ЛАБОРАТОРНАЯ РАБОТА №2

**По курсу «Интернет-программирование»**

Основы создания динамических интернет-страниц

с помощью HTML, CSS и JavaScript

**Цель работы.**

Ознакомиться с принципами создания динамических HTML документов с использованием CSS и JavaScript. Практически освоить написание и отладку сценариев JavaScript.

**Приобретаемые компетенции:**

1. Понимание основных принципов работы динамических HTML документов с использованием CSS и JavaScript.
2. Настройка рабочего пространства Интернет-разработчика при отладке CSS и JavaScript.
3. Работа с элементами и событиями HTML документа с помощью JavaScript.

# **Объектная модель документа**

Исходный код HTML документа может выглядеть сравнительно просто см. Листинг 1.

Листинг 1. Простой HTML документ

<!DOCTYPE html>  
<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />

<title>МИЭТ</title>

</head>

<body>

<center>Здравствуй мир!</center>

</body>

</html>

Все содержимое между тегами **html** делится на два блока заголовок: **<head>** и тело документа **<body>**.

Все, что описано внутри тега **<head>** загружается до начала отображения содержимого документа.

Все, что описано внутри тега **<body>** может загружаться в момент, когда уже началось отображение содержимого документа. Например, при медленных каналах связи можно видеть, как картинки сначала загружаются в виде пустых прямоугольников и только через некоторое время эти прямоугольники заполняютмся изображениями\*.

\* - важно представлять порядок загрузки содержимого HTML документа т.к. обращение к незагруженным ресурсам может вызвать как отображаемые, так и не отображаемые ошибки. К неотображаемым ошибкам относятся ошибки, которые не видны без дополнительных действий (например, вызова отладчика)

При более детальном даже такой простой документ, как в Листинг 1 (впрочем как и любой другой), загружается в виде объектной модели DOM (Document Object Model), содержащей множество объектов см. рис. 1.

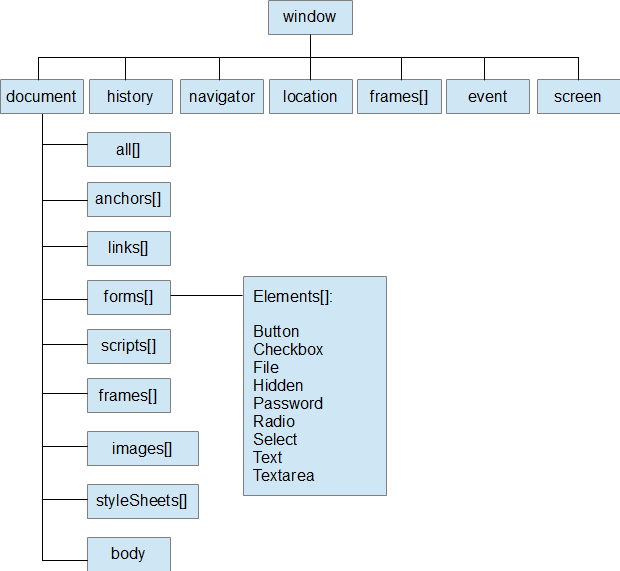


Рис.1. Объектная модель документа

На верхнем уровне DOM располагается объект **window**, содержащий объекты окна броузера в котором отображается документ. Внутри объекта **body** располагаются уже все объекты-узлы документа в иерархическом виде, определяемом вложенностью тегов - см. рис. 2.

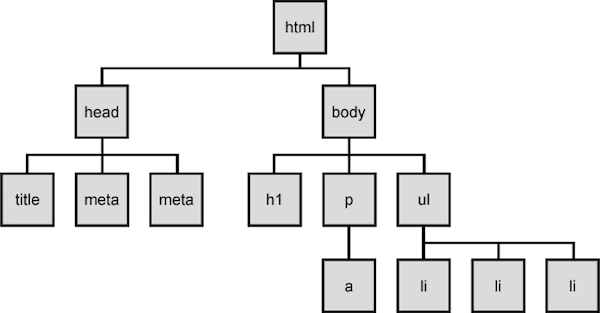


Рис.2. Пример вложенности тегов в объекте body

Описание объектов **window** приведено в таблице 1.

**Таблица 1. Объекты уровня window**

|  |  |
| --- | --- |
| **Объект** | **Описание** |
| document | Содержит DOM страницы и специальные массивы элементов страницы:   * anhors – все внутренние ссылки-якоря * links – все внешние ссылки * forms.elements – все поля форм * frames – все вложенные фреймы типа iframe * images – все картинки * scripts – все JavaScript скрипты * styleSheets – все CSS стили * body – объекты документа в иерарахии вложенности |
| navigator | Содержит информацию о браузере — название браузера и номер его версии. |
| frames | Для фреймсодержащего документа позволяет ссылаться на окна набора фреймов. |
| history | Позволяет перемещаться по страницам из истории посещений, в рамках одной закладки броузера, но не позволяет читать URL истории, оставаясь на текущей страницы, из соображений безопасности. |
| location | Содержит адрес URL страницы, загруженной в окно браузера. Изменяя свойства этого объекта, можно загрузить в окно браузера новую страницу |
| event | Содержит информацию о возникающих событиях |
| screen | Содержит информацию о текущем диплее и окне броузера: размер, разрешение, цветность и т.д. |

## Работа с DOM c помощью отладчика браузера

Для исследования DOM удобно использовать специальный режим работы броузеров, предназначенных для разхработчика. В броузере Chrome его можно вызвать компбинацией «Ctrl+Shift+I» или кликнув правой клавишей мыши на любом элементе документа и выбрав в появившемся меню пункт «Посмотреть код». В результате окроется фрейм с инструментами исследования и отладки страницы — см. рис. 3.

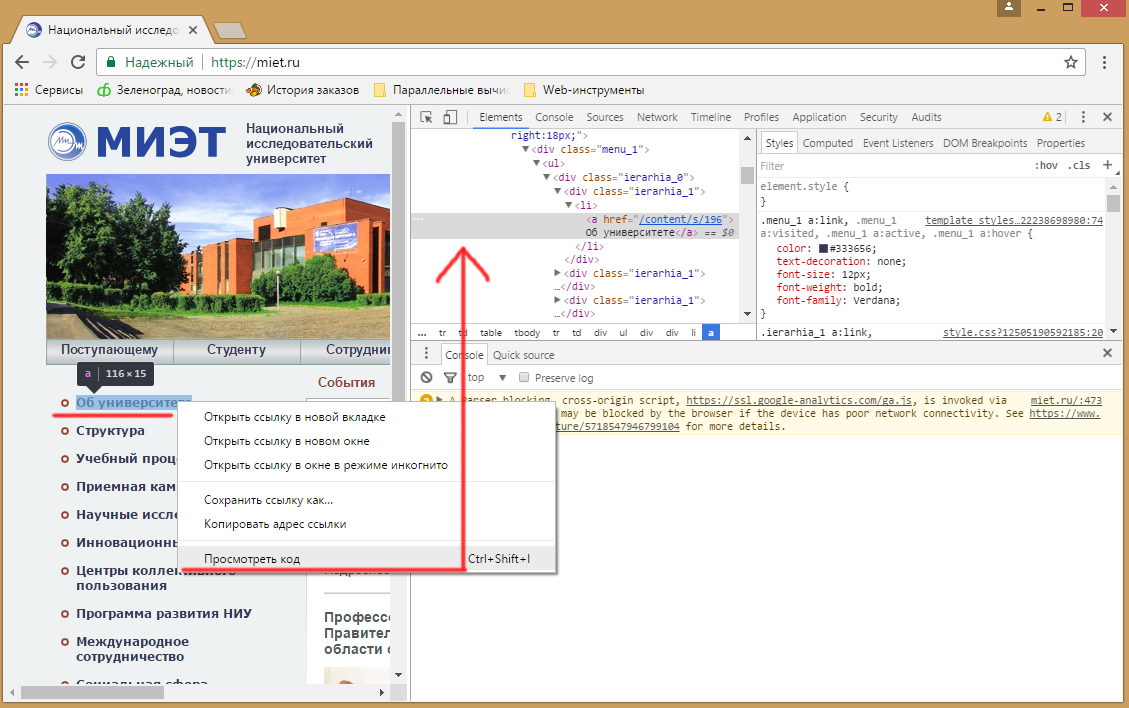
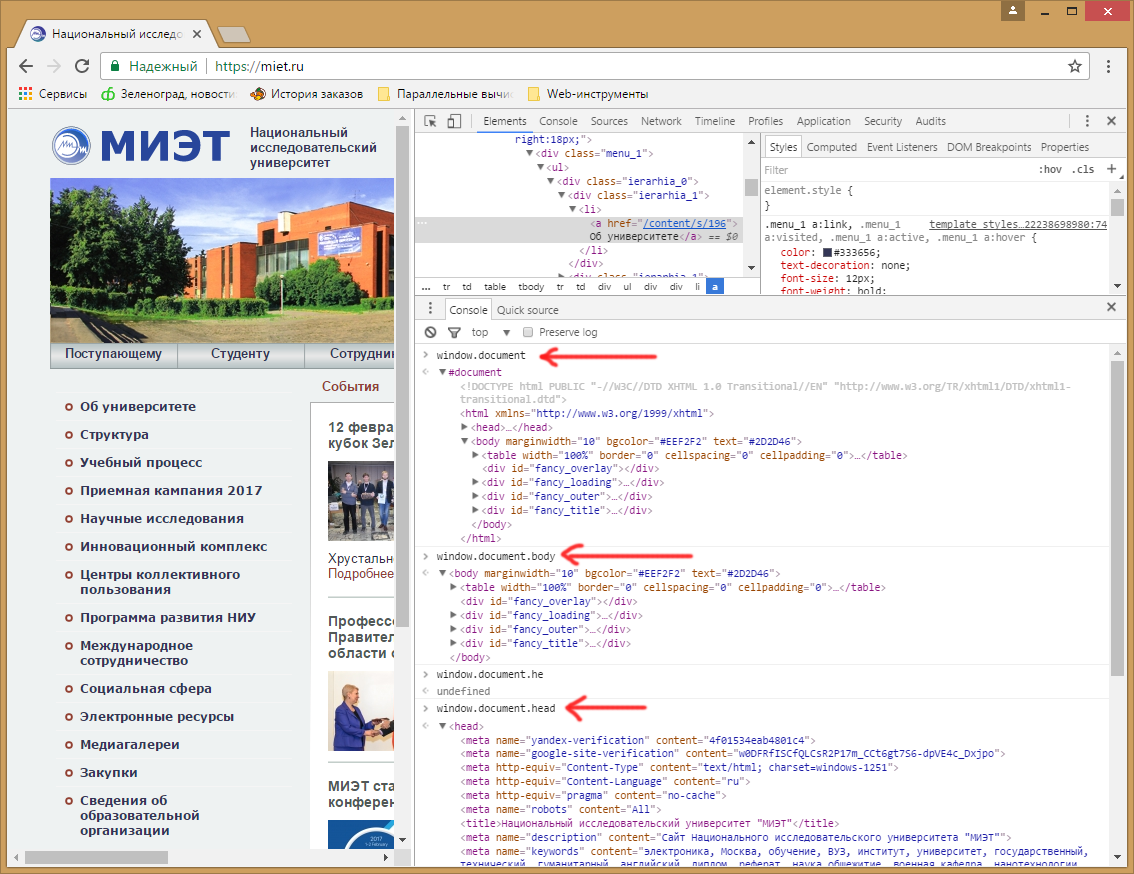
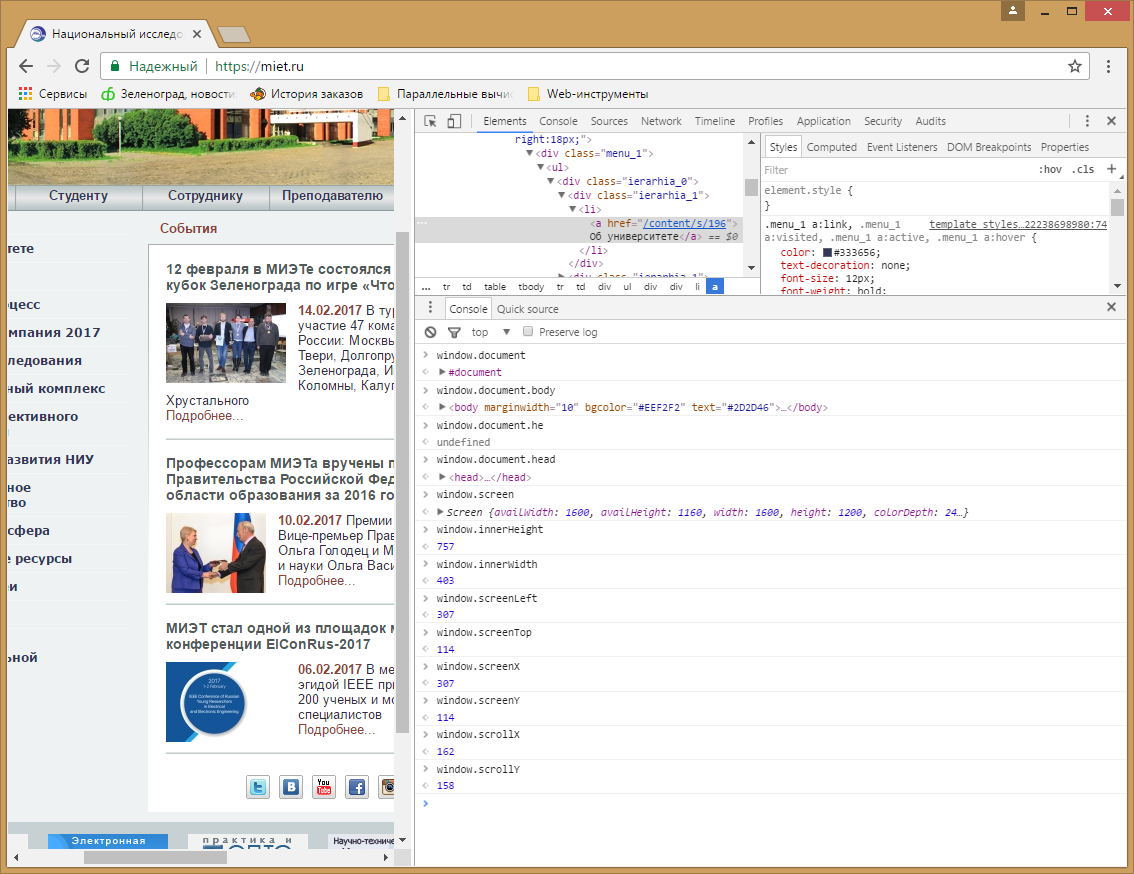


Рис. 3. Вид окна браузера Chrome в режиме отладки

Как видно на рис. 3. в левом верхнем окне показан код элемента в теле документа, в правом верхнем окне — свойства этого элемента (по умолчанию это стили), а ниже находится консоль (Console). Консоль позволяет выполнять команды. Например, с помощью конслоли можно исследовать DOM документа, вызывая показ описания его элементов командами вида: window.document.body, window.document.screen и т. д. см. рис. 4-5. При наборе слова в консоле появляется всплывающая контекстная подсказка, которая облегчает ввод.

Рис. 4. Демонстрация иерархичности DOM

Рис.5. Демонстрация получения геометрических свойств окна (window)

В режиме отладчика можно вносить изменения в код текущего загруженного документа для его отладки и тестирования. Можно поменять сам HTML код документа или CSS стили и эти изменения сразу отобразятся в видимой части документа. Для внесения изменений достаточно кликнуть на элементе правой кнопкой мыши для вызова контекстного меню в котором есть пункты для редактирования, например: «Edit Attribute», «Edit as HTML» см. рис. 6. На рис. 6. также показана открытой закладка «Properties», где видно, что текст «Приемная кампания 2017» яляется дочерним узлом (childNodes) тега ссылки «<a>».

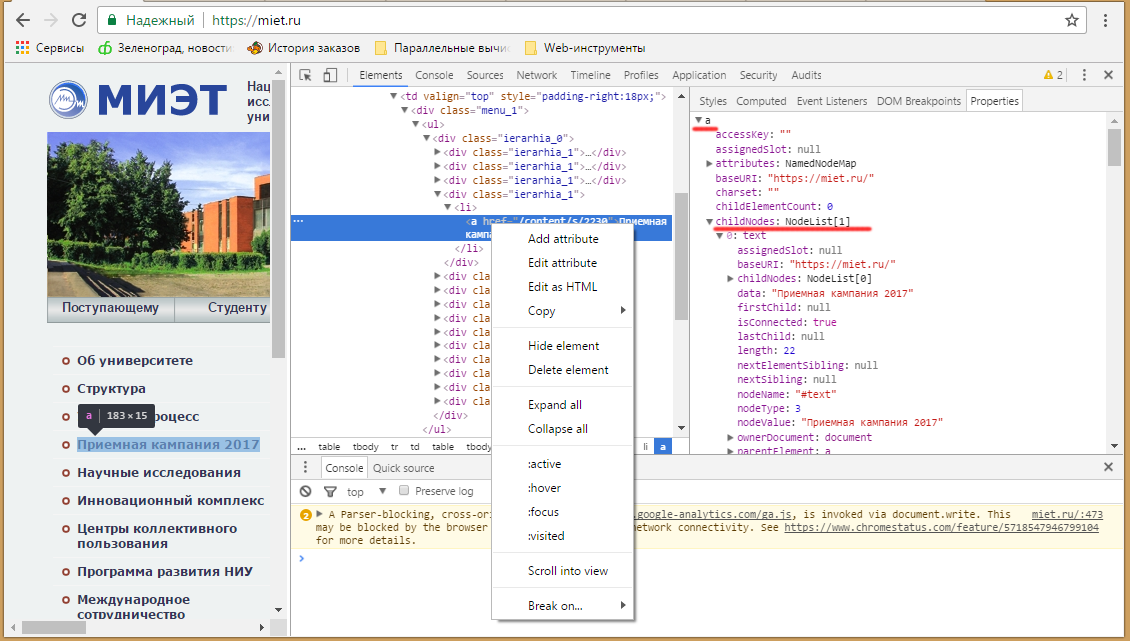


Рис. 6. Демонстрация изменения свойств элемента

Изменения содержимого элемента можно производить и с помощью Консоли. Но для этого элемент сначала нужно извлечь для редактирования. На рис. 7. показано, как с помощью метода document.getElementByClassName() извлекается первый элемент вертикального меню сайта МИЭТ и в нем заменяется текст «Об университете» на текст “Здравствуй ВУЗ”.

Все изменения, осуществляемые с помощью отладчика производятся над локальной копией документа, загруженной в браузер. При перезагрузке документа с сайта будет скачана его оригинальная копия и все внесенные изменения пропадут (за исключением брекпоинтов поставленных для отладки JavaScript)

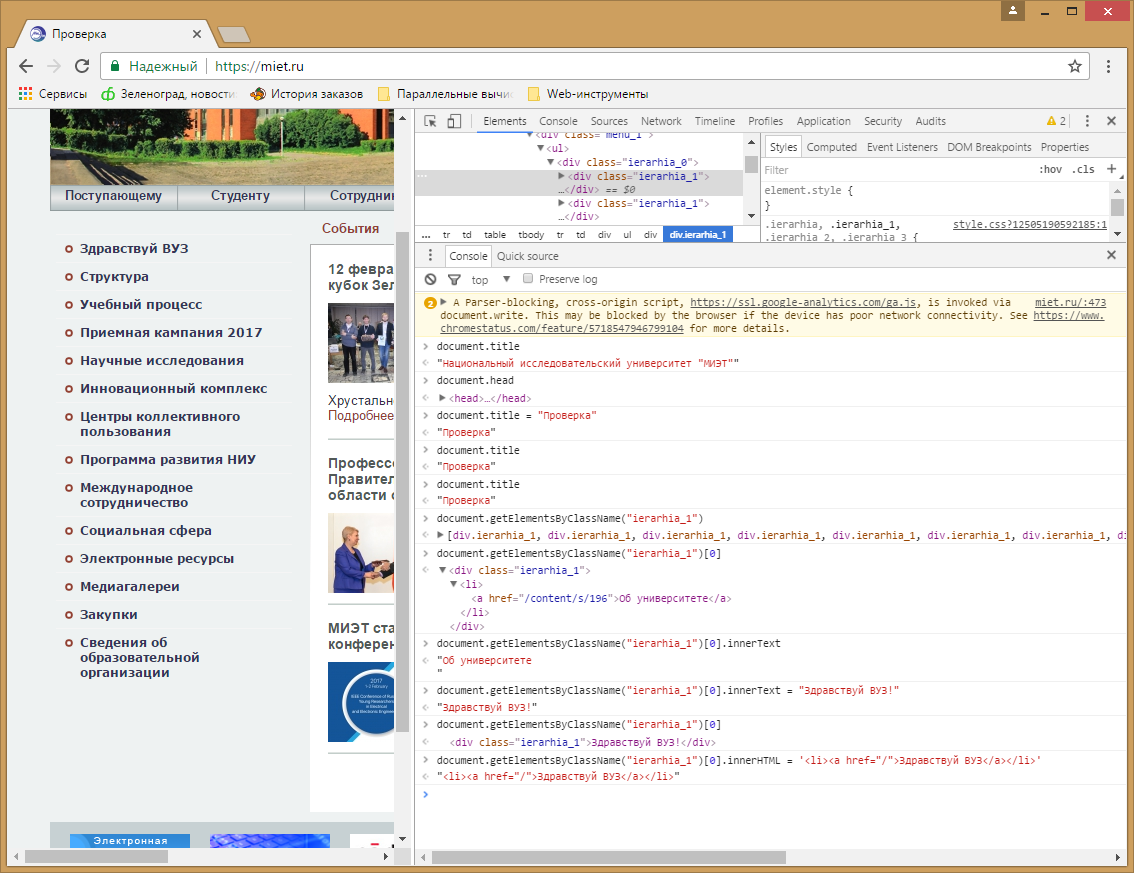


Рис. 7. Демонстрация изменения локальной копии документа с помощью отладчика

Важным удобством отладчика является то, что он показывает для объекта все его свойства и все допустимые для него методы. Например, набрав **document.get**  можно увидеть, что кроме получения объекта по имени класса — **getElementByClassName**, элемент можно получить по уникальному идетнификатору — **getElementByID**, по имени тега - **getElementByTagName** и т. д.

С помощью языка JavaScript элементами страницы можно манипулировать также, как из консоли. Например, замену показанную на рис.7. на JavaScript можно сделать так:

**Листинг 2. Пример JavaScript**

<script>

document.getElementByClassName(“ierarhia\_1”)[0].innerText = “Здравствуй вуз!”;

</script>

# Основы JavaScript

JavaScript это язык с помощью которого можно управлять содержимым загруженного HTML документа, в том числе и динамически обновляя его, путем получения новых данных и отправки запросов на сервер.

Программа «Здравствуй вуз!» на JavaScript может выглядеть так:

**Листинг 3. Программа «Здравствуй вуз!»**

<!DOCTYPE html>  
<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />

<title>МИЭТ</title>

</head>

<body>

<script>

window.alert("Здравствуй вуз!");

document.write("<h1>Здравствуй вуз!</h1>");

</script>

</body>

</html>

Метод **alert** объекта **window** показывает всплывающее модальное окно с текстом «Здравствуй вуз!», а метод write объекта document “вписывает” этот текст, в форме заголовка первого уровня в тело документа. Объект первого уровня **window** можно не писать. Например, вместо **window.alert(...)** можно писать только **alert(...)**. Как видно из примера кода, программа на JavaScript с одной стороны “видит” все объекты DOM при этом сама являясь частью документа.

При работе с динамическими страницами наиболее востребованы следующие «сценарии»:

1. Проверка полей форм ввода перед отправкой на сервер
2. Изменение размеров, положения и оформления элементов страницы при ресайзинге и прокрутке документа в окне браузера
3. Управление показом контента: фотогалереи, меню, закладки и подобное.
4. Отправка запросов на сервер, прием и публикация ответов
5. On-line диалоговые системы: калькуляторы, системы заказа, чаты и подобное.

В данной лабораторной работе будут рассматриваться сценарии 1 и 2 из этого списка.

## Основные синтаксические конструкции JavaScript\*

Переменные. Комментарии. Условия. Массивы. Циклы. Функции.

\* для сокращения вместо JavaScript далее будет использоваться JS

Операторы в JavaScript разделяются символом точка с запятой - «;»

### Переменные

Переменные объявляются с помощью ключевого слова var\*.

\* - если переменная находится внутри функции и ее имя совпадает с именем переменной, находящейся вне функции, то без использования ключевого слова var будет взята внешняя переменная, а с использованием var будет создана локальная переменная в этой функции.

**Листинг 4. Пример использования переменных**

<script>

var test = '8'; //присваиваем символ

var temp = 8; //присваиваем число

alert(test+temp); //выведет 88

alert(test\*1+temp); //выведет 16

</script>

Как видно из примера в JS существует неявное приведение типов. Если с символьной переменной выполняется арифметическая операция, кроме «+», то символьная переменная автоматически преобразуется в число. Операция сложения «+» в JS также выполняет роль конкатенации (склейки) строк. Поэтому при использовании «+» по умолчанию содержимое правого операнда добавляется к содержимому левого.

Если переменной test присвоить значение, которое не может быть преобразовано к число (например, var test='miet'), то в результате попытки преобразования будет возвращено значение NaN, что означает ошибку числовой операции, если она не корректна. Значение NaN далее распространяетcя на все выражение. Поэтому даже если все остальные операторы корректны то результатом всего выражения будет NaN.

Значением переменных также могут быть:

* Infinity – бесконечность (результат деления на 0)
* null – значение переменной неизвестно
* true/false – логическое значение
* undefined – значение не присвоено (не инициализированно)

Кавычки в JS двух типов:

* одинарные - ''
* двойные - “”

Двойные кавычки применяются, когда в коде имеет смысл устанавливать специальные символы разметки из которых наиболее востребован символ перевода строки - «\n»

Код alert("Первая строка\nВторая строка"); Выведет текст «Вторая строка» под текстом «Первая строка»

### Комментарии

**Листинг 4. Пример использования однострочных и многострочных комментариев**

<script>

//Однострочный комментарий

/\*

Многострочный

комментарий

\*/

</script>

### Условия

В JS определены логические операторы условий:

* == РАВНО с неявным приведением типа. Пример: '1'==1 //вернет true
* === РАВНО без приведения типа. Пример: '1'===1 //вернет false
* && - И
* || - ИЛИ
* ! - НЕ
* < - меньше
* > - больше
* <= - меньше или равно
* >= - больше или равно
* != - не равно

Эти операторы интерпретируют любое значение как логическое.

В JS определены три конструкции логических операторов:

**Листинг 5. Пример конструкции if-else**

//Предположим где-то был определена переменная year

if (year < 2017) {

alert( year+' меньше 2017' );

} else if (year > 2017) {

alert( 'year+' больше 2017' );

} else {

alert( 2017 );

}

**Листинг 6. Пример тернарного оператора ?**

//Присваиваем переменной success значение true | false в зависимости от величины year

var success = (year > 2017) ? true : false;

**Листинг 7. Пример конструкции switch**

//Предположим где-то был определена переменная year

switch (year) {

case 2017:

alert( 'Это число 2017' );

case "2017":

alert( 'Это строка 2017' );

break;

default:

alert( 'Это не 2017, а что-то другое' );

}

Как видно из примера switch использует сравнение без неявного приведения типа

### Массивы

**Листинг 8. Пример объявления массивов с числовыми индексами и операции с ними**

var test = []; //Определяем пустой массив

var temp = ["МИЭТ", "лучший", "вуз"]; //Определяем массив со значениями

test[0] = "Первое"; //Присваивание значения по индексу

test.push("Второе"); //Добавление элемента в конец массива

test.unshift("Нулевое"); //Добавляет элемиент в начало массива

temp.shift(); //Удаляем первый элемент массива

temp.pop(); //Удаляем последний элемент массива

alert( temp ); // лучший

alert( test ); // Нулевое,Первое,Второе

test.length; // 3 элемента

test.length = 1; //Укоротили массив справа до одного элемента. test содержит «Нулевое»

Допустимо создавать многомерные массивы вкладывая один массив в другой

**Листинг 9. Пример объявления массивов с символьными индексами и операции с ними**

var test = {}; //пустой ассоциативный массив

var temp = {

vuz:"МИЭТ",

quality: ["лучший", "вуз"],

danu: "однозначно"

}; //многомерный массив со значениями

test.name = "Первое"; //Создание свойства и присваивание значения

temp.quality[2] = "!"; //Присваивание значения

alert( test.name ); // Первое

alert( test["name"] ); // Так тоже вернет Первое

alert( test.family ); // undefined

alert( temp.quality ); //лучший,вуз,!

Как видно, обращение к несуществующему свойству ассоциативного массива не приводит к ошибке, а возвращает undefined.

### Циклы

В JS определены 4-ре конструкции циклов. 3 универсальных: while, do-while, for и четвертый for-in для перебора свойств ассоциативных массивов.

**Листинг 10. Примеры циклов**

var i = 0;

while (i < 3) { //Цикл с предусловием

alert( i ); i++;

}

i = 0;

do { //Цикл с постусловием

alert( i ); i++;

} while (i < 3);

for (i = 0; i < 3; i++) { //Цикл со счетчиком

alert( i );

}

var temp = {

vuz:"МИЭТ",

quality: ["лучший", "вуз"],

danu: "однозначно"

};

for (key in temp) { //Цикл для перебора ключей ассоциативного массива

alert(temp[key]); //выведем по очереди все определенные значения ключей

}

Для управления ходом итераций цикла предусмотрены операторы break и continue, работающие также, как в других языках программирования.

### Функции

Как и в других языках программирования функции предназначены для структурирования кода приложения.

Примечание. Кроме функций JS поддерживает работу с объектами в парадигме ООП, но это будет рассмотрено в следующих лабораторных.

**Листинг 11. Пример функции**

var test = "МИЭТ - лучший";

var temp = "Неужели";

var attentions = {}; //глобальный объект предупреждений

var errors = {}; //глобальный объект ошибок

//Определение функции

function showTest(data, text, flag) {

var Result = true;

if( flag === undefined ) {

Result = false;

attentions.showTest = 'Флаг не передан';

}

alert( data + " " + test + " " + text );

// test берется извне функции

// будет выведено - "2017 год МИЭТ - лучший однозначно"

return Result;

}

//Вызов функции с проверкой возвращаемого значения

if( !showTest("2017 год", "однозначно") ) {

if( attentions.showTest )

alert( attentions.showTest ); //сработает т.к. флаг не был передан

}

# Взаимодействие JavaScript и DOM

Как было показано в примерах кода: Листинг 2 и Листинг 3 JS имеет доступ к объектам DOM. Это позволяет получить в коде JS объекты DOM и их свойства. Поскольку свойств у объектов много, удобно пользоваться справочником отладчика броузера, который при выборе элемента на странице показывает все свойства и методы допустимые для манипулирования с ним см. рис. 8.

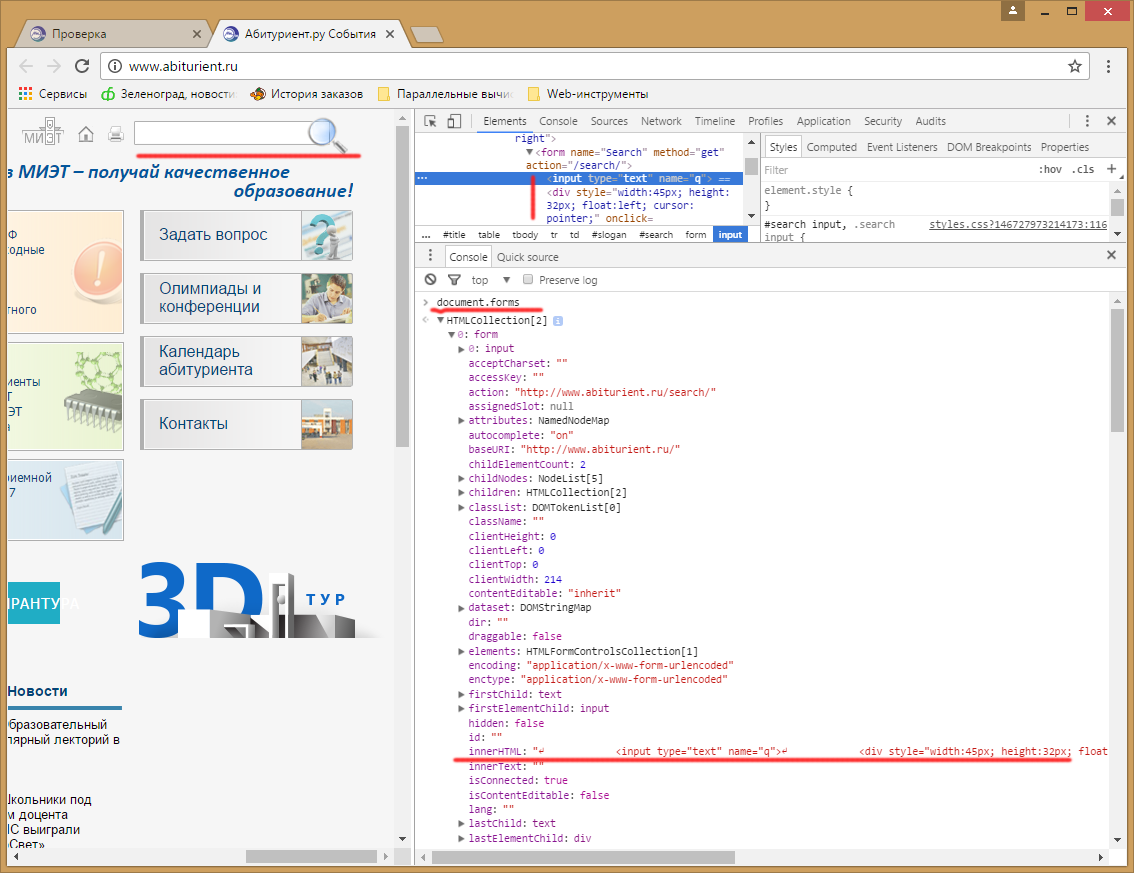


Рис. 8. Просмотр свойств и методов элементов страницы

## Применение JavaScript для задач управления размером и позиционированием элементов

Предположим есть трехколоночный макет сайта, в котором в первой и третьей колонках определены по два контейнера. Предположим, что при изменении размеров экрана до ширины менее 900 пикселей второй контейнер из правой колонки нужно переместить в левую колонку. Сделать это средствами CSS — невозможно т. к. CSS не может изменить иерархию DOM.

Составим алгоритм задачи

1. «Взять» второй контейнер из правой колонки
2. Разместить взятый контейнер в левой колонке между первым и вторым контенером

**Листинг 12. HTML код**

<div class="layout">

<div class="leftCol">

<div id="leftFirst">

Первый в левой колонке

</div>

<div id="leftSecond">

Второй в левой колонке

</div>

</div>

<div class="rightCol">

<div id="rightFirst">

Первый в правой колонке

</div>

<div id="rightSecond">

Второй в правой колонке

</div>

</div>

<div class="centerCol">Колонка 2</div>

</div>

Поскольку для контейнеров определены id “взять” контейнер c ID=”rightSecond” можно с помощью метода getElementByID. Затем контейнер нужно разместить после контейнера с ID=”leftFirst”. Эту операцию можно оформить в виде фукции в которую передается ID извлекаемого контейнера, ID контейнера назначения и flag операции: вставить до, вставить после.

**Листинг 13. Функция перемещения контейнера**

function moveDiv(IDFrom, IDTo, flag) {

if( IDFrom === undefined || IDTo === undefined )

return false;

var obj\_from = document.getElementById(IDFrom);

var obj\_to = document.getElementById(IDTo);

var parent\_obj\_to = obj\_to.parentNode; //Определяем родительский узел

if( obj\_from === null || obj\_to === null )

return false;

if( obj\_to.parentNode === obj\_from.parentNode )

return false; //Объект уже перемещен. Выходим

if( flag === undefined || flag === true) {

parent\_obj\_to.insertBefore(obj\_from, obj\_to);

//метод insertBefore вставляет элемент obj\_from перед элментом obj\_to

//внутри родительского элемента parent\_obj\_to

} else if (flag === false) {

parent\_obj\_to.insertBefore(obj\_from, obj\_to.nextSibling);

//метод nextSibling получает следующий элемент

//метод previousSibling получает предыдущий элемент

}

};

Теперь нужно вызывать эту функцию при изменении размеров экрана. Для этого нужно отслеживать эти изменения с помощью механизма событий DOM. С каждым объектом DOM ассоциированы различные события. У объекта window определены 6 событий:

* Load – вызывается после загрузки документа в окно
* Unload — вызывается перед выгрузкой из окна
* Focus — окно (документ) получает фокус
* Blur — окно (документ) теряет фокус
* Resize — окно меняет размеры
* Error — в коне произошла ошибка

Нам нужно, чтобы код срабатывал при изменении размеров окна, значит нам нужно «повесить обработчик» на событие Resize. Это делается с помощью метода addEventListener, которому передается имя события и обработчик, который нужно вызывать при наступлении этого события см. Листинг 14.

**Листинг 14. Отслеживание ресайза окна**

var startWidth = 900;

window.addEventListener("resize", function() {

//проверяем, что текущая ширина меньше заданной

if( window.innerWidth < startWidth) {

moveDiv("rightSecond", "leftFirst", false);

} else {

moveDiv("rightSecond", "rightFirst", false);

//перемещаем контейнер на прежнее место, если ширина больше заданной

}

});

Все возможные параметры окна показаны на рис. 9 и 10.

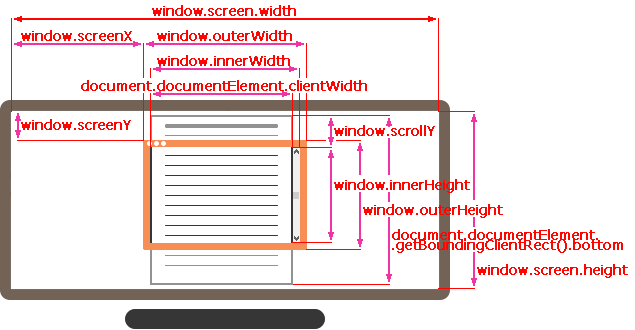


Рис. 9. Размеры окна

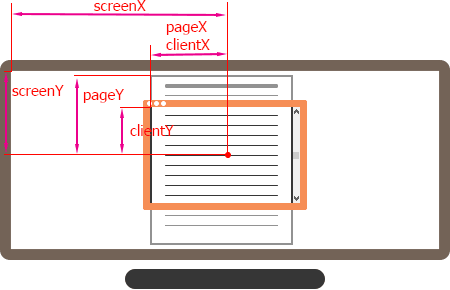


Рис. 10. Размеры при позиционировании путем скролинга

Кроме задания отслеживания через addEventListener, возможно также задавать отслеживание непосредственно в коде тега элемента.

Например так: <div onClick=”alert('Кликнули по этому контейнеру');”></div>

Если требуется определить положение конкретного элемента относительно текущих границ окна броузера, то используется метод getBoundingClientRect(), который возвращает объект, содержащий свойства прямоугольника ограничивающего элемент. У этого объекта есть свойства top и left, которые и показывают положение элеменат страницы относительно окна броузера. Тогда, если переменная obj содержит элмент страницы, полученный например с помощью метода getElementByID и elem = obj.getBoundingClientRect(), то clientY = elem.top, а clientX = elem.left. Координаты элемента относительно страницы, с учетом ее прокрутки можно получить с помощью pageY= clientY + window.pageYOffset иpageX= clientX + window.pageXOffset.

## Применение JavaScript для обработки элементов форм ввода

Как было показано на рис. 1 HTML страница имеет несколько типов форм ввода для отправки информации на сервер. Обработка значений введенных в формы возможна как при их вводе/выводе в каждый элемент формы с помощью событий onChange, onBlur и т.д. так и перед отправкой всех данных на сервер с помощь обработки событий onSubmit.

Для иллюстрации обработки элементов форм предположим, что необходимо контролировать отправку содержимого формы ввода поискового запроса по следующим правилам:

1. Не отправлять пустую форму
2. Перед оправкой контролировать содержимое и иметь возможность отменить отправку

Для этого составляем простую форму и «вешаем» обработчики на событие потери фокуса (**onblur**) поля ввода поискового запроса и событие отправки формы (**onsubmit**) см. Листинг 15.

**Листинг 15. HTML код формы**

<div class="centerCol">

<form name="TestForm" method="POST" action="index.php" onsubmit="return checkAllFileds(this)">

Введите текст в поле поиска

<input type="text" name="SearchString" value="" onblur="emptyControl(this);" />

<input type="submit" name="Submit" value="Отправить" disabled/>

</form>

</div>

После этого создаем функции emptyColor и checkAllFields. Как видно в из листинга «Листинг 15» внутрь этих функций передаются сами объекты для которых определено событие.

**Листинг 16. JS код обработчиков формы**

function emptyControl(obj) {

if(obj.value === "") {

TestForm.Submit.disabled = true;

alert("Вы не ввели данные в поле поиска\nКнопка Отправить - заблокирована");

} else {

TestForm.Submit.disabled = false;

}

}

function checkAllFileds(form) {

for(i=0; i<form.length; i++) {

//Контроль отправленных данных

if ( form[i].type == "text") {

if( !window.confirm(form[i].name+"\n"+form[i].value+"\nОтправляем?") )

return false; //Если не подтверждаем отправку, то она отменяется

}

}

return true;

}

При отладке JS кода удобно пользоваться отладчиком, который вызывается в Chrome по закладке Sources см. рис. 11.

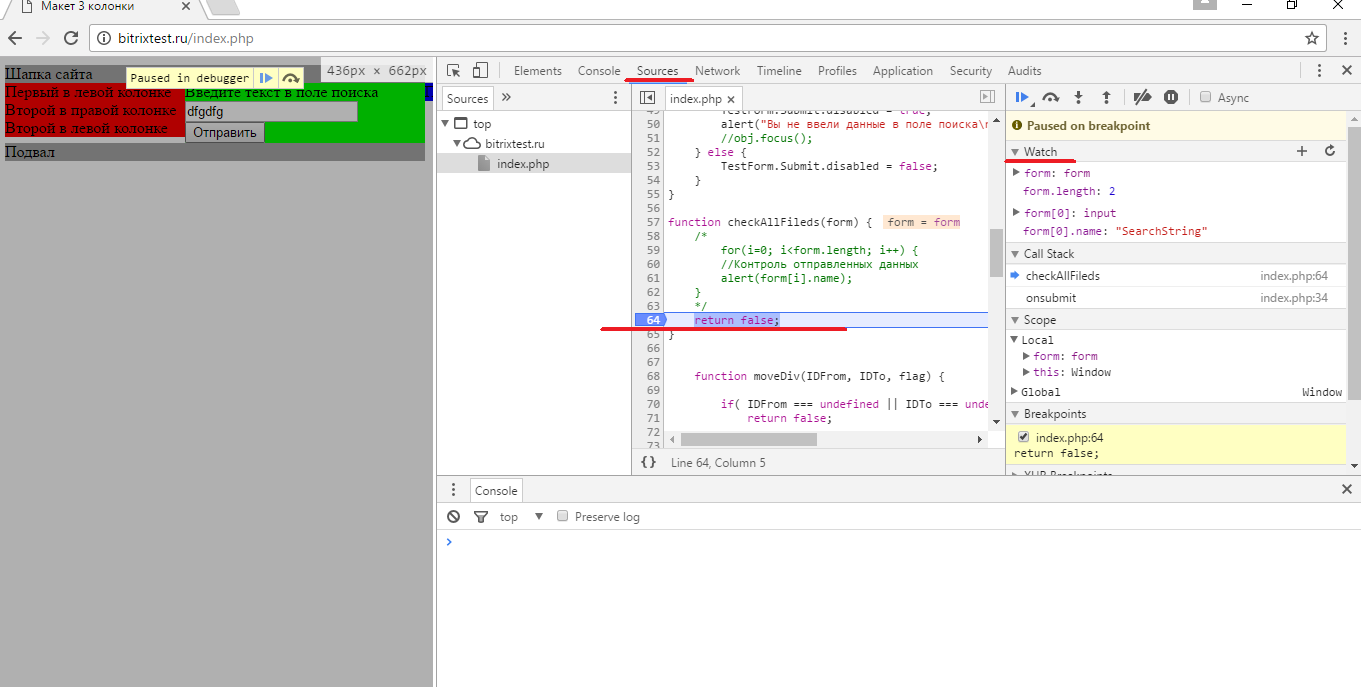


Рис. 11. Демонстрация режима отладки JS кода

## Хранение данных в браузере между вызовами страницы

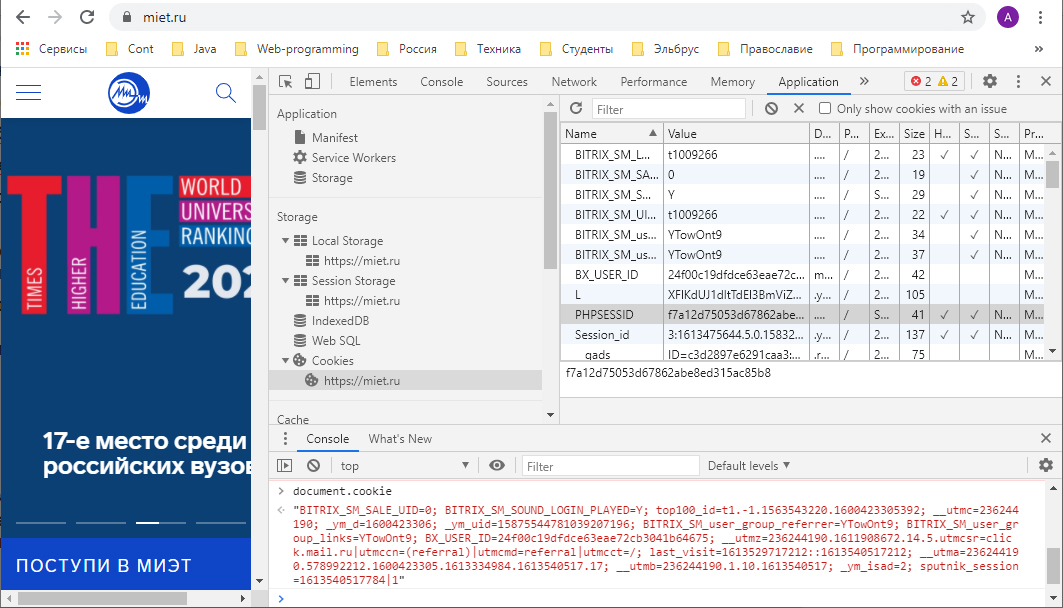
Данные, которые обрабатывает JavaScript, в том числе данные, введенные в поля форм, не сохраняются, если страница будет перезагружена. Такая ситуация может возникнуть, например, из-за сбоя соединения при отправке данных из формы. Так-же сохранение данных может потребоваться для восстановления front-end окружения клиента, повторно зашедшего на сайт, после неявного выхода: закрытие вкладки или браузера. На сегодняшний день существуют три способа хранения данных в браузерах (не зависящие от браузера) и реализованные во всех браузерах:

1. **Cookie** (употребляемые русские сокращения: «куки» или «ключики»)
2. **LocalStorage + SessionStorage**
3. **IndexedDB**

Все эти способы предлагают механизм хранения данных в виде ассоциаций **ключ=значение**.

Инструмент WebSQL предусматривает хранение данных в виде реляционной БД, однако с 2010 года, когда был предложен W3C реализован не во всех браузерах (Chrome, Safary, Opera, Android) и, на сегодняшний момент, не рекомендован W3C к применению.

Данные, содержащиеся в Cookie, LocalStorage, SessionStorage и IndexedDB можно посмотреть как с помощью скрипта на JavaScript (например, командой document.cookie), так и с помощью специального интерфейса отладки бразузера (например, в Chrome закладка «Application», пункт меню «Storage») см. рис. 12.

  
Рис.12. Демонстрация просмотра локальных данных в браузере Chrome

### Cookie

Cookie (ключик), иcторически, первый тип хранения пользовательских данных в браузере. В переводе с английского cookies - «печеньки = хлебные крошки». Такое название появилось потому, что первоначально cookies предназначались для хранения информации о посещенных страницах сайта и организации навигации вида: **адрес сайта | раздел | раздел | … | страница**. Однако, возможность записи в «ключики» информации вида **ключ=значение** как с помощью front-end (JavaScript), так и с помощью back-end (PHP, Ruby, JavaScript, Java и т. д.) обусловило широкое применение ключиков для хранения пользовательских данных на стороне клиента в браузере. Cookie являются частью HTTP протокола, поэтому back-end программы, формирующие HTTP ответ, могут включать cookie в заголовок документа. Так-же cookie записанные с помощью JavaScript – отправляются на сервер с запросом документа.

Безопасность использование cookie обеспечивается на уровне браузера. Разные сайты (домены) не могут читать cookie друг друга. Существуют так-же настройки для разграничения видимости cookie между страницами и разделами одного сайта (домена).

В данной ЛР рассматривается работа с cookie только со стороны front-end. Работа с cookie со стороны Back-end будет рассмотрена в других ЛР.

Запись ключика с помощью JavaScript очень проста:

**Листинг 17. Запись и чтение cookie**

<script>

document.cookie = “test=Spintex”;

alert(document.cookie); //Вернет test=Spintex

</script>

К сожалению, нет встроенных методов JavaScript для получения значения cookie по ключу. document.cookie возвращает все ключики данного домена в виде строки и для извлечения значения нужного ключа нужно писать специальную функцию разбора, например, с использованием регулярных выражений см. Приложение 4.

По умолчанию, доступ к ключику имеет тот домен, скрипт страницы которого произвел запись. При этом поддомены (например, forum.miet.ru) этого домена (miet.ru) не будут иметь доступа к данным ключиков.

Так-же по умолчанию, значения cookie (если не указано иное) хранятся до закрытия браузера.

Для решения проблем с поддоменами и хранения cookie между открытиями браузера существует расширенный формат записи cookie с параметрами: domain, path, expires, max-age:

**Листинг 18. Запись и чтение cookie с параметрами**

<script>

document.cookie = “test=Spintex; domain=institut-spintex.ru; path=/proba; expires=Wed, 24 Feb 2021 03:14:07 GMT”;

alert(document.cookie); //Вернет test=Spintex

</script>

В листинге 18 указано, что корневым доменом является institut-spintex.ru. При этом все поддомены вида \*. institut-spintex.ru могут иметь доступ к ключику test. Однако, далее идет указание конкретной папки к которой относится ключик - path=/proba. Это означает, что все подпапки этой папки будут «видеть» ключик, но все старницы, что выше или на одному уровне c /proba — не будет иметь доступа. Наконец, параметр expires определяет «срок жизни» ключика до 24 февраля 2021 года. Чтобы не писать сложный формат даты, его можно получить с помощью объекта Date, см. Листинг 19.

**Листинг 19. Преобразование даты для cookie**

<script>

let date = new Date(Date.now()+86400); //Текущую дату увеличиваем на сутки

date = date.toUTCString(); //Преобразуем дату в строку

document.cookie = "user=John; expires=" + date;

</script>

Второй вариант указания времени «жизни ключика» — использовать вместо expires параметр max-age, который задает время жизни ключика в секундах от текущего времени. Например, max-age=3600, установит срок жизни ключика в 1 час. **Важно, что установка прошедшей даты приводит к удалению ключика.**

### Local Storage и SessionStorage

Хранение данных с помощью ключиков обладает рядом неудобств:

1. Ограниченный размер данных (зависит от браузера), которые можно сохранить Спецификация W3C рекомендует 4кб на один ключ и количество ключей до 300 на один сайт (домен). Однако, реализация этих рекомендаций в различных браузерах — различается, например, от 20 ключей на сайт в IE7 до 176 в Chrome 58.
2. Все ключики передаются с клиента на сервер и с сервера на клиент при каждом запросе, хотя многие из них могут быть нужны только локально в течение одной сессии. Отсюда, если есть 50 ключей по 4 кб и из них только 3 нужно пересылать при каждом запросе (логин, пароль, код сессии), то паразитный трафик составит (50-3)\*4=188Кб.
3. Хранение данных в ключиках уязвимо для атак XSRF
4. Ключики не имеют удобного механизма записи, изменения и удаления значения по ключу

Перечисленные неудобства были устранены в механизмах хранения (объектах) LocalStorage и SessionStorage. Эти механизмы отличаются временм хранения данных. Данные SessionSrorage сохраняются только пока закладка со страницей не была закрыта. LocalSorage хранит данные между перезапусками браузера.

Методы обоих объектов одинаковы:

1. setItem(key, value) – сохранить пару ключ/значение.
2. getItem(key) – получить данные по ключу key.
3. removeItem(key) – удалить данные с ключом key.
4. clear() – удалить всё.
5. key(index) – получить ключ на заданной позиции.
6. length – количество элементов в хранилище.

Хранилища LocalStorage и SessionStorage позволяют хранить не менее 2 Мб данных, не передаются на сервер и не получаются с сервера. Ключи и их значения могут быть только строками. Данные LocalStorage не имеют «времени истечения». Удалять их надо в явном виде методов removeItem или clear. В отличие от ключиков данные могут быть привязаны не только к домену и протоколу, но и к порту.

**Листинг 20. Работа с LocalStorage**

<script>

localStorage.test = “Spintex”; // установить значение для ключа

alert( localStorage.test ); // возвращает значение Spintex ключа test

</script>

### IdexedDB

Хранилище IndexedDB (офиц. документация — <https://w3c.github.io/IndexedDB/>) можно считать эволюцией хранилища LocalStorage в сторону поддержки надежности операций (транзакционность), введения типов данных и ограниченной реализации запросов по ключам и значениям.

1. Хранилище IndexedDB поддерживает:
2. Транзакции
3. Запросы по ключам и индексам полей объектов
4. Типы ключей: примитивные (Number, String) и объектные (Date, ArrayBuffer, Array)
5. Типы значений: String, Date, Object, Array, File, Blob, ImageData
6. Больший объем данных, чем у LocalStorage

Примеры использования IndexedDB - <https://learn.javascript.ru/indexeddb>

# Библиотеки JavaScript

Как можно было заметить в главе «Взаимодействие JavaScript и DOM», обращение к элементам DOM требует написания повторяющегося кода document.getElementById();. Для сокращения кода при обращениям к элементам DOM и манипулировании ими написан ряд библиотек. Наиболее распростарненной библиотекой на сегодня является jQuery (<https://jquery.com/>).

Библиотека jQuery встраивается в страницы сайта кодом -

<script src="путь/[query.js](https://jqueryvalidation.org/files/lib/jquery.js)"></script>.

Тогда можно обращаться к элементам страницы с помощью инструкции

$(селектор).метод(параметры).

Предположим, в коде страницы существует элемент <div id=”Test” val=”100”>. Тогда для получения значения 100 в переменную test можно написать

var test = $(“#Test”).attr(“val”).

Селекторы могут быть переменными и выражениями JavaScript. Правила формирования конечного вида селекторов совпадают с таковыми для CSS. Актуальная документация по методам jQuery расположена по адресу <https://api.jquery.com/>.

Удобство библотеки jQuery сделало ее платформой для многих производных библиотек. Например, для проверки полей форм написана библиотека jQuery Validation Plugin (<https://jqueryvalidation.org/>).Библиотека позволяет использовать как определенные в ней по умолчанию способы проверки, так и добавлять новые, а также управлять видом отображения сообщений об ошибках см. Приложение 3.

**Задание на лабораторную работу**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Задание | Баллы | % |
| 1 | Освоить работу с отладчиком клиентской части интернет-приложения на базе браузера Chrome или аналога | 1 | 20 |
| 2 | Разместить в средней колонке соответствующие номеру задания поля формы и обеспечить их проверку по заданию. | 1 | 20 |
| 3 | При уменьшении размеров экрана менее 480 пикселей оставить видимой только центральную колонку с формой | 1 | 20 |
| 4 | Осуществить проверку полей формы из пункта 3 с помощью библиотеки jQuery Validation Plugin | 1 | 20 |
| 5 | Реализовать сохранение введенных в поля данных при закрытии браузера или страницы с помощью хранилищ Cookie или LocalStorage | 1 | 20 |
|  |  | 5 | 100 |

|  |  |
| --- | --- |
| 1 | Поля формы и проверки:  text – проверка наличия только чисел  textarea — проверка минимальной длины символов — 20 и максимальной 200  radio – 3 позиции, проверка обязательного заполнения первой позиции |
| 2 | Поля формы и проверки:  text – проверка наличия только букв русского и латинского алфавитов  select – проверка, что выбрана хотя-бы одна позиция  checkbox – 3 позиции, проверка обязательного заполнения первой и последней позиции |
| 3 | text – проверка наличия только чисел  checkbox – 3 позиции, проверка обязательного заполнения первой и последней позиции  radio – 3 позиции, проверка обязательного заполнения последней позиции |
| 4 | text — проверка минимальной длины символов — 20 и максимальной 200  text – проверка номера телефона по формату +\*(\*\*\*)\*\*.\*\*.\*\*  checkbox – 3 позиции, проверка исключения заполнения одновременно первой и последней позиции |
| 5 | text – проверка исключения символов: %,$,#,@,^  checkbox – 3 позиции, проверка обязательного заполнения первой и последней позиции  radio – 3 позиции, проверка обязательного заполнения последней позиции |
| 6 | text – проверка номера телефона по формату +\*(\*\*\*)\*\*.\*\*.\*\*  checkbox – 3 позиции, проверка исключения заполнения одновременно первой и последней позиции  textarea — проверка минимальной длины символов — 5 и максимальной 2000 |

**Контрольные вопросы:**

1. Что такое DOM?
2. Какие объекты первого уровня находятся в объекте window?
3. Какие объекты первого уровня находятся в объекте document?
4. Как посмотреть и изменить CSS свойства элемента в отладчике?
5. Как посмотреть и изменить код элемента в отладчике?
6. Как обратиться из JS к элементам DOM?
7. Как привязать обработчик события к элементу DOM?
8. Какие виды массивов существуют в JavaScript?
9. Какие виды циклов существуют в JavaScript?
10. Как осуществляется неявное преобразование типов в JavaScript?
11. Чем отличаются сравнения == и ===?
12. Какие элементы форм вы знаете? Для чего они предназначены?
13. Какие способы сохранения данных в Браузере вы знаете? Чем они отличаются?
14. Что такое библиотека JQuery? В чем преимущества ее использования?

**Дополнительные вопросы**

(при подготовке рекомендуется использовать источники из Приложения 1)

1. Расскажите об элементе form
2. Расскажите об элементе input типа text
3. Расскажите об элементе radio
4. Расскажите об элементе select
5. Расскажите об элементе checkbox
6. Расскажите об элементе input типа button

# Приложение 1

# Список литературы и ссылок

* <http://htmlbook.ru/> - справочник по HTML и CSS
* <https://webref.ru/ref> - справочник по HTML и CSS
* [http://javascript.ru](http://javascript.ru/) – справочник по JavaScript
* <http://learn.javascript.ru/> - учебник по JavaScript
* <http://es5.javascript.ru/> - спецификация JavaScript на русском языке
* <https://api.jquery.com/> - документация jQuery
* <https://jqueryvalidation.org/> - документация плагина jQuery.validator
* <https://learn.javascript.ru/data-storage> — Примеры использования Cookies, LocalStorage и IndexedDB

# Приложение 2

# Свойства, методы и события форм

Свойства, методы и события объекта  Form

|  |  |  |
| --- | --- | --- |
| Свойства | Методы | События |
| length  action  method  target  encoding  elements[] | reset()  submit() | Reset  Submit |

Свойства, методы и события объекта Text

|  |  |  |  |
| --- | --- | --- | --- |
| Свойства | Методы | Обработчики событий | |
| defaultValue  value  size  maxLength  disabled  readOnly | focus()  blur()  select() | onChange  onSelect  onFocus  onBlur  onClick  onDblClick | onMouseOver  onMouseOut  onMouseDown  onMouseUp  onKeyPress  onKeyDown  onKeyUp |

Объект Select

|  |  |  |
| --- | --- | --- |
| Свойства | Методы | Обработчики событий |
| options[]  size  length  multiple  selectedIndex | focus()  blur()  add()  remove() | onBlur  onChange  onFocus |

Объект Option

|  |  |  |
| --- | --- | --- |
| Свойства | Методы | События |
| defaultSelected  selected  index  text  value | нет | нет |

# Приложение 3

Пример кода проверки формы библиотекой jQuery Validation Plugin

<script>

$(document).ready(function(){

//Задание текстов описания ошибок

jQuery.validator.messages.required = "Это поле обязательно для заполнения";

jQuery.validator.messages.email = "Некорректный e-mail";

jQuery.validator.messages.minlength = "Недостаточное количество символов в поле";

jQuery.validator.messages.maxlength = "Превышено количество символов в поле";

jQuery.validator.messages.phone = "Синтаксически неверный номер телефона";

jQuery.validator.messages.symbols = "Введены запрещенные символы";

//Метод проверки телефона

jQuery.validator.addMethod("phone", function(phone\_number, element) {

phone\_number = phone\_number.replace(/\s+/g, "");

return this.optional(element) || !SimpleCheckOnExist(' +()-0123456789', phone\_number);

}, "Некорректный номер телефона.");

//Метод проверки HTML Тегов

jQuery.validator.addMethod("symbols", function(text, element) {

text = text.replace(/\s+/g, "");

return this.optional(element) || !text.match(/(\<|\>)/);

}, jQuery.validator.messages.symbols+" HTML разметки: < и/или >");

//Функция проверки наличия недопустимых символов

function SimpleCheckOnExist(checkStop, checkStr)

{

var check = false;

var i = 0;

var j = 0;

for (i = 0; i < checkStr.length; i++)

{

simbol = checkStr.charAt(i);

for (j = 0; j < checkStop.length; j++)

{

if (simbol == checkStop.charAt(j))

{ check = true; break; }

}

if (check == false)

{ return(true); break; }

check = false;

}

}

//Вывод сообщения при успешной отправке

$.validator.setDefaults({

submitHandler: function() {

alert("submitted!");

}

});

//Задание на проверку полей формы

$().ready(function() {

// validate the comment form when it is submitted

$("#ContactForm").validate({

rules: {

name: {

required: true,

minlength: 3

},

email: {

required: true,

minlength: 3,

email: true

},

phone: {

required: true,

minlength: 7,

maxlength: 36,

phone: true

},

service: {

required: true,

//required: "#face:checked",

minlength: 2

},

"comment": {

required: false,

maxlength: 3000,

symbols: true

}

},

messages: {

name: {

required: "Введите название",

minlength: "Введите не менее 3-х символов"

},

email: "Введите верный email адрес",

service: "Выберите не менее 2-х опций"

}

});

});

});

</script>

<style>

#ask\_yurist {

min-width:330px;

min-height:300px;

background-color:#9AC43C;

border-radius:20px;

color:#fff;

#ask\_yurist .title {

#ask\_yurist .title h3 {

font-size:22px;

text-transform:uppercase;

font-weight:bold;

text-align:center;

margin:0;

padding:0;

#ask\_yurist .body {

#ask\_yurist form {

#ask\_yurist fieldset {

padding:10px;

display:inline-block;

width:47%;

min-height:250px;

color:#fff;

font-weight:normal;

vertical-align:top;

min-width:270px;

#ask\_yurist fieldset.face,

#ask\_yurist fieldset.comments {

#ask\_yurist legend {

border:none;

padding:0 10px 0 10px;

width:auto;

}

#ask\_yurist label,

#ask\_yurist fieldset.face label span,

#ask\_yurist fieldset.services label span {

overflow:hidden;

display:inline-block;

margin-right:10px;

color:#fff;

font-weight:normal;

font-size:18px;

#ask\_yurist fieldset.face label span,

#ask\_yurist fieldset.services label span {

width:150px;

display:inline-flex;

#ask\_yurist input[type="text"],

#ask\_yurist input[type="mail"],

{

#ask\_yurist div.b\_send {

#ask\_yurist input[type="Submit"] {

padding:10px 30px 12px 30px;

font-size:22px;

border-radius:10px;

border:none;

background-color:#000;

color:#fff;

}

#ask\_yurist input[type="Submit"]:hover {

color:#000;

@media (max-width: 980px) {

#ask\_yurist fieldset,

#ask\_yurist fieldset.face,

#ask\_yurist fieldset.comments {

float:none;

}

#ContactForm label.error {

margin: 0;

width: auto;

display: none;

width:100%;

text-align:right;

background:#fff;

color:#ff0000;

padding:5px;

}

#ContactForm input.error{border:1px dotted red}

#newsletter\_topics label.error {

display: none;

margin-left: 103px;

}

</style>

<div id="ask\_yurist">

<div class="title">

<h3>Вопрос юристу</h3>

</div>

<div class="body">

<form id="ContactForm" name="ContactForm" method="post" action="">

<fieldset class="contacts">

<legend>Контакты</legend>

<p>

<label for="name">Имя <em>\*</em></label><input type="text" id="name" name="name" required>

</p>

<p>

<label for="email">E-mail <em>\*</em></label><input type="email" id="email" name="email" required>

</p>

<p>

<label for="phone">Телефон <em>\*</em></label><input type="text" id="phone" name="phone" required>

</p>

</fieldset>

<fieldset class="face">

<legend>Лицо</legend>

<p>

<label><span>Юридическое</span><input type="radio" id="face" value="1" name="face" required></label>

</p>

<p>

<label><span>Физическое</span><input type="radio" id="face" value="2" name="face"></label>

</p>

<p>

<label><span>ИП</span><input type="radio" id="face" value="3" name="face"></label>

</p>

<label for="face" class="error"></label>

</fieldset>

<fieldset class="services">

<legend>Услуги</legend>

<p>

<label><span>Гражданские</span><input type="checkbox" class="checkbox" id="service1" value="1" name="service"></label>

</p>

<p>

<label><span>Арбитражные</span><input type="checkbox" class="checkbox" id="service2" value="2" name="service"></label>

</p>

<p>

<label><span>Уголовные</span><input type="checkbox" class="checkbox" id="service3" value="3" name="service"></label>

</p>

<label for="service" class="error"></label>

</fieldset>

<fieldset class="comments">

<legend>Комментарий</legend>

<p>

<textarea id="comment" name="comment" style="width:95%" rows="8"></textarea>

</p>

</fieldset>

<div class="b\_send">

<input type="submit" name="Submit" value="Отправить">

</div>

</form>

</div>

</div>