окончания процесса

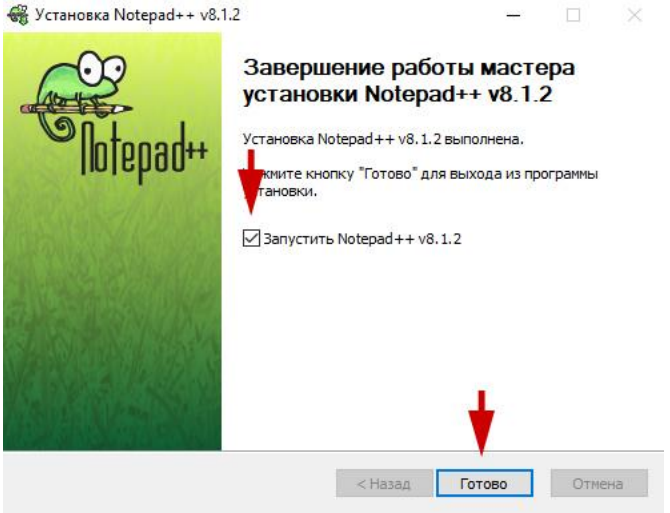

Рис. 3.5.9. Завершение установки

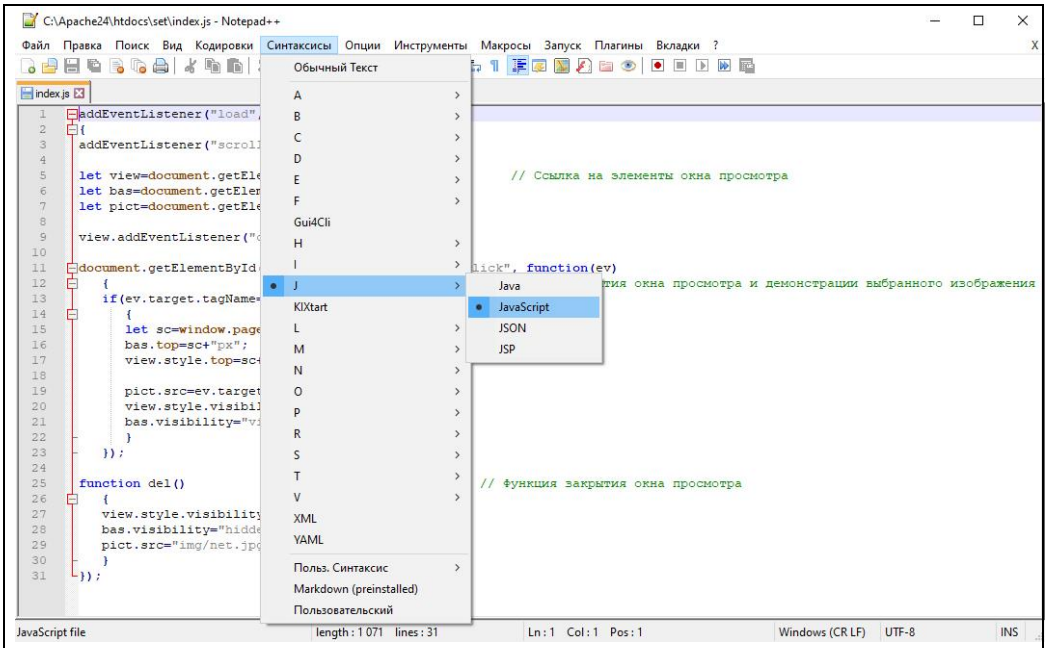

Рис. 3.5.10. Редактор Notepad++ с открытым в нем файлом проекта

4. Тестирование программ

Как ни хороши современные web-обозреватели, но кое-чем они мне категорически не нравятся, а именно тем, что в них все очень тщательно продумано и предусмотрено. В результате, даже получив некачественный код, браузеры понимают, чего хотел добиться программист, и демонстрируют правильную web-страницу. Тем самым браузеры провоцируют разработчиков небрежно относиться к своим обязанностям. В результате миллионы сайтов за красивой оболочкой содержат множество ошибок и нарушений стандартов, которые web-обозреватели успешно исправляют.

Удивительно, но даже в ресурсах лидеров мировой web-индустрии ошибок и нарушений немало.

Большинство таких сайтов сделаны на популярных «движках», как иногда называют всем известные CMS. Ошибки и отклонения от стандартов заложены у них в модулях, формирующих страницы.

Автор относится к тому немногочисленному отряду программистов, которые выпускают продукцию в свет, только добившись абсолютно чистого кода. Поэтому я проверил разметку, стили и сценарии проекта самым тщательным образом. Вот перечень проведенных работ:

– проверка работоспособности сайта и редакторов в различных браузерах, в том числе достижение идентичности результатов показа страниц и результатов редактирования в различных браузерах;

– проверка кода с помощью валидаторов и устранение ошибок;

– проверка работы сайта и редакторов с одновременным наблюдением за качеством исполнения кода в консоли.

Подробнее об этом — в следующих разделах.

4.1. Тестирование сайта и редакторов в браузерах

Один из важнейших этапов тестирования файлов нашего проекта — проверка их работоспособности в разных web-обозревателях. Написав программу и убедившись, что она корректно выполняется в одном браузере, нельзя останавливаться на достигнутом. Ведь может оказаться, что в другом браузере сайт или один из редакторов будут работать совсем не так, как мы ожидаем.

Чем в большем количестве web-обозревателей проведены испытания, тем лучше для нашего проекта. Это гарантирует, что все посетители сайта увидят на его страницах именно то, что мы хотели продемонстрировать, а редакторы будут надежно работать в любых браузерах, независимо от того, каким из них пользуется администратор ресурса.

На мой взгляд, обязательный набор разработчика должен включать минимум 5 web-обозревателей. Если вы решили серьезно заниматься написанием визуальных редакторов и CMS, советую установить на свой компьютер эти браузеры. Расскажу о них по порядку.

1. Браузер **Microsoft Edge**. Он входит в состав операционной системы Windows 10, поэтому не нуждается в скачивании и установке. Многие опытные программисты не любят данный браузер и игнорируют его при проверке сценариев. Я считаю, что это серьезная ошибка. Дело в том, что Microsoft Edge является браузером по умолчанию в операционной системе Windows 10. Многие пользователи, не такие продвинутые, как программисты, просто используют то, что у них изначально установлено на ПК. Следовательно, большой процент посетителей Интернета прибегают к услугам Microsoft Edge.

2. **Google Chrome**. На мой субъективный взгляд — лучший браузер. И судя по некоторым опросам, наиболее популярный. Он моложе некоторых своих конкурентов, но снабжен самыми передовыми решениями и очень быстро развивается. И это неудивительно — ведь за ним стоит такой гигант, как компания Google. Скачать ПО можно здесь: **https://www.google.ru/chrome/**.

3. **Яндекс.Браузер**. После его создания компания Яндекс вложила заметные средства в рекламу и популяризацию своей разработки. Поэтому данный браузер стоит у многих владельцев компьютеров с ОС Windows. Этих людей обязательно надо учитывать. Скачать дистрибутив можно здесь: **https://browser.yandex.ru/**.

4. **Opera** — один из старейших браузеров, существовавших еще во времена борьбы за лидерство между Netscape Navigator и Internet Explorer. В России у него много поклонников. Недаром среди российских пользователей Интернета показатель его популярности в несколько раз выше, чем общемировой уровень. Кстати, браузер Opera был одним из первых, кто начал поддерживать таблицы стилей. Скачать программное обеспечение можно здесь: **https:// www.opera.com/ru**.

5. **Mozilla Firefox**. В этом браузере сценарии необходимо проверять в обязательном порядке. У него достаточно высокий уровень популярности. При этом есть ряд отличий в обработке кода по сравнению с четырьмя перечисленными выше браузерами. Что-то Mozilla Firefox обрабатывает аналогично остальным браузерам, а что-то по-своему. Во всяком случае, я неоднократно сталкивался с ситуациями, когда код, отлично работавший в других браузерах, начинал «капризничать» в Mozilla Firefox (рис. 4.1.1). Скачать браузер можно здесь: **https://www.mozilla.org/ru/firefox/**.

И уж совсем идеальной можно считать проверку, которую, кроме всего прочего, удалось провести на устройствах Apple с операционной системой macOS и браузером **Safari**.

Думаю, что цель таких масштабных испытаний с использованием группы разных браузеров понятна: автор считал необходимым добиться совершенно одинаковой работы демонстрационного сайта и редакторов во всех браузерах. Только получив этот результат, можно считать свои программы вполне качественными.

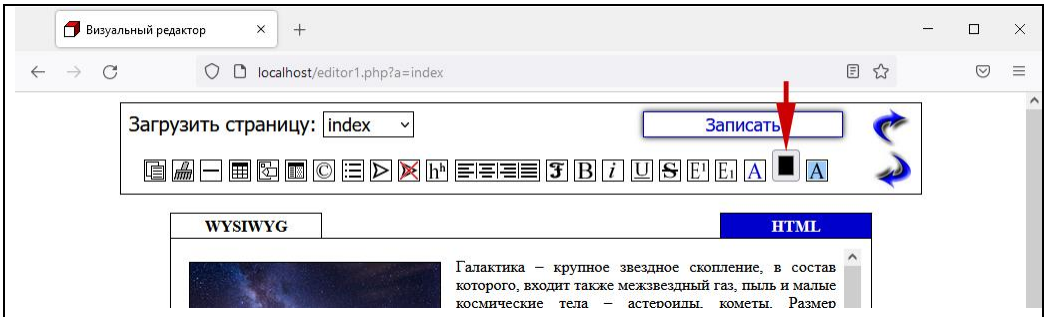

Рис. 4.1.1. Пример того, как браузер Mozilla Firefox игнорирует заданную таблицей стилей высоту поля выбора цвета. В остальных браузерах поле отображается правильно

Отступление от темы

Много лет тому назад двумя основными браузерами — и одновременно конкурентами — были Internet Explorer и Netscape Navigator. Первый выделялся большим набором возможностей для программиста, второй лучше соответствовал принятым на то время стандартам. Сначала сдал Netscape Navigator. Его поддержка прекратилась в 2008 году. Internet Explorer продержался дольше, но на сегодняшний день — это реликт, которым пользуются, наверное, только его самые фанатичные поклонники. Хотя этот обозреватель встроен в ОС Windows 10, для просмотра современных web-страниц он совершенно не приспособлен. В наши дни конкуренция между браузерами возросла, а наиболее популярный из них — Google Chrome.

4.2. Тестирование сайта и редакторов в валидаторах

Валидатор — это специальная программа для проверки кода, написанного на одном из языков, предназначенных для создания web-проектов. Задача такой программы — найти ошибки, если они есть, и выдать их перечень разработчику или сообщить, что код безупречный, если ошибок нет.

Можно сказать и по-другому: назначение того или иного валидатора — проверять код на соответствие стандартам языка программирования.

Стандарты HTML и CSS разрабатывает Консорциум Всемирной Паутины — World Wide Web Consortium (W3C). Он же предоставляет и валидаторы для проверки разметки страниц и написания таблиц стилей. Вот их адреса в сети:

– валидатор HTML5 — **https://validator.w3.org/**;

– валидатор CSS3 — **http://jigsaw.w3.org/css-validator/**.

Данные валидаторы очень удобны. Они позволяют проверять разметку и таблицы стилей:

– страниц, уже загруженных в сеть;

– файлов, загруженных в валидатор с вашего компьютера;

– в коде, скопированном из вашего файла и помещенном в специальное текстовое поле.

Например, после обработки кода разметки все ошибки и предупреждения с комментариями на английском языке появятся на странице HTML-валидатора в ее нижней части. При этом ошибки будут сопровождаться надписью **Error** на розовом фоне, а предупреждения — надписью **Warning** на желтом фоне. Например, строка кода **<script type="text/javascript">** получит предупреждение, так как по современным стандартам указывать тип скрипта не надо, но в то же время данная ошибка не критична, так как не мешает нормальному отображению страницы. Зато следующая строка **<img src="img/im0.jpg">** будет отмечена надписью **Error**, так как у рисунка пропущен обязательный, согласно стандартам HTML 5, атрибут **alt**.

У HTML-валидатора есть еще одна полезная функция: по завершении проверки он показывает, сколько времени было потрачено на загрузку вашей страницы.

Если ошибок нет, валидаторы сообщат вам об этом. Любопытно заметить, что о безупречности HTML-кода вы получите простое текстовое подтверждение. А вот качественную таблицу стилей Консорциум Всемирной Паутины предлагает отметить специальным знаком — что-то вроде знака качества, который вам будет предложено разместить на странице сайта (по крайней мере, так было на момент написания книги).

Естественно, разметка и таблицы стилей нашего сайта и всех редакторов в обязательном порядке проверялись в этих валидаторах. Результаты такой проверки вы можете видеть на рисунках 4.2.1 и 4.2.2.

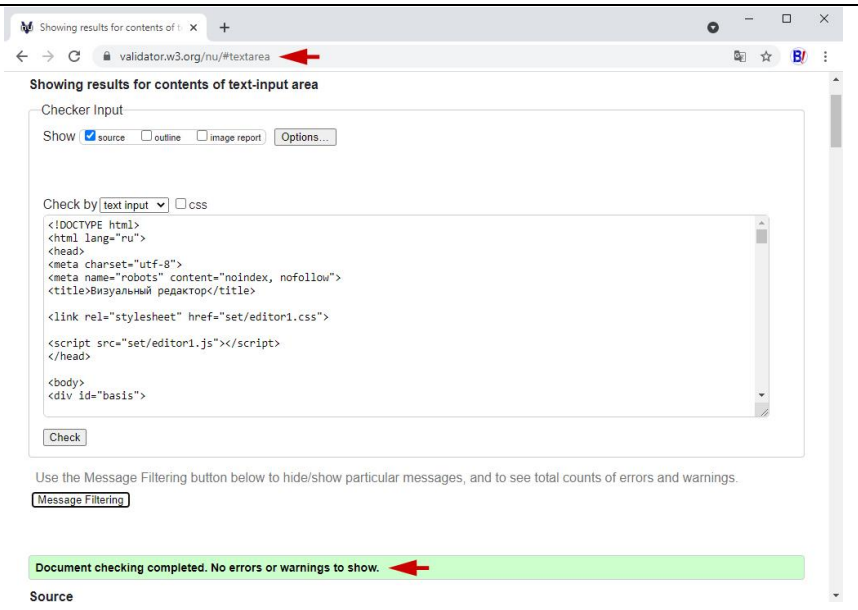

Рис. 4.2.1. Проверка кода одного из редакторов в HTML-валидаторе. Как видите, ошибок нет

Что касается валидатора JavaScript, то здесь ситуация несколько иная. Таких валидаторов много. Я опробовал несколько и выбрал из них вариант, в котором реализован самый строгий подход к оценке кода. Вот его адрес: валидатор JavaScript — **https://beautifytools.com/javascript-validator.php**.

Он позволяет вставить в специальное текстовое поле ваш код, после чего сразу, без нажатия каких-либо кнопок, начинается проверка скрипта. Единственное неудобство этого валидатора, обнаруженное мною на момент написания книги, — сообщения об ошибках, предупреждения и рекомендации он выдает в общем списке, не выделяя пункты этого списка по степени важности.

Думаю, читатели уже догадываются, что все сценарии проекта, написанные на JavaScript, были проверены в данном валидаторе. На момент написания книги отклонений от стандартов языка и нарушений синтаксиса в сценариях не было (рис. 4.2.3).

Особенность всех упомянутых валидаторов в том, что они находят все ошибки в разметке, в таблицах стилей, в коде программ, в том числе пропущенные символы, опечатки (но не в обычном тексте), необъявленные переменные и т. д. Понятно, что задача хорошего программиста — исправить эти ошибки и получить идеально чистый и правильный код. Что и было сделано автором.

Автор пытался тестировать и код PHP, но, хотя PHP-валидаторы существуют, ни одного надежного на просторах сети обнаружить не удалось.

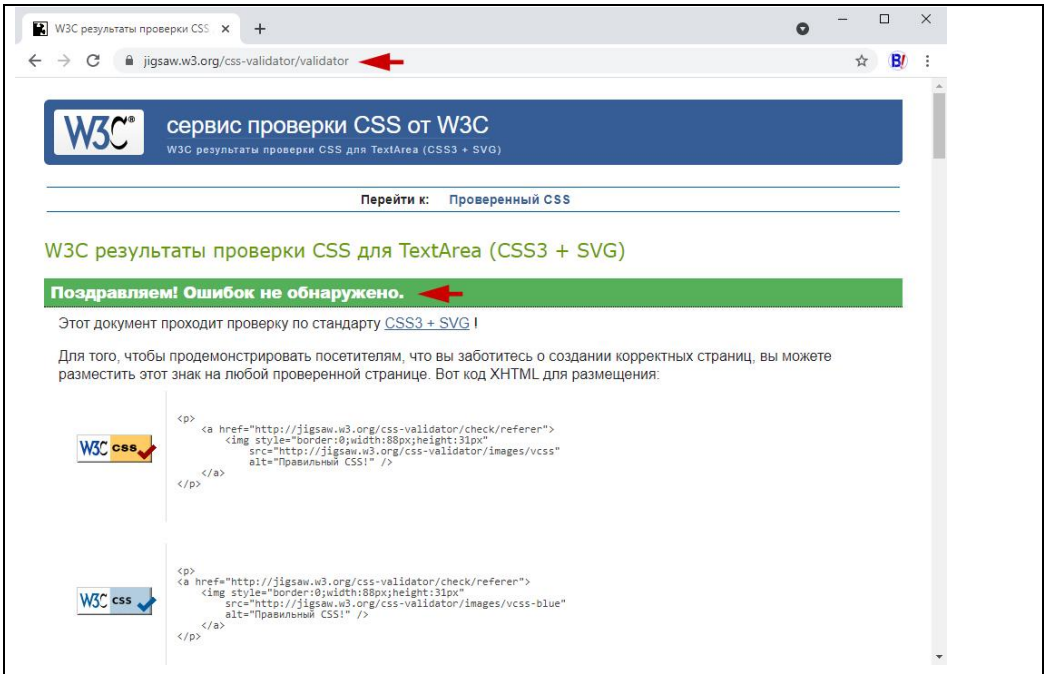

Рис. 4.2.2. Проверка таблицы стилей одного из редакторов в CSS-валидаторе. Как видите, и здесь ошибок нет

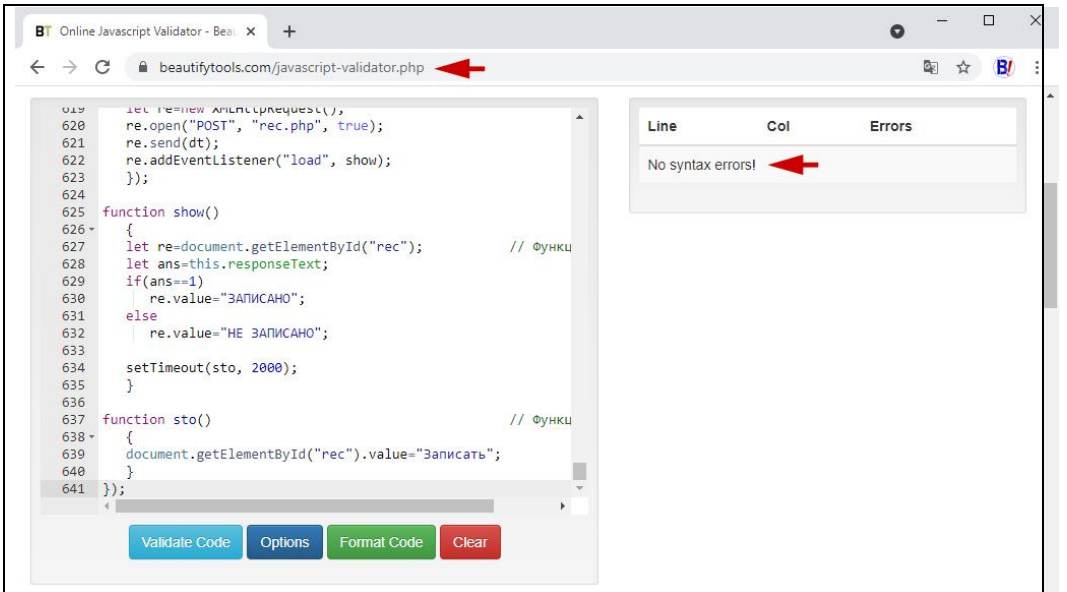

Рис. 4.2.3. Проверка сценария одного из редакторов. И опять ни одной ошибки

4.3. Проверка работы сайта и редакторов в консоли

Еще один способ проверить в первую очередь код JavaScript — посмотреть результаты его работы в консоли браузера.

Чтобы зайти в консоль Microsoft Edge:

– справа вверху кликните на иконке в виде трех точек «Настройки и прочее»;

– в списке выберите пункт «Другие инструменты»;

– в появившейся вкладке щелкните на пункте «Средства разработчика»;

– в нижней части браузера или справа откроется дополнительное окно, в котором нужно выбрать вкладку «Консоль».

Чтобы зайти в консоль Google Chrome: иконка «Настройка и управление Google Chrome» (три точки) — «Дополнительные инструменты» — «Инструменты разработчика» — «Console» (рис. 4.3.1).

Чтобы зайти в консоль Яндекс.Браузера: «Настройки Яндекс.Браузера» (три черты) — «Дополнительно» — «Дополнительные инструменты» — «Инструменты разработчика» — «Console».

Чтобы зайти в консоль Opera: «Меню» (слева вверху) — «Разработка» — «Инструменты разработчика» — «Console».

Чтобы зайти в консоль Mozilla Firefox: «Открыть меню приложения» (три черты) — «Другие инструменты» — «Инструменты веб-разработчика» — «Консоль».

Работа сценариев проверялась следующим образом. В браузер загружался очередной редактор, после чего поочередно выполнялись все действия по

внесению изменений в текст главной страницы. Каждый раз автор смотрел, не появляются ли в консоли сообщения об ошибках.

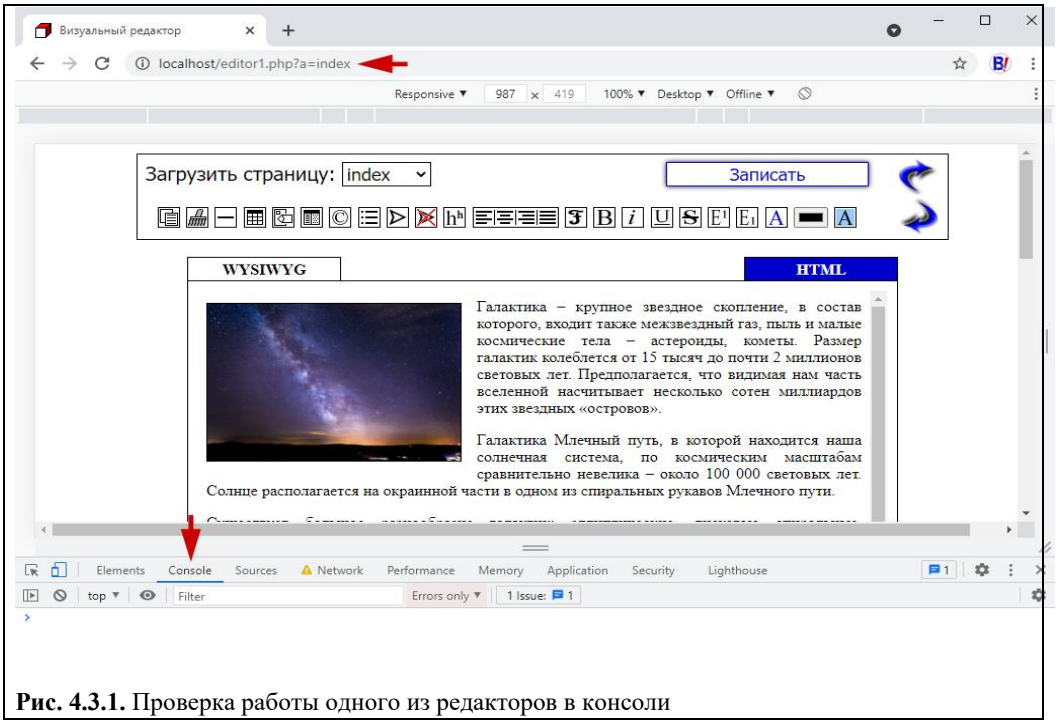

Рис. 4.3.1. Проверка работы одного из редакторов в консоли

5. Структура проекта

Прежде, чем начать детальное рассмотрение проекта, разберемся с его «комплектующими».

Первым делом поговорим про **zip-архив**. Его можно скачать для своих экспериментов. Особо обращаю внимание читателей на следующий момент: вы можете абсолютно свободно использовать файлы проекта в своих разработках, не указывая автора. Говоря высокопарно, это свободно распространяемое ПО с открытым исходным кодом.

Затем посмотрим, как организовано взаимодействие компонентов сайта и редактора. Чтобы рассуждения автора выглядели более понятными, снабдим описание блок-схемами, иллюстрирующими взаимосвязь разных файлов.

5.1. Состав zip-архива

Напомню, что архив с полным набором всех компонентов находится по адресу: **https://editjs.ru/EDIT.zip**.

Что вы найдете в архиве?

1. Файл **index.php**, отвечающий за создание страниц демонстрационного сайта. Обратите внимание: редакторов у нас четыре, но все они работают с одним сайтом.

2. Файлы **editor1.php**, **editor2.php** и **editor4.php**. Это файлы страниц первого, второго и четвертого редакторов.

3. Файлы **editor3.php** и **index_ed3.php**. Они отвечают за формирование страницы третьего редактора.

4. **rec-real.php** — файл записи отредактированных данных.

5. **rec.php** — файл-заглушка, используемый при экспериментах с редактором. Служит для предотвращения записи данных. В коде редакторов при нажатии кнопки записи происходит обращение к файлу с таким именем. Соответственно, реальная запись не выполняется. Чтобы привести редакторы в рабочее состояние, этот файл необходимо удалить (на всякий случай где-нибудь на компьютере сохранив его копию), а **rec-real.php** переименовать в **rec.php**.

6. **vacant.html** — пустая web-страница. «Участвует» в проекте, чтобы угодить требованиям Консорциума Всемирной Паутины. Дело в том, что в самом первом редакторе мы использовали фрейм, в который загружается редактируемое содержимое. При этом никакие дополнительные файлы не нужны. Но если страницу с подобным кодом проверить в валидаторе HTML, то он выдаст сообщение об ошибке, так как у фрейма отсутствует атрибут **src**. Поэтому мы сначала загружаем во фрейм пустую страницу с реальным адресом (**src="vacant.html"**), а затем уже меняем содержимое фрейма на данные, которые необходимо редактировать.

7. Папка **set**. В ней файлы сценариев и таблиц стилей. **index.js** — сценарий, необходимый для просмотра увеличенных изображений на страницах сайта. **editor1.js**, **editor2.js**, **editor3.js** и **editor4.js** — сценарии, выполняющие все функции редактирования контента. **index.css**, **editor1.css**, **editor2.css**,

editor3.css и **editor4.css** — таблицы стилей сайта и редакторов. **cont.css** — дополнительная таблица стилей, необходимая для второго редактора.

8. Папка **content**. Содержит txt-файлы с основным контентом страниц.

9. **cosmos** — папка с изображениями, используемыми в основном контенте и в блоке рекламы.

10. **img —** папка с изображениями, используемыми в дизайне сайта и панелей настроек редакторов.

11. **pict** — папка с изображениями, используемыми в дизайне главной панели редакторов.

Скачайте zip-архив на свой компьютер. Распакуйте. Скопируйте все содержимое архива и переместите его в папку **htdocs** сервера Apache по адресу: **C:\Apache24\htdocs**. Сайт нужно смотреть в браузере по адресу **http://localhost/**. Страницы первого, второго и четвертого редакторов — по адресу **http://localhost/editorN.php?a=index**, где N — номер редактора. Чтобы зайти в третий — креативный — редактор, укажите адрес **http://localhost/index_ed3.php?a=index&d=admin**, введите в левом верхнем поле пароль **admin** и нажмите клавишу «Enter». Подробнее о механизме входа в третий редактор будет рассказано в главе 12.

Для удобства читателей все файлы имеют необходимый набор комментариев, объясняющих, для чего служит тот или иной фрагмент.

5.2. Сайт

Итак, часть файлов из архива образуют структуру сайта, а другая часть — редакторов. Сначала выясним, как взаимодействуют файлы, необходимые для работы сайта.

«Скелет» любой страницы — HTML-код с разметкой (рис. 5.2.1). Непосредственно в теле документа прописаны все его элементы, кроме:

– основного содержания;

– таблицы стилей;

– сценариев на JavaScript (они необходимы для реализации ряда функций сайта).

Созданием разметки на странице управляет «оболочка», написанная на PHP.

Рис. 5.2.1. Взаимодействие файлов сайта

Таблица стилей и сценарии добавляются в HTML-код в процессе загрузки документа, минуя «посредников». А вот основное содержание вставляется с помощью PHP-оболочки из соответствующего txt-файла. В PHP-коде предусмотрен механизм определения, какой контент должен быть загружен для той или иной страницы.

У оболочки есть еще одна функция: проверять, соответствует ли адрес в строке запроса адресу реально существующей страницы. Если искомая страница отсутствует, то никаких предупреждений не выводится, а просто загружается главная страница.

5.3. Редакторы

Устройство редакторов № 1, 2 и 4 отчасти похоже на то, что мы видели у сайта. Также есть основа в виде HTML-разметки и PHP-оболочка. Точно таким же способом внедряются таблица стилей и сценарии. Аналогично выясняется, соответствует ли адрес в строке запроса адресу реально существующей страницы. Но дальше следуют отличия (рис. 5.3.1).

Первое — оболочка загружает контент из файла **index.php**, формирующего запрошенную страницу. Затем основное содержимое извлекается и помещается в окно редактирования.

Второе — файл со сценариями не только участвует в обеспечении функций редактирования. Одна из его задач — отправка данных программе **rec.php** для их записи в соответствующий текстовый файл.

Компоновку третьего редактора мы изучим в главе 12.

Рис. 5.3.1. Взаимодействие файлов редактора

6. Сборка демонстрационного сайта

Демонстрационный сайт находится по адресу **https://editjs.ru/**.

Думаю, не имеет смысла долго объяснять, что задача проекта — не впечатлить читателей навороченным дизайном и сложным контентом, а рассказать, как можно создавать, менять и редактировать этот контент. Поэтому автор решил сделать максимально простой сайт с максимально простой структурой и максимально простым дизайном. Каждая страница ресурса, опять же для простоты, имеет фиксированную ширину (чтобы не усложнять дело адаптивной версткой) и включает три компонента (тоже фиксированной ширины):

– блок меню (имеет еще и фиксированную высоту) с элементами навигации по страницам ресурса;

– блок рекламы — имитирует анонсы новых статей на сайте;

– контейнер для основного содержания, которое может редактироваться.

Также необходимо напомнить, что:

– тема ресурса — астрономия;

– всего сделано 5 страниц;

– страница «Галактики» является одновременно главной — область контента в ней заполнена;

– страницы «Звезды», «Планеты», «Астероиды» и «Кометы» — пустые.

6.1. Пишем заготовку

Компоновка страницы сайта показана на рисунке 6.1.1.

Чтобы получить данную структуру, напишем вот такой HTML-шаблон:

```
<!DOCTYPE html>
<html lang="ru">
<head>
<meta charset="utf-8">
<title>Космические объекты</title>
 _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
|                                                 |
| Подключаем таблицы стилей и файл сценария       |
|_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _|

</head>

<body>
<div id="basis">

   <table id="tab">
   <tr><td id="menu" colspan="2">
    _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
   |                                                 |
   | Меню                                            |
   |_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _|

   </td></tr>
```

```
    <tr>
    <td id="main">
       <div id="cont">
        _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
       |                                                 |
       | Основное содержание                             |
       |_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _|

       </div>
    </td>
    <td id="rec">
     _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
    |                                                 |
    | Реклама                                         |
    |_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _|

    </td></tr>
    </table>

</div>
</body>
</html>
```

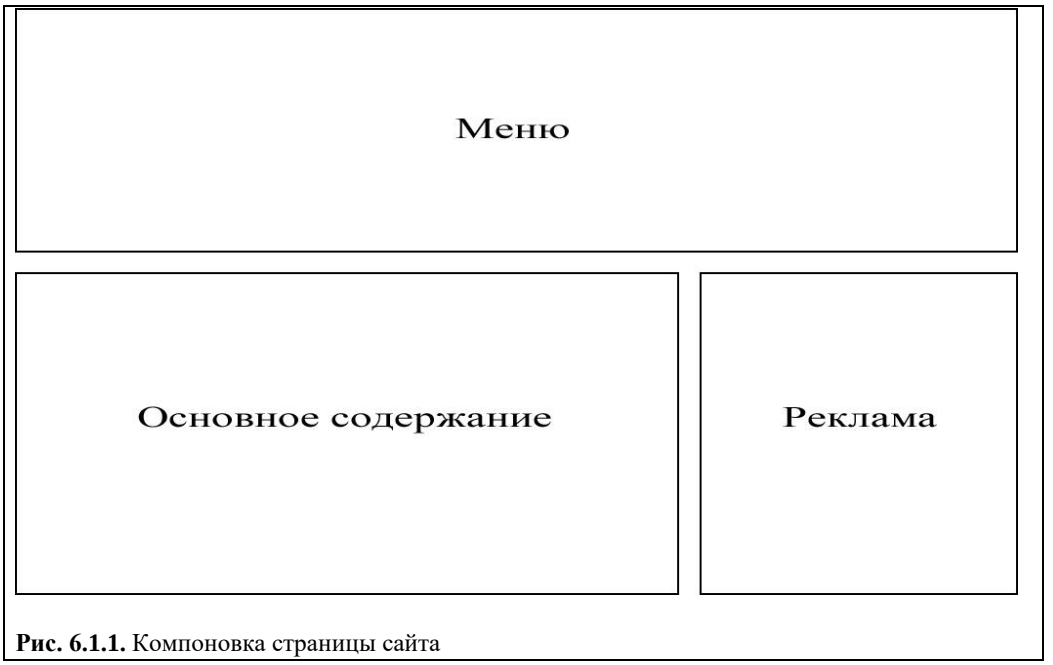

Рис. 6.1.1. Компоновка страницы сайта

Как видите, сайт получил название в соответствии с темой:

```
<title>Космические объекты</title>
```

Заголовочная часть — стандартная для ресурсов, написанных на HTML 5. Есть указание типа документа, языка и кодировки:

```
<!DOCTYPE html>
```

```
<html lang="ru">
...
<meta charset="utf-8">
```

Здесь же подключаются файлы таблиц стилей и сценариев.

Основой каждой страницы является слой **div**:

```
<div id="basis">
...
</div>
```

Его настройки в таблице стилей таковы, что ширина страницы **1000px**, а располагается она строго посередине окна браузера:

```
#basis {position: relative; width: 1000px; margin: auto;}
```

Для «прочности» структуры все элементы видимой части документа помещаются в ячейках таблицы:

```
<table id="tab">
<tr><td id="menu" colspan="2">
...
</td></tr>
<tr>
<td id="main">
...
</td>
<td id="rec">
...
</td></tr>
</table>
```

Верхняя строка таблицы имеет одну ячейку, нижняя разбита на две:

– основное содержание (ширина — 700px);

– блок рекламы (ширина — 300px).

Также обращаю внимание, что ячейка для контента (**id="main"**) имеет внутри слой

```
<div id="cont">
...
</div>
```

Он необходим для размещения основного содержания и взаимодействия с редакторами.

Ссылки, которые необходимы для навигации по страницам, размещены в секции **nav**:

```
<nav id="sect">
<a href="index.php?a=index">
<h4 class="lin">ГАЛАКТИКИ</h4></a>
<a href="index.php?a=star">
<h4 class="lin">ЗВЕЗДЫ</h4></a>
```

```
<a href="index.php?a=planet">
<h4 class="lin">ПЛАНЕТЫ</h4></a>
<a href="index.php?a=asteroid">
<h4 class="lin">АСТЕРОИДЫ</h4></a>
<a href="index.php?a=comet">
<h4 class="lin">КОМЕТЫ</h4></a>
</nav>
```

Если на странице в области контента есть какое-либо изображение, посетитель может увеличить его, щелкнув мышью по картинке. При этом страница будет закрыта белым полупрозрачным слоем с увеличенным изображением. Если теперь кликнуть на нем или прокрутить страницу, увеличенная картинка и полупрозрачный слой исчезнут. Управляет данным процессом сценарий, написанный на JavaScript. Его мы разберем позже, а пока выясним, что необходимо добавить в HTML-код страниц для просмотра увеличенных фотографий.

Автор добавил в разметку вот такой фрагмент:

```
<div id="bas"></div>
<div id="view">
<img id="pict" src="img/net.jpg" alt="Фото">
</div>
```

Как видите, здесь 2 слоя. Нижний

```
<div id="bas"></div>
```

имеет белый фон с уровнем непрозрачности **0.95**. Верхний

```
<div id="view">
...
</div>
```

служит контейнером для увеличенного изображения:

```
<img id="pict" src="img/net.jpg" alt="Фото">
```

В исходном состоянии на данный слой загружено изображение **net.jpg** белого цвета размером 1×1 пиксель. Сделано это для того, чтобы в очередной раз угодить требованиям стандартов Консорциума Всемирной Паутины, которые не допускают в разметке изображений с пустым атрибутом **src**.

При загрузке увеличенного рисунка белый фон занимает все пространство окна браузера, а картинка располагается строго посередине (рис. 6.1.2).

Для вставки данного фрагмента кода автор выбрал место сразу после открывающего тега **body**:

```
...
<body>
<div id="bas"></div>
```

```
<div id="view">
<img id="pict" src="img/net.jpg" alt="Фото">
</div>

<div id="basis">
...
```

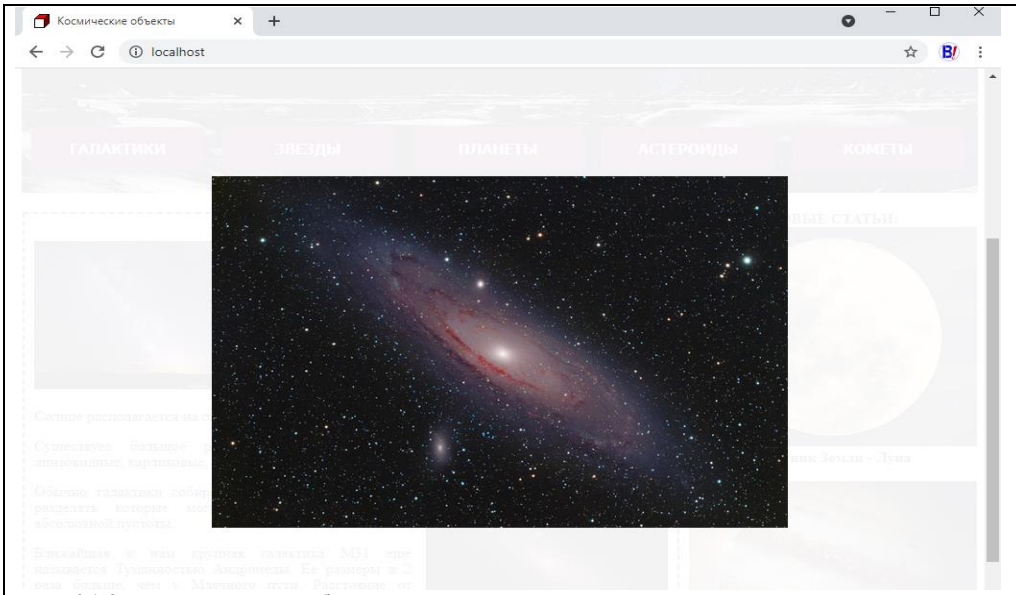

Рис. 6.1.2. Увеличенное изображение на странице

6.2. Файл таблицы стилей

Файлы с таблицами стилей подключаем обычным способом, поместив необходимые инструкции в головной части документа:

```
<link rel="stylesheet" href="set/index.css">
<link rel="stylesheet" href="set/cont.css"
```

Отступление от темы

Многие программисты привыкли внедрять таблицы стилей методом, показанным выше, и даже не догадываются, что существует еще один способ. Добавить файл CSS к HTML-документу можно с помощью директивы ***@import****. В этом случае подключение необходимо выполнить внутри блока head таким образом:*

```
<style>
@import url(name.css);
</style>
```

где ***name.css*** *— адрес файла с таблицей стилей.*

Важно! Вторая таблица стилей с адресом **set/cont.css** актуальна только для второго редактора. Если вы в собственной версии сайта планируете использовать другой редактор, удалите ссылку на данную таблицу стилей. Зачем и когда она нужна, мы выясним позже.

6.3. Файл со сценариями

В отличие от двух файлов CSS, файл JavaScript у сайта всего один. Подключаем его в заголовочной части документа следующим образом:

```
<script src="set/index.js"></script>
```

Что мы найдем в файле сценария? Две функции. Первая необходима для демонстрации увеличенного изображения, вторая — для выхода из режима просмотра.

Регистрация необходимых обработчиков событий выполняется после загрузки страницы:

```
addEventListener("load", function()
{
...
});
```

Для сокращения кода, в котором несколько раз происходят обращения к одним и тем же компонентам, создадим три переменные:

```
let view=document.getElementById("view");
let bas=document.getElementById("bas").style;
let pict=document.getElementById("pict");
```

Теперь зарегистрируем обработчик для клика в области основного содержания. Его роль у нас будет выполнять анонимная функция, получающая в качестве аргумента объект события:

```
document.getElementById("cont").
       addEventListener("click", function(ev)
   {
   ...
   });
```

После щелчка, произошедшего на слое **cont**, первым делом надо выяснить, был ли этот щелчок на изображении или нет.

```
if(ev.target.tagName=="IMG")
   {
   ...
   }
```

Если да, определяем, насколько прокручена страница по вертикали

```
let sc=window.pageYOffset;
```

а затем «передвигаем» слой фона и слой с изображением на величину прокрутки:

```
bas.top=sc+"px";
view.style.top=sc+"px";
```

Вычисляем адрес «кликнутого» фото и присваиваем этот адрес атрибуту **src** картинки со слоя:

```
pict.src=ev.target.src;
```

Завершающий этап — включение режима просмотра:

```
view.style.visibility="visible";
bas.visibility="visible";
```

Чтобы выйти из режима просмотра, зарегистрируем один обработчик для двух разных событий:

```
view.addEventListener("click", del);
addEventListener("scroll", del);
```

Первый запускает функцию **del**, когда мы щелкаем по увеличенному изображению или фону, а второй — при прокрутке страницы.

В самой функции всего три инструкции:

```
function del()
   {
   view.style.visibility="hidden";
   bas.visibility="hidden";
   pict.src="img/net.jpg";
   }
```

Первая и вторая скрывают фон и картинку, а третья присваивает рисунку со слоя адрес исходного изображения. Этого момента коснемся подробнее.

В принципе, рисунку со слоя можно было бы и не присваивать исходный адрес точечного изображения. Но! В таком случае при низкой скорости соединения с сервером можно столкнуться с неприятным эффектом: допустим, вы посмотрели одно фото, завершили просмотр и щелкнули на второй фотографии. Вновь запустится режим просмотра, но из-за медленной загрузки второго снимка вы некоторое время будете видеть первый. А это, согласитесь, не очень хорошо. Присваивая **src** значение **img/net.jpg**, мы избавляемся от подобных неприятностей, ведь маленькая белая точка загружается очень быстро и совершенно не будет видна в момент медленной загрузки второго снимка. То есть мы просто будем видеть белый полупрозрачный фон до тех пор, пока не появится следующее изображение.

6.4. Дизайн сайта

Все подготовительные работы завершены, и настало время смотреть, что за сайт у нас получился. Его главную страницу вы можете видеть на рисунке 6.4.1.

Здесь особые рассуждения и описания не нужны, за исключением нескольких простых комментариев.

1. Штриховая рамка вокруг области основного содержания сделана просто для того, чтобы сконцентрировать внимание читателя и посетителя на этом фрагменте страницы, подчеркнув, что именно данная часть и подвергается обработке в редакторе.

2. Все картинки, рисунки, фотографии, использованные в качестве элементов дизайна, а также в качестве контента, предназначенного для добавления на страницы, взяты с сайта бесплатных изображений. Данный сайт разрешает использовать эти изображения без указания авторства — в любых целях, в том числе коммерческих.

3. Все иконки для кнопок главной панели разработаны автором.

4. Текст главной страницы написан автором на основе данных из нескольких источников.

4. Блок рекламы носит чисто изобразительную нагрузку. Анонсированные статьи, как и ссылки на них, не существуют.

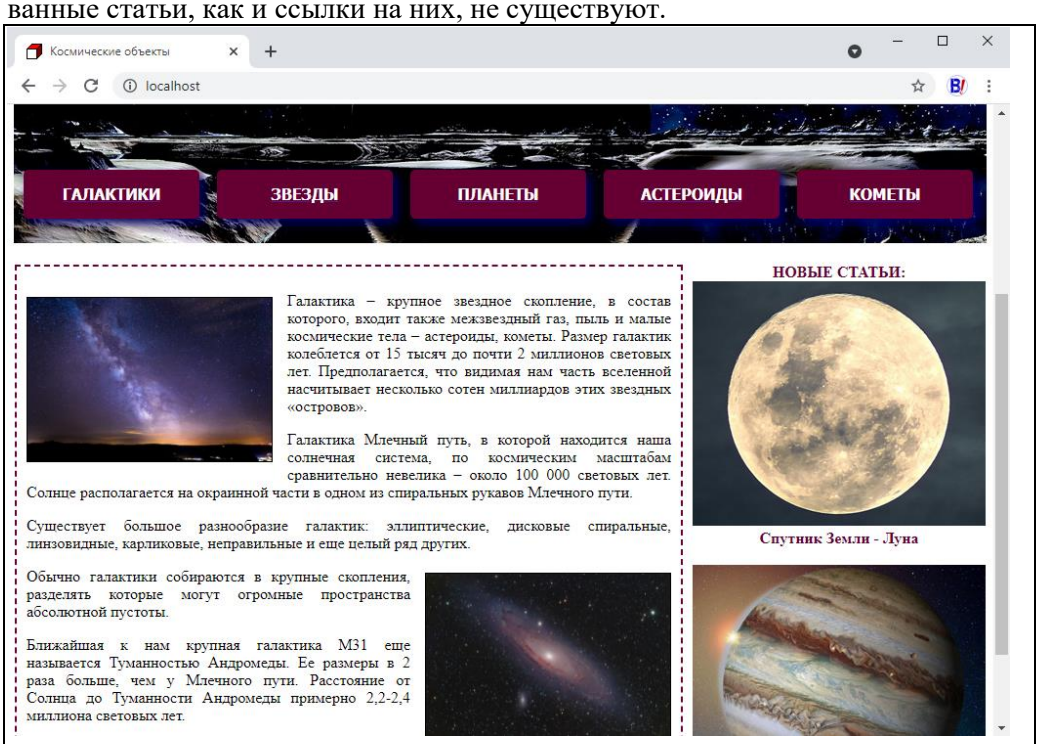

Рис. 6.4.1. Главная страница сайта

6.5. Выделяем редактируемую область

Совсем короткий, но важный раздел. Акцент на области основного содержания актуален еще и по той причине, что все без исключения редакторы получают данные в окно редактирования именно из контейнера

```
<div id="cont">
...
</div>
```

а не из текстового файла, содержащего данный контент.

Включить содержимое текстового файла в HTML-разметку можно двумя способами.

Первый: добавить код из файла с помощью JavaScript методом Ajax — сразу после загрузки документа. Данный подход плох тем, что поисковые системы могут не видеть такого контента, так как он отсутствует в изначальном коде.

Поэтому более правильным видится иной вариант: включение содержимого текстового файла в разметку непосредственно в момент ее «прочтения» браузером с помощью PHP (для этих целей можно использовать и другие языки программирования, но автор изначально сделал выбор в пользу PHP как наиболее подходящего). Подробнее об этом — в следующем разделе.

6.6. Подключаем PHP

Если вы внимательно читали главу 5, то сразу вспомните, что файл-сборщик сайта (**index.php**) — это фактически HTML-файл, заключенный в оболочку PHP. Повторюсь, код данного языка программирования при компоновке страницы используется два раза: для проверки адреса запрошенной страницы и для вывода данных из соответствующего текстового файла в разметку соответствующей страницы.

Разберемся с этими двумя пунктами.

Как вы помните, мы установили правило: если клиент запрашивает несуществующую страницу, то в его браузер загружается главная. Поэтому начинается проверка с того, что мы создаем переменную и присваиваем ей адрес главной страницы:

```
$adr="index";
```

Следом узнаем, передан ли в запросе параметр с адресом страницы:

```
if(isset($_GET["a"]))
   {
   ...
   }
```

Если не передан, то загружаем главную, так как значение переменной **$adr** не изменилось. Например, введя в браузере адрес **https://editjs.ru/**, вы получите страницу «Галактики», которая и является главной.

Если параметр передан, присваиваем его значение новой переменной **$test_adr**:

```
if(isset($_GET["a"]))
   {
   $test_adr=$_GET["a"];
   ...
   }
```

и проверяем, существует ли текстовая страница с таким адресом. Для этого поочередно выясняем имена всех файлов из папки **content**:

```
$opdir=opendir("content");
while($redir=readdir($opdir))
   {
   ...
   }
closedir($opdir);
```

и сравниваем их с адресом, переданным в запросе:

```
if($test_adr.".txt"==$redir)
   {
   ...
   }
```

Совпадения не обнаружены? Значит, переменная **$adr** не изменит своего значения и загрузится главная страница.

Совпадение обнаружено? Переменная **$adr** получает новое значение — адрес страницы, которая должна быть загружена по запросу клиента:

```
$adr=$test_adr;
```

после чего цикл перебора файлов из папки **content** прерывается:

```
break;
```

Вот такой код у нас получился в итоге:

```
$adr="index";

if(isset($_GET["a"]))
   {
   $test_adr=$_GET["a"];

   $opdir=opendir("content");
   while($redir=readdir($opdir))
      {
      if($test_adr.".txt"==$redir)
```

```
        {
        $adr=$test_adr;
        break;
        }
      }
   closedir($opdir);
   }
```

Теперь необходимо загрузить нужную страницу. Но поскольку все страницы формируются по одному шаблону, записанному в файле **index.php**, нам остается только в нужном месте разметки добавить код из текстового файла, адрес которого мы определили выше.

Фрагмент кода, отвечающего за включение содержимого текстового файла в разметку, совсем простой и короткий:

```
<div id="cont">

<?php
$stran="content/".$adr.".txt";
$soder=file_get_contents($stran);
echo $soder;
?>

</div>
```

Составляем путь к файлу на основе полученного адреса страницы **$adr**:

```
$stran="content/".$adr.".txt";
```

за один раз считываем все содержимое файла

```
$soder=file_get_contents($stran);
```

и выводим его на страницу:

```
echo $soder;
```

В заключение несколько рассуждений о безопасности. Использованный нами метод загрузки страницы хорош по двум причинам.

1. Мы не указываем в адресе расширение к имени файла, а добавляем его программным методом. Это расширение строго ограничено одним вариантом — **.txt**.

2. Мы сравниваем имя файла с реально существующими именами из соответствующей папки и формируем строго ограниченный путь.

Таким образом, даже если недоброжелатель захочет ввести какой-либо иной адрес, какой-либо иной путь, чтобы с помощью нашей программы **index.php** открыть или запустить на сервере файл, не предназначенный для посторонних, у него ничего не выйдет. При любых действиях просто откроется главная страница сайта.

7. Структура редакторов

Как выглядит типичный визуальный редактор? Примерно таким образом: окно редактирования и набор кнопок, выполняющих определенные действия с текстом, изображением, таблицами и т. п. Чаще всего в стандартном редакторе еще предусмотрен режим переключения на вкладку с HTML-кодом, чтобы опытный пользователь мог при желании подправить какие-либо фрагменты. При этом в окне визуального редактирования сразу видно, что получится после записи содержимого в файл или базу данных. И лишь одно вы не сможете посмотреть заранее, еще до записи: коррелируются ли результаты вашего творчества с дизайном страницы и сочетается ли созданный вами контент с остальными элементами.

Наш проект избавлен от этого недостатка. Все представленные в книге редакторы позволяют администратору сайта видеть, как будет выглядеть страница еще до записи внесенных изменений. Такой результат достигается за счет того, что в редакторах № 1, 2 и 4 есть встроенные фреймы, в которые загружаются редактируемые страницы. При этом каждое изменение в окне редактора тут же отображается и в области контента страницы. За счет этого вы можете сразу оценить совпадение стилей редактируемых элементов с дизайном сайта.

Редактор № 3 также позволяет «в прямом эфире» наблюдать за соответствием вносимых изменений дизайну ресурса, но иным способом, о чем автор расскажет в свое время.

Основная, базовая структура редакторов № 1, 2 и 4 идентична. В них есть:

- вкладка с окном визуального редактирования;
- вкладка с окном HTML-редактора;
- главная панель с набором основных кнопок;
- скрытые панели настроек элементов, которые появляются при нажатии соответствующих кнопок главной панели;
- фрейм со страницей, основное содержание которой загружено в редактор.

Структура редактора № 3 несколько отличается от изложенной выше. Ее мы разберем в главе 12.

7.1. Пишем заготовку

Компоновка страниц редакторов № 1, 2 и 4 показана на рисунке 7.1.1.

Чтобы получить данную структуру, напишем следующий HTML-шаблон:

```
<!DOCTYPE html>
<html lang="ru">
<head>
<meta charset="utf-8">
<meta name="robots" content="noindex, nofollow">
<title>Визуальный редактор</title>
 _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
|                                           |
| Подключаем таблицы стилей и файл сценария |
|_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _|
```

```
</head>

<body>
<div id="basis">
 _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
|                                           |
| Вкладки настроек                          |
|_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _|

   <div id="panel">
    _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
   |                                           |
   | Главная панель                            |
   |_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _|

   </div>
 _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
|                                           |
| Кнопки переключения между редакторами     |
|_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _|
 _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
|                                           |
| Текстовый и WYSIWYG-редакторы             |
|_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _|
 _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
|                                           |
| Фрейм со страницей сайта                  |
|_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _|

</div>
</body>
</html>
```

Первое новшество, по сравнению с кодом разметки сайта, — мета-тег **robots** в заголовочной части:

```
<meta name="robots" content="noindex, nofollow">
```

Данный фрагмент запрещает поисковым роботам индексирование документа и переход с него по ссылкам, если они есть в текстах данной страницы.

Остальная часть разметки довольно простая. Есть основа каждой страницы — слой **div**:

```
<div id="basis">
...
</div>
```

Его настройки в таблице стилей таковы, что ширина страницы **1000px**, высота — **1410px**, а располагается она строго посередине окна браузера:

```
#basis {position: relative; width: 1000px;
        height: 1410px; margin: auto;}
```

Думаю, назначение остальных компонентов структуры понятно без лишних комментариев. Подробнее об окнах редактирования, элементах главной панели, вкладок будет рассказано в главе 8.

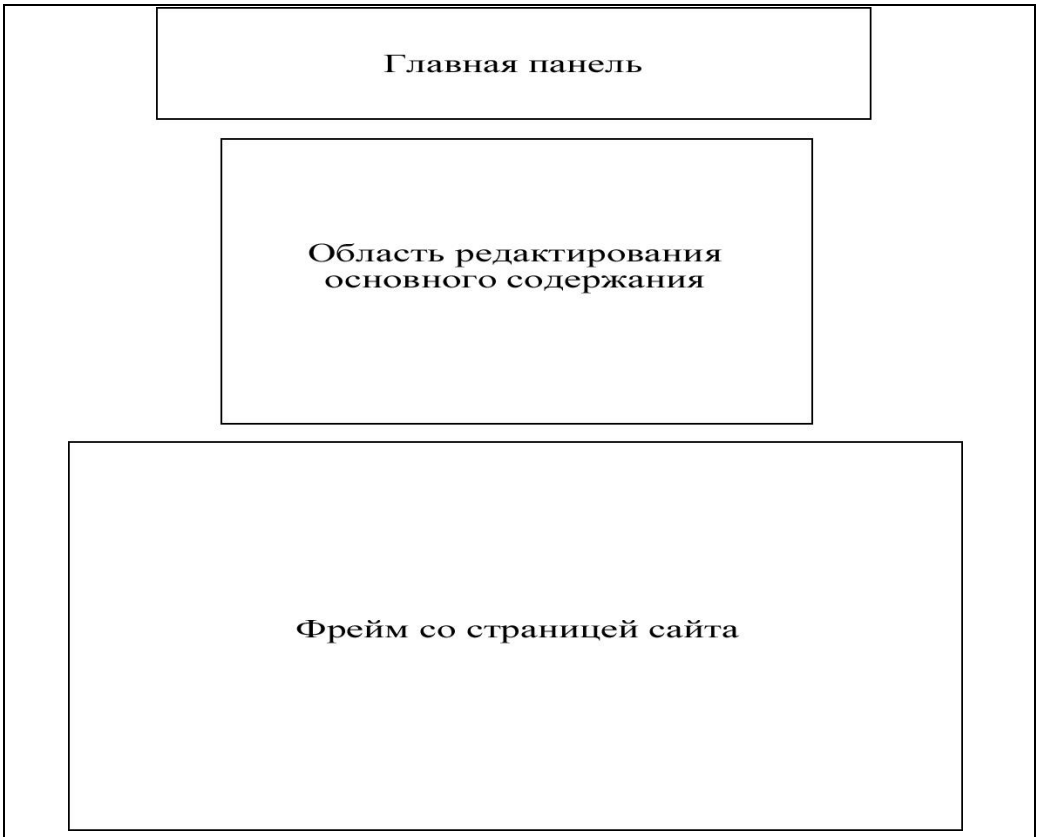

Рис. 7.1.1. Компоновка страницы редактора (за исключением третьего — креативного)

7.2. Файл таблицы стилей

Файлы с таблицами стилей, как и у сайта, подключаем обычным способом, поместив необходимые инструкции в головной части документа. Так как в структурах редакторов есть некоторые отличия, каждый из них имеет свой собственный файл CSS. Создавая проект, автор решил не писать единой таблицы стилей, так как в этом случае, выбрав какой-то один редактор, вы получили бы к нему CSS-файл с избыточным кодом. Это несколько «утяжеляет» zip-архив, но «облегчает» каждый редактор в отдельности.

На примере второго редактора покажем внедрение таблиц стилей в код страницы:

```
<link rel="stylesheet" href="set/editor2.css">
<link rel="stylesheet" href="set/cont.css">
```

Важно! Вторая таблица стилей с адресом **set/cont.css** импортируется только во второй редактор. Другим редакторам она не нужна.

7.3. Файл со сценариями

Каждый редактор имеет персональный файл сценариев на JavaScript. Покажем его подключение на примере редактора № 1:

```
<script src="set/editor1.js"></script>
```

Как и в ситуации с таблицами стилей, автор решил не писать единого файла сценария, так как в этом случае получилась бы огромная программа с таким количеством кода, что разбираться в нем было бы проблематично. Отдельные файлы, хотя и содержат повторяющиеся фрагменты, заметно короче и проще читаются. При этом повторяется ситуация, когда несколько программ «утяжеляют» zip-архив, но «облегчают» каждый редактор в отдельности.

Основные функции таких файлов:

– выбор загружаемой страницы;

– загрузка основного содержания в окно визуального редактора;

– обеспечение взаимодействия WYSIWYG и текстового редакторов;

– открытие и закрытие вкладок с настройками;

– простое редактирование текста — преобразование его фрагментов к полужирному, подчеркнутому, наклонному начертанию и т. п.;

– удаление форматирования из текста;

– добавление различных элементов — рисунков, таблиц, линий и т. д.;

– непрерывная передача результатов из окна редактора в область контента редактируемой страницы для наблюдения за соответствием внесенных изменений дизайну сайта;

– навигация по внесенным изменениям с возможностью отмены и возврата промежуточных результатов;

– отправка данных для записи;

– вывод сообщений, подтверждающих успешную запись.

7.4. Добавляем фрейм для просмотра страницы

Как уже было сказано выше, для наблюдения за соответствием вносимых изменений дизайну сайта в нижней части страницы редакторов № 1, 2 и 4 есть фрейм, куда загружается редактируемая страница:

```
<iframe id="edit" src="vacant.html"></iframe>
```

Назначение страницы **vacant.html** мы уже объясняли: она носит чисто декоративную функцию на случай проверки разметки редактора в валидаторе HTML.

7.5. Дизайн редакторов

Полную страницу редактора вы можете видеть на рисунке 7.5.1.

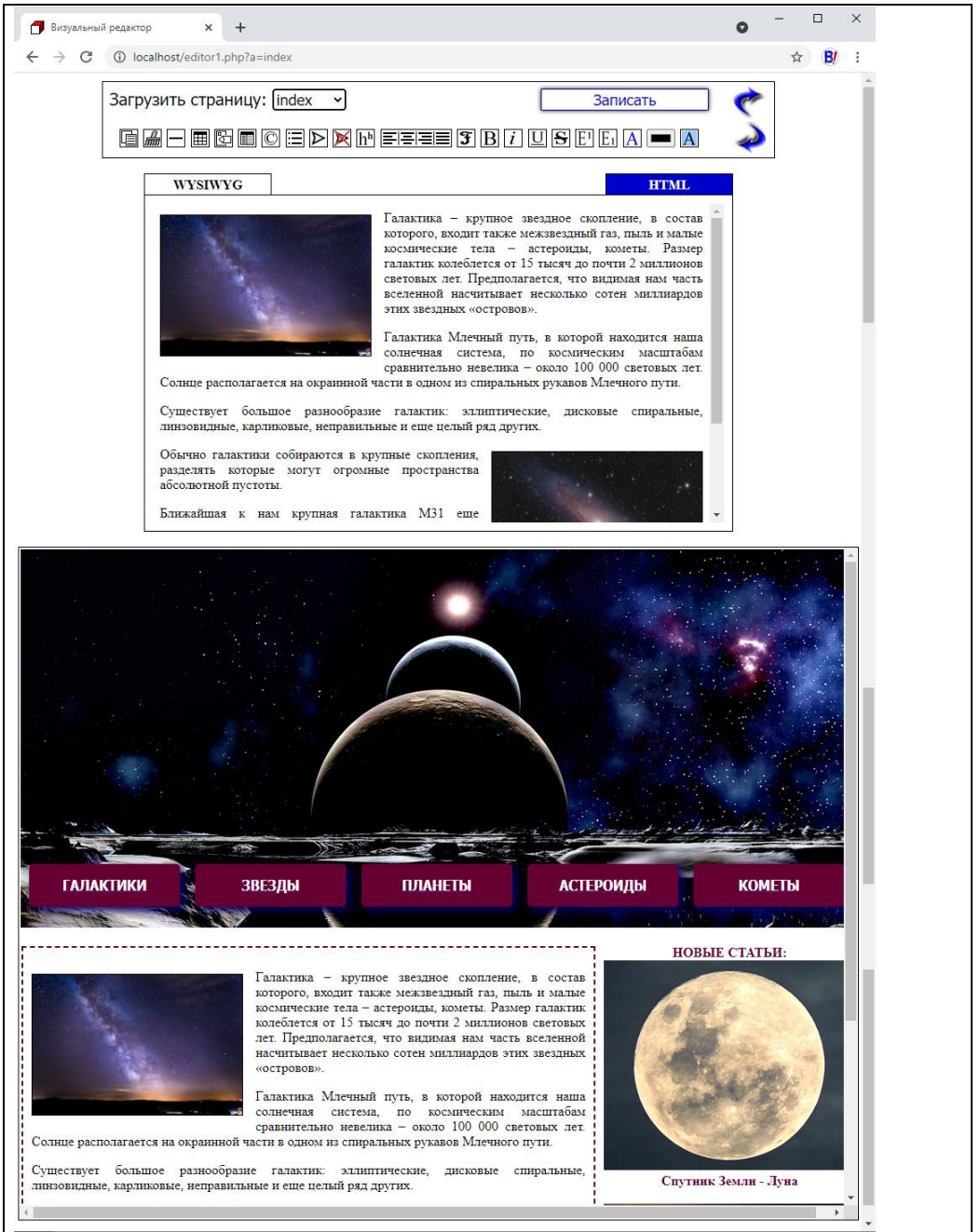

Рис. 7.5.1. Редактор № 1 с загруженной главной страницей

В исходном состоянии вкладки настроек не видны. Они выводятся поверх главной панели при нажатии соответствующих кнопок редактирования. Чтобы скрыть вкладку, необходимо щелкнуть на пиктограмме закрытия или на кнопке «Вставить».

Главная панель, помимо кнопок редактирования, содержит:

– список страниц, которые могут быть загружены в редактор;

– кнопку записи результатов редактирования;

– кнопки навигации по внесенным изменениям.

Для переключения с визуального редактора на HTML и обратно используются вкладки WYSIWYG и HTML. Вкладка неактивного в данный момент окна редактирования выделяется синим цветом.

7.6. Подключаем PHP

Оболочка PHP любого из редакторов используется чаще, чем аналогичная оболочка сайта. Совпадает одна функция — проверки адреса запрошенной страницы. Этот адрес нужен, чтобы:

– при отправке отредактированных данных сообщить программе rec.php, в какой файл необходимо произвести запись;

– загрузить во фрейм необходимую страницу и получить из нее редактируемое содержимое.

Также PHP требуется для загрузки списка существующих на сервере фотографий, списка внутренних страниц сайта (для внутренних ссылок) и списка страниц, которые можно загрузить в редактор. О последних трех случаях использования PHP мы поговорим в главе 8.

Кроме того, считаю необходимым напомнить, что во всех редакторах, за исключением «креативного», не предусмотрена проверка, откуда поступает запрос: от авторизованного источника или от недоброжелателя. Поэтому в файлы редакторов № 1, 2 и 4 нужно будет добавить код проверки, который зависит от способа аутентификации и авторизации администратора. Выбор способа — за вами.

Итак, проверка введенного адреса. Сначала выясняем, указан ли в запросе параметр с адресом страницы:

```
if(isset($_GET["a"]))
```

Если да, то присваиваем значение параметра переменной **$test_adr**:

```
$test_adr=$_GET["a"];
```

Если нет, это, скорее всего, означает, что-либо ошибся администратор, либо в редактор пытается проникнуть недоброжелатель. Выводим предупреждение об ошибке и на всякий случай прерываем выполнение программы (а не загружаем главную страницу, как в файле сайта):

```
else
   {
```

```
   echo "НЕКОРРЕКТНЫЙ ЗАПРОС!";
   return;
   }
```

Данные поступили? В этом случае создаем переменную-идентификатор **$ch**:

```
$ch=0;
```

и проверяем, существует ли текстовая страница с адресом из запроса. Для этого поочередно выясняем имена всех файлов из папки **content**

```
$opdir=opendir("content");
while($redir=readdir($opdir))
   {
   ...
   }
closedir($opdir);
```

и сравниваем их с адресом, переданным в запросе:

```
if($test_adr.".txt"==$redir)
   {
   ...
   }
```

Совпадения не обнаружены? Опять проблема может быть в ошибке администратора или в попытке недоброжелателя загрузить страницу, не предназначенную для доступа постороннему. В этой ситуации переменная-идентификатор не изменит своего значения, что служит сигналом для вывода предупреждения и остановки программы:

```
if($ch==0)
   {
   echo "НЕКОРРЕКТНЫЙ АДРЕС !";
   return;
   }
```

Совпадение обнаружено? Следуют три действия:
– создаем новую переменную **$adr**, которая получает значение — адрес страницы, предназначенной для загрузки в редактор:

```
$adr=$test_adr;
```

– присваиваем идентификатору новое значение:

```
$ch=1;
```

– прерываем цикл:

```
break;
```

Полный код блока проверки:

```
if(isset($_GET["a"]))
   $test_adr=$_GET["a"];
else
   {
   echo "НЕКОРРЕКТНЫЙ ЗАПРОС !";
   return;
   }

$ch=0;
$opdir=opendir("content");
while($redir=readdir($opdir))
   {
   if($test_adr.".txt"==$redir)
      {
      $adr=$test_adr;
      $ch=1;
      break;
      }
   }
closedir($opdir);

if($ch==0)
   {
   echo "НЕКОРРЕКТНЫЙ АДРЕС !";
   return;
   }
```

Отступление от темы
Любопытно, что PHP 8 стал более строг по отношению к параметрам. Раньше можно было просто написать ***$test_adr=$_GET["a"]*** *и не заботиться о том, существует ли данный параметр в запросе или нет. В любом случае проверка условия* ***if($test_adr.".txt"==$redir)*** *прошла бы корректно. В PHP 8 такой подход при отсутствии в запросе параметра вызовет ошибку. Поэтому автор рекомендует всегда проверять наличие параметра, например так:* ***if(isset($_GET["a"]))****.*

Следующий этап — присвоение имени страницы скрытому полю, из которого при передаче данных программа на сервере извлечет информацию о том, в какой файл необходимо производить запись:

```
<form name="dat">
...
<input type="hidden" name="str"
                     value="<?php echo $adr; ?>">
</form>
```

Последний этап — загрузка той страницы, чье имя мы получили из параметра:

```
<iframe id="page"
src="<?php echo "index.php?a=".$adr; ?>">
</iframe>
```

8. Базовые компоненты редакторов

Готовя материал для книги, автор хотел показать самые разные варианты создания редакторов. Например, первый из них построен на основе фрейма в качестве окна визуального редактирования с включенным свойством **designMode** (**designMode="on"**). В остальных окно редактирования — это слой **div** с атрибутом **contentEditable="true"**. В первых трех редакторах для создания элементов и оформления текста использованы методы **createElement** и **surroundContents**. В четвертом — ретро-метод **execCommand**. В редакторах № 1, 3 и 4 администратор сам настраивает большинство параметров стилей. Во втором редакторе варианты стилей ограничены предустановленным набором.

Но несмотря на все различия, каждый редактор имеет определенный набор одинаковых компонентов. Во-первых, это панели. В общей сложности их 10: главная и 9 панелей для выбора параметров добавляемых в страницу элементов. Во-вторых, это окно редактирования визуального представления кода (только у «креативного» редактора оно отсутствует). В-третьих, окно HTML-редактора. В-четвертых, кнопки переключения между редакторами. И в-пятых, фрейм, в котором мы видим необходимую страницу (отсутствует у «креативного» редактора).

Фрейм из пятого пункта мы уже посмотрели в предыдущей главе и выяснили, как в него загружается необходимая страница. Устройство остальных компонентов разберем подробнее.

8.1. WYSIWYG и текстовый режимы

Окна визуального и текстового редакторов (рис. 8.1.1 и 8.1.2) имеют на странице одинаковые координаты, прописанные в соответствующей таблице стилей. Для переключения режимов есть специальные вкладки (или кнопки — кому как удобнее называть) с надписями «WYSIWYG» и «HTML». Вкладка неактивного в данный момент окна редактирования выделяется синим цветом. По умолчанию активным окном сразу после загрузки является визуальное. При переключении на HTML-режим вкладка визуального окна меняет фон на синий, а цвет букв — на белый.

Читателям необходимо обратить внимание, что кнопки редактирования в HTML-режиме не работают!

Все изменения, которые произошли с контентом в визуальном режиме, передаются в HTML-окно в момент переключения закладок с WYSIWYG на HTML. В обратном направлении — в визуальное окно — передача кода выполняется при обратном переключении закладок. Кнопка «Записать» активна только в визуальном режиме, но при этом все данные для записи берутся из текстового поля. Такой способ выбран, чтобы вся информация концентрировалась в одном месте в одном контейнере. Добавим, что текстовые поля во всех редакторах идентичны.

Контейнер **form** необходим для передачи данных серверной программе записи. Отправка данных выполняется по технологии **Ajax** методом **POST**, то есть информация поступает не в строке запроса, а в его теле.

В контейнере два поля: текстовое с кодом основного содержания

```
<textarea id="tex" name="tex"></textarea>
```

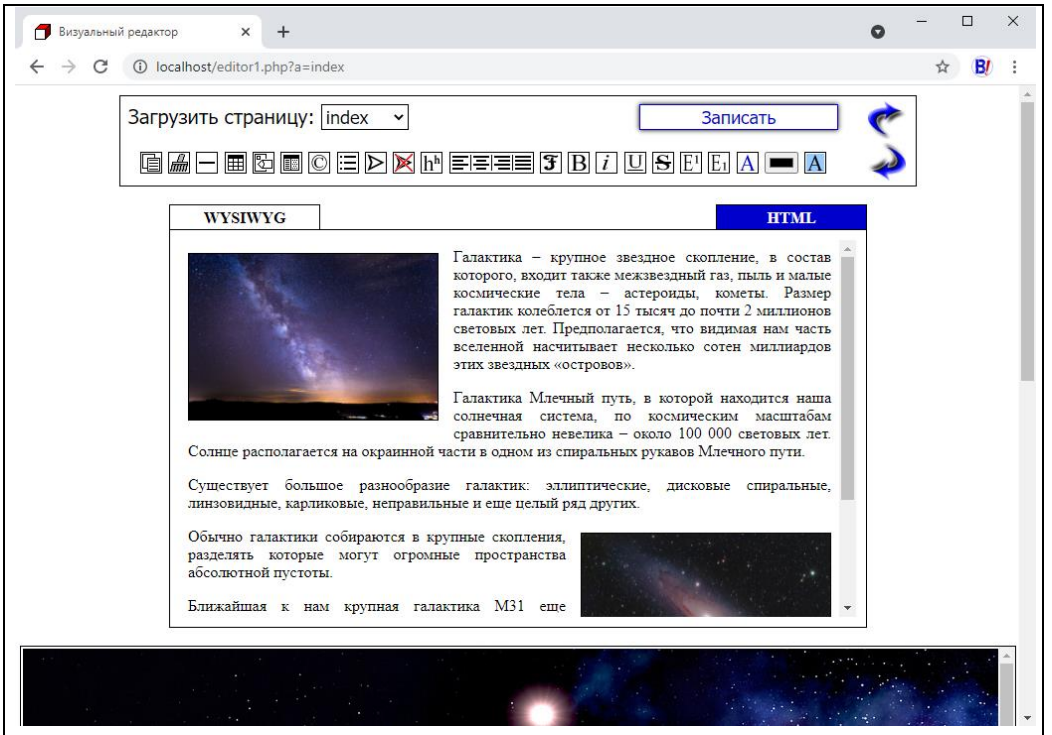

Рис. 8.1.1. Окно визуального редактора

и скрытое — с именем файла, предназначенного для перезаписи

```
<input type="hidden" name="str"
                     value="<?php echo $adr; ?>">
```

Как уже было сказано, во всех редакторах, включая «креативный», есть вкладка перехода в HTML-окно:

```
<div id="htm">HTML</div>
```

Во всех редакторах (но не в «креативном») есть вкладка перехода в WYSIWYG-режим:

```
<div id="wys">WYSIWYG</div>
```

Окно визуального представления в первом редакторе образуется фреймом:

```
<iframe id="edit" src="vacant.html"></iframe>
```

После загрузки редактора пустой файл **vacant.html** подменяется основным содержанием страницы.

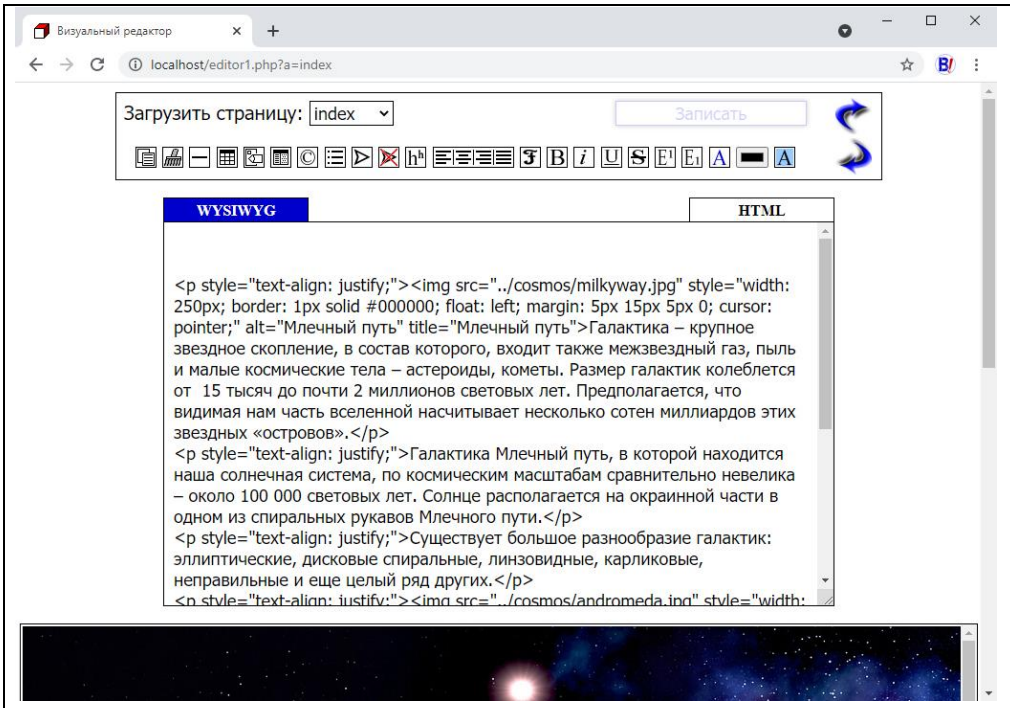

Рис. 8.1.2. Окно текстового редактора

Во втором и четвертом редакторах окно визуального представления создано на основе **div**:

```
<div id="edit" contenteditable="true"></div>
```

В очередной раз напоминаю, что третий редактор у нас довольно нестандартный, поэтому о его особенностях будет отдельный разговор в главе 12.

8.2. Главная панель

Главная панель находится в самой верхней части любого редактора (рис. 8.2.1). Она содержит:

– список для выбора загружаемой страницы;

– кнопку записи данных;

– кнопки навигации по внесенным изменениям (находятся у правого края панели);

– 21 кнопку и окно выбора цвета, выстроенные в ряд.

Разберемся с этими элементами по порядку.

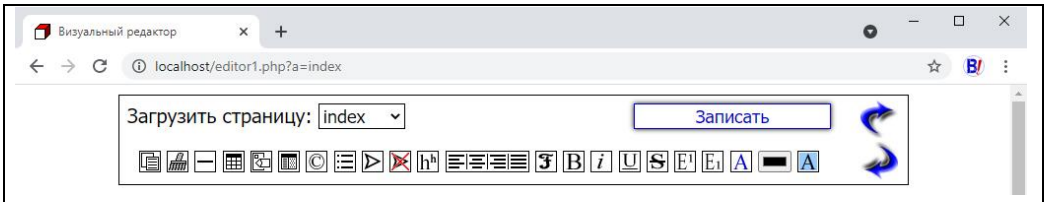

Рис. 8.2.1. Главная панель

Со списком все очень просто: раскрываете его, выбираете интересующую страницу, нажимаете на ее имя, и страница загружается в редактор.

Чтобы вывести перечень страниц, напишем вот такой код:

```
Загрузить страницу:
<select id="sel">
<option selected value="<?php echo $adr; ?>">
                          <?php echo $adr; ?></option>
<?php
$opdir=opendir("content");
while($redir=readdir($opdir))
   {
   if(!preg_match("/$adr/", $redir)
                              && strpos($redir, "."))
      {
      $rep=str_replace(".txt", "", $redir);
      echo '<option value="'.$rep.'">'.$rep.
                                          '</option>';
      }
   }
closedir($opdir);
?>
</select>
```

Основное действующее лицо здесь — PHP.

Во-первых, сценарий формирует установленную по умолчанию строку в выпадающем списке на основе переменной **$adr**, значение которой получено в процессе загрузки страницы:

```
<option selected value="<?php echo $adr; ?>">
                          <?php echo $adr; ?></option>
```

Во-вторых, сценарий открывает папку **content** и считывает из нее имена всех текстовых файлов:

```
$opdir=opendir("content");
while($redir=readdir($opdir))
```

```
   {
   ...
   }
closedir($opdir);
```

К сожалению, в список попадут:

– страница, которая уже выведена первой строкой в списке;

– такие элементы, как точка и две точки (**.** и **..**), которые обозначают текущий и вышестоящий каталоги.

Чтобы преодолеть эти проблемы, устроим проверку:

```
if(!preg_match("/$adr/", $redir)
                              && strpos($redir, "."))
   {
   ...
   }
```

Первая часть условия

```
!preg_match("/$adr/", $redir)
```

проверяет, не является ли текущее имя файла совпадением с именем из переменной **$adr**.

Вторая часть условия «отсекает» обозначения каталогов:

```
strpos($redir, "."))
```

Точка и две точки вернут ноль — индекс первого вхождения элемента. Условие **if** воспримет данный результат как **false** и не включит эти элементы в список. Зато у остальных файлов, у которых точка стоит перед расширением после нескольких символов имени, индекс вхождения окажется больше нуля, и условие примет значение **true**. То есть реальные файлы пройдут «тест».

Печать очередной строки происходит только при одновременном выполнении двух условий: полученное в цикле значение не совпадает со значением, указанным первым пунктом списка, и не является символом какого-либо каталога.

Первый этап печати — удаление расширения **.txt** из имени файла:

```
$rep=str_replace(".txt", "", $redir);
```

Так сделано для упрощения списка. Позже, при загрузке следующей страницы, это расширение будет опять добавлено к имени.

Второй и последний этап — собственно формирование кода очередной строки из списка:

```
echo '<option value="'.$rep.'">'.$rep.'</option>';
```

Теперь про кнопку записи данных. Она выполняет также функцию вывода сообщений о результатах этого процесса. Если запись была успешной, на кнопке на 2 секунды возникает текст «ЗАПИСАНО». При неудаче появляется сообщение «НЕ ЗАПИСАНО» такой же продолжительности. После чего возвращается исходная надпись «Записать». HTML-код кнопки очень простой:

```
<input type="button" value="Записать" id="rec">
```

Навигация по этапам редактирования напоминает аналогичную функцию, например из программы Word. Верхняя изогнутая стрелка служит для перемещения по промежуточным результатам редактирования от исходного состояния к финишному. Нижняя стрелка загружает промежуточные результаты в обратном порядке. Сигналом для записи промежуточного состояния контента в память служит нажатие одного из знаков препинания, изменение форматирования текста или вставка какого-либо элемента.

Кнопки навигации записываются так:

```
<img src="pict/forw.jpg" id="forw" alt="Вперед"
                                        title="Вперед">
<img src="pict/back.jpg" id="bac" alt="Назад"
                                         title="Назад">
```

В качестве основы для главной панели служит слой **div** и таблица с жестко заданными размерами:

```
<div id="panel">

<table id="pan">
<tr>

   <td id="td1">
   Загрузить страницу:
   <select id="sel">
   ...
   <!-- Список страниц -->
   ...
   </select>
   </td>

   <td id="td2">
   <!-- Кнопка записи -->
   </td>

</tr>
<tr>

   <td id="td3" colspan="2">
   ...
   <!-- Кнопки редактирования -->
   ...
   </td>
```

```
</tr>
</table>

<!-- Кнопки навигации по внесенным изменениям -->
</div>
```

Хотя кнопки при наведении указателя мыши дают подсказки, для чего они предназначены (рис. 8.2.2), на всякий случай перечислим их функции. Будем считать кнопки слева направо.

1. Копирование фрагмента. В редакторах № 1–3 копирует только текстовое содержимое. Если необходимо скопировать фрагмент вместе с форматированием и элементами, выделите его, нажмите правую кнопку мыши и в контекстном списке выберите пункт «Копировать». В редакторе № 4 копирует все выделенное содержимое, включая теги форматирования и элементы. Если вам в четвертом редакторе хочется сохранить систему копирования такую же, как и в первых трех, то в разделе 13.2 вы найдете подсказку, как это сделать.

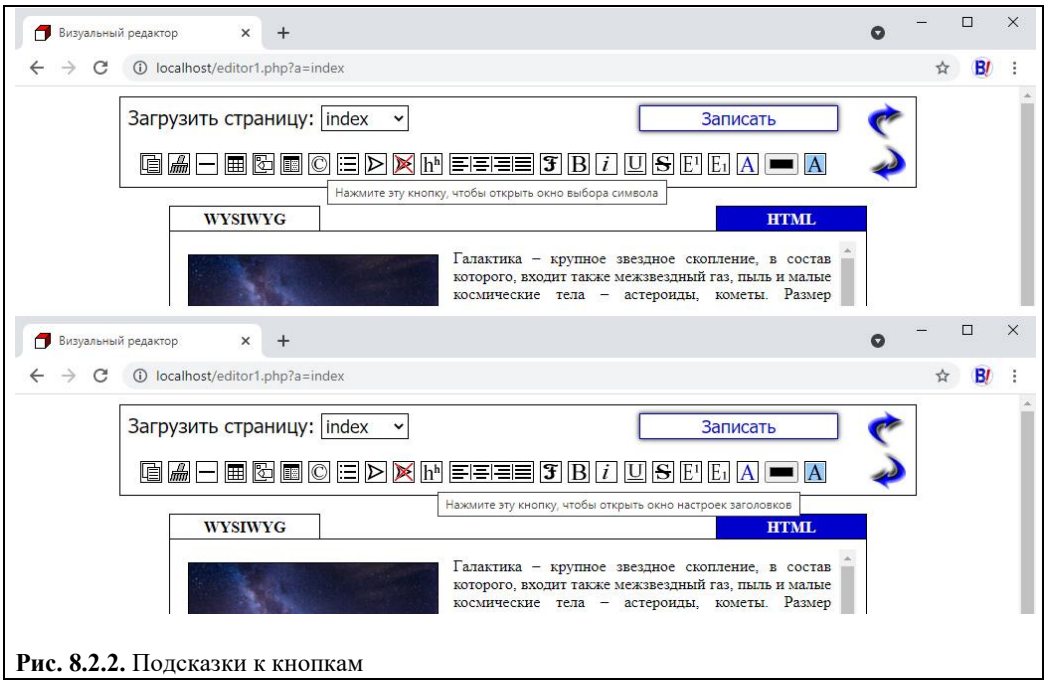

Рис. 8.2.2. Подсказки к кнопкам

2. Удаление форматирования. В редакторах № 1–3 удаляет все HTML-теги из выделенного фрагмента. В редакторе № 4 — только теги форматирования текста и стили ссылок.

3. Вставка линии. При нажатии этой кнопки открывается вкладка с настройками линии. Закрывается либо при нажатии на пиктограмму в виде крестика, либо при вставке линии в окно редактирования. Правило

распространяется на все остальные вкладки, поэтому в дальнейшем автор не станет напоминать о нем.

4. Вставка таблицы. Открывает вкладку с настройками таблицы.
5. Вставка рисунка. Открывает панель выбора и настроек рисунка.
6. Вставка фрейма. Открывает окно с настройками фрейма.
7. © Выбор символа. Выводит на экран список символов, которые можно вставить в текст.
8. Маркированный список. Необходимо сделать выделение и нажать данную кнопку, чтобы превратить текст в маркированный список.
9. Добавить ссылку. Открывает окно настроек ссылки.
10. Удалить ссылку. Удаляет выделенную ссылку, оставляя только ее текст.
11. h^h Добавить заголовок. Открывает вкладку с настройками заголовков.
12. Выравнивание текста. Выводит панель с настройками выравнивания.
13. Выбор шрифта. Открывает окно с настройками шрифта.
14. **B** Выделение жирным. Необходимо выделить текст и нажать кнопку. У следующих пяти кнопок принцип действия аналогичен. Нужно сделать выделение в тексте и нажать кнопку.
15. *i* Выделение курсивом.
16. U Выделение подчеркиванием.
17. S Выделение зачеркиванием.
18. E^1 Сделать фрагмент текста надстрочным.
19. E_1 Сделать фрагмент текста подстрочным.

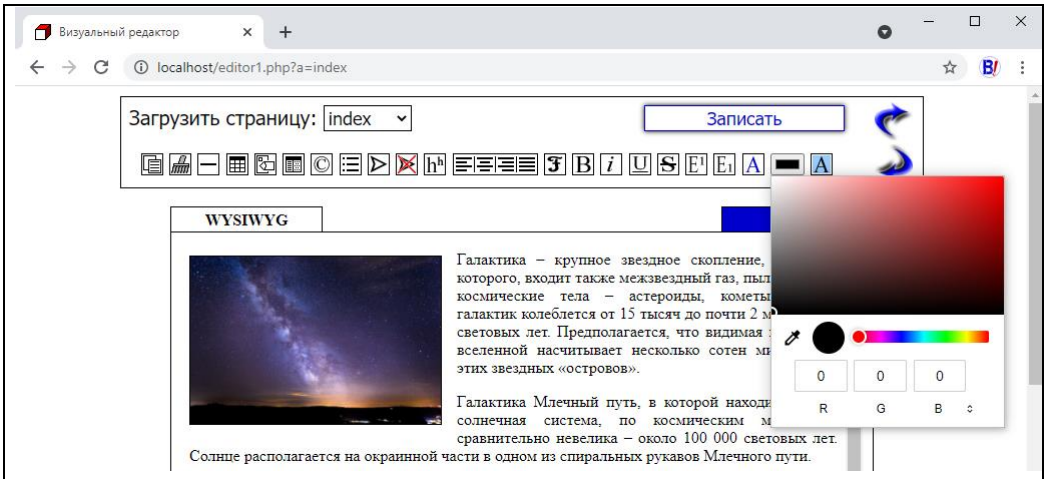

Рис. 8.2.3. Окно выбора цвета в редакторах № 1, 3 и 4

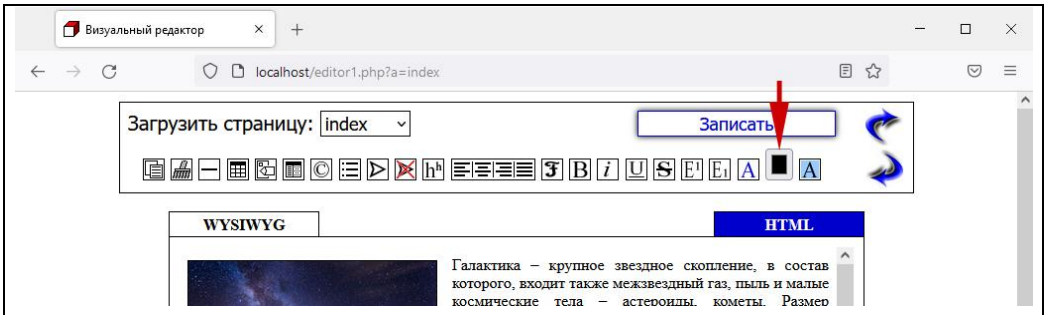

Рис. 8.2.4. Поле выбора цвета у браузера Firefox игнорирует размеры, установленные в таблице стилей

20. A Сделать текст цветным. Сначала необходимо открыть окно настроек цвета, выбрать интересующий оттенок, выделить фрагмент и нажать данную кнопку.

21. Открыть окно выбора цвета. Открывает вкладку с палитрой цветов (рис. 8.2.3). Я уже показывал, как браузер Firefox отличился с полем выбора цвета. У всех остальных — Microsoft Edge, Google Chrome, Яндекс.Браузер и Opera — поле выбора цвета **<input type="color">** подчиняется настройкам из таблицы стилей, а у Firefox — нет. На рисунке 8.2.4 видно, что это поле выбивается по высоте из ряда кнопок. И это не единственная «особенность» — у данного браузера отличается также палитра цветов (рис. 8.2.5). Недаром автор предупреждал читателя о необходимости особенно внимательно тестировать проект в Firefox.

22. A Сделать фон текста цветным. Сначала необходимо открыть окно настроек цвета, выбрать интересующий оттенок, выделить фрагмент и нажать данную кнопку.

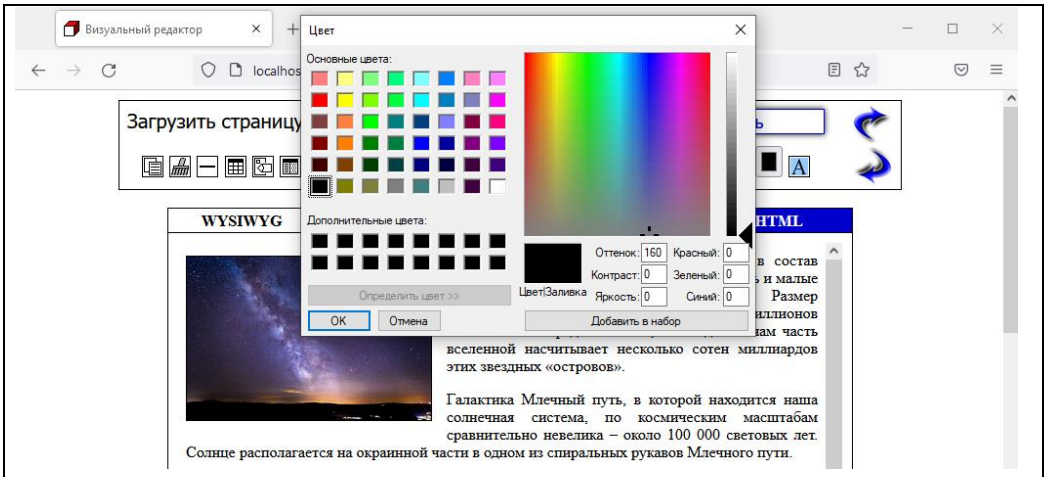

Рис. 8.2.5. Окно выбора цвета у браузера Firefox

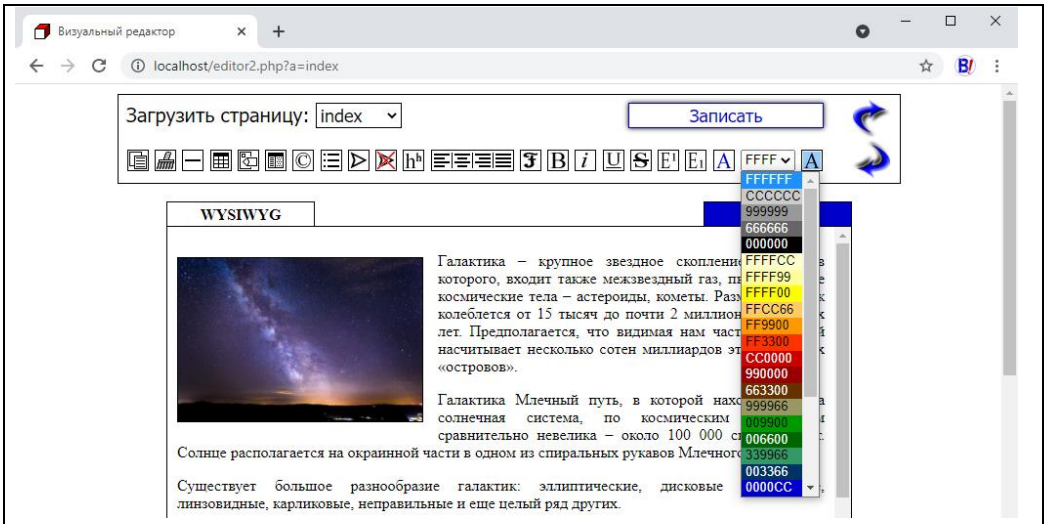

Рис. 8.2.6. Выпадающий список выбора цвета на главной панели второго редактора

У главной панели второго редактора есть единственное отличие: она предоставляет администратору не окно выбора цвета, а выпадающий список, в котором 28 цветов (рис. 8.2.6). Список был составлен автором на произвольной основе. Читатели, повторяя этот редактор, могут изменить, сократить или расширить палитру.

8.3. Кнопки простого редактирования

Под простым редактированием автор подразумевает операцию изменения или копирования текста без предварительных настроек. Кнопок, работающих по данному принципу, 10. На рисунке 8.3.1 они обозначены стрелками. Как видите — разными. Тонкие продолговатые стрелки указывают на кнопки, нажатие которых в редакторах обрабатывается различными функциями. Короткими широкими стрелками обозначены кнопки, которые в редакторах обращаются к одной и той же функции.

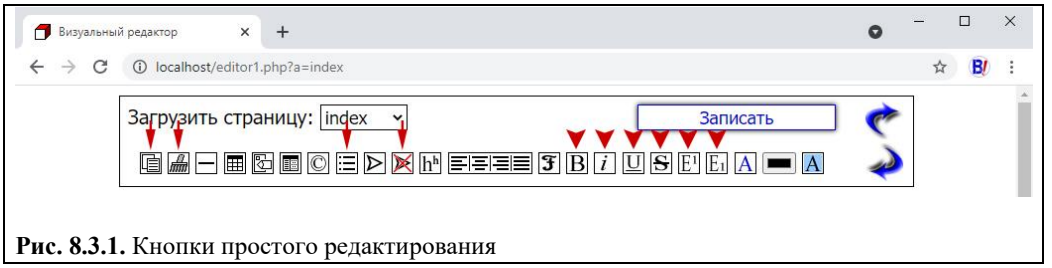

Рис. 8.3.1. Кнопки простого редактирования

Необходимо отметить, что данный принцип распространяется на редакторы № 1–3. В редакторе № 4 все обозначенные стрелками кнопки взаимодействуют с одной и той же функцией.

Перечень кнопок простого редактирования:

–  копирование фрагмента;
–  удаление форматирования;
–  маркированный список;
–  удаление ссылки;
–  выделение жирным;
–  выделение курсивом;
–  выделение подчеркиванием;
–  выделение зачеркиванием;
–  сделать фрагмент текста надстрочным;
–  сделать фрагмент текста подстрочным.

В разметке код этих кнопок выглядит следующим образом:

```
<img src="pict/copy.jpg" id="cop" class="pers"
                        alt="Копировать" title="...">
<img src="pict/format.jpg" id="forma" class="pers"
                    alt="Очистить формат" title="...">
<img src="pict/ul_li.jpg" id="ulli" class="pers"
     alt="Создать маркированный список" title="...">
<img src="pict/del_link.jpg" id="delin" class="pers"
                    alt="Удалить ссылку" title="...">
<img src="pict/bold.jpg" id="bol" class="but"
                           alt="Жирный" title="...">
<img src="pict/ital.jpg" id="ital" class="but"
                           alt="Курсив" title="...">
<img src="pict/under.jpg" id="under" class="but"
                         alt="Подчеркнутый" title="...">
<img src="pict/strike.jpg" id="strik" class="but"
                         alt="Зачеркнутый" title="...">
<img src="pict/sup.jpg" id="sup" class="but"
                        alt="Надстрочный" title="...">
<img src="pict/sub.jpg" id="sub" class="but"
                        alt="Подстрочный" title="...">
```

Несмотря на полное визуальное сходство кнопок во всех редакторах, они могут принадлежать к разным стилевым классам. Сделано так из соображений удобства регистрации обработчиков событий (объяснения этому будут даны позже). В редакторах № 1–3 первые четыре кнопки относятся к классу **pers**, остальные — к классу **but**. В последнем редакторе все кнопки простого редактирования имеют атрибут **class="but"**.

Кнопки открытия вкладок с настройками создаются так же, как и перечисленные выше. Единственное отличие — они принадлежат к стилевому классу **swit**.

Кнопки простого редактирования, как ясно из их названия, работают максимально просто: выделяете необходимую часть текста, нажимаете кнопку — и фрагмент копируется, избавляется от форматирования, меняет

начертание и т. д, в зависимости от функции, которая «приписана» к нажатой кнопке. Если вы нажмете кнопку, не сделав выделение, откроется диалоговое окно с предупреждением «Вы не выделили текст!» (рис. 8.3.2).

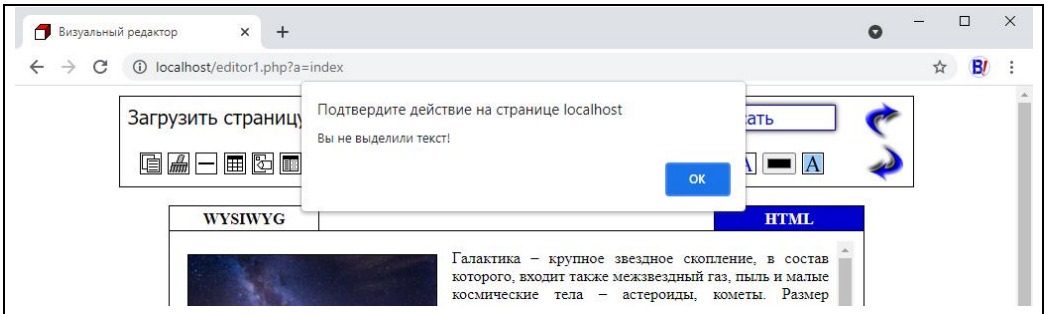

Рис. 8.3.2. Предупреждение о том, что текст не выделен

8.4. Блок настроек линий

Начиная с этого раздела, мы станем разбирать так называемые кнопки сложного редактирования. Этот термин подразумевает, что перед созданием какого-либо элемента вам необходимо предварительно настроить его внешний вид. Для этого предназначены специальные вкладки, которые появляются на экране при нажатии соответствующей кнопки.

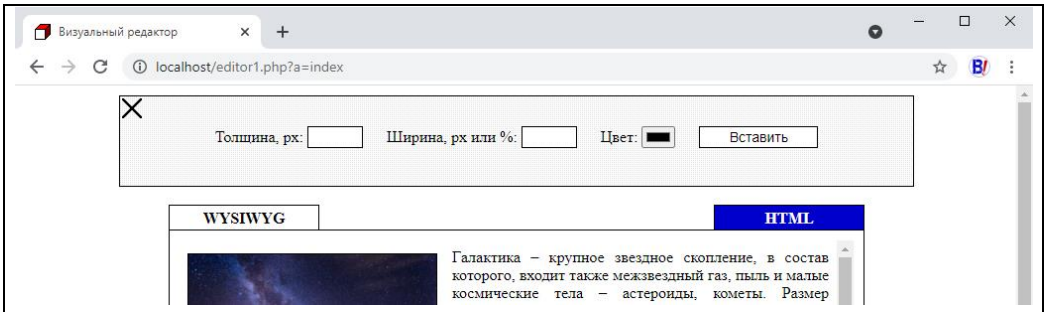

Рис. 8.4.1. Окно настроек линий в редакторах № 1, 3 и 4

Первой на очереди кнопка вставки линии. Вкладка с элементами настройки линий в редакторах № 1, 3 и 4 показана на рисунке 8.4.1. Вы можете выбрать толщину, ширину и цвет. При введении толщины необходимо указать числовое значение и размерность — пиксели, записанные латинскими буквами в сокращенном виде (**px**). Например, так: **3px**. Ширина может быть назначена в пикселях или процентах. Соответственно, в этом поле необходимо указать значения так: **200px** или **50%.**

Обратите внимание! Для всех остальных вкладок действует аналогичный принцип: вводя числовые параметры, необходимо

указывать не только число, но также единицы измерения, указанные в тексте перед соответствующим полем (без пробелов после числа)!

Завершив выбор настроек, вставьте курсор в то место окна редактирования, куда вы хотите поместить линию, и нажмите кнопку «Вставить». Вкладка скроется, а линия появится в окне редактирования и одновременно в области основного содержания страницы, загруженной во фрейм. Если вы забыли поместить курсор в окно редактирования, то линия окажется в его самой верхней части. Кстати, это правило соблюдается и при добавлении всех остальных самостоятельных элементов: таблицы, рисунка, фрейма и символа. Распространяется оно на редакторы № 1–3. Запомните это. В четвертом редакторе при отсутствии курсора в окне редактирования элемент просто не будет добавлен.

Можно вставить линию, не вводя никаких значений. В этом случае она будет черной, толщиной **1** пиксель и шириной **100%**.

Ниже приведен код, создающий данную вкладку на странице редакторов № 1, 3 и 4:

```
<div id="hr_d" class="cust">
<img src="img/close.png" id="hr_i" class="cldiv"
                          alt="Закрыть" title="...">

   <div class="deploy">
   Толщина, px: <input id="hr1" class="inptx">
   Ширина, px или %: <input id="hr2" class="inptx">
   Цвет: <input type="color" id="hr3c"
                        class="colored" title="...">
   <input type="button" id="buthr" class="butin"
                       value="Вставить" title="...">
   </div>

</div>
```

Как уже было сказано в начале этой главы, во втором редакторе перечень стилей ограничен предустановленным набором. Вы не можете указать произвольную толщину, ширину или цвет линии. Вариант только один: найти подходящее значение из того набора, что предоставляют выпадающие списки (рис. 8.4.2). Этот принцип соблюдается для всех вкладок со всеми настройками второго редактора. Недаром он получил название «с фиксированными стилями». Исключение составляют настройки отдельных параметров, которые вводятся вручную:

– количество строк и столбцов в таблицах;

– адреса внешних изображений, фреймов и ссылок;

– подписи к рисункам.

Думаю, ситуация понятна и нет необходимости каждый раз напоминать о данной особенности второго редактора при рассмотрении других вкладок.

Обращаю также ваше внимание, что вместо окна выбора цвета во втором редакторе — выпадающий список, аналогичный тому, что вы видели у данного редактора на главной панели.

Отступление от темы
По умолчанию любой браузер создает у линии тень. Поэтому, даже если вы не ввели никаких настроек, каждый редактор добавляет в стиль линии свойство ***border*** *со значением* ***0*** *(чтобы убрать эту тень).*

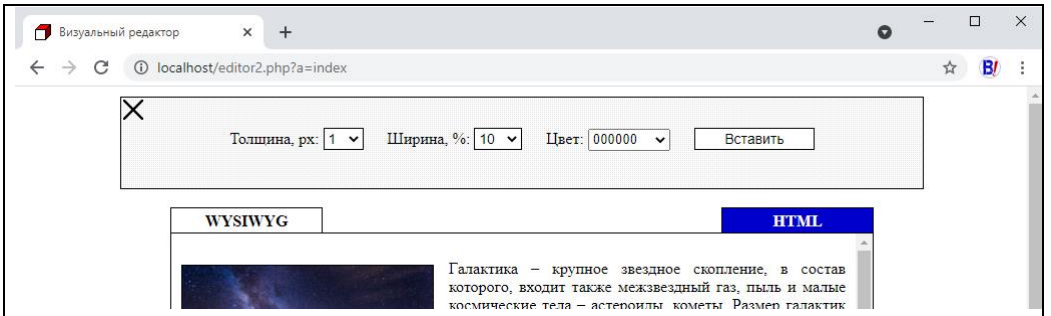

Рис. 8.4.2. Окно настроек линий второго редактора

Код, создающий вкладку настройки линий на странице второго редактора:

```
<div id="hr_d" class="cust">
<img src="img/close.png" id="hr_i" class="cldiv"
                         alt="Закрыть" title="...">

   <div class="deploy">
   Толщина, px: <select id="hr1" class="sele">
   <option selected value="he1">1</option>
   <option value="he2">2</option>
   ...
   <option value="he10">10</option>
   </select>

   Ширина, px или %: <select id="hr2" class="sele">
   <option selected value="perwi10">10</option>
   <option value="perwi20">20</option>
   ...
   <option value="perwi100">100</option>
   </select>

   Цвет: <select id="hr3c" class="colored">
   <option selected class="ba000000 coFFFFFF"
                   value="ba000000">000000</option>
   <option class="baFFFFFF"
                   value="baFFFFFF">FFFFFF</option>
   ...
   <option class="ba660066 coFFFFFF"
                   value="ba660066">660066</option>
   </select>

   <input type="button" id="buthr" class="butin"
                      value="Вставить" title="...">
   </div>

</div>
```

8.5. Блок настроек таблиц

Вкладка с элементами настройки таблиц в редакторах № 1, 3 и 4 показана на рисунке 8.5.1. Вы можете выбрать:

– ширину таблицы;

– количество строк;

– количество столбцов;

– внутренний зазор (то есть расстояние от границ ячейки до ее содержимого);

– внешний зазор (то есть расстояние от границ таблицы до соседних элементов);

– шрифт, его размер и цвет;

– толщину рамки и ее цвет;

– цвет фона таблицы;

– положение в тексте или среди других элементов.

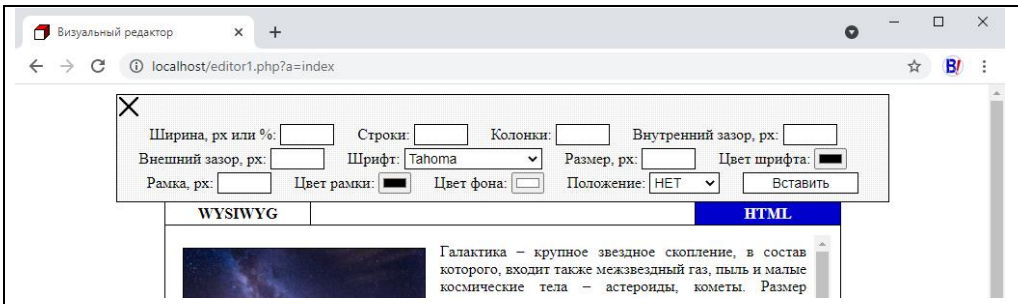

Рис. 8.5.1. Окно настроек таблиц в редакторах № 1, 3 и 4

Аналогичная вкладка редактора с фиксированными стилями имеет два поля для введения произвольных значений: «Строки» и «Колонки». Остальные настройки выбираются из значений выпадающих списков.

Во всех редакторах программа не даст создать таблицу, если вы не укажете количество строк или столбцов. Об этом появится предупреждение, которое вы можете видеть на рисунке 8.5.2.

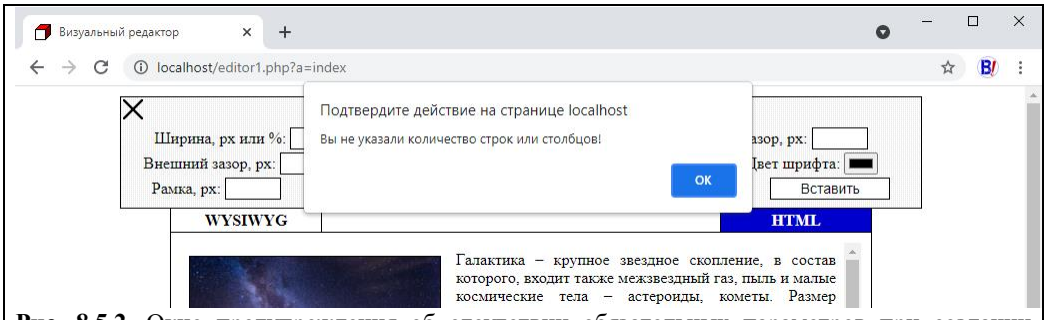

Рис. 8.5.2. Окно предупреждения об отсутствии обязательных параметров при создании таблицы

Если вы укажете в настройках только количество строк и столбцов, то будет создана таблица с рамкой черного цвета толщиной 1 пиксель и шириной, которая зависит от содержимого ячеек.

Фрагмент разметки страницы редакторов № 1, 3 и 4, создающий данную вкладку, приведен ниже:

```
<div id="tabl_d" class="cust bighei">
   <img src="img/close.png" id="tabl_i"
             class="cldiv" alt="Закрыть" title="...">

   <div class="deploy">
   Ширина, px или %: <input id="tab1" class="inptx">
   Строки: <input id="tab2" class="inptx">
   Колонки: <input id="tab3" class="inptx">
   Внутренний зазор, px: <input id="tab4"
                                       class="inptx">

   <br>

   Внешний зазор, px: <input id="tab5" class="inptx">
   Шрифт: <select id="tab6" class="sele">
   <option selected value="Tahoma">Tahoma</option>
   <option value="Arial">Arial</option>
   ...
   <option value="'Times New Roman'">
                               Times New Roman</option>
   </select>
   Размер, px: <input id="tab7" class="inptx">
   Цвет шрифта: <input type="color" id="tab8c"
                         class="colored" title="...">

   <br>

   Рамка, px: <input id="tab9" class="inptx">
   Цвет рамки: <input type="color" id="tab10c"
                          class="colored" title="...">
   Цвет фона: <input type="color" id="tab11c"
         class="colored" value="#FFFFFF" title="...">
   Положение: <select id="tab12" class="sele">
   <option selected value="no">НЕТ</option>
   <option value="left">Слева</option>
   <option value="right">Справа</option>
   </select>
   <input type="button" id="butab" class="butin"
                        value="Вставить" title="...">
   </div>

</div>
```

У редактора № 2 все устроено аналогично, только текстовые поля и окно выбора цвета заменены выпадающими списками.

8.6. Блок настроек рисунков

Вкладка выбора и настроек изображений показана на рисунке 8.6.1. В данном окне вы можете:

– указать адрес картинки с внешнего сайта;

– указать имя фото из папки **cosmos** нашего проекта;

– ввести подпись, которая будет записана в атрибуты **alt** и **title** рисунка;

– позиционировать снимок относительно других элементов или текста;

– настроить внешний зазор (т. е. расстояние от внешнего контура изображения до соседних элементов);

– указать ширину картинки;

– выбрать размер и цвет рамки.

Скрипт добавления рисунка требует двух обязательных параметров:

– адреса внешнего фото или имени внутреннего (одно из двух);

– подписи к рисунку (обязательное условие Консорциума Всемирной Паутины — атрибут **alt** с текстом должен присутствовать в теге **img**).

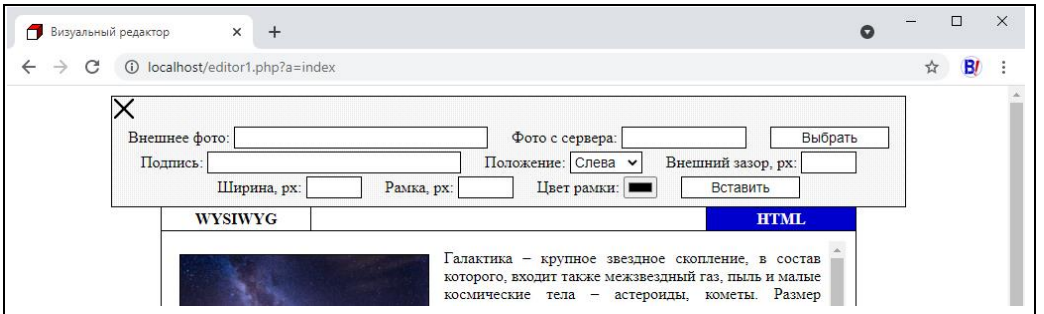

Рис. 8.6.1. Окно выбора и настройки изображения

Если вы оставите остальные поля незаполненными, то будет вставлена картинка шириной **250px** с рамкой черного цвета толщиной **1px**.

Код вкладки выбора изображения:

```
<div id="img_d" class="cust bighei">
<img src="img/close.png" id="img_i" class="cldiv"
                            alt="Закрыть" title="...">

   <div class="deploy">
   Внешнее фото: <input id="ima1" class="inpbig">
   Фото с сервера: <input id="ima2" class="inpmid">
   <input type="button" id="list" class="swit butin"
                         value="Выбрать" title="...">

   <br>

   Подпись: <input id="ima3" class="inpbig">
   Положение: <select id="ima4" class="sele">
   <option selected value="no">НЕТ</option>
   <option value="left">Слева</option>
   <option value="right">Справа</option>
   </select>
   Внешний зазор, px: <input id="ima5" class="inptx">
   <br>
```

```
    Ширина, px: <input id="ima6" class="inptx">
    Рамка, px: <input id="ima7" class="inptx">
    Цвет рамки: <input type="color" id="ima8c"
                          class="colored" title="...">
    <input type="button" id="butima" class="butin"
                         value="Вставить" title="...">
    </div>

</div>
```

Выбор загружаемой в область редактирования картинки происходит по следующей схеме:

– если введен адрес внешнего рисунка, то добавляется внешний рисунок;

– если введено имя внутреннего снимка, то добавляется внутренний снимок;

– если заполнены оба поля, то преимущество имеют данные внешнего фото — вставляется оно.

Чтобы разместить в основном содержании картинку из папки **cosmos**, необходимо нажать кнопку «Выбрать». Откроется вкладка со всеми изображениями на сервере (рис. 8.6.2). Щелкните на необходимом фото. Вкладка закроется, а имя картинки (с расширением) появится в поле «Фото с сервера». Чтобы просто закрыть вкладку без выбора рисунка, нажмите пиктограмму «крестик» в левом верхнем углу.

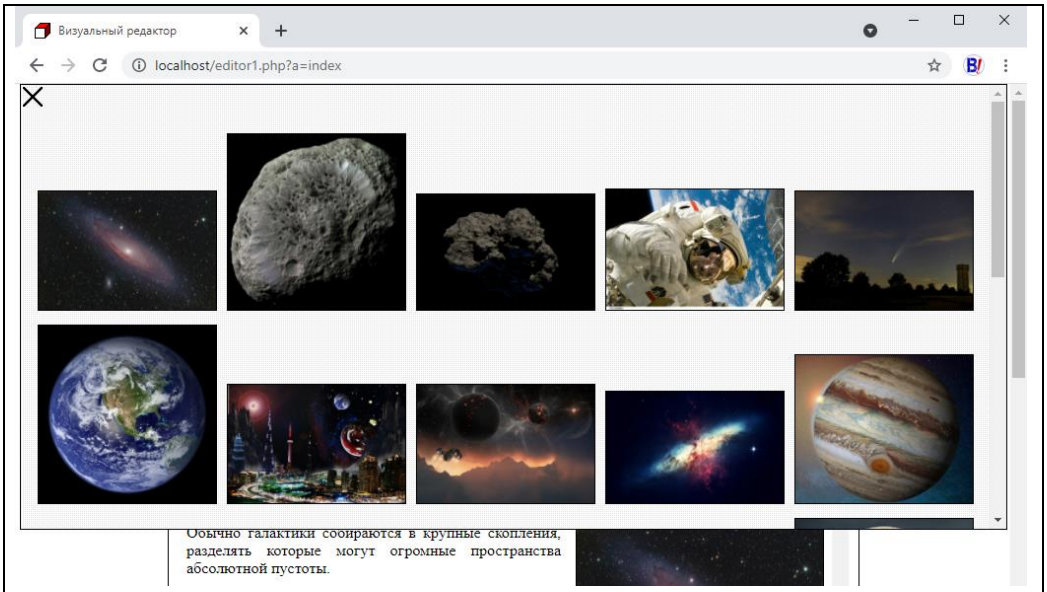

Рис. 8.6.2. Панель выбора рисунков из папки **cosmos** нашего проекта

HTML-код окна выбора внутреннего снимка:

```
<div id="list_d">
```

```
<img src="img/close.png" id="list_i" class="cldiv"
                          alt="Закрыть" title="...">
<p id="p_list">
<?php
$opdir=opendir("cosmos");
while($redir=readdir($opdir))
   {
   if(strpos($redir, "."))
      echo '<img class="roster"
                src="cosmos/'.$redir.'" alt="Фото">';
   }
closedir($opdir);
?>
</p>
</div>
```

Как видите, здесь вновь задействован PHP. Сначала мы открываем папку **cosmos**

```
$opdir=opendir("cosmos");
```

и считываем из нее все содержимое:

```
while($redir=readdir($opdir))
   {
   ...
   }
closedir($opdir);
```

Напомню, что в список попадут такие элементы, как точка и две точки (**.** и **..**), которые символизируют текущий и вышестоящий каталоги. Чтобы удалить их, выполняем следующую проверку:

```
if(strpos($redir, "."))
```

Точка и две точки возвратят индекс первого вхождения элемента — ноль. Условие **if** воспримет данный результат как **false** и не включит эти элементы в список изображений. Зато у остальных картинок, у которых точка стоит перед расширением после нескольких символов имени файла, индекс вхождения окажется больше нуля, и условие примет значение **true**. То есть реальные фотографии окажутся на вкладке:

```
echo '<img class="roster" src="cosmos/'.$redir.'"
                                        alt="Фото">';
```

8.7. Блок настроек фреймов

Окно с элементами настройки фреймов показано на рисунке 8.7.1. Обязательным параметром является адрес страницы. При его отсутствии появится предупреждение: «Вы не указали адрес загружаемой страницы!».

Кроме того, вы можете:

– настроить размеры фрейма;

– выбрать его позиционирование (графа «Положение»);

– установить толщину рамки и ее цвет;

– задать внешний зазор от окружающих элементов или текста.

При отсутствии необходимых параметров вставляется фрейм с черной рамкой толщиной **1px** и размерами в соответствии с внутренними установками браузера (обычно ширина такого фрейма примерно 300 пикселей).

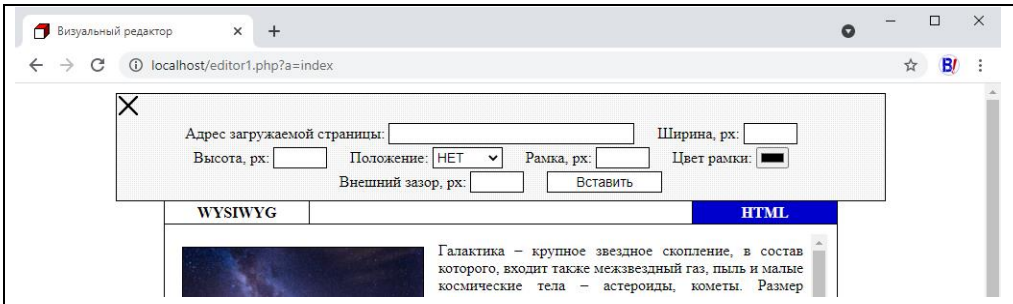

Рис. 8.7.1. Окно настройки фреймов

Обратите внимание: в окне фрейма будет показана страница, загруженная только по протоколу https. Так происходит во всех браузерах — из соображений безопасности.

Код вкладки с настройками фреймов:

```
<div id="ifr_d" class="cust bighei">
<img src="img/close.png" id="ifr_i" class="cldiv"
                            alt="Закрыть" title="...">

   <div class="deploy">
   Адрес загружаемой страницы:
                       <input id="ifr1" class="inpbig">
   Ширина, px: <input id="ifr2" class="inptx">

   <br>

   Высота, px: <input id="ifr3" class="inptx">
   Положение: <select id="ifr4" class="sele">
   <option selected value="no">НЕТ</option>
   <option value="left">Слева</option>
   <option value="right">Справа</option>
   </select>
   Рамка, px: <input id="ifr5" class="inptx">
   Цвет рамки: <input type="color" id="ifr6c"
                         class="colored" title="...">

   <br>

   Внешний зазор, px: <input id="ifr7" class="inptx">
   <input type="button" id="butifr" class="butin"
                        value="Вставить" title="...">
```

```
   </div>
</div>
```

Отступление от темы
Среди некоторых программистов бытует мнение, что фреймы устарели. Это не соответствует действительности. Даже такие передовые лидеры web-индустрии, как Google или YouTube, используют фреймы. Например, первый — для встраивания пользователем карт местности, а второй — в функции «Поделиться видео».

8.8. Блок выбора символов

Панель с различными математическими и не только символами показана на рисунке 8.8.1. Работать с ней очень просто: вставляете курсор в необходимое место окна редактирования и нажимаете кнопку требуемого символа, после чего панель закрывается, а символ появляется в тексте.

Вообще, полезных символов очень много — но автор выбрал лишь 50 наиболее часто применяемых (на его взгляд).

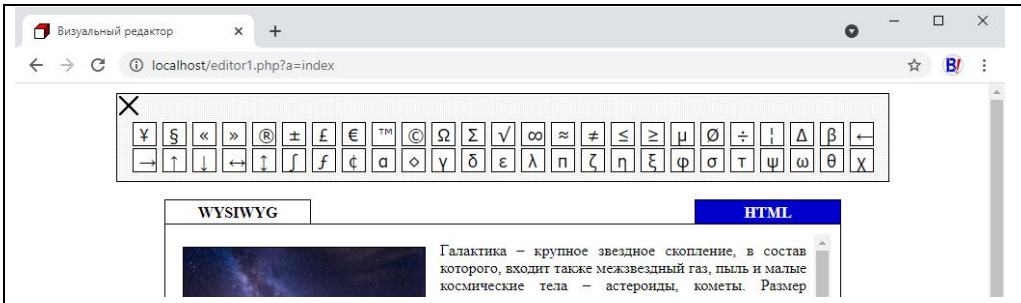

Рис. 8.8.1. Окно выбора символов

Код вкладки занимает в документе довольно большой объем, поэтому приведем его в сокращенном виде:

```
<div id="copyr_d" class="cust">
<img src="img/close.png" id="copyr_i" class="cldiv"
                          alt="Закрыть" title="...">

   <div id="copyr_sym">
   <input type="button" class="sym" id="s1"
                                     value="&#165;">
   ...
   <input type="button" class="sym" id="s25"
                                     value="&#8592;">

   <br>

   <input type="button" class="sym" id="s26"
                                     value="&#8594;">
   ...
```

```
   <input type="button" class="sym" id="s50"
                                        value="&#967;">
   </div>

</div>
```

В коде страницы почти все символы записаны в виде HTML-сущностей: **¥**, **§** и так далее. В окне редактирования они преобразуются в реальные символы.

8.9. Блок настроек ссылок

Блок настроек ссылок показан на рисунке 8.9.1.

Окно редактирования позволяет создать ссылку на внешний ресурс, на внутреннюю страницу сайта или на электронную почту.

Помимо адреса вы можете указать:

– для внешней ссылки — открывать ли ее в новом окне или нет (не действует для внутренней ссылки);

– тип протокола соединения — **https**, **http** или **mailto**;

– нужно ли подчеркивать ссылку или нет;

– цвет текста ссылки.

Выбор между внешней и внутренней ссылкой происходит по следующей схеме:

– если введен адрес стороннего ресурса, то добавляется именно он;

– в остальных случаях добавляется ссылка на внутреннюю страницу.

Начальные установки вкладки после ее открытия: выбрана внутренняя страница, цвет ссылки — черный, с подчеркиванием.

Чтобы добавить ссылку, откройте вкладку, выполните требуемые настройки, выделите текст и нажмите кнопку «Вставить». Забыли сделать выделение? Появится окно с предупреждением.

Код вкладки:

```
<div id="link_d" class="cust bighei">
<img src="img/close.png" id="link_i" class="cldiv"
                            alt="Закрыть" title="...">

   <div class="deploy">
   Добавить ссылку на внешний ресурс:
                     <select id="link1" class="selena">
   <option selected value="https://">https://</option>
   <option value="http://">http://</option>
   <option value="mailto:">mailto:</option>
   </select>
   <input id="link2" class="inpbig">

   <br>

   Открывать ссылку в новом окне:
                       <select id="link3" class="sele">
   <option selected value="yes">ДА</option>
   <option value="no">НЕТ</option>
```

```
    </select>
    Добавить ссылку на страницу:
                        <select id="link4" class="sele">
    ...
    </select>

    <br>

    Оформление: <select id="link5" class="sele">
    <option selected value="yes">
                                  С подчеркиванием</option>
    <option value="no">Без подчеркивания</option>
    </select>
    Цвет: <input type="color" id="link6c"
                             class="colored" title="...">
    <input type="button" id="butlink" class="butin"
                            value="Вставить" title="...">
    </div>

</div>
```

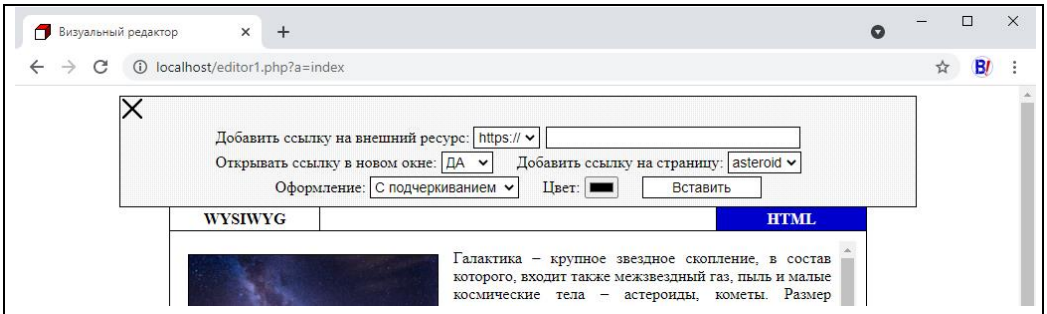

Рис. 8.9.1. Окно настроек ссылок

Отдельно разберем, как создается выпадающий список «Добавить ссылку на страницу»:

```
<select id="link4" class="sele">
...
</select>
```

Здесь мы в который раз прибегаем к помощи PHP:

```
<?php
$ident=0;
$opdir=opendir("content");
while($redir=readdir($opdir))
{
if(strpos($redir, "."))
   {
   $rep=str_replace(".txt", "", $redir);
   if($ident==0)
      {
      echo '<option selected value="'.$rep.'">'
                                    .$rep.'</option>';
```

```
      $ident=1;
      }
   else
      echo '<option value="'.$rep.'">'.$rep.
                                          '</option>';
   }
}
closedir($opdir);
?>
```

Очень похоже на то, что мы делали, когда формировали список файлов для главной панели. Здесь практически все то же самое, но за тремя исключениями.

1. У панели на вершине списка было имя загруженной страницы. Теперь нам не требуется создавать преимущество какому-либо файлу. На вершине оказывается имя первого обнаруженного в цикле просмотра папки **content**.

2. Из первого пункта автоматически вытекает второй: сейчас проверка на имя файла не нужна. Нам достаточно убедиться, что очередной элемент из цикла не является символом текущего или вышестоящего каталога:

```
if(strpos($redir, "."))
```

3. Введена дополнительная переменная **$ident**, которая служит идентификатором заполнения первой строки перечня. Соответственно, в цикле делается проверка:

```
if($ident==0)
   {
   ...
   }
else
```

Первая строка формируется следующим образом:

```
if($ident==0)
   {
   echo '<option selected value="'.$rep.'">'
                                    .$rep.'</option>';
   ...
   }
```

После этого идентификатору присваивается новое значение:

```
$ident=1;
```

Теперь выражение

```
if($ident==0)
```

дает результат **false** и выполняется второй блок условия, в котором записана инструкция печати остальных имен файлов:

```
echo '<option value="'.$rep.'">'.$rep.'</option>';
```

8.10. Блок настроек заголовков

Вкладка настроек заголовков изображена на рисунке 8.10.1. В данном окне вы можете выбрать:

– уровень заголовка от h1 до h6;

– шрифт, размер шрифта и его цвет.

Как видно из рисунка, по умолчанию для заголовка установлен первый уровень и шрифт «Tahoma» черного цвета. Пустует только поле для ввода кегля (размера). Если все так и оставить, размер для h1 будет примерно 32px.

Откройте вкладку, выполните требуемые настройки, выделите текст, который станет заголовком, и нажмите кнопку «Вставить».

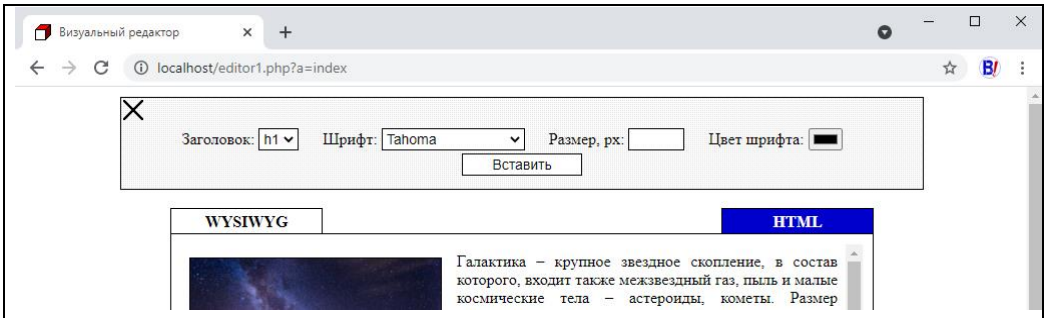

Рис. 8.10.1. Вкладка настроек заголовков

HTML-код блока настроек заголовков:

```
<div id="h16_d" class="cust">
<img src="img/close.png" id="h16_i" class="cldiv"
                          alt="Закрыть" title="...">

   <div class="deploy">
   Заголовок: <select id="h161" class="sele">
   <option selected value="h1">h1</option>
   <option value="h2">h2</option>
   ...
   <option value="h6">h6</option>
   </select>
   Шрифт: <select id="h162" class="sele">
   <option selected value="Tahoma">Tahoma</option>
   <option value="Arial">Arial</option>
   ...
   <option value="'Times New Roman'">
                              Times New Roman</option>
   </select>
   Размер, px: <input id="h163" class="inptx">
```

```
    Цвет шрифта: <input type="color" id="h164c"
                        class="colored" title="...">

    <br>

    <input type="button" id="buth16" class="butin"
                        value="Вставить" title="...">
    </div>
</div>
```

8.11. Блок настроек выравнивания текста в абзацах

Панель настроек выравнивания текста в абзацах изображена на рисунке 8.11.1. Как видите, здесь 5 полей. Создавая такую вкладку, автор руководствовался следующим соображением: выделяя часть текста в отдельный абзац, администратор, наверное, предполагает, что текст будет иметь какое-то оформление. Так почему этот процесс надо разбивать на две или три операции, когда все можно совместить в одной? Делать все настройки за один присест, на взгляд автора, рациональнее, быстрее и проще.

Поэтому в данном окне вы можете выбрать:

– выравнивание по левому или правому краю, по центру, по ширине страницы;

– шрифт, его размер и цвет;

– отступ первой строки абзаца от левого края.

На третий пункт обратите особое внимание. В этом параметре можно указывать не только положительные числа, но также и отрицательные, например так: **–50px**. В последнем случае образуется не отступ, а выступ начала первой строки левее левого края остального текста.

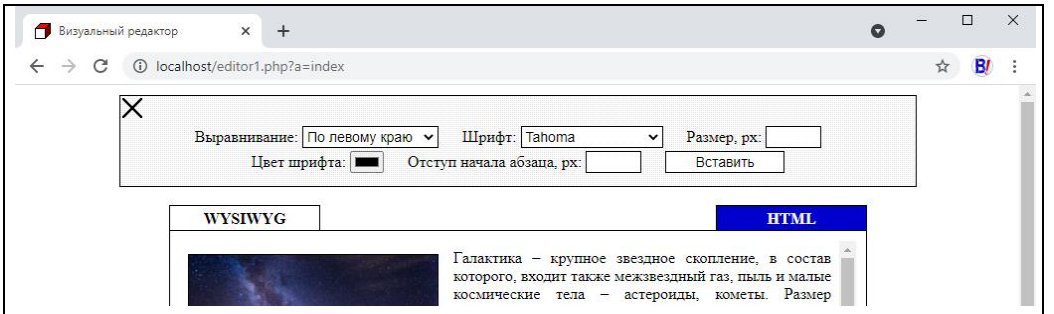

Рис. 8.11.1. Панель настроек выравнивания текста в абзацах

Код блока настроек выравнивания текста:

```
<div id="equa_d" class="cust">
<img src="img/close.png" id="equa_i" class="cldiv"
                          alt="Закрыть" title="...">

   <div class="deploy">
```

```
    Выравнивание: <select id="equa1" class="sele">
    <option selected value="left">По левому краю
                                                </option>
    <option value="center">По центру</option>
    <option value="right">По правому краю</option>
    <option value="justify">По ширине</option>
    </select>
    Шрифт: <select id="equa2" class="sele">
    <option selected value="Tahoma">Tahoma</option>
    <option value="Arial">Arial</option>
    ...
    <option value="'Times New Roman'">
                               Times New Roman</option>
    </select>
    Размер, px: <input id="equa3" class="inptx">

    <br>

    Цвет шрифта:
   <input type="color" id="equa4c"
                            class="colored" title="...">
    Отступ начала абзаца, px:
                         <input id="equa5" class="inptx">
    <input type="button" id="butequa" class="butin"
                           value="Вставить" title="...">
    </div>

</div>
```

8.12. Блок настроек шрифтов

Окно настроек шрифтов показано на рисунке 8.12.1. У читателя может появиться вопрос: зачем такое окно, если все предварительные установки можно выполнить, создавая абзац? Ответ очень простой: отдельная настройка необходима на случай, если надо выделить одно или несколько слов шрифтом, отличающимся от остального текста.

Панель имеет уже хорошо знакомые вам три параметра: шрифт, цвет и размер в пикселях.

Так как в этом окне нет каких-либо особенностей, на которые необходимо обратить внимание, просто посмотрим его HTML-код. Он максимально прост:

```
<div id="font_d" class="cust">
<img src="img/close.png" id="font_i" class="cldiv"
                              alt="Закрыть" title="...">

    <div class="deploy">
    Шрифт: <select id="font1" class="sele">
    <option selected value="Tahoma">Tahoma</option>
    <option value="Arial">Arial</option>
    ...
    <option value="'Times New Roman'">
                               Times New Roman</option>
    </select>
    Размер, px: <input id="font2" class="inptx">
    Цвет шрифта: <input type="color" id="font3c"
                            class="colored" title="...">
```

```
    <input type="button" id="butfont" class="butin"
                         value="Вставить" title="...">
    </div>
</div>
```

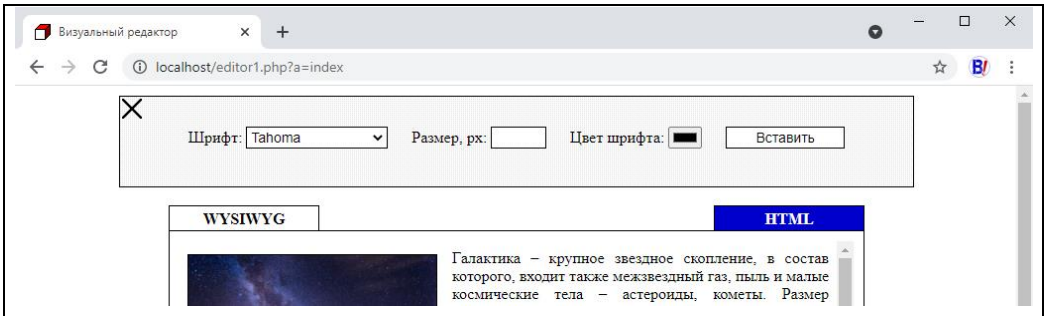

Рис. 8.12.1. Блок настроек шрифтов

8.13. Блок настроек цвета текста

Последняя из рассматриваемых функций — настройка цвета текста — не имеет персонального окна. Все делается непосредственно с главной панели. На рисунке 8.13.1 стрелками показаны кнопки, выполняющие заявленные операции. Их три: изменение цвета текста, выбор цвета, изменение цвета фона под текстом.

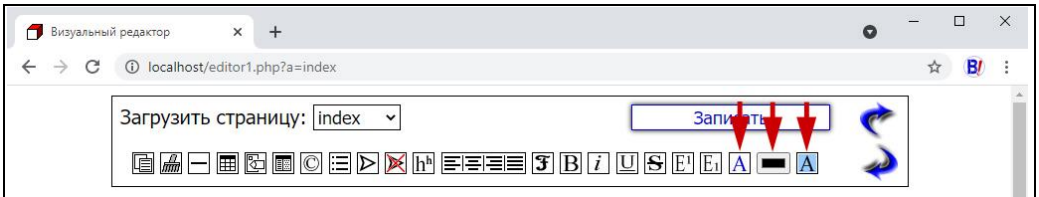

Рис. 8.13.1. Кнопки настроек цвета текста

Работает все очень просто. Нажмите кнопку открытия окна выбора цвета (на рисунке 8.13.1 эта кнопка обозначена средней стрелкой). Выберите требуемый тон, передвигая круглый ползунок по цветовой шкале (обозначен нижней стрелкой на рисунке 8.13.2). Перемещая круглый курсор (обозначен верхней стрелкой на рисунке 8.13.2) по прямоугольнику палитры, выберите необходимый оттенок. Выделите текст. Для изменения цвета букв нажмите кнопку с символом «А» синего цвета (левая на рисунке 8.13.1). Для изменения цвета фона под текстом нажмите кнопку с буквой «А» на синем фоне (правая на рисунке 8.13.1).

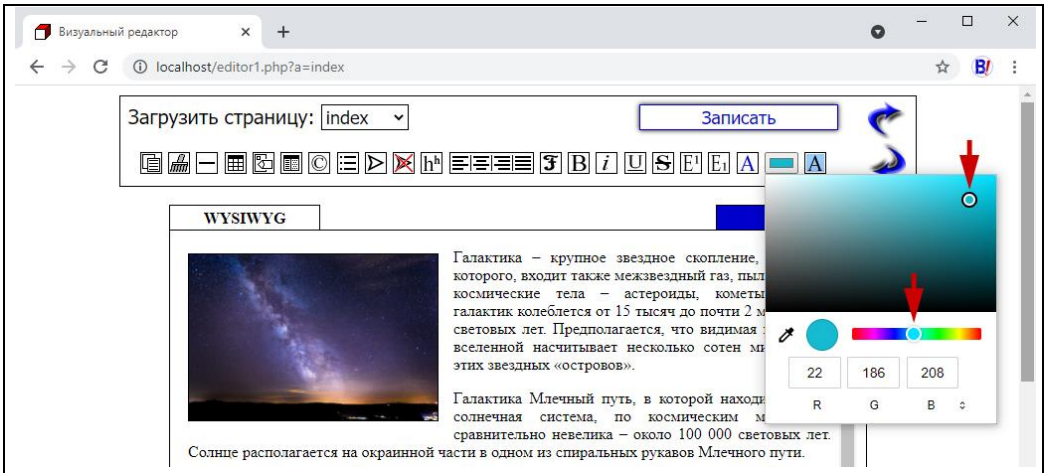

Рис. 8.13.2. Выбор цвета текста или его фона

Код, создающий элементы управления цветом:

```
<img src="pict/color.jpg" id="col" class="colo"
                        alt="Цвет шрифта" title="...">
<input type="color" id="ccc" title="...">
<img src="pict/backgr.jpg" id="backgr" class="colo"
                    alt="Цвет фона шрифта" title="...">
```

8.14. Удаление форматирования и ссылки

В заключение — короткая справка о том, как работают кнопки удаления форматирования и удаления ссылки в разных версиях редакторов.

Во-первых, в редакторах № 1–3 действие этих кнопок абсолютно одинаково — они исключают из текста все HTML-теги, которые там есть, оставляя только сам текст. Таким образом, если в выделение попадет, например, ссылка, она тоже будет удалена вместе с такими тегами форматирования, как **p**, **span**, **b**, **i** и т. д. На резонный вопрос, зачем тогда две разные кнопки — удаления форматирования и удаления ссылки, если они выполняют одну и ту же функцию, есть резонный ответ: в последнем, четвертом редакторе эти кнопки действуют по-разному. Поэтому для соблюдения идентичности главной панели во всех версиях редакторов в первых трех были сохранены обе кнопки.

Во-вторых, чтобы удалить форматирование из фрагмента текста, в редакторах № 1–3 необходимо сделать выделение на один символ больше слева и справа. Как это выглядит на практике? Допустим, у вас есть слово «визу**аль**ный» с буквами **аль** полужирного начертания. Чтобы привести их к нормальному отображению, надо выделить мышью буквы у**аль**н и нажать кнопку удаления форматирования (выделение должно выглядеть так: визуальный). Если у вас есть фрагмент текста «этот **визуальный** редактор», в котором полужирным шрифтом набрано слово «**визуальный**», то для его

приведения к исходному состоянию надо выделить не только само слово, но и пробелы до соседних слов (вот так: этот **визуальный** редактор).

В-третьих, про последний, четвертый, редактор. В нем кнопка удаления форматирования удаляет из текста действительно только теги форматирования и стили из попавших в выделение ссылок, не трогая последние. Кнопка удаления ссылки удаляет только ее, оставляя служащий основой ссылки текст. При этом в обоих случаях дополнительные символы по краям выделять не надо.

На этом разговор о кнопках и вкладках завершен. Теперь перейдем к самому главному — сценариям на JavaScript, которые управляют всеми процессами редактирования и записи данных.

9. Общие фрагменты сценариев

Любой редактор состоит из трех «ингредиентов»: файла страницы, таблицы стилей и файла со сценариями, написанными на JavaScript. По сути, JavaScript — это квинтэссенция, альфа и омега, основа нашего проекта. Именно сценарии меняют или создают контент для сайта. При этом каждый редактор обеспечен своим персональным набором сценариев, объединенных в одну программу.

Уже неоднократно было упомянуто, что редакторы имеют как определенные различия, так и набор одинаковых или почти одинаковых компонентов. Про отличия мы будем говорить, когда станем подробно разбирать каждую версию в отдельности. Сейчас речь пойдет о блоках кода JavaScript, которые есть в любом из редакторов и которые в одних случаях полностью, а в других почти полностью (за некоторыми незначительными различиями) схожи.

Перечислим совпадающие компоненты наших JavaScript-файлов:

– «главный» обработчик, служащий контейнером для всех остальных обработчиков и функций;

– инструкции выбора загружаемой страницы;

– инструкции, обеспечивающие взаимодействие визуального и текстового редакторов;

– всего одна строка кода для фиксации выделенной области в окне визуального редактора;

– функции открытия и закрытия вкладок с настройками;

– сценарий выбора рисунка с сервера (то есть из папки **cosmos**);

– блок навигации по истории внесенных изменений с возможностью зафиксировать начальный, конечный или любой промежуточный результат;

– блок записи контента отредактированной страницы.

Двинемся по порядку.

9.1. Регистрация «главного» обработчика

Один из самых простых фрагментов кода, абсолютно одинаковый для всех редакторов. Как вы помните, файл JavaScript мы подключаем в заголовочной части документа. Следовательно, прежде чем регистрировать обработчики нажатия различных кнопок, необходимо сначала дождаться полной загрузки страницы, чтобы браузер «увидел» эти кнопки. Значит, самым первым, «главным» обработчиком должен быть следующий:

```
addEventListener("load", function()
   {
   ...
   });
```

После загрузки документа в окно браузера можно регистрировать все остальные обработчики:

```
addEventListener("load", function()
   {
   Регистрация обработчика 1
   Регистрация обработчика 2
   ...
   Регистрация обработчика N
   });
```

Обычно сами функции, выполняющие роли обработчиков, располагают вне блока:

```
addEventListener("load", function()
   {
   ...
   });
```

Однако из соображений упрощения кода автор поместил внутрь этого блока не только инструкции по регистрации обработчиков, но и функции, обрабатывающие события. Это несколько сокращает программы и одновременно улучшает их читабельность. Таким образом, у нас получается главная функция, в которую вложены все остальные.

Отступление от темы
Анализатор браузера при загрузке документа считывает его «сверху вниз». Поэтому, если сценарий располагается или загружается в головной части документа, будет ошибкой пытаться сразу (напрямую) зарегистрировать обработчики, например для нажатия кнопок. Ведь эти обработчики web-обозреватель найдет еще до того, как будет построена объектная модель документа. То есть браузер еще не видит кнопок, а мы уже пытаемся привязать к ним событие ***click****. В итоге обработчики не будут зарегистрированы.*
Чтобы избежать таких неприятностей, автор советует поступать следующим образом: регистрировать сначала обработчик уровня окна браузера для события ***load****, а уже внутри него записывать остальные обработчики (можно, в отличие от нашего случая, все функции вынести за пределы обработчика загрузки страницы).*

9.2. Выбор загружаемой страницы

По умолчанию при запуске того или иного редактора в нем оказывается главная страница, рассказывающая о галактиках. Чтобы загрузить другую страницу, необходимо открыть выпадающий список и щелкнуть на имени файла. В этот момент происходит событие выбора — **change**. К нему привязан обработчик, роль которого выполняет анонимная функция:

```
document.getElementById("sel").
                addEventListener("change", function()
   {
```

```
...
});
```

В ней всего два оператора. Первый присваивает переменной **a** текущее значение из выпадающего списка

```
let a=document.getElementById("sel").value;
```

а второй загружает в браузер выбранную страницу, присваивая свойству **location** новый адрес

```
window.location="редакторN.php?a="+a;
```

где:
– **редакторN.php** — имя PHP-файла редактора с номером **N**;
– **a** (следующее после разделителя **?**) — параметр, который содержит адрес страницы.

Можно было обойтись и одной строкой, сразу присваивая свойству **location** адрес файла, а параметру **a** — значение из выпадающего списка **sel**. Но запись в две строки более читабельна.

Вот как выглядит код, например, в первом редакторе:

```
document.getElementById("sel").
                addEventListener("change", function()
   {
   let a=document.getElementById("sel").value;
   window.location="editor1.php?a="+a;
   });
```

Для всех остальных редакторов функция выбора страницы точно такая же, за исключением имени PHP-файла. Имя должно быть **editor2.php** у второго редактора, **editor4.php** — у четвертого. Только в третьем редакторе адрес представляет собой более сложную конструкцию. Но об этом — в главе 12.

9.3. Взаимодействие области контента, WYSIWYG и текстового редакторов

Во всех сценариях различные функции неоднократно обращаются к области контента загруженной во фрейм страницы, а также к окнам визуального и текстового редакторов. Поэтому для сокращения кода мы в самом начале создадим ссылки на эти окна.

Разберемся, как это делается на примере редактора № 1. В нем есть два фрейма. Первый служит окном визуального представления, а во второй загружается текущая страница. Соответственно эти фреймы имеют индексы **0** и **1**. Сначала запишем ссылку на слой **div** с основным содержанием:

```
let co=frames[1].document.getElementById("cont");
```

затем на фрейм визуального редактора

```
let ed=frames[0].document;
```

и последней будет ссылка на текстовое поле:

```
let te=document.getElementById("tex");
```

Теперь передадим код из области контента в область визуального редактирования

```
ed.body.innerHTML=co.innerHTML;
```

а затем из области редактирования в текстовое поле:

```
te.value=ed.body.innerHTML;
```

Редакторы устроены так, что любое действие в окне визуального представления тут же передается во фрейм со страницей, а точнее, в ее слой **div** с основным содержанием. Сделано так, чтобы администратор в любой момент видел, как меняется облик редактируемой страницы. Для этого обрабатываются два события: щелчок мышью в окне визуального представления

```
ed.addEventListener("click", tran);
```

и нажатие любой кнопки на клавиатуре

```
ed.addEventListener("keyup", function(ev)
   {
   tran();
   ...    // Код, который будет пояснен в разделе 9.7
   });
```

Как видите, в обоих случаях запускается функция **tran**. Ее задача — отправить данные в слой с основным содержанием:

```
function tran()
   {
   co.innerHTML=ed.body.innerHTML;
   }
```

Помимо такого взаимодействия, предусмотрено еще одно. Оно происходит во время переключения режима работы с окна визуального представления на текстовое поле и обратно. Это взаимодействие включает несколько этапов, которые объединены в две анонимные функции. Поскольку в обеих встречаются обращения к одним и тем же элементам редактора, для сокращения кода запишем четыре ссылки: на стили кнопки визуального редактора

```
let wys=document.getElementById("wys").style;
```

на стили кнопки HTML-редактора

```
let htm=document.getElementById("htm").style;
```

на стили текстового поля

```
let tex=document.getElementById("tex").style;
```

на кнопку записи

```
let rec=document.getElementById("rec");
```

Первая из функций запускается, когда администратор переключается с визуального на HTML-редактор:

```
document.getElementById("htm").
                addEventListener("click", function()
   {
   ...
   });
```

Сначала происходит изменение цвета фона вкладки с надписью «WYSIWYG»

```
wys.background="#0000CC";
```

и цвета текста:

```
wys.color="#FFFFFF";
```

Затем аналогичные операции выполняются для вкладки с текстом «HTML»:

```
htm.background="#FFFFFF";
htm.color="#000000";
```

Вкладка «WYSIWYG» станет синей с белыми буквами, вкладка «HTML» — белой с черными буквами.

Следом передаем информацию из окна визуального представления в текстовое поле

```
te.value=ed.body.innerHTML;
```

и делаем его видимым:

```
tex.visibility="visible";
```

За счет того, что **z-index** текстового поля выше, чем у окна визуального представления, текстовое поле оказывается наверху, а визуальное окно скрывается.

Мы условились, что в текстовом режиме отправка данных серверной программе невозможна. Поэтому деактивируем кнопку «Записать», установив ей атрибут **disabled**:

```
rec.setAttribute("disabled", true);
```

А чтобы подчеркнуть отключение кнопки, делаем ее полупрозрачной:

```
rec.style.opacity=0.2;
```

При обратном переключении вкладок запускается вторая функция:

```
document.getElementById("wys").
                addEventListener("click", function()
   {
   ...
   });
```

Первым делом вкладкам «WYSIWYG» и «HTML» возвращаются их первоначальные цвета:

```
wys.background="#FFFFFF";
wys.color="#000000";
htm.background="#0000CC";
htm.color="#FFFFFF";
```

Теперь передаем код из текстового поля в окно визуального представления

```
ed.body.innerHTML=te.value;
```

а из него — в область основного контента страницы:

```
co.innerHTML=ed.body.innerHTML;
```

То есть при обратном переключении вкладок результаты действий администратора в текстовом поле тоже становятся видны на странице сайта, загруженной во фрейм.

Осталось скрыть текстовое поле

```
tex.visibility="hidden";
```

вернуть кнопке исходный уровень непрозрачности

```
rec.style.opacity=1;
```

и активировать ее, удалив атрибут **disabled**:

```
rec.removeAttribute("disabled");
```

На этом перечень взаимодействий области контента, визуального и текстового редакторов завершен.

9.4. Фиксация в памяти выделенного фрагмента

Прежде, чем вносить изменения в контент, необходимо выделить какой-либо фрагмент текста или установить курсор в нужную позицию в окне визуального редактирования. После этого в памяти компьютера создается объект **Selection**, который хранит выделенный текст, а также начальную и конечную позиции курсора (если курсор просто вставлен в текст, то начальные и конечные точки совпадают).

В редакторах № 2–4 этот процесс описывается так:

```
let sel=document.getSelection();
```

В редакторе № 1 чуть иначе, поскольку фрейм представляет собой иной документ, нежели страница редактора:

```
let sel=ed.getSelection();
```

Когда выделенная область создана, ее можно обрамлять различными тегами, а также вставлять любые элементы, предусмотренные на главной панели редактора.

9.5. Открытие и закрытие вкладок с настройками

Уже упоминалось, что кнопки сложного редактирования относятся к стилевому классу **swit** (к этому же классу относится и кнопка «Выбрать», открывающая вкладку с перечнем изображений в папке **cosmos)**. Данный фактор используется в универсальной функции открытия вкладок с настройками элементов.

На первом этапе мы получаем массив всех кнопок, относящихся к упомянутому классу:

```
let s=document.querySelectorAll(".swit");
```

А затем в цикле регистрируем список обработчиков нажатия этих кнопок:

```
for(let i=0; i<s.length; i++)
   {
   let idc=s[i].id;
   let nid=idc+"_d";
   document.getElementById(idc).
      addEventListener("click", ins.bind(null, nid));
   }
```

Сначала получаем **id** очередной кнопки

```
let idc=s[i].id;
```

после чего выясняем имя вкладки, которая должна быть открыта:

```
let nid=idc+"_d";
```

Каждый редактор написан так, что определенной кнопке с некоторым **id** соответствует вкладка, идентификатор которой отличается добавлением нижнего подчеркивания и символа **d** к **id** кнопки.

Третий шаг — регистрация обработчика нажатия очередной кнопки:

```
document.getElementById(idc).
      addEventListener("click", ins.bind(null, nid));
```

Обращаю внимание читателя, что метод **bind** создаёт новую функцию, которая при вызове устанавливает в качестве контекста выполнения некоторое значение, а также принимает набор аргументов. В нашем случае аргумент один — **id** вкладки с настройками**.** К **bind** мы прибегли, так как иначе не смогли бы передать в функцию-обработчик необходимый параметр. Поскольку мы не привязываем к функции контекст выполнения, то первым пунктом указываем **null**.

Функция **ins**, которая записана в качестве обработчика для нажатия любой из кнопок сложного редактирования, содержит всего одну инструкцию — открытия вкладки с **id**, переданным в аргументе:

```
function ins(k)
   {
   document.getElementById(k).style.visibility=
                                          "visible";
   }
```

Для закрытия вкладок служит функция **clo**:

```
function clo(c)
   {
   document.getElementById(c).style.visibility=
                                           "hidden";
   }
```

Она срабатывает после нажатия иконки с крестиком или после нажатия кнопки «Вставить» на вкладке с настройками, а также после нажатия на любом из знаков вкладки специальных символов.

В качестве аргумента функция получает **id** окна, которое необходимо закрыть. Следующий код регистрирует функцию **clo** в качестве обработчика нажатия иконок с крестиком:

```
let t=document.querySelectorAll(".cldiv");
```

```
for(let i=0; i<t.length; i++)
   {
   let idc=t[i].id;
   let nid=idc.replace("_i", "_d");
   document.getElementById(idc).
      addEventListener("click", clo.bind(null, nid));
   }
```

Иконки закрытия окон относятся к стилевому классу **cldiv**. Получаем массив иконок, в цикле выясняем их **id** и заменяем в идентификаторах букву **d** на **i** (**id** любой иконки закрытия отличается от **id** вкладки символом **i** вместо **d** после нижнего подчеркивания). Последний этап — регистрация функции **clo** в качестве обработчика щелчка на каждой из иконок.

Как формируется обращение к функции **clo** после выбора настроек на вкладках, мы рассмотрим позже, когда перейдем к подробному описанию редакторов.

9.6. Выбор рисунка

В окне настроек изображений есть кнопка «Выбрать». Если ее нажать, то откроется вкладка с галереей фото из папки **cosmos** (рис. 9.6.1). Допустим, вы изучили все варианты и остановились на каком-то рисунке. Нажмите на него. Вкладка закроется, а имя файла выбранной картинки появится в текстовом поле рядом с надписью «Фото с сервера».

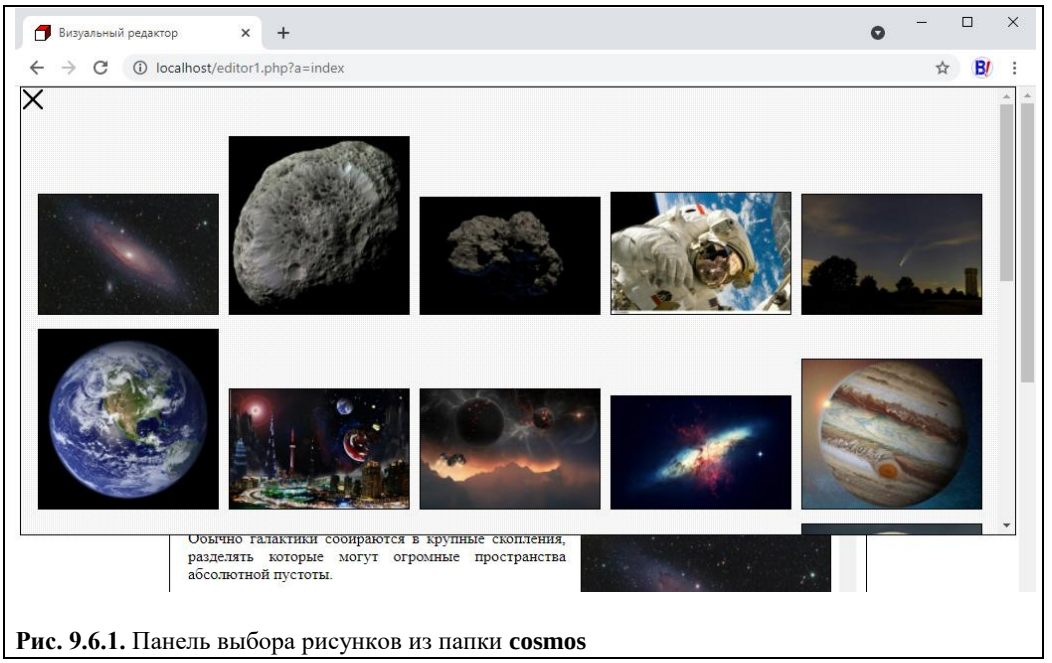

Рис. 9.6.1. Панель выбора рисунков из папки **cosmos**

Управляет процессом выбора одна довольно простая анонимная функция, которая в качестве аргумента получает объект события и запускается при нажатии на любое фото из представленной галереи:

```
document.getElementById("p_list").
                addEventListener("click", function(ev)
   {
   ...
   });
```

Как видите, мы регистрируем не все случаи нажатия на каждый рисунок, а только одно событие — клик в области

```
<p id="p_list">
...
</p>
```

окна выбора.

Дальше узнаем, пришелся ли щелчок на пустое пространство или на изображение:

```
if(ev.target.tagName=="IMG")
   {
   ...
   }
```

Если условие верно, выясняем имя файла выбранной картинки. Для этого берем у объекта события атрибут **src** и разбиваем полученное из него значение на две части. В качестве разделителя служит имя папки с фотографиями **cosmos/**:

```
let pho=ev.target.src.split('cosmos/');
```

Второй элемент **pho[1]** из образовавшегося массива и будет содержать имя файла. Передаем это имя в текстовое поле

```
document.getElementById("ima2").value=pho[1];
```

и закрываем вкладку:

```
document.getElementById("list_d").style.visibility=
                                                "hidden";
```

9.7. Навигация по внесенным изменениям

Все редакторы спроектированы таким образом, что записывают в память промежуточные результаты изменений, внесенных в контент. Графические кнопки «Вперед» и «Назад» позволяют перемещаться по этим этапам в прямом и обратном направлении так же, как это делается, например, в программе Word.

Допустим, вы вставили в окно редактирования рисунок, а потом выделили одно из слов цветом, отличным от цвета остального текста. Если теперь один раз нажать кнопку «Назад», то исчезнет цветовое выделение. Нажимаем второй раз — исчезает картинка и контент в окне редактирования приобретает первоначальный вид. Теперь нажмем кнопку «Вперед» — картинка вновь займет свое место. Нажмем кнопку еще раз — вновь появится цветовое выделение слова. То есть контент примет тот же вид, что был достигнут по завершении последнего шага редактирования. Думаю, из этого описания алгоритм навигации по внесенным изменениям понятен.

Пошаговая запись промежуточных данных выполняется в двух случаях:

– при внесении изменений с помощью кнопок редактирования главной панели (то есть при изменении форматирования текста или добавлении любых элементов);

– при нажатии на клавиатуре кнопок со знаками препинания или «Enter».

Администратор должен учитывать второй случай, так как введенный текст будет записан в память только после нажатия одной из перечисленных кнопок.

Сохранение в памяти промежуточных данных выполняет специальная функция **inter**. Чтобы обеспечить ее корректную работу, предварительно необходимо объявить несколько переменных. Сначала создаем пустой массив, в котором очередной промежуточный результат будет следующим элементом:

```
let arr=[];
```

Понятно, что первый элемент — HTML-код исходного состояния контента в окне редактирования (дается пример из JavaScript-файла первого редактора):

```
arr[0]=ed.body.innerHTML;
```

Объявляем еще две переменные:

– счетчик внесенных изменений

```
let cna=0;
```

– идентификатор начала и конца просмотра

```
let vnc=0;
```

Все готово, можно написать функцию:

```
function inter()
   {
   cna++;
   vnc++;
   arr[cna]=ed.body.innerHTML;
   }
```

Как видите, при каждом ее вызове значения идентификатора и счетчика увеличиваются на единицу, а в массив добавляется новый элемент с новыми данными из окна визуального представления:

```
arr[cna]=ed.body.innerHTML;
```

Когда происходит вызов функции **inter**? Допустим, вы нажали одну из кнопок простого редактирования. Будет запущена функция, обрабатывающая это событие. После того как контент изменится, одна из последних инструкций вызовет функцию **inter**. Если вы добавляете в окно редактирования какой-то элемент, то запускается функция, отзывающаяся на событие нажатия кнопки «Вставить». И опять одна из ее последних инструкций вызовет функцию **inter**.

Для событий нажатия клавиш со знаками препинания или перевода строки у нас существует еще одна функция. Мы уже упоминали ее в разделе 9.3:

```
ed.addEventListener("keyup", function(ev)
   {
   tran();
   ...    // Код, который будет пояснен в разделе 9.7
   });
```

Пришло время показать код, скрытый за многоточием:

```
for(let i=0; i<symb.length; i++)
   {
   if(symb[i]==ev.key)
      inter();
   }
```

Здесь в цикле проверяется, совпадает ли символ нажатой клавиши с символом из заранее объявленного массива **symb**:

```
let symb=["Enter", "!", "?", ";", ":", ",", "."];
```

Если совпадение обнаружено, вызывается функция **inter**.

Теперь рассмотрим, как выполняется навигация. Начнем с движения в обратном направлении. Для его реализации файл содержит соответствующую анонимную функцию, запускающуюся при щелчке на графической кнопке «Назад»:

```
document.getElementById("bac").
                addEventListener("click", function()
   {
   if(vnc>0)
      {
      vnc--;
      ed.body.innerHTML=arr[vnc];
      co.innerHTML=arr[vnc];
      }
   });
```

Первым делом надо убедиться, что значение идентификатора **vnc** больше нуля (то есть результаты изменений не перемотаны до исходного состояния):

```
if(vnc>0)
   {
   ...
   }
```

Если это так, уменьшаем значение **vnc** на единицу

```
vnc--;
```

после чего загружаем в окно редактирования и в область основного содержания страницы предыдущие данные:

```
ed.body.innerHTML=arr[vnc];
co.innerHTML=arr[vnc];
```

Перемоткой промежуточных результатов вперед управляет другая функция:

```
document.getElementById("forw").addEventListener("click",
function()
   {
   if(vnc<arr.length-1)
      {
      vnc++;
      ed.body.innerHTML=arr[vnc];
      co.innerHTML=arr[vnc];
      }
   });
```

Здесь проверяется, не достигло ли значение идентификатора **vnc** верхней границы:

```
if(vnc<arr.length-1)
   {
   ...
   }
```

В случае истинности условия значение идентификатора увеличивается на единицу

```
vnc++;
```

после чего вновь загружаем в окно редактирования и в область основного содержания страницы следующие данные:

```
ed.body.innerHTML=arr[vnc];
co.innerHTML=arr[vnc];
```

Осталось добавить, что в остальных редакторах рассмотренные функции имеют очень незначительные отличия, которые мы не станем разбирать. Читатель сам легко обнаружит их, если посмотрит нужные файлы из zip–архива.

9.8. Запись отредактированной страницы

Отправка данных серверной программе для записи в файл и получение ответа происходят в фоновом режиме. Для этого мы используем технологию Ajax.

Схема процесса выглядит следующим образом. После нажатия кнопки «Записать» HTML-код из области визуального представления считывается в текстовый редактор, который является элементом формы. Вторым элементом служит скрытое поле, содержащее адрес записываемой страницы. Данные из формы пересылаются методом POST скрипту **rec.php** (о нем — в главе 14). Скрипт проверяет входные параметры и, если они корректные, производит запись в файл. В случае успешной записи **rec.php** возвращает **1**, при неудаче — **2**. Сообщения о результатах на 2 секунды появляются на кнопке отправки данных, после чего ей возвращается исходный текст (рис. 9.8.1).

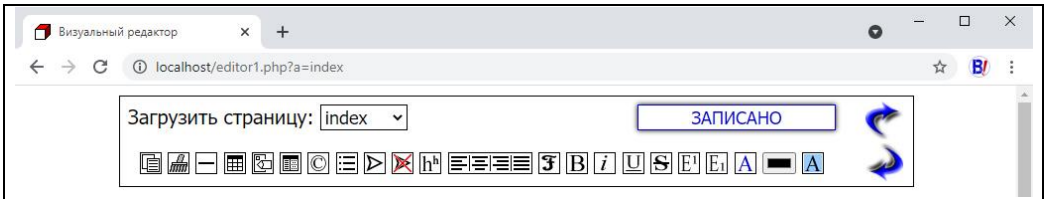

Рис. 9.8.1. Сообщение о результатах записи на кнопке отправки данных

Для реализации перечисленных этапов у нас существуют три функции.

Первая запускается после щелчка на кнопке:

```
document.getElementById("rec").
                addEventListener("click", function()
   {
   ...
   });
```

Процесс передачи данных сопровождается одним не совсем приятным явлением: в текстовом поле перед кодом появляются два перевода строки. Их легко увидеть в любом редакторе, например на главной странице, переключая режимы с визуального на HTML. Чтобы каждый раз не удалять их вручную, мы поручили эту задачу программе (на примере первого редактора):

```
let edin=ed.body.innerHTML.replace(/\n{2}/, "");
```

Теперь можно копировать данные в **textarea**:

```
te.value=edin;
```

Далее создаем:
– новый объект с данными из формы

```
let dt=new FormData(document.forms.dat);
```

– новый объект интерфейса **XMLHttpRequest**

```
let re=new XMLHttpRequest();
```

Методом **open** формируем запрос

```
re.open("POST", "rec.php", true);
```

и отправляем его серверному скрипту:

```
re.send(dt);
```

Также устанавливаем, что после загрузки ответа должна запускаться функция **show**:

```
re.addEventListener("load", show);
```

Ее задача на основе полученного ответа вывести то или иное сообщение на кнопке отправки данных:

```
function show()
   {
   let re=document.getElementById("rec");
   let ans=this.responseText;
   if(ans==1)
      re.value="ЗАПИСАНО";
   else
      re.value="НЕ ЗАПИСАНО";

   setTimeout(sto, 2000);
   }
```

Последняя инструкция сообщает программе, что через 2 секунды должна быть запущена функция **sto**. У нее очень простое назначение — вернуть кнопке исходную надпись:

```
function sto()
   {
   document.getElementById("rec").value="Записать";
   }
```

Мы рассмотрели ряд компонентов, которые очень схожи во всех четырех редакторах. Теперь можно приступать к детальному разбору каждого в отдельности. Главными темами всех последующих глав будут методы и функции, используемые для изменения или создания контента.

10. Традиционный редактор

Что собой представляют визуальные редакторы, которые используют последние 10–15 лет в CMS? При их создании существует два похода: можно применить в качестве окна редактирования фрейм или сделать такое окно из контейнера **div**. Мы начнем рассмотрение этих подходов по порядку возникновения. Сначала (его можно назвать классическим) появился редактор на основе фрейма с включенным свойством **designMode**. Именно на данном принципе и будет построен наш первый — традиционный — редактор.

Кстати, вы, наверное, уже обратили внимание, что большинство примеров кода, рассмотренных в предыдущих главах, относились именно к первому редактору. Мы продолжим это начинание и на примере традиционной системы подробно разберем механизмы изменения или создания контента. При разборе остальных редакторов будем обращать внимание только на их принципиальные отличия.

Но прежде чем приступить к делу, хочу посоветовать во время чтения последующих глав держать открытыми на компьютере соответствующие файлы из zip-архива. А еще лучше дополнительно запустить локальный хостинг и не только в теории, но и на практике наблюдать результаты работы тех или иных функций, написанных на JavaScript. Так вы достигнете максимально ясного и объемного восприятия материала.

Перечень компонентов традиционного редактора:
– основной файл — **editor1.php**;
– файл сценариев на JavaScript — **set/editor1.js**;
– файл таблицы стилей — **set/editor1.css**.

Можно также обойтись без локального хостинга. В помощь читателю, как вы помните, есть сайт с демонстрационными версиями редакторов. Все они действующие (только не позволяют записывать результаты ваших экспериментов). Первый из них можно видеть по адресу **https://editjs.ru/editor1.php?a=index**.

10.1. Включение режима редактирования

После того как страница запущена в окне браузера, данные из области основного содержания загружаются во фрейм, предназначенный для изменения или создания контента. Как это делается, мы говорили в разделе 9.3:

```
ed.body.innerHTML=co.innerHTML;
```

Но это еще не все. Необходимо включить режим редактирования. Он активируется в файле **editor1.js** одновременно с передачей данных во фрейм:

```
ed.designMode="on";
```

Теперь режим включен и можно приступать к редактированию контента.

10.2. Копирование текста

Одно из самых простых действий. Регистрируем анонимную функцию в качестве обработчика события щелчка на кнопке копирования:

```
document.getElementById("cop").
                addEventListener("click", function()
   {
   ...
   });
```

Проверяем, сделано ли выделение фрагмента текста. Если нет, выводим предупреждение:

```
if(sel=="")
   alert("Вы не выделили текст!");
```

Если текст выделен, копируем его, используя свойство **clipboard** интерфейса **writeText**:

```
else
   navigator.clipboard.writeText(sel);
```

Напоминание: будет скопирован только текст, без находящихся в нем элементов и тегов форматирования.

10.3. Удаление форматирования и ссылок

Как вы помните, в редакторах № 1–3 удаление форматирования и ссылок — это один и тот же процесс. Поэтому в качестве обработчика для нажатия соответствующих кнопок используем только одну функцию.

Регистрируем обработчики:

```
document.getElementById("forma").
                     addEventListener("click", nofr);
document.getElementById("delin").
                     addEventListener("click", nofr);
```

Первая операция в функции **nofr** — уже знакомая нам проверка выделения текста:

```
function nofr()
   {
   if(sel=="")
      alert("Вы не выделили текст!");
   ...
   }
```

Если она прошла успешно, удаляем все теги из выбранного фрагмента. Для этого указываем, что будет обрабатываться та часть, которую охватывает выделение:

```
let rng=sel.getRangeAt(0);
```

Дальше создаем новый текстовый узел, в который помещаем только текстовое содержимое выделения:

```
let cr=ed.createTextNode(sel.toString());
```

Удаляем выделенный фрагмент

```
sel.deleteFromDocument();
```

и вставляем на это место чистый текст:

```
rng.insertNode(cr);
```

В самом конце переносим получившийся контент из окна редактирования в страницу нижнего фрейма

```
tran();
```

и записываем новые данные в массив промежуточных этапов редактирования

```
inter();
```

10.4. Простое редактирование текста

Чтобы задействовать кнопки простого редактирования, сначала создадим два массива:
– с **id** кнопок

```
let mid=["bol", "ital", "under", "strik", "sup", "sub"];
```

– с перечнем элементов, которые создают эти кнопки

```
let mco=["b", "i", "u", "s", "sup", "sub"];
```

Обратите внимание, что в первом массиве порядковый номер кнопки с **id="bol"** соответствует порядковому номеру создаваемого этой кнопкой элемента **b** во втором массиве. И так для всех остальных кнопок.

Поскольку кнопок много, для сокращения кода станем регистрировать обработчики в цикле. Первым делом создадим массив кнопок, ориентируясь на то, что все они (и только они) принадлежат к классу **but**,

```
let m=document.querySelectorAll(".but");
```

а затем зарегистрируем для всех один и тот же обработчик

```
for(let i=0; i<m.length; i++)
   document.getElementById(mid[i]).
   addEventListener("click", emb.bind(null, mco[i]));
```

Как видите, в функцию **emb** передается в качестве аргумента имя тега из массива **mco**.

Функция **emb**, которая вставляет теги в текст:

```
function emb(v)
   {
   if(sel=="")
      alert("Вы не выделили текст!");
   else
      {
      let rng=sel.getRangeAt(0);
      let elm=ed.createElement(v);
      rng.surroundContents(elm);

      tran();
      inter();
      }
   }
```

Здесь четыре знакомых нам фрагмента кода:

– проверка выделения;

– создание на основе выделения диапазона для обработки;

– перенос контента из окна редактирования в страницу нижнего фрейма;

– запись данных в массив промежуточных этапов редактирования.

После указания диапазона создаем в редакторском фрейме методом **createElement** новый парный тег

```
let elm=ed.createElement(v);
```

и обрамляем этим тегом выделенный текст (методом **surroundContents**):

```
rng.surroundContents(elm);
```

Как видите, термин «простое редактирование» использован автором неслучайно. Действительно, такие действия элементарны: выделил текст, нажал кнопку и получил искомый результат.

10.5. Изменение цвета букв или фона текста

Описание этого процесса начнем с функции изменения цвета:

```
function clr(b)
```

```
{
...
}
```

Что за аргумент получает функция, станет ясно по ходу рассказа.

Функция имеет уже ставшие привычными инструкции:

– проверку выделения;

– создание диапазона для обработки;

– перенос контента из окна редактирования в страницу нижнего фрейма;

– запись данных в массив промежуточных этапов редактирования;

– добавление парного тега к выделению.

Чтобы раскрасить текст, мы используем парный тег **span**:

```
let elm= ed.createElement("span");
```

После его создания устанавливаем стиль тега, где код цвета определяется значением из поля с идентификатором **ccc**:

```
elm.style=b+"color: "+
                document.getElementById("ccc").value;
```

Если нужно выделить цветом буквы, то аргумент **b** имеет пустое значение, так как этот способ выделения задается свойством **color**. Когда необходимо изменить цвет фона текста, то в **b** передается значение **background-**, так как данное выделение устанавливается свойством **background-color**.

Поэтому регистрация обработчика для событий щелчка на кнопках выглядит следующим образом. При нажатии кнопки выбора цвета букв в функцию передается пустой аргумент:

```
document.getElementById("col").
       addEventListener("click", clr.bind(null, ""));
```

При нажатии кнопки выбора цвета фона в функцию в качестве аргумента передается строка **background-**:

```
document.getElementById("backgr").
                  addEventListener("click",
                      clr.bind(null, "background-"));
```

10.6. Добавление разметки маркированного списка

Важное замечание к этой функции: чтобы создать список, используйте перевод на следующую строку, а не на следующий абзац (то есть нажимайте комбинацию клавиш «Shift»+«Enter», а не просто «Enter»).

Регистрируем обработчик:

```
document.getElementById("ulli").
                 addEventListener("click", function()
```

```
   {
   ...
   });
```

Проверяем выделение. Если оно есть, присваиваем выделенный фрагмент новой переменной:

```
let ulli=sel;
```

Разбиваем выделенный текст на отдельные строки, используя в качестве разделителя символ перевода строки **\n**:

```
let masulli=ulli.toString().split("\n");
```

Добавляем к строкам парный тег **li**:

```
let sum="";
for(let i=0; i<masulli.length; i++)
   sum+="<li>"+masulli[i]+"</li>";
```

Создаем элемент списка **ul**:

```
let rng=sel.getRangeAt(0);
let elm=ed.createElement("ul");
```

Определяем, есть ли другие списки в окне редактирования, сколько их:

```
ed.getElementsByTagName("UL").length
```

и одновременно создаем **id** для нового списка

```
let len="sul"+ed.getElementsByTagName("UL").length;
```

Идентификатор состоит из символов **sul** и порядкового номера списка, полученных на предыдущем шаге:

```
elm.id=len;
```

Если созданный элемент — первый, то у списка **id** будет **sul0**, если элемент второй, то **id** списка будет **sul1** и т. д.

Вставляем тег **ul** в окно редактирования

```
rng.surroundContents(elm);
```

и помещаем в его «внутренности» элементы списка:

```
ed.getElementById(len).innerHTML=sum;
```

Последний этап — перенос контента из окна редактирования в страницу нижнего фрейма и запись данных в массив промежуточных этапов редактирования:

```
tran();
inter();
```

10.7. Вставка линии

С этого раздела начинается описание добавления элементов, которые нуждаются в предварительных настройках. Поэтому сначала расскажем, какая схема действий реализована в их функциях. Порядок операций следующий:

– получаем данные из полей формы соответствующей вкладки;

– проверяем, все ли критически важные параметры введены администратором;

– компонуем данные, полученные из формы.

Дальше возможны два варианта действий:

– если настройки элемента уникальны, создаем его в текущей функции, помещаем элемент в документ и добавляем ему скомпонованные данные;

– если настройки элемента универсальны, передаем все его данные отдельной функции, которая обслуживает создание элементов, происходящее по одному и тому же алгоритму.

Из этого вытекает, что перенос контента из окна редактирования в страницу нижнего фрейма и запись данных в массив промежуточных этапов редактирования происходят либо в текущей функции, либо во внешней.

А теперь перейдем к созданию линий.

Регистрируем обработчик:

```
document.getElementById("buthr").
                addEventListener("click", function()
   {
   ...
   });
```

Первый шаг — получение данных из панели настройки:

```
let hr1_v=document.getElementById("hr1").value;
let hr2_v=document.getElementById("hr2").value;
let hr3_v=document.getElementById("hr3c").value;
```

На случай, если администратор заполнил не все поля, создаем две пустые переменные:

```
let hr1="";
let hr2="";
```

Напомню, что на вкладке выбора цвета по умолчанию всегда установлен черный, поэтому данное поле не бывает пустым.

Если толщина линии введена, записываем в переменную **hr1** фрагмент настройки стиля:

```
if(hr1_v!="")
   hr1="height: "+hr1_v+"; ";
```

Если нет, то устанавливаем линии толщину **1px**:

```
else
   hr1="height: 1px; ";
```

Проверяем, введено ли значение ширины линии. Если да, то записываем во вторую переменную свойство **width**:

```
if(hr2_v!="")
   hr2="width: "+hr2_v+"; ";
```

Компонуем строку с настройками стиля

```
let shr=hr1+hr2+'background: '+hr3_v+'; border: 0;';
```

Напомню, что по умолчанию все браузеры создают линиям тени. Этот фрагмент кода убирает тень:

```
border: 0;
```

Теперь передаем все параметры функции **place**, которая вставляет в окно редактирования некоторые элементы, в том числе линии:

```
place("hr", "", shr, "", "hr_d");
```

Здесь первый аргумент — тег вставляемого элемента, третий — настройки стилей, пятый — идентификатор закрываемой вкладки.

Как уже было сказано, функция **place** создает разные элементы, в том числе такие, которым требуется 4 (фреймы) или 5 (рисунки) аргументов. Создание линии обходится только тремя, поэтому второй и четвертый аргументы пустые.

Функция **place**:

```
function place(x, y, z, w, q)
   {
   let rng=sel.getRangeAt(0);
   let elm=ed.createElement(x);

   if(y!="")
      elm.src=y;

   elm.style=z;

   if(w!="")
```

```
      {
      elm.alt=w;
      elm.title=w;
      }

   rng.surroundContents(elm);

   tran();
   inter();
   clo(q);
   }
```

Здесь второй аргумент — адрес фрейма или рисунка, четвертый — текст альтернативной подписи к рисунку. Все остальные инструкции вам уже знакомы по предыдущим разделам.

10.8. Вставка фрейма

Регистрируем обработчик:

```
document.getElementById("butifr").
                addEventListener("click", function()
   {
   ...
   });
```

Получаем данные:

```
let ifr1_v=document.getElementById("ifr1").value;
...
let ifr7_v=document.getElementById("ifr7").value;
```

Проверяем, введен ли адрес страницы, загружаемой во фрейм, и, если администратор забыл это сделать, выводим предупреждение:

```
if(ifr1_v=="")
   alert("Вы не указали адрес загружаемой страницы!");
```

Когда адрес введен, наступает следующий этап. Объявляем 5 пустых переменных

```
let ifr2="";
let ifr3="";
let ifr4="";
let ifr5="";
let ifr7="";
```

и на их основе формируем значения свойств стиля:

```
if(ifr2_v!="")
   ifr2="width: "+ifr2_v+"; ";
```

```
if(ifr3_v!="")
   ifr3="height: "+ifr3_v+"; ";

if(ifr4_v!="no")
   ifr4="float: "+ifr4_v+"; ";

if(ifr5_v!="")
   ifr5="border: "+ifr5_v+" solid "+ifr6_v+";";
else
   ifr5="border: 1px solid "+ifr6_v+";";

if(ifr7_v!="")
   ifr7=" margin: "+ifr7_v+";";
```

Затем компонуем эти данные в одну строку

```
let sifr=ifr2+ifr3+ifr4+ifr5+ifr7;
```

и запускаем функцию **place** с набором из четырех заполненных аргументов и одного пустого:

```
place("iframe", ifr1_v, sifr, "", "ifr_d");
```

10.9. Вставка рисунка

Уже привычная регистрация обработчика:

```
document.getElementById("butima").
                addEventListener("click", function()
   {
   ...
   });
```

Получаем данные из полей вкладки:

```
let ima1_v=document.getElementById("ima1").value;
...
let ima8_v=document.getElementById("ima8c").value;
```

Выясняем, введен ли хотя бы один адрес картинки (внешней или внутренней). В случае забывчивости администратора делаем ему предупреждение:

```
if(ima1_v=="" && ima2_v=="")
   alert("Вы не указали адрес рисунка!");
```

Проверяем, создана ли альтернативная подпись к рисунку (согласно требованиям Консорциума Всемирной Паутины, атрибут **alt** является обязательным в теге изображения). В противном случае опять выносим предупреждение:

```
if(ima3_v.length<2)
   alert("Вы не ввели подпись к рисунку!");
```

Когда введен адрес внешнего рисунка, понятно, что загружен будет он. Если введены сразу два адреса — внешнего и внутреннего фото — то приоритет отдается внешнему. Если выбрана только внутренняя картинка, то в этом случае будет загружена она:

```
if(ima1_v!="")
   ima=ima1_v;
else
   ima="../cosmos/"+ima2_v;
```

Процедуры формирования переменных для записи свойств стилей уже знакомы вам по двум предыдущим разделам. Вы можете подробно изучить их, открыв файл **editor1.js** из папки **set** zip-архива.

Сформировав промежуточные переменные, сводим их в одну:

```
let sima=ima4+ima5+'width: '+ima6+';
            border: '+ima7+' solid '+ima8_v+
                                '; cursor: pointer;';
```

Обратите внимание: при наведении указателя мыши на рисунок курсор будет принимать вид «руки», чтобы посетитель догадывался — на фото можно кликнуть и посмотреть его в увеличенном формате.

Последний этап — обращение к функции **place**:

```
place("img", ima, sima, ima3_v, "img_d");
```

На этот раз мы задействовали и четвертый аргумент функции, передав через него альтернативную подпись к изображению.

10.10. Вставка таблицы

Создание таблицы выполняет одна из наиболее «массивных» функций — из-за обилия настроек в ней 77 строк кода. Естественно, для экономии места автор не станет приводить ее целиком, а покажет только основные части. Полностью код этой функции можно посмотреть в файле **editor1.js** из папки **set** zip-архива.

Обработчик клика на кнопке **butab**:

```
document.getElementById("butab").
                addEventListener("click", function()
   {
   ...
   });
```

Данные настроек:

```
let tab1_v=document.getElementById("tab1").value;
...
let tab12_v=document.getElementById("tab12").value;
```

Обязательные параметры для таблицы — количество строк и столбцов. Проверяем их наличие. Если администратор забыл указать эти числа или одно из них, открываем диалоговое окно с предупреждением:

```
if(tab2_v=="" || tab3_v=="")
   alert("Вы не указали количество
                              строк или столбцов!");
```

На основе полученных значений из вкладки формируем набор свойств таблицы и объединяем их в одну строку:

```
let tbf=tab1+'border-collapse: collapse;
       background: '+tab11_v+'; '+tab5+tab7+
             'font-family: '+tab6_v+'; color: '+
                                  tab8_v+';'+tab12;
```

Обратите внимание, мы свели границы ячеек

```
border-collapse: collapse;
```

чтобы таблица имела единую внешнюю рамку.

Теперь надо сформировать сами ячейки. Делаем это в циклах:

```
let sum="";
for(let i=0; i<tab2_v; i++)
   {
   sum+='<tr>';

   for(let t=0; t<tab3_v; t++)
       sum+='<td style="border: '+tab9+
           ' solid '+tab10_v+';'+tab4+'"> </td>';

   sum+='</tr>';
   }
```

Внешний цикл совершает количество проходов, равное количеству строк, а внутренний — количеству столбцов. Переменная **tab9** задает толщину рамки, а **tab10_v** — цвет рамки для каждой ячейки. Переменная **tab4** определяет внутренний зазор между содержимым ячейки и ее границами. Пробел ** ** добавлен в каждую ячейку специально. Если этого не сделать, таблица может сжаться так, что в ячейки невозможно будет вставить курсор.

Теперь создадим таблицу и добавим ей стили:

```
let rng=sel.getRangeAt(0);
let elm=ed.createElement("table");
elm.style=tbf;
```

Следующий шаг — присвоение таблице **id** (то же самое мы делали при создании маркированного списка):

```
let len="tbv"+ed.getElementsByTagName("TABLE").length;
elm.id=len;
```

Помещаем таблицу в окно редактирования

```
rng.surroundContents(elm);
```

и вставляем в нее ячейки:

```
ed.getElementById(len).innerHTML=sum;
```

Завершается функция тремя инструкциями

```
tran();
inter();
clo("tabl_d");
```

которые выполняют:

– перенос контента из окна редактирования в страницу нижнего фрейма;
– запись данных в массив промежуточных этапов редактирования;
– закрытие вкладки с настройками таблицы.

10.11. Вставка символов

Символов очень много — 50 штук. Регистрировать для каждого из них обработчик отдельной строкой — хлопотное и нерациональное занятие. Поэтому все события нажатия кнопок символов будем регистрировать в цикле.

Создадим массив кнопок, ориентируясь на то, что все они (и только они) принадлежат к классу **sym**:

```
let q=document.querySelectorAll(".sym");
```

Зарегистрируем для всех кнопок один и тот же обработчик

```
for(let i=0; i<q.length; i++)
   {
   let idc=q[i].id;
   let smb=q[i].value;
   document.getElementById(idc).
     addEventListener("click", sign.bind(null, smb));
   }
```

Здесь мы последовательно выясняем идентификатор очередной кнопки

```
let idc=q[i].id;
```

получаем символ этой кнопки

```
let smb=q[i].value;
```

и регистрируем один обработчик для событий нажатия любой кнопки:

```
document.getElementById(idc).
     addEventListener("click", sign.bind(null, smb));
```

Функция **sign(s)** работает очень просто. Создаем новый текстовый узел, который содержит переданный в аргументе символ

```
let elm=ed.createTextNode(s);
```

и вставляем его в позицию курсора в окне редактирования

```
let rng=sel.getRangeAt(0);
rng.insertNode(elm);
```

В конце выполняем уже привычные завершающие операции:

```
tran();
inter();
clo("copyr_d");
```

10.12. Добавление ссылки

В этом модуле сразу после регистрации обработчика

```
document.getElementById("butlink").
                addEventListener("click", function()
   {
   ...
   });
```

выполняется проверка выделения с типичным предупреждением в случае его отсутствия:

```
if(sel=="")
   alert("Вы не выделили текст!");
```

Дальше все идет привычным образом. Сначала получаем данные:

```
let link1_v=document.getElementById("link1").value;
...
let link6_v=document.getElementById("link6c").value;
```

Затем формируем стиль, а заодно определяемся, будет ли ссылка с подчеркиванием:

```
if(link5_v=="no")
   link5='text-decoration: none; color: '+link6_v+';';
else
   link5='color: '+link6_v+';';
```

Следом выясняем, какой должен быть адрес ссылки — внешний или внутренний:

```
if(link2_v!="")
   ares=link1_v+link2_v;
else
   ares="index.php?a="+link4_v;
```

Создаем ссылку в документе:

```
let rng=sel.getRangeAt(0);
let lin=ed.createElement("a");
```

Добавляем адрес и стили:

```
lin.href=ares;
lin.style=link5;
```

Определяемся, в текущей или новой странице должна быть открыта ссылка (если добавляется внешняя ссылка):

```
if(link2_v!="" && link3_v=="yes")
   lin.target="_blank";
```

«Оборачиваем» выделенный текст в ссылку

```
rng.surroundContents(lin);
```

и завершаем процесс:

```
tran();
inter();
clo("link_d");
```

Хочу также обратить ваше внимание, что при создании ссылок действует правило, уже знакомое вам по модулю выбора фотографий: когда введены адреса внешней и внутренней ссылок, преимущество имеет внешний адрес.

10.13. Добавление заголовка

С этого раздела начинается описание трех обработчиков, каждый из которых на финише запускает одну и ту же функцию — **wrap**:

```
function wrap(r, u, j)
   {
```

```
   if(sel=="")
      alert("Вы не выделили текст!");
   else
      {
      let rng=sel.getRangeAt(0);
      let elm=ed.createElement(r);
      elm.style=u;
      rng.surroundContents(elm);

      tran();
      inter();
      clo(j);
      }
   }
```

Функция получает три аргумента:
– название создаваемого элемента;
– настройки стиля элемента;
– идентификатор закрываемой вкладки.

Инструкции функции уже так хорошо нам знакомы, что разбирать их не имеет смысла. Поэтому перейдем к добавлению заголовков.

Регистрируем обработчик:

```
document.getElementById("buth16").
                addEventListener("click", function()
   {
   ...
   });
```

Получаем данные настроек:

```
let h161_v=document.getElementById("h161").value;
...
let h164_v=document.getElementById("h164c").value;
```

Формируем строку стилей:

```
let h163="";

if(h163_v!="")
   h163=" font-size: "+h163_v+";";

let h16res='font-family: '+h162_v+';'+h163+
                                ' color: '+h164_v+';';
```

И отправляем информацию функции **wrap**

```
wrap(h161_v, h16res, "h16_d");
```

которая добавляет выделенному тексту один из тегов **h1**–**h6** и свойства стиля.

10.14. Выравнивание абзаца

Как всегда, пишем оболочку для обработчика:

```
document.getElementById("butequa").
                  addEventListener("click", function()
   {
   ...
   });
```

На основе данных из вкладки

```
let equa1_v=document.getElementById("equa1").value;
...
let equa5_v=document.getElementById("equa5").value;
```

формируем строку свойств абзаца

```
let equa3="";
let equa5="";

if(equa3_v!="")
   equa3=" font-size: "+equa3_v+";";

if(equa5_v!="")
   equa5=" text-indent: "+equa5_v+";";

let equares='text-align: '+equa1_v+
              '; font-family: '+equa2_v+';'+
                  equa3+' color: '+equa4_v+';'+equa5;
```

и отправляем необходимые параметры функции **wrap**:

```
wrap("p", equares, "equa_d");
```

10.15. Настройка шрифта

В этом модуле тоже нет ничего особенного — только неоднократно разобранные инструкции. Так как код модуля довольно короткий, приведем его целиком. Думаю, читатель без проблем поймет, что и как устроено в этой функции:

```
document.getElementById("butfont").addEventListener("click",
function()
   {
   let font1_v=document.getElementById("font1").value;
   let font2_v=document.getElementById("font2").value;
   let font3_v=document.getElementById("font3c").value;

   let font2="";

   if(font2_v!="")
      font2=" font-size: "+font2_v+";";
```

```
   let    fontres='font-family:    '+font1_v+';'+font2+'    color:
'+font3_v+';';

   wrap("span", fontres, "font_d");
   });
```

Мы подробно (точнее — почти подробно) разобрали устройство первого редактора. При разборе остальных не будем досконально описывать уже знакомые вам функции, а станем обращать внимание в первую очередь на принципиальные отличия в модулях, выполняющих аналогичные действия.

11. Редактор с фиксированными стилями

Придерживаясь хронологии возникновения разных систем, вторым рассмотрим редактор, построенный на основе контейнера **div** с включенным атрибутом **contenteditable**.

Помимо данного отличия от первой системы, у второго редактора есть еще одна особенность, которая понятна из его названия. В этом редакторе нельзя произвольно настраивать параметры элементов. Выбор делается из наборов чисел или кодов (последние — для цветовой гаммы), которые установлены изначально. Поэтому в большинстве случаев текстовые поля для параметров заменены во втором редакторе на выпадающие списки с фиксированными значениями. Если, например, в первом редакторе вы могли выбирать любую ширину вставляемой линии в процентах с шагом 1%, то во втором — только от 10 до 100 процентов с шагом 10%. Если в первом редакторе можно задать любую ширину вставляемого рисунка, то во втором — только от 50 до 600 пикселей с шагом 50px. Думаю, после этих примеров различия понятны.

Зачем такие изменения в конструкции второй системы?

Дело в том, что первый редактор имеет один не то чтобы недостаток, но не совсем приятный нюанс. Если вы, просматривая в первом редакторе главную страницу, переключитесь на вкладку HTML, то увидите, что стили всех элементов оказываются встроенными в их теги. Согласно корпоративным правилам, распространенным среди программистов, это не очень хорошо.

Более правильным считается писать в теге атрибут **class** и уже в его параметрах указывать, к какому или каким классам таблицы стилей относятся свойства данного элемента. Второй редактор как раз и позволяет добиться такого результата. В нем каждое значение из любого выпадающего списка принадлежит к определенному классу. Соответственно, при вставке элемента в окно редактирования в теге окажется не атрибут **style**, а **class** со списком классов для тех или иных свойств. Если в первом редакторе, добавив фото, вы увидите в его теге строку

```
style="margin: 6px; width: 350px;
  border: 2px solid rgb(204, 0, 0); cursor: pointer;"
```

то во втором —

```
class="mar6 wi350 bor2 bosty borCC0000 cursp"
```

что полностью соответствует корпоративным правилам.

Перечень компонентов второго редактора:

– основной файл — **editor2.php**;

– файл сценариев на JavaScript — **set/editor2.js**;

– основной файл таблицы стилей — **set/editor2.css**;

– дополнительный файл таблицы стилей — **set/cont.css**.

Демонстрационная страница редактора:
https://editjs.ru/editor2.php?a=index.

11.1. Установка режима редактирования

В отличие от предыдущего редактора, в этом ничего включать не надо. Требуемый режим задан непосредственно в разметке страницы **editor2.php**:

```
<div id="edit" contenteditable="true"></div>
```

11.2. Копирование текста

Одно предварительное замечание. В редакторах с первого по третий включительно использованы методы **createElement** и **surroundContents**. При этом **createElement** имеет один недостаток. Это метод уровня документа: **document.createElement(element)**. Соответственно, сначала в документе нужно создать какой-то элемент, а потом уже поместить его в окно редактирования. В первой системе этот момент не критичен, так как **createElement** работает не со страницей редактора, а с фреймом, который и является рабочим документом. Таким образом, все элементы вставляются только в этот фрейм и больше никуда. А что произойдет, если во втором редакторе попытаться создать элемент, забыв вставить курсор в окно визуального представления? Получится, что последний элемент из текущего документа, имевший фокус перед нажатием кнопки «Вставить», — это панель настройки. В нее и будет добавлена таблица или рисунок (чего ни в коем случае быть не должно). Чтобы не происходили такие казусы, после нажатия кнопки «Вставить» необходимо устанавливать фокус на окне редактирования.

Может появиться вопрос: сейчас мы разбираем копирование текста и никаких элементов не создаем. Причем тут фокусировка? Дело в том, что ее отсутствие в данной функции приводит к тому, что можно скопировать любой фрагмент текста из любой части страницы. Вводя фокусировку, мы тем самым ограничиваем область копирования редакторским окном.

Теперь к делу. Регистрируем анонимную функцию в качестве обработчика события щелчка на кнопке копирования:

```
document.getElementById("cop").
                addEventListener("click", function()
   {
   ...
   });
```

Устанавливаем фокус на поле редактирования:

```
ed.focus();
```

А дальше пишем основную часть функции, которая уже знакома вам по аналогичному коду первого редактора:

```
if(sel=="")
   alert("Вы не выделили текст!");
else
   navigator.clipboard.writeText(sel);
```

11.3. Удаление форматирования и ссылок

Данный блок кода почти один в один повторяет аналогичный модуль из предыдущего редактора. Но есть два отличия.

Во-первых, это перемещение фокуса на окно редактирования:

```
ed.focus();
```

Во-вторых, окно визуального представления теперь является частью текущего документа, а не внешнего, расположенного во фрейме. Поэтому и текстовый узел создается в текущем документе:

```
let cr=document.createTextNode(sel.toString());
```

Все остальные инструкции — точные копии тех, что мы разбирали в разделе 10.3.

11.4. Простое редактирование текста

Способ регистрации обработчиков для кнопок простого редактирования остался прежним. В самой функции emb те же изменения, что и в функции предыдущего раздела:
– перемещаем фокус на окно редактирования

```
ed.focus();
```

– новый элемент создаем в текущем документе

```
let elm=document.createElement(v);
```

11.5. Изменение цвета букв или фона текста

С этого раздела начинаются более заметные изменения. Так как при «раскраске» текста и создании настраиваемых элементов мы теперь используем не **style**, а **class**, то последующие функции будут заниматься компоновкой списков классов, использованных для оформления элементов.

Регистрация обработчиков данного модуля выглядит по-новому:

```
document.getElementById("col").
     addEventListener("click", clr.bind(null, "co"));
```

Здесь мы передаем в функцию **clr** часть имени класса **co**, которое в соединении со значением из выпадающего списка образует полное имя класса. Символы **co** означают, что данный класс служит для цветового оформления букв, а записывается он в файле **cont.css** так:

```
.coXXXXXX {color: #XXXXXX;}
```

где **XXXXXX** — буквенно-цифровой код цвета.
Соответственно, для цветового выделения фона текста обработчик выглядит следующим образом:

```
document.getElementById("backgr").
     addEventListener("click", clr.bind(null, "ba"));
```

Символы **ba** означают, что данный класс служит для цветового оформления фона, а записывается он в файле **cont.css** так:

```
.baXXXXXX {background: #XXXXXX;}
```

где **XXXXXX** — все тот же буквенно-цифровой код цвета.

В самой функции **clr** кроме наведения фокуса есть и другие перемены. Одна из них опять связана с документом, в котором создается элемент **span**:

```
let elm=document.createElement("span");
```

Другая определяется тем, что теперь мы формируем список классов (в данной функции он один, но в других их будет больше):

```
elm.className=b+document.getElementById("ccc").value;
```

В этой строке к переданному в функцию аргументу **b** прибавляется буквенно-цифровой код цвета из выпадающего списка **ccc**.

11.6. Добавление разметки маркированного списка

В данной функции опять всего два отличия, связанные с изменением рабочего документа с внешнего на внутренний. Это проявляется во время создания элемента **ul**

```
let elm=document.createElement("ul");
```

и наполнения списка маркированных строк

```
document.getElementById(len).innerHTML=sum;
```

11.7. Вставка линии

Этот блок кода стал намного короче — вместо 21 строки в файле **editor1.js** всего 10 строк в файле **editor2.js**. Так, кстати, будет и с другими функциями — во втором редакторе они подчас заметно «легче». Почему?

Дело в том, что второй редактор обеспечивает администратору гораздо меньше возможностей по вводу произвольных значений. Исключение составляют настройки отдельных параметров, которые вводятся вручную:

– количество строк и столбцов в таблицах;

– адреса внешних изображений, фреймов и ссылок;

– подписи к рисункам.

Все остальные параметры уже имеют в выпадающих списках изначально установленные значения. Если списки вообще не трогать, то элементам будут даны наборы свойств из установок по умолчанию. Из этого следует, что операции проверки существования данных и их типа во многих функциях либо вообще отсутствуют, либо представлены в незначительном объеме.

В результате во втором редакторе код функции вставки линий очень простой.

Получаем данные из панели

```
let hr1_v=document.getElementById("hr1").value;
let hr2_v=document.getElementById("hr2").value;
let hr3_v=document.getElementById("hr3c").value;
```

формируем строку с перечнем классов

```
let shr=hr1_v+" "+hr2_v+" "+hr3_v+" bor0";
```

добавляя в нее класс **bor0** (**.bor0 {border: 0;}**), и отправляем данные функции **place**

```
place("hr", "", shr, "", "hr_d");
```

11.8. Вставка фрейма

Еще одна более короткая функция. Начинается она как обычно:

– получаем данные из панели настроек;

– проверяем, указан ли адрес загружаемой страницы;

– создаем необходимую переменную.

Если администратор выбрал положение для фрейма, то получаем имя класса данного размещения

```
if(ifr4_v!="no")
   ifr4=" "+ifr4_v;
```

и формируем строку с перечнем классов, используемых для оформления элемента:

```
let sifr=ifr2_v+" "+ifr3_v+ifr4+" "+ifr5_v+
                          " bosty "+ifr6_v+" "+ifr7_v;
```

Здесь **bosty** — **.bosty {border-style: solid;}**,

Последний шаг — отправляем информацию функции **place:**

```
place("iframe", ifr1_v, sifr, "", "ifr_d");
```

11.9. Вставка рисунка

В первом редакторе при добавлении рисунка помимо его подписи и адреса проверяется наличие еще 5 параметров и в зависимости от их присутствия или отсутствия формируются свойства стиля. Во втором редакторе проверяемых параметров только два, так как остальные имеют установки по умолчанию.

Естественно, обязательно выясняется, какая фотография должна быть отправлена в окно редактирования — внешняя или внутренняя:

```
let ima="";

if(ima1_v!="")
   ima=ima1_v;
else
   ima="../cosmos/"+ima2_v
```

Кроме того, определяется, будет ли у снимка позиционирование относительно других элементов или нет:

```
let ima4="";

if(ima4_v!="no")
   ima4=ima4_v+" ";
```

Затем формируется строка с набором классов:

```
let sima=ima4+ima5_v+" "+ima6_v+" "+ima7_v+
                             " bosty "+ima8_v+" cursp";
```

Здесь **bosty** — **.bosty {border-style: solid;}**, а **cursp** — **.cursp {cursor: pointer;}**.

Функции **place** отправляем все пять аргументов:

```
place("img", ima, sima, ima3_v, "img_d");
```

11.10. Вставка таблицы

Поскольку функция вставки таблицы выполняет все операции самостоятельно, без участия «посредников», то начинается она с установки фокуса на окне редактирования:

```
ed.focus();
```

Перечень классов формируется в следующей строке:

```
let tbf=tab1_v+" bocollap "+tab11_v+" "+tab5_v+
              " "+tab6_v+" "+tab7_v+" "+tab8_v+tab12;
```

где **bocollap** — класс **.bocollap {border-collapse: collapse;}**, необходимый для сведения границ ячеек, а переменная **tab12** содержит информацию о наличии или отсутствии позиционирования таблицы:

```
let tab12="";

if(tab12_v!="no")
   tab12=" "+tab12_v;
```

Изменилась строка, формирующая в циклах отдельные ячейки. Теперь она записывается так:

```
sum+='<td class="'+tab9_v+' bosty '+tab10_v+
                           ' '+tab4_v+'"> </td>';
```

Здесь **bosty** — все тот же класс, задающий границу таблицы и ячеек в виде непрерывной линии.

Инструкция создания элемента теперь выглядит так:

```
let elm=document.createElement("table");
```

Вместо набора стилей добавляем таблице набор классов:

```
elm.className=tbf;
```

Содержимое таблицы внедряется в ее теги следующим образом:

```
document.getElementById(len).innerHTML=sum;
```

11.11. Вставка символов

Функции первого и второго редакторов практически идентичны, за исключением типовых изменений. Во втором редакторе:

– добавлена фокусировка;

– текстовый узел создается в текущем документе.

11.12. Добавление ссылки

Главное отличие данного варианта функции — делая выбор между ссылкой с подчеркиванием или без, мы создаем не часть стиля, а часть списка классов:

```
if(link5_v=="no")
   link5='tedecno '+link6_v;
else
   link5=link6_v;
```

В данной записи **tedecno** — это **.tedecno {text-decoration: none;}**.

Сам список классов состоит всего из одного пункта:

```
lin.className=link5;
```

Ну и куда же без установки фокуса и создания элемента в текущем документе с переносом тегов ссылки в окно редактирования.

11.13. Добавление заголовка

Сокращение кода не коснулось этой функции — в ней 16 строк, как и у ее предшественницы. Несколько иначе выглядит инструкция определения — выбран или нет размер шрифта:

```
if(h163_v!="no")
      h163=" "+h163_v;
```

Чуть по-другому компонуются настройки свойств:

```
let h16res=h162_v+" "+h163+" "+h164_v;
```

Все остальное осталось неизменным.

11.14. Выравнивание абзаца

Данная функция выиграла четыре строки кода от применения выпадающих списков. Определяем только один параметр — есть ли у первой строки абзаца отступ или нет:

```
if(equa5_v!="no")
   equa5=" "+equa5_v;
```

После этого формируем строку классов

```
let equares=equa1_v+" "+equa2_v+" "+equa3_v+" "+
                                        equa4_v+equa5;
```

и отправляем все данные функции **wrap**:

```
wrap("p", equares, "equa_d");
```

11.15. Настройка шрифта

Эти настройки обрабатываются еще проще. Не надо выяснять наличие или отсутствие каких-то значений. Получаем данные

```
let font1_v=document.getElementById("font1").value;
let font2_v=document.getElementById("font2").value;
let font3_v=document.getElementById("font3c").value;
```

готовим список классов

```
let fontres=font1_v+" "+font2_v+" "+font3_v;
```

и отправляем информацию функции **wrap**:

```
wrap("span", fontres, "font_d");
```

На этом глава 11 подошла к концу.

Мы уже рассмотрели два варианта визуальных редакторов. Настало время перейти к самому, на мой взгляд, интересному.

12. Креативный редактор

Мы уже неоднократно, описывая те или иные компоненты проекта, делали оговорку, что в третьем редакторе есть заметные отличия от остальных. Вполне возможно, у читателей сложилось впечатление о третьем редакторе как об особенном, необычном и нестандартном. Это правильное впечатление. Недаром он получил название «креативный».

Все дело в том, что в этой части нашего проекта сайт и редактор представляют собой единое целое. Заметим, что, в отличие от базового сайта, адрес его новой версии (совмещенной с креативным редактором) — **index_ed3.php**. Сделано так для того, чтобы сделать третий редактор совершенно автономным.

Введите в браузере в строке адреса следующий запрос: **https://editjs.ru/index_ed3.php?a=index**. Откроется главная страница сайта. При этом ничто ни на самой странице, ни в ее коде не указывает, что к ней привязано еще какое-то программное обеспечение. Если в строку запроса добавить **&d=admin** (**https://editjs.ru/index_ed3.php?a=index&d=admin**), то на странице в ее левом верхнем углу появится поле ввода пароля. В демонстрационной версии он очень простой: **admin**. Введите его и нажмите клавишу «Enter». Редактор загрузится в страницу сайта (рис. 12*а*).

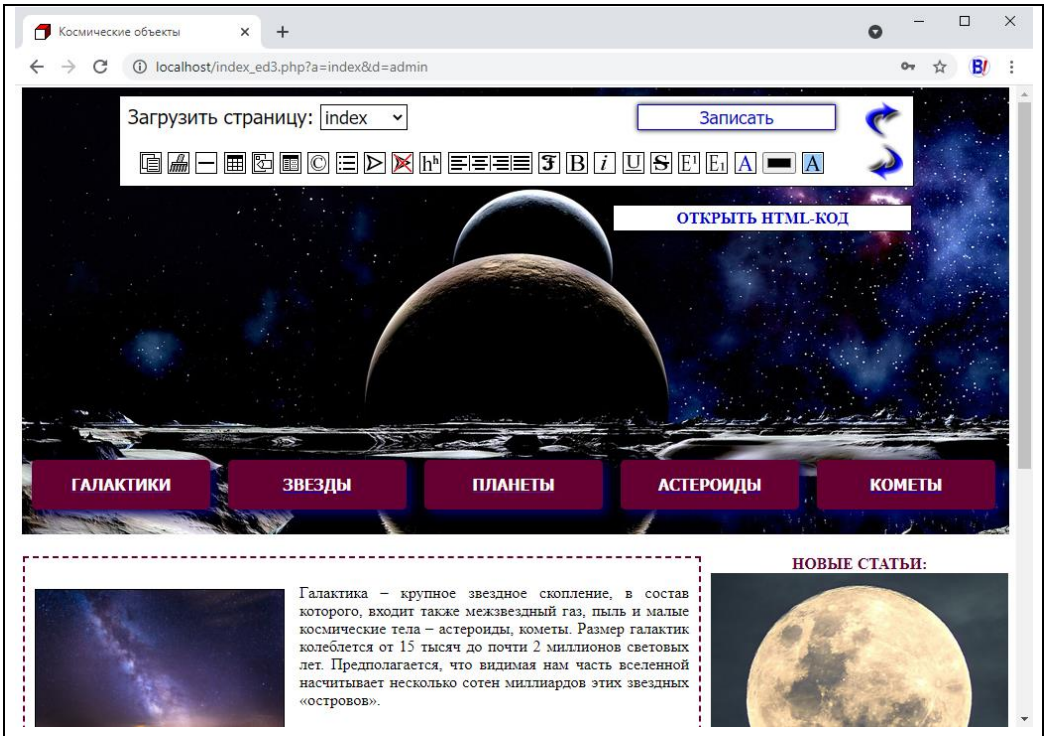

Рис. 12*а*. Вид третьего редактора

Тут вас ждет еще одно новшество: редакторская правка, создание текста, добавление элементов происходят непосредственно в области основного содержания страницы, которая загружена во фрейм просмотра. Ясно, что здесь мы обошлись без посредника в виде отдельного окна редактирования. Соответственно, из двух окон у нас осталось только одно — HTML-редактора, которое появляется в результате нажатия кнопки «ОТКРЫТЬ HTML-КОД». Чтобы закрыть вкладку, надо нажать ту же самую кнопку (только теперь у нее надпись «СКРЫТЬ HTML-КОД»).

Но и это еще не все. Новая «фишка»: главная панель со всеми вкладками и окном HTML-редактора свободно перемещается по вертикали. Нажмите, не отпуская, левую клавишу мыши на свободном участке панели и перемещайте ее вверх-вниз, насколько это надо. Отпустите клавишу — панель зафиксируется в новом месте. Такая функциональная возможность делает процесс редактирования очень удобным: всегда можно расположить панель рядом с какой-либо частью контента (рис. 12*б*).

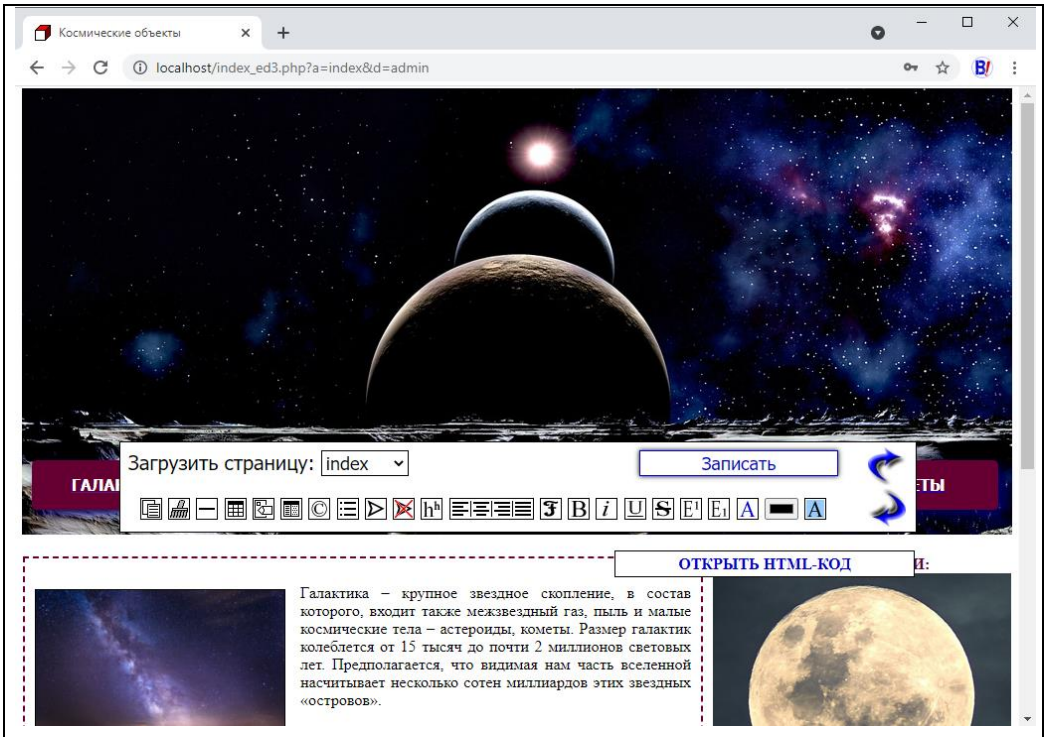

Рис. 12*б*. Перемещение главной панели

Теперь еще один важный момент. Навигация по страницам сайта осуществляется нажатием кнопок меню. Если вы находились в редакторе главной страницы и нажали кнопку меню «КОМЕТЫ», то загрузится эта страница в чистом виде. Чтобы перемещаться по страницам редактора, пользуйтесь выпадающим списком с перечнем файлов (как это делалось в

предыдущих редакторах). При этом чтобы зайти в следующую страницу редактора, нужно будет снова вводить пароль. Если это неудобно, предусмотрите какой-либо механизм запоминания администратора после однократного ввода пароля. Например, с использованием cookie или сессий.

Также необходимо указать, что в этой системе мы вновь вернемся к практике размещения стилей непосредственно в тегах элементов.

На мой вкус, креативный редактор отлично подходит для небольшого проекта с ограниченным количеством страниц и не нуждающегося в «услугах» полноценной CMS.

Перечень компонентов редактора:

– файл сайта — **index_ed3.php**;
– файл редактора — **editor3.php**;
– файл сценариев сайта — **set/index.js**;
– файл сценариев редактора — **set/editor3.js**;
– файл таблицы стилей сайта — **set/index.css**;
– файл таблицы стилей редактора — **set/editor3.css**.

Обратите внимание! Если вы будете делать сайт именно с таким редактором, то в этом случае необходимо переименовать файл **index_ed3.php** в **index.php**. В этом же файле в блоке кода

```
echo '<form method="POST"
      action="index_ed3.php?a='.$adr.'&d=admin"
      style="position: absolute; left: 20px;
      top: 20px; z-index: 10;">
      <input type="password" name="pr"></form>';
```

замените адрес

```
action="index_ed3.php?a='.$adr.'&d=admin"
```

на

```
action="index.php?a='.$adr.'&d=admin"
```

В этом же файле поменяйте в блоке меню адреса страниц с **index_ed3.php?a=** на **index.php?a=**.

В файле **editor3.js**, в модуле выбора загружаемой страницы, тоже необходимы коррективы: строку

```
window.location="index_ed3.php?a="+a+"&d=admin";
```

замените на

```
window.location="index.php?a="+a+"&d=admin";
```

Кроме того, сообщаю, что пароль вшит в код файла **index_ed3.php**. Поэтому откройте данный файл и в блоке

```
if(isset($_POST["pr"]))
{
$pr=$_POST["pr"];
if($pr=="admin")
   $ide=1;
}
```

замените пароль **admin** на придуманный вами (желательно достаточно сложный).

12.1. Файлы редактора и сайта

Начнем с того, что рассмотрим структуру нового редактора (рис. 12.1.1). Центральная часть системы похожа на то, что мы наблюдали у остальных редакторов: есть HTML-код разметки, помещенный в PHP-оболочку. Когда система запускается только как сайт, подключаются 3 файла. Таблица стилей и сценарии сайта импортируются непосредственно в HTML-код, а основное содержание внедряется в страницу через PHP-оболочку.

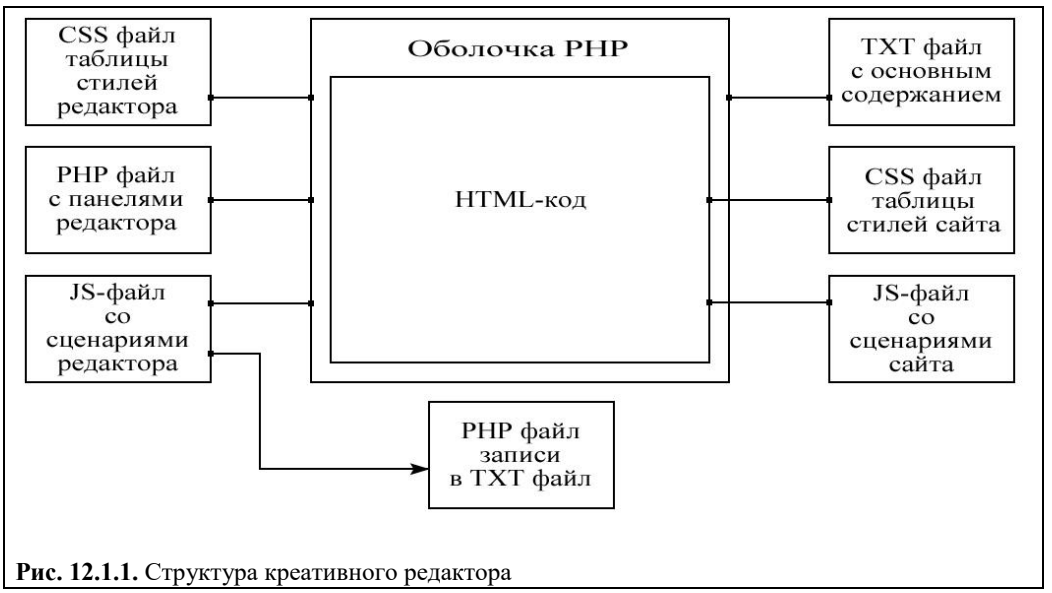

Рис. 12.1.1. Структура креативного редактора

Если перейти в режим редактора, то PHP-оболочка добавляет в код еще 3 компонента:

– PHP-файл с элементами редактора;
– таблицу стилей редактора;
– файл со сценариями редактора.

Как и в остальных случаях, файл со сценариями участвует не только в редактировании контента. Он также отправляет данные программе **rec.php** для их записи.

Функции PHP-оболочки здесь гораздо сложнее и многообразнее.

Если система запущена в качестве сайта, то выполняется только проверка ссылки на корректность (как и в оригинальной версии сайта). Но дополнительно объявляется переменная

```
$ide=0;
```

задача которой — служить идентификатором переключения в режим редактора. Когда значение этой переменной **0**, программа считает, что вы просматриваете только сайт.

Если добавить в конец строки запроса параметр **d** со значением **admin** (**&d=admin**) и нажать клавишу «Enter», то на странице появится поле ввода пароля. Как это происходит?

Вот код, создающий поле:

```
<?php
if($ide==0)
   {
   if(isset($_GET["d"]))
      {
      $dm=$_GET["d"];
      if($dm=="admin")
         echo '<form method="POST"
         action="index_ed3.php?a='.$adr.'&d=admin"
         style="position: absolute; left: 20px;
         top: 20px; z-index: 10;">
         <input type="password" name="pr"></form>';
      }
   }
?>
```

Поскольку значение идентификатора **$ide** пока еще равно **0**, вход не осуществлен и поле выводится на страницу (когда переменная **$ide** равна единице, вход состоялся и форму показывать не надо).

Разберем приведенный выше фрагмент. Если параметр **d** существует

```
if(isset($_GET["d"]))
```

получаем его значение

```
$dm=$_GET["d"];
```

и сравниваем с требуемым

```
if($dm=="admin")
```

В случае удачного сравнения печатаем форму с полем для ввода пароля:

```
echo '<form method="POST"
action="index_ed3.php?a='.$adr.'&d=admin"
style="position: absolute; left: 20px;
top: 20px; z-index: 10;">
<input type="password" name="pr"></form>';
```

После ввода пароля и нажатия клавиши «Enter» выполняем проверку, был ли введен пароль и правильный ли он:

```
if(isset($_POST["pr"]))
   {
   $pr=$_POST["pr"];
   if($pr=="admin")
      $ide=1;
   }
```

В случае успешной проверки меняем значение идентификатора **$ide** на **1**.

Дальше следует ряд процедур, в которых проверяется значение идентификатора и подключаются необходимые компоненты к сайту, чтобы преобразовать его в редактор.

В заголовочную часть документа вписываем meta-тег, запрещающий индексирование страниц редактора и переход по их ссылкам:

```
<?php
if($ide==1)
   echo '<meta name="robots"
                     content="noindex, nofollow">';
?>
```

Загружаем таблицу стилей и файл сценариев редактора:

```
<?php
if($ide==1)
   echo '<link rel="stylesheet"
                           href="set/editor3.css">';
?>
...
<?php
if($ide==1)
   echo '<script src="set/editor3.js"></script>';
?>
```

Наконец, последний этап — подключение файла редактора к телу документа:

```
<?php
if($ide==1)
   include 'editor3.php';
?>
```

Файл **editor3.php** содержит такой же набор компонентов, как и остальные редакторы, за исключением окна визуального представления контента.

Фактически, **editor3.php** — это содержимое контейнера ** ... ** из других редакторов. Правда, в креативном редакторе этот контейнер назван **basis3** (чтобы отличать его от контейнера **basis** сайта).

12.2. Включение режима редактирования

Еще один важный момент. При загрузке редактора необходимо добавить атрибут **contenteditable** слою **div** с основным содержанием. Делается это в файле **editor3.js** следующим образом:

```
ed.setAttribute("contenteditable", true);
```

12.3. Открытие и закрытие HTML-редактора

Как вы помните, обращение к различным окнам редакторов происходит достаточно часто. Поэтому мы неоднократно применяли сокращения в виде ссылок. Поступим аналогичным образом и в этот раз.

Создадим сокращенную запись для текстового редактора:

```
let te=document.getElementById("tex");
```

Добавим ссылку на стили текстового поля:

```
let tex=document.getElementById("tex").style;
```

Объявим переменную, которая будет выступать в роли идентификатора открытия текстового поля:

```
let cou=1;
```

В анонимной функции

```
document.getElementById("htm").
                addEventListener("click", function()
   {
   ...
   })
```

управляющей открытием и закрытием текстового редактора, два блока. Первый запускается, когда текстовое поле скрыто, идентификатор имеет значение **1** и мы нажали кнопку «ОТКРЫТЬ HTML-КОД»:

```
if(cou==1)
   {
   ...
   }
```

Начальная операция этого блока — передача HTML-кода из слоя **cont** в текстовую область:

```
te.value=ed.innerHTML;
```

Вторая операция — открытие текстового поля:

```
tex.visibility="visible";
```

Следующие две операции в точности повторяют аналогичные инструкции из предыдущих редакторов — делаем кнопку записи неактивной и полупрозрачной:

```
rec.style.opacity=0.2;
rec.setAttribute("disabled", true);
```

Теперь выводим на кнопку HTML-редактора новый текст

```
document.getElementById("htm").innerHTML=
                                    "СКРЫТЬ HTML-КОД";
```

и меняем значение идентификатора:

```
cou=2;
```

Чтобы скрыть вкладку HTML-кода, снова нажмем на кнопку «СКРЫТЬ HTML-КОД». После этого начнутся обратные процессы:
– данные из текстового поля поступят в контейнер основного содержимого страницы

```
ed.innerHTML=te.value;
```

– вкладка текстового редактора скроется

```
tex.visibility="hidden";
```

– кнопка записи снова вернется в рабочее состояние

```
rec.style.opacity=1;
rec.removeAttribute("disabled");
```

– на вкладке появится исходный текст

```
document.getElementById("htm").innerHTML=
                                    "ОТКРЫТЬ HTML-КОД";
```

– идентификатор опять получит значение 1

```
cou=1;
```

12.4. Взаимодействие HTML-редактора и области контента

Вообще, по очередности эти операции более ранние, нежели изложенные в разделе 12.3. Но описывать рабочие процессы удобнее в порядке, определенном в оглавлении.

Как и в предыдущем разделе, создадим сокращенную запись — теперь для области редактирования (она же область основного содержания в странице)

```
let ed=document.getElementById("cont");
```

и передадим исходные данные в текстовое поле:

```
te.value=ed.innerHTML;
```

Поскольку отдельной вкладки визуального редактирования у нас сейчас нет, то из третьего редактора выпала функция **tran** (**inter** осталась).

Напомню, что кроме обмена информацией при загрузке системы данные также передаются между областью контента и текстовым полем при открытии/закрытии HTML-редактора (как мы это видели в предыдущем разделе).

12.5. Копирование текста

Функция копирования текста третьего редактора аналогична той, что мы видели во втором редакторе. Нет абсолютно никаких различий.

12.6. Удаление форматирования и ссылок

У этого модуля единственное отличие от того, что мы видели во втором редакторе, — отсутствие вызова функции **tran** за неимением последней.

12.7. Простое редактирование текста

Здесь те же самые массивы **mid** и **mco** с идентификаторами кнопок простого редактирования и списком тегов для оформления текста. Точно так же в цикле регистрируется обработчик нажатия кнопок. В функции **emb** опять единственное отличие — отсутствие вызова функции **tran**. Думаю, что дальше не имеет смысла без конца указывать на эту разницу. Просто помните о ней.

12.8. Изменение цвета букв или фона текста

Этот блок кода имеет некоторые фрагменты, общие как с первым редактором, так и со вторым.

Обработчик регистрируется так же, как и в первом редакторе, с теми же аргументами:

```
document.getElementById("col").
       addEventListener("click", clr.bind(null, ""));
document.getElementById("backgr").
                    addEventListener("click",
                       clr.bind(null, "background-"));
```

Установка фокуса

```
ed.focus();
```

и создание элемента **span**

```
let elm=document.createElement("span");
```

идентичны аналогичному коду второго редактора.

12.9. Добавление разметки маркированного списка

Код добавления маркированного списка в третьем редакторе аналогичен тому, что мы видели во втором.

12.10. Вставка линии

Код вставки линии до мельчайших подробностей совпадает с тем, что написан для аналогичного блока редактора № 1.

Функция **place** имеет отличие от такой же функции редактора № 2 всего в одном пункте — вместо набора классов добавляется набор свойств стиля:

```
elm.style=z;
```

12.11. Вставка фрейма

Модуль добавления фрейма в третьем редакторе — один в один такой же модуль из первого редактора.

12.12. Вставка рисунка

Блок вставки изображения в третьем редакторе — точно такой же, как и в первом редакторе.

12.13. Вставка таблицы

Функция вставки таблицы третьего редактора отличается от этой же функции первого тремя компонентами:

– наличием фокусировки области основного содержания

```
ed.focus();
```

– созданием таблицы в текущем документе

```
let elm=document.createElement("table");
```

– отсутствием вызова функции **tran**.

12.14. Вставка символов

Регистрация обработчиков для кнопок с символами во всех уже рассмотренных редакторах одинаковая.

Функция вставки символов третьего редактора имеет отличия от функции **sign** первого редактора по тем же трем пунктам, что были упомянуты в разделе 12.13:

– наличию фокусировки области основного содержания;
– созданию текстового узла в текущем документе;
– отсутствию вызова функции **tran**.

12.15. Добавление ссылки

Опять те же три отличия от соответствующего блока кода первого редактора. По всем остальным пунктам — полное совпадение.

12.16. Добавление заголовка

Полная аналогия с анонимной функцией добавления заголовка редактора № 1. В функции **wrap** — три уже ставших привычными отличия.

12.17. Выравнивание абзаца

Полная идентичность с такой же функцией первого редактора.

12.18. Настройка шрифта

Опять все то же самое, что и в первом редакторе.

12.19. Перемещение панели

Модуль перемещения главной панели в файле **editor3.js** расположен в самом начале. Но поскольку его роль второстепенна, мы рассматриваем эту часть кода в последнюю очередь.

Для выполнения необходимых манипуляций зарегистрируем сначала два обработчика:

– нажатия кнопки мыши на контейнере с панелью

```
addEventListener("load", function()
   {
   document.getElementById("basis3").
                  addEventListener("mousedown", coor);
   });
```

– отпускания кнопки мыши в любой точке окна браузера

```
addEventListener("mouseup", stco);
```

Введем переменную, предназначенную для хранения значения разности между точкой клика на панели и вертикальной координатой верхнего края панели (тем самым мы определяем, на каком расстоянии от указателя все время должен находиться верхний край панели в процессе движения):

```
let dv=0;
```

Этот параметр необходим, чтобы при перемещении панели она двигалась равномерно, а указатель мыши все время находился в одной и той же точке панели.

Напишем функцию **coor**, которая запускается после нажатия кнопки мыши на панели:

```
function coor()
   {
   ...
   }
```

Первая операция — определение вертикальной координаты указателя:

```
let v=event.pageY;
```

Следом узнаем координаты верхнего левого угла панели:

```
let sv=document.getElementById("basis3").offsetTop;
```

Вычисляем разницу (рис. 12.19.1)

```
dv=v-sv;
```

и регистрируем обработчик для события перемещения мыши по странице:

```
addEventListener("mousemove", neco);
```

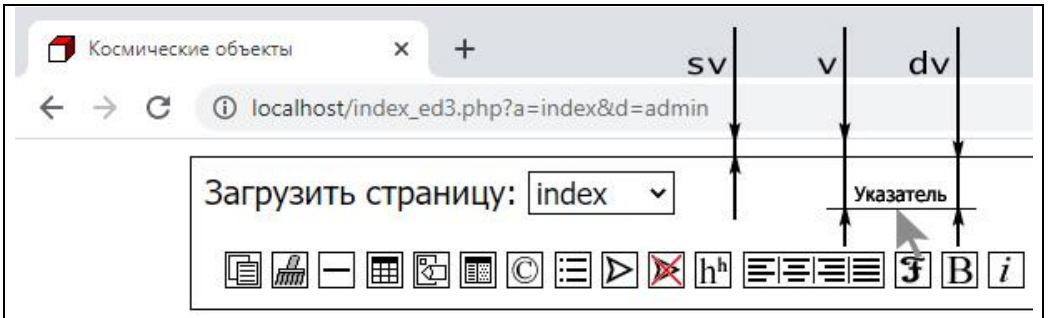

Рис. 12.19.1. Вычисление начального смещения панели относительно верхней границы окна браузера

Как только мы начинаем перемещать панель, запускается функция **neco**:

```
function neco()
   {
   ...
   }
```

В ней сначала вычисляется новая вертикальная координата указателя

```
let nv=event.pageY;
```

потом из значения новой координаты **nv** вычитается значение переменной **dv** с уже вычисленным начальным смещением

```
let fv=nv-dv;
```

и тем самым мы получаем новое значение координаты, в которой должен оказаться верхний край панели. Перемещаем панель в эту точку:

```
document.getElementById("basis3").style.top=fv+"px";
```

Повторные вычисления текущих координат и смещение панели в новое положение продолжаются до тех пор, пока мы перемещаем мышь и пока нажата ее левая кнопка. Как только мы отпускаем кнопку, «срабатывает» функция, в которой всего одна инструкция — удалить обработчик события перемещения мыши:

```
function stco()
{
removeEventListener("mousemove", neco);
}
```

Движение панели прекратится, и она займет новое положение, то, которое наиболее удобно администратору при редактировании контента.

Отступление от темы

Автор видел много визуальных редакторов, но с перемещающейся панелью — ни разу. Если такие все же существуют, значит, мне не удалось охватить своим вниманием все разнообразие WYSIWYG-систем. Если же визуальных редакторов с перемещающейся панелью раньше не было, то мне будет приятно внести скромную лепту новизны в развитие CMS.

13. Ретро-редактор

Вопрос: если первые три редактора, как мы убедились, вполне успешно выполняют свои функции, стоит ли изучать редактор, построенный на использовании ретро-метода **execCommand**? Мой ответ — стоит.

Хочу напомнить, что такая система имеет ряд преимуществ. Повторим их:

– метод продолжает работать в требуемых нам объемах во всех браузерах;

– **execCommand** до сих пор используется во многих визуальных редакторах (а значит, не одни мы доверяем этому методу);

– благодаря тому, что большинство команд метода унифицированы, редакторы на его основе занимают меньше всего места в памяти компьютера;

– написать редактор на основе **execCommand** гораздо легче — в такой системе будут более лаконичные и простые функции;

– нет полной уверенности, что на этом методе окончательно поставлено клеймо устаревшего.

Есть и другие преимущества. Например, в редакторах № 1–3, добавляя элемент — таблицу или рисунок, мы в обязательном порядке сначала устанавливали фокус окна визуального представления. Помните, почему? Потому что иначе, забыв установить курсор в окне, мы тем самым помещаем элемент не в окно, а в панель настроек, так как в этом случае в момент нажатия кнопки «Вставить» именно панель будет находиться в фокусе. В ретро-редакторе устанавливать фокус не надо. Забыли вставить курсор в окно визуального представления — элемент просто не будет создан.

Или еще один момент. В разделе 8.14 мы говорили, что для удаления ссылки или форматирования необходимо выделить в тексте справа и слева на один символ больше. В ретро-редакторе таких сложностей нет — надо выделять только необходимый текст без дополнительных символов по краям.

Наконец, редактор с **execCommand**, в отличие от предыдущих трех, при удалении форматирования сохраняет в тексте все HTML-теги, которые не имеют отношения к форматированию.

Так что мой совет — попробовать, поэкспериментировать.

Перечень компонентов редактора:

– основной файл — **editor4.php**;

– файл сценариев на JavaScript — **set/editor4.js**;

– файл таблицы стилей — **set/editor4.css**.

Демонстрационная страница редактора: **https://editjs.ru/editor4.php?a=index**. Зайдя на нее, вы увидите, что мы опять вернулись к традиционному построению редактора в виде отдельной страницы.

13.1. Установка режима редактирования

Режим редактирования в данной системе устанавливается так же, как это делалось во втором редакторе:

```
<div id="edit" contenteditable="true"></div>
```

13.2. Простое редактирование текста

Смена метода форматирования текста и вставки элементов привела к тому, что в этом блоке появились заметные изменения.

Теперь функция простого редактирования обслуживает следующие кнопки:

–  копирование фрагмента;

–  удаление форматирования;

–  преобразование в маркированный список;

–  удаление ссылки.

По-прежнему относятся к блоку простого редактирования, как и в других редакторах, шесть кнопок: выделение жирным, курсивом, подчеркиванием, зачеркиванием, а также преобразование текста в надстрочный и подстрочный.

В результате расширился список элементов массива **mid** — их 10:

```
let mid=["cop", "forma", "ulli", "delin", "bol",
              "ital", "under", "strik", "sup", "sub"];
```

Массив **mco** не только увеличился. Теперь его элементы хранят совершенно иные значения — команды метода **execCommand**, необходимые для выполнения копирования и преобразования текста:

```
let mco=["Copy", "RemoveFormat",
"InsertUnorderedList", "Unlink", "Bold",
        "Italic", "Underline", "strikeThrough",
                          "superscript", "subscript"];
```

Ниже приведена таблица 13.2.1 с расшифровкой команд:

Таблица 13.2.1

Метод **execCommand**

Команда	**Описание**
Copy	Выполняет копирование выделенного фрагмента, включая текст, его форматирование и различные элементы
RemoveFormat	Удаляет форматирование из выделенного текста. Не затрагивает находящиеся в тексте элементы
InsertUnorderedList	Преобразует выделенный фрагмент в маркированный список
Unlink	Удаляет выделенную ссылку, оставляя образующий ее текст
Bold	Добавляет/удаляет выделение жирным
Italic	Добавляет/удаляет выделение курсивом
Underline	Добавляет/удаляет выделение подчеркиванием
strikeThrough	Добавляет/удаляет выделение зачеркиванием

Продолжение табл.

Команда	Описание
superscript	Добавляет/удаляет верхний индекс у выделенного фрагмента
subscript	Добавляет/удаляет нижний индекс у выделенного фрагмента

У последних шести команд есть свои особенности. Рассмотрим их на конкретном примере. Если выделить слово, набранное обычным шрифтом, и нажать кнопку «Bold», то буквы станут «жирными». Если выделить «жирное» слово и нажать кнопку «Bold», то начертание букв вернется к обычному виду. Остальные пять кнопок из упомянутой шестерки команд действуют по аналогичной схеме. На взгляд автора, это очень удобно при оформлении текста.

Регистрация обработчика для перечисленных выше кнопок происходит так же, как и в остальных редакторах.

Функция **emb** сохранила проверку выделения

```
if(sel=="")
   alert("Вы не выделили текст!");
```

но при этом стала короче. Теперь в ней всего одна команда форматирования текста:

```
document.execCommand(v);
```

Обратите внимание! Поскольку окно редактирования у нас является компонентом текущего документа, то и метод работает с объектом **document**.

Завершается процесс стандартным вызовом двух функций:

```
tran();
inter();
```

Еще один важный момент. По сравнению с редакторами № 2 и 3 в этой функции отсутствует фокусировка. Более того, ее нет ни в одной другой функции четвертого редактора. Метод **execCommand** не требует концентрации внимания на окне редактирования. Если вы забыли выделить текст, то его форматирование не произойдет. Если вы не вставили курсор, то элемент — таблица, рисунок, линия — не будет создан вообще. **execCommand** работает только при наличии выделения (или установленного курсора) в окнах, у которых есть атрибут **contentEditable** или включено свойство **designMode**.

Теперь обратим внимание на самую первую команду **Copy**. С ее помощью в выделенном фрагменте копируется текст и различные элементы. Но тот же самый результат можно получить, выделив фрагмент, щелкнув на нем правой кнопкой мыши и выбрав из контекстного меню пункт «Копировать». А это значит, что совершенно безболезненно можно отказаться от команды **Copy** и сохранить возможность копировать «чистый» текст без HTML-

элементов (как в других редакторах). В этом случае в массивах **mid** и **mco** нужно удалить элементы с индексом **0**, а кнопке  назначить точно такой же обработчик, как, например, во втором редакторе.

13.3. Изменение цвета букв или фона текста

Регистрация обработчика нажатия кнопок цветового оформления такая же, как в редакторах 1 и 3.

Запуская функцию **clr**, в обязательном порядке проверяем выделение. В остальном данная функция мало напоминает то, что мы видели раньше.

Здесь мы используем команду **insertHTML** метода **execCommand**. Таким способом можно вставить блок HTML-кода, отдельный элемент или простой текст. Для выполнения данной команды методу требуется передать три аргумента: имя команды; значение, указывающее, отображать или нет пользовательский интерфейс; HTML-код или текст. Второй аргумент обычно указывают **false**. Это означает, что третий аргумент передается в метод непосредственно из программы.

На первом этапе определимся с третьим аргументом. Он представляет собой выделенный текст, обрамленный открывающим и закрывающим тегами **span**. Стилевое оформление «внутренностей» тега мы получаем из панели выбора цвета:

```
document.getElementById("ccc").value
```

Полный код, формирующий третий аргумент:

```
let cl='<span style="'+b+'color: '+
   document.getElementById("ccc").value+';">'
                                      +sel+'</span>';
```

Напоминание. Если аргумент **b**, переданный в функцию **clr**, пустой, то цветом выделяется текст. Если значение этого аргумента **background-**, то меняется цвет фона текста. **sel** — объект, содержащий выделенный текст.

На втором шаге создаем цветовое оформление:

```
document.execCommand("insertHTML", false, cl);
```

Завершается процесс вызовом функций **tran** и **inter**.

13.4. Вставка линии

До 17 строки функция вставки линии похожа на то, что мы видели, например, в первом редакторе. Но дальше начинаются отличия. В файле **editor4.js** при создании какого-либо элемента используется команда **insertHTML**. Поэтому сейчас наша задача — сформировать не просто набор

свойств элемента, а его полный код. Делаем это в два приема. Сначала формируем стили

```
let shr=' style="'+hr1+hr2+'background: '+hr3_v+
                                        '; border: 0;"';
```

а потом компонуем линию:

```
let hrres='<hr'+shr+'>';
```

Теперь можно отправлять данные функции **place**:

```
place(hrres, "hr_d");
```

В новой версии она принимает только два аргумента: HTML-код создаваемого элемента и **id** закрываемой вкладки:

```
function place(x, y)
   {
   ...
   }
```

«Начинка» функции **place** состоит из четырех инструкций. Первая создает в окне редактора новый элемент:

```
document.execCommand("insertHTML", false, x);
```

Назначение остальных вам уже известно:

```
tran();
inter();
clo(y);
```

13.5. Вставка фрейма

Функция отличается от аналогичных тремя последними строками:
– составляем набор свойств фрейма

```
let sifr=' style="'+ifr2+ifr3+ifr4+ifr5+ifr7+'"';
```

– «собираем» его HTML-код

```
let ifres='<iframe src="'+ifr1_v+'"'+sifr+'>
                                        </iframe>';
```

– поручаем функции **place** создание фрейма

```
place(ifres, "ifr_d");
```

13.6. Вставка рисунка

Такая же картина вырисовывается при создании изображения:
– составляем набор свойств картинки

```
let sima='style="'+ima4+ima5+'width: '+ima6+
                      '; border: '+ima7+' solid '
                       +ima8_v+'; cursor: pointer;"';
```

– пишем ее HTML-код

```
let imares='<img src="'+ima+'" '+sima+' alt="'
                    +ima3_v+'" title="'+ima3_v+'">';
```

– вызываем функцию **place** для вставки рисунка

```
place(imares, "img_d");
```

13.7. Вставка таблицы

После выяснения, какие настройки введены для создания таблицы, начинаем создавать первую часть ее HTML-кода:

```
let tb='<table style="'+tab1+
'border-collapse: collapse; background: '+
tab11_v+'; '+tab5+tab7+'font-family: '+tab6_v+
                    '; color: '+tab8_v+';'+tab12+'">';
```

Затем наполняем таблицу ячейками:

```
let sum="";
for(let i=0; i<tab2_v; i++)
   {
   sum+='<tr>';

   for(let t=0; t<tab3_v; t++)
      sum+='<td style="border: '+tab9+
           ' solid '+tab10_v+';'+tab4+'"> </td>';

   sum+='</tr>';
   }
```

Завершаем формирование кода:

```
let tabres=tb+sum+'</table>';
```

В четвертом редакторе, в отличие от предыдущих, создает таблицу тоже функция **place**:

```
place(tabres, "tabl_d");
```

13.8. Вставка символов

И этим процессом тоже занимается функция **place**. Соответственно, обработчик нажатия кнопок символов регистрируется в цикле следующим образом:

```
document.getElementById(idc).
                addEventListener("click",
                   place.bind(null, smb, "copyr_d"));
```

13.9. Добавление ссылки

Как и в остальных редакторах, функция добавления ссылки выполняет все операции самостоятельно.

Проверяем, сделано ли выделение в тексте. Получаем данные из панели настроек. Создаем переменную

```
let ares="";
```

которая предназначена для сбора результирующего кода ссылки. Формируем стиль в зависимости от того, какого типа ссылку мы хотим видеть — с подчеркиванием или без:

```
if(link5_v=="no")
   link5=' style="text-decoration: none; color: '+
                                          link6_v+';"';
else
   link5=' style="color: '+link6_v+';"';
```

Если ссылка ведет на сторонний сайт, то выясняем — открывать ее в новом окне или текущем:

```
if(link2_v!="")
   {
   let tar="";

   if(link3_v=="yes")
      tar=' target="_blank"';

   ...
   }
```

После чего формируем код внешней ссылки:

```
if(link2_v!="")
   {
   ...
   ares='<a href="'+link1_v+link2_v+'"'+tar+link5+
                                       '>'+sel+'</a>';
   }
```

Если ссылка ведет на внутреннюю страницу нашего сайта, то выполняется вторая часть условия и компоновка происходит следующим образом:

```
else
   ares='<a href="index.php?a='+link4_v+'"'+link5+
                                        '>'+sel+'</a>';
```

Предпоследний этап — добавление ссылки к тексту:

```
document.execCommand("insertHTML", false, ares);
```

«На закуску» — вызов трех функций:

```
tran();
inter();
clo("link_d");
```

13.10. Добавление заголовка

Как это ни удивительно для четвертого редактора, но код функции формирования заголовка полностью совпадает с аналогичной функцией первого редактора. А вот функция **wrap**, добавляющая заголовки, выравнивания и шрифты, значительно отличается от таких же функций из других редакторов. Хотя аргументы те же: первый — имя создаваемого тега, второй — настройки стиля, третий — идентификатор закрываемой вкладки.

Функция **wrap** формирует строку HTML-кода форматирования:

```
let res='<'+r+' style="'+u+'">'+sel+'</'+r+'>';
```

Как видите, в этой строке находится и текст из выделения **sel**.

Добавляя теги к выделенному тексту, мы на самом деле меняем выделенный фрагмент на строку HTML-кода, полученную на предыдущем шаге:

```
document.execCommand("insertHTML", false, res);
```

Таким образом, исходный фрагмент сохраняется, но по его краям появляются теги того или иного типа.

13.11. Выравнивание абзаца

Полное совпадение с аналогичной функцией из первого редактора наблюдается и в анонимной функции выравнивания абзаца.

13.12. Настройка шрифта

Все, что сказано в предыдущем случае, можно отнести и к функции настройки шрифта: она совпадает с такой же функцией из файла **editor1.js**.

14. Файл записи данных

Для записи измененного или только что созданного контента все редакторы обращаются к одной и той же программе, написанной на PHP, — **rec.php** (от слова record — запись).

Как вы помните, в демонстрационном ресурсе такая процедура запрещена, поэтому там файл **rec.php** выполняет роль заглушки. Его задача — имитировать успешную запись данных и возвратить сообщение об этом. Файл для реальной записи на демонстрационной платформе отсутствует.

Иная ситуация с zip-архивом. В нем есть оба файла: заглушка **rec.php** и «настоящий» — **rec-real.php**.

После того, как скопируйте содержимое zip-архива и поместите его в папку htdocs сервера Apache (эта процедура более подробно описана в разделе 5.1), можно на время экспериментов оставить файл-заглушку, а можно сразу использовать действующий скрипт. Опять же напомню: чтобы привести редакторы в рабочее состояние, файл-заглушку необходимо удалить, а **rec-real.php** переименовать в **rec.php**.

14.1. Заглушка

Этот файл максимально прост:

```
<?php
echo "1";
```

Редактор отправляет файлу адрес страницы и контент. Эти данные просто игнорируются. В качестве ответа программа возвращает число 1, что JavaScript-кодом редактора воспринимается как успешная запись, о чем и выводится сообщение на кнопке отправки данных.

14.2. Рабочий файл

Первым делом хочу подробнее обсудить уже сказанное в разделе 2.8 по теме безопасности.

Файл **rec-real.php** выполняет следующие проверки:

– наличия адреса страницы, переданного методом POST (страницы, предназначенной для записи данных);

– соответствия переданного адреса имени файла одной из существующих страниц;

– наличия параметра с данными для записи.

В то же время не предусмотрена проверка, откуда поступают данные: от авторизованного источника или из формы, созданной недоброжелателем. Поэтому в данный файл нужно будет добавить код проверки, который зависит от способа аутентификации и авторизации администратора. Выбор способа — за вами.

А теперь приступим к делу. Как уже сказано, работа программы начинается с выяснения, передан ли параметр с адресом страницы:

```
if(isset($_POST["str"]))
   $str=$_POST["str"];
else
   {
   echo "НЕКОРРЕКТНЫЙ ЗАПРОС !";
   return;
   }
```

Если параметр существует, присваиваем его значение переменной **$str**:

```
$str=$_POST["str"];
```

Если такой параметр отсутствует или передан методом GET, выводим в браузер текстовое предупреждение и прерываем выполнение программы:

```
echo "НЕКОРРЕКТНЫЙ ЗАПРОС !";
return;
```

Допустим, мы получили необходимый параметр. Следующая проверка — соответствует ли содержащийся в нем адрес одной из существующих в папке **content** страниц:

```
$ch=0;
$opdir=opendir("content");
while($redir=readdir($opdir))
   {
   if($str.".txt"==$redir)
      {
      $ch=1;
      break;
      }
   }
closedir($opdir);
```

Переменная **$ch** служит у нас идентификатором результатов проверки.

Запускаем цикл **while**, в котором проходим по всем файлам из папки **content**

```
$opdir=opendir("content");
while($redir=readdir($opdir))
   {
   ...
   }
```

и сравниваем адрес, переданный в запросе, с именами файлов из этой папки. Если произошло совпадение, присваиваем идентификатору **$ch** новое значение и прерываем цикл:

```
if($str.".txt"==$redir)
```

```
   {
   $ch=1;
   break;
   }
```

Совпадение не обнаружено? Значит, был передан несуществующий адрес. Проверяем идентификатор:

```
if($ch==0)
   {
   echo "НЕКОРРЕКТНЫЙ АДРЕС !";
   return;
   }
```

Если результаты проверки положительные, переменная **$ch** получила значение 1, условие

```
if($ch==0)
```

ложно и программа переходит к следующему шагу. Если результаты проверки отрицательные, условие истинно. В этом случае выводим предупреждение и прерываем выполнение программы:

```
echo "НЕКОРРЕКТНЫЙ АДРЕС !";
return;
```

Когда указанные выше барьеры преодолены, можно сформировать адрес файла, в который будет произведена запись:

```
$adr="content/".$str.".txt";
```

У администратора сайта может возникнуть необходимость временно удалить весь контент с одной из страниц. В этом случае программе записи будет отправлен пустой параметр, что вызовет ошибку при записи в файл. Избежать ошибки можно, если, например, заменить «пустоту» на пробел. В этом случае программа посчитает, что данные из параметра переданы корректно, и запишет в файл пробел. Но визуально страница останется пустой.

Исходя из этих соображений, построена проверка параметра с данными контента:

```
$tex=" ";
if(isset($_POST["tex"]))
   {
   if($_POST["tex"]!="")
      $tex=$_POST["tex"];
   }
else
   {
   echo "НЕКОРРЕКТНЫЙ ЗАПРОС !";
   return;
   }
```

Создаем переменную **$tex** и помещаем в нее пробел:

```
$tex=" ";
```

Выясняем, передан ли параметр с данными для записи:

```
if(isset($_POST["tex"]))
```

Если нет, то выводим предупреждение и останавливаем выполнение программы:

```
echo "НЕКОРРЕКТНЫЙ ЗАПРОС !";
return;
```

Если да, то проверяем, не пусто ли в данных:

```
if($_POST["tex"]!="")
```

Когда данные присутствуют, помещаем их в переменную **$tex**:

```
$tex=$_POST["tex"];
```

Данные отсутствуют? Значит, переменная **$tex** остается с пробелом.

Последний этап — запись в файл:

```
if(file_put_contents($adr, $tex))
   echo "1";
else
   echo "2";
```

При успешной записи программа возвращает число 1 и в редакторе на кнопке отправки данных выводится сообщение «ЗАПИСАНО». В случае неудачи возвращается число 2 и появляется сообщение «НЕ ЗАПИСАНО».

15. Подведем итоги

Итак, главные результаты нашей работы:

– мы написали несколько вариантов визуальных редакторов, каждый из которых может найти свое применение в составе системы управления сайтом или быть подключенным к сайту напрямую;

– наша продукция тщательно проверена и представляет собой набор файлов, написанных в строгом соответствии с синтаксисами примененных языков программирования и в полном согласии с требованиями Консорциума Всемирной Паутины.

В одной из своих книг по программированию на JavaScript я написал буквально следующее:

«Очень придирчиво относитесь к выбору популярных платформ, движков и CMS для создания сайтов. Да, такие системы очень облегчают труд дизайнера и верстальщика. Но почти любая из них содержит в своем „ядре“ многочисленные ошибки, которые потом перекочевывают в ваши разработки. Если вам необходимо выполнить крупный проект — смиритесь с неизбежным: без платформы, движка или CMS, скорее всего, не обойтись. Если проект небольшой и не нуждается в системе управления контентом, попробуйте самостоятельно написать его „от корки до корки“. Так вы можете создать ресурс с абсолютно чистым кодом. Есть еще один вариант действий: выберите CMS, которая вам нравится больше всего, скачайте дистрибутив на свой компьютер и попробуйте исправить его исходный код. Работа большая и трудоемкая, зато у вас появится исправленный экземпляр CMS, на копиях которого вы в дальнейшем станете делать „чистые“ сайты».

Теперь перед вами открывается третий путь: вы можете написать всю CMS или хотя бы визуальный редактор самостоятельно. И при этом создать чистый продукт, который не содержит ошибок и нарушений стандартов, существующих в web-индустрии. Автор надеется, что его книга будет хорошим помощником в таком деле. Если вы изберете этот путь, мне только останется пожелать вам удачи.

Валерий Викторович ЯНЦЕВ

JAVASCRIPT

ВИЗУАЛЬНЫЕ РЕДАКТОРЫ

Учебное пособие

Зав. редакцией
литературы по информационным технологиям
и системам связи *О. Е. Гайнутдинова*
Ответственный редактор *Н. А. Кривилёва*
Корректор *М. А. Лауконен*
Выпускающий *Е. А. Романова*

ЛР № 065466 от 21.10.97
Гигиенический сертификат 78.01.10.953.П.1028
от 14.04.2016 г., выдан ЦГСЭН в СПб

Издательство «ЛАНЬ»
lan@lanbook.ru; www.lanbook.com
196105, Санкт-Петербург, пр. Юрия Гагарина, д.1, лит. А.
Тел.: (812) 336-25-09, 412-92-72.
Бесплатный звонок по России: 8-800-700-40-71

Подписано в печать 11.01.22.
Бумага офсетная. Гарнитура Школьная. Формат 70×100 $^{1}/_{16}$.
Печать офсетная/цифровая. Усл. п. л. 13,65. Тираж 30 экз.

Заказ № 083-22.

Отпечатано в полном соответствии
с качеством предоставленного оригинал-макета
в АО «Т8 Издательские Технологии»
109316, г. Москва, Волгоградский пр., д. 42, к. 5.

В. В. ЯНЦЕВ

JAVASCRIPT ВИЗУАЛЬНЫЕ РЕДАКТОРЫ

Учебное пособие

ЛАНЬ

· САНКТ-ПЕТЕРБУРГ · МОСКВА · КРАСНОДАР ·
· 2022 ·

В. В. ЯНЦЕВ

JAVASCRIPT ВИЗУАЛЬНЫЕ РЕДАКТОРЫ

Учебное пособие

ЛАНЬ

· САНКТ-ПЕТЕРБУРГ · МОСКВА · КРАСНОДАР ·
· 2022 ·