Работа с DOM-моделью и пользовательский интерфейс

По материалам курса University of Washington

http://www.cs.washington.edu/education/courses/cse190m/07sp/index.shtml

Введение в DOM

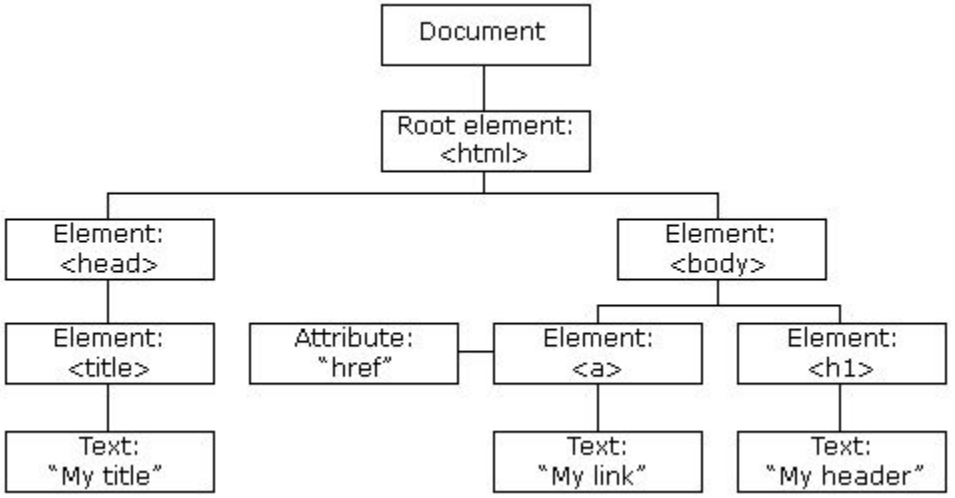
DOM – Document Object Model:

```
<!DOCTYPE html ...>
<html xmlns="...">

<head>
  <title>My title</title>
</head>

<body>
  <a href="...">
    My link
  </a>
  <h1>My header<h1>
</body>

</html>
```



Элементы (`Element`), один из них - корневой

Атрибуты (`Attribute`)

Текстовые узлы (`Text`)

Работа с узлами в DOM

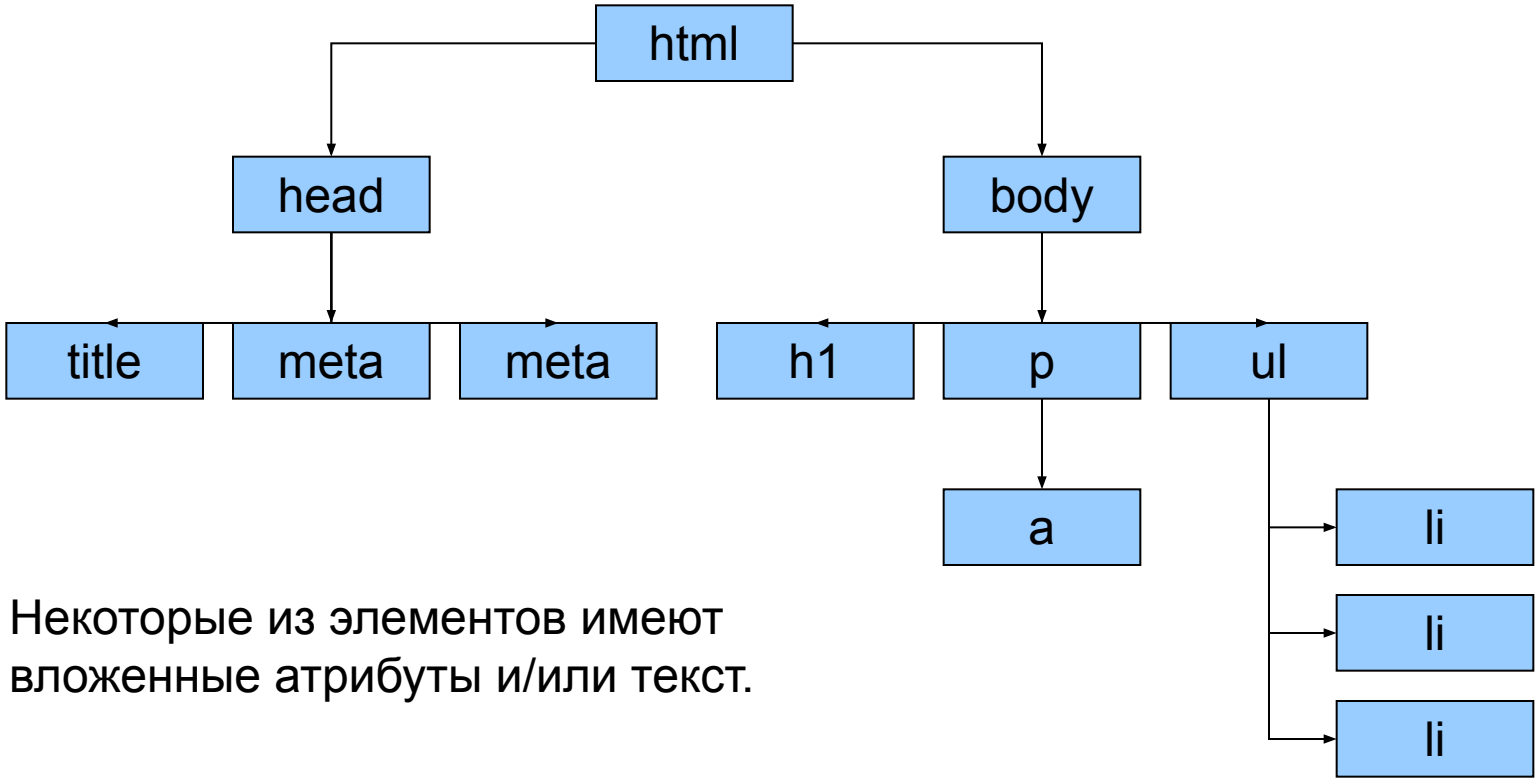
Есть несколько способов получить объект, представляющий узел, через глобальный объект `document` или уже имея ссылку на другой узел:

1. Если узел имеет уникальный идентификатор (атрибут `id`), то узел можно найти с помощью метода `document.getElementById(id)`.
2. Можно найти массив элементов с заданными тегами с помощью метода `document.getElementsByTagName(tag)`.
3. Можно перейти от узла к его непосредственным потомкам `node.firstChild, node.lastChild` или к предку `node.parentNode`.
4. Можно перейти от узла к его соседям `node.nextSibling`, `node.previousSibling`.

Пример страницы

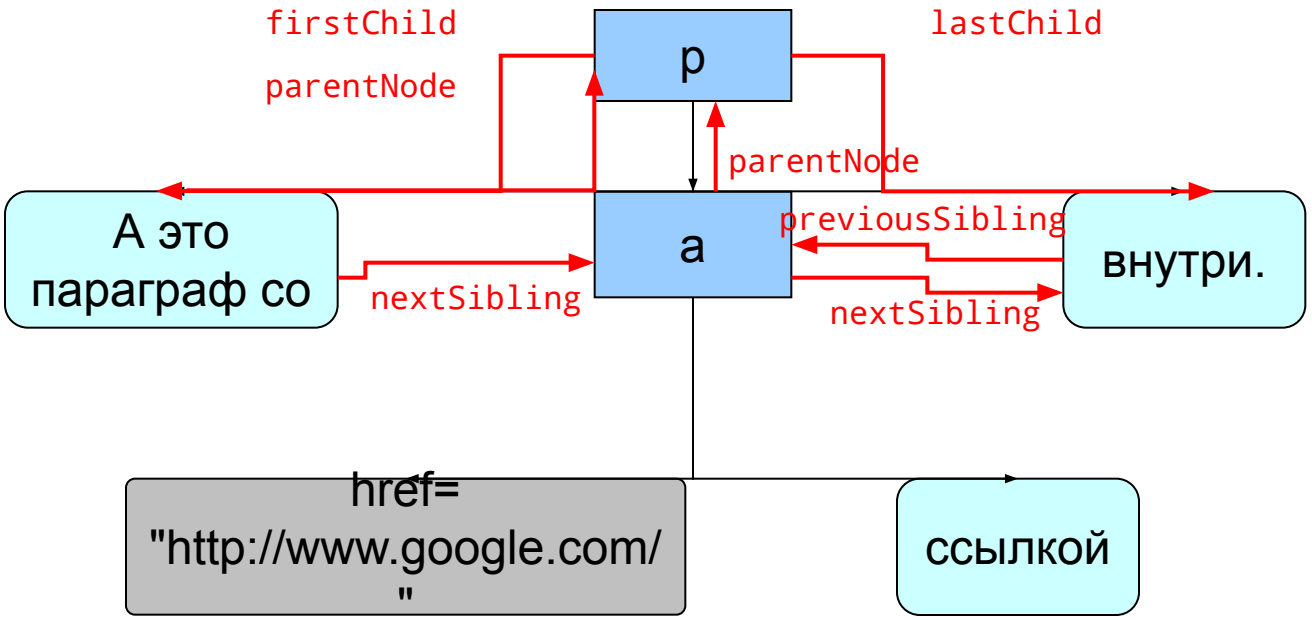
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
                "http://www.w3.org/TR/XHTML1/DTD/XHTML1-strict.dtd">
<html xmlns="http://www.w3.org/1999/XHTML" xml:lang="en" lang="en">
<head>
  <title>Заголовок страницы</title>
  <meta http-equiv="content-type"
        content="text/html;charset=windows-1251"/>
  <meta http-equiv="Content-Language" content="ru"/>
</head>
<body>
   <h1>Это заголовок</h1>
   <p>А это - параграф со
      <a href="http://www.google.com/">ссылкой</a> внутри.
   </p>
   <ul>
     <li>элемент списка</li>
     <li>еще один элемент</li>
     <li>третий элемент списка</li>
   </ul>
</body>
</html>
```

Дерево элементов для этой страницы



Некоторые из элементов имеют вложенные атрибуты и/или текст.

Полное дерево одного из элементов



- элемент

- атрибут

- текстовый элемент

Если pNode – указатель на "p",
то заменим "ссылкой" на "звездой".

```
pNode.firstChild.nextSibling.
lastChild.nodeValue = "звездой";
```

change.html

Изменение структуры DOM страницы

Следующие методы применимы ко всем элементам DOM:

1. `element.appendChild(node)` – добавление нового узла в конец списка "детей".
2. `element.insertBefore(newNode, oldNode)` – добавление нового узла в список детей перед заданным.
3. `element.removeChild(node)` – удаление указанного узла из списка "детей".
4. `element.replaceChild(newNode, oldNode)` – замена в списке "детей" существующего элемента на новый.

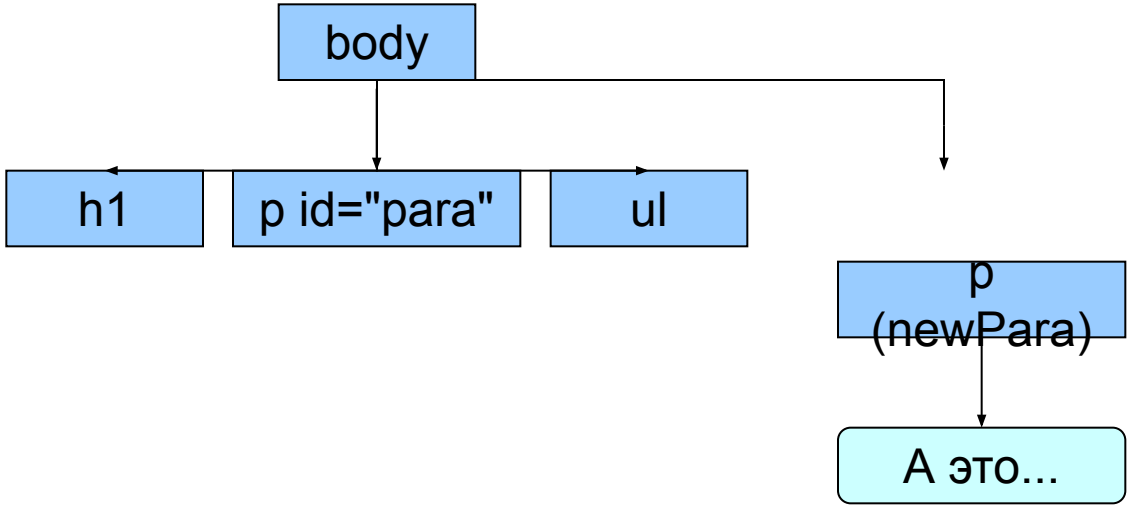
Новый элемент (атрибут, текстовый узел) можно создать с помощью следующих методов:

1. `document.createElement(tag)` – создание нового элемента с заданным тегом.
2. `document.createAttribute(name)` – создание нового атрибута с заданным именем.
3. `document.createTextNode(data)` – создание текстового узла.

Динамическое добавление узлов

Добавим новый параграф сразу после заданного:

```
function insertNewParagraph() {
  var pNode = document.getElementById('para');
  var newPara = document.createElement('p');
  var newText = document.createTextNode(
      'А это динамически добавленный параграф!');
  newPara.appendChild(newText);
  pNode.parentNode.insertBefore(newPara, pNode.nextSibling);
}
```



insert.html

Отделение Javascript-кода от HTML

Поместим теперь весь код полностью в отдельную javascript-страницу:

```
window.onload = initBody;

function initBody() {
  var pNode = document.getElementById('para');
  pNode.onclick = insertNewParagraph;
}

function insertNewParagraph() {
  var pNode = document.getElementById('para');
  var newPara = document.createElement('p');
  newPara.style.color = 'red';
  newPara.style.marginLeft = '50px';
  var newText = document.createTextNode(
      'А это динамически добавленный параграф!');
  newPara.appendChild(newText);
  pNode.parentNode.insertBefore(newPara, pNode.nextSibling);
}
```

separate.html

Динамическое добавление реакций на события

Вместо определения значений атрибута `onclick` можно добавлять элементам реакции на события. Например, вместо

```
window.onload = initBody;
```

можно написать

```
window.addEventListener('load', initBody, false);
```

Преимущества:

1. Соответствие стандарту (метод `addEventListener` применим не только для HTML-страниц, но и для любых XML-документов).
2. Можно определить несколько реакций на одно и то же событие, при этом все они будут выполняться последовательно.
3. Можно управлять распространением событий (третий аргумент `addEventListener`).

Удалить реакцию можно с помощью метода `window.removeEventListener` с теми же аргументами.

В примере реакции динамически добавляются и удаляются. *dyna.html*

Использование групповой обработки

Массовую обработку элементов можно производить с помощью метода `getElementsByTagName`, например:

```
var emElements = document.getElementsByTagName('em');
for (var i = 0; i < emElements.length; ++i) {
  emElements[i].style.backgroundColor = 'yellow';
}
```

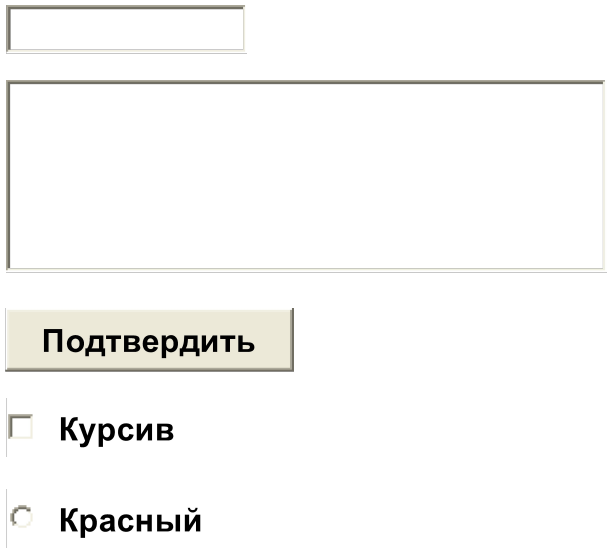
В примере показано изменение стиля элементов.

group.html

Элементы интерфейса с пользователем

Имеется большой набор элементов интерфейса с пользователем:

- простое окно ввода;
- многострочное окно ввода;
- кнопка;
- флажок;
- переключатель;



Чаще всего эти элементы используются в составе «форм». Содержимое «формы» может быть передано web-серверу в виде параметров запроса.

Все элементы считаются «элементами ввода» различных типов и появляются в виде элементов `<input type="button">`, но некоторые могут иметь и свои собственные теги, например, `<button>`.

Кнопки

Пример использования кнопок.

```
<body>
  <h1>Заголовок страницы</h1>
  <button onclick="alert('Превед медвед!');">Жми сюда!</button>
</body>
```

button.html

Внутри тела реакции `this` означает ссылку на сам элемент (кнопку).

```
<body>
  <h1>Заголовок страницы</h1>
  <button onclick="this.firstChild.nodeValue += ' быстрее!';
                   this.parentNode.appendChild(
                        document.createTextNode(' Молодец!'));">
    Жми сюда!
  </button>
</body>
```

more-button.html

Осторожно! Внутри функции `this` означает ссылку на контекст!

Многострочный текст

```
<script>
   function changeProperty(butt, prop) {
     if (typeof(butt.flag) == 'undefined') butt.flag = 1;
     var ta = document.getElementById('myText');
     ta[prop] = butt.flag == 1;
     butt.flag = 1 - butt.flag;
   }
</script>

<body>
   <h1>Это область ввода текста</h1>
   <p><textarea cols="20" rows="4" id="myText">
        Сюда вводим текст.</textarea></p>
   <p>
     <button onclick="changeProperty(this, 'disabled');">
         Disable/Enable</button>
     <button onclick="changeProperty(this, 'readOnly');">
         ReadOnly yes/no</button>
   </p>
</body>
```

textarea.html

Списки выбора

```
<body>
   <h1>Выберите из списка:</h1>
   <p><select>
     <option value="0">Пики</option>
     <option value="1">Трефы</option>
     <option value="2">Бубны</option>
     <option value="3">Черви</option>
   </select></p>
   <p>Вы выбрали: <span id="choice">&bksp;</span></p>
</body>
```

При выборе элемента происходит событие change, при этом можно узнать, какое значение выбрано и какой текст связан с этим выбором.

Еще атрибуты:

select.html

`selectNode.options` – массив узлов, представляющих элементы списка

`selectNode.size` – число показываемых элементов

`selectNode.multiple` – разрешен ли мульти-выбор

`selectNode.selectedIndex` – индекс выбранного элемента

Списки выбора (продолжение)

Список может быть визуально разбит на группы

```
<select>
  <optgroup label="Овощи">
    <option value="0">Репка</option>
  </optgroup>
  <optgroup label="Люди">
    <option value="1">Дедка</option>
    <option value="2">Бабка</option>
    <option value="3">Внучка</option>
  </optgroup>
  <optgroup label="Животные">
    <option value="4">Жучка</option>
    <option value="5">Кошка</option>
    <option value="6">Мышка</option>
  </optgroup>
</select>
```

Поля ввода

Элемент с тегом `input` может заменять и дополнять многие из рассмотренных ранее элементов ввода. Его представление и функциональность зависят от типа. Этот элемент никогда не имеет внутреннего содержания, только атрибуты.

```
<input type="тип элемента ввода" />
```

где «тип элемента ввода» может иметь следующие значения:

`text`	текстовое поле ввода, похожее на `textarea`
`password`	текстовое поле ввода со скрытым отображением символов
`checkbox`	флажок
`radio`	элемент выбора
`button`	кнопка (такая же, как в элементе `button`)

Кроме этих типов есть еще типы, предназначенные исключительно для представления элементов ввода внутри форм для передачи информации на web-сервер. Это типы:

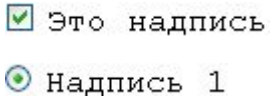
```
submit reset file hidden
```

Поля ввода (продолжение)

```
<input type="text" />
<input type="password" />
```

имеют атрибуты `maxLength`, `size`, `disabled`, `readOnly`, `value`.

```
<input type="checkbox" />
<input type="radio" />
```

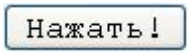


имеют атрибуты `defaultChecked`, `disabled`, `checked`, `value`.

Чтобы приделать надписи к флажкам и элементам выбора, используют элемент `label`: `<label for="input-id">текст</label>`.

Чтобы собрать элементы выбора в одну группу, используют атрибут `name`: `<input type="radio" name="group-name"/>`.

```
<input type="button" />
```



имеет атрибуты `disabled`, `value`.

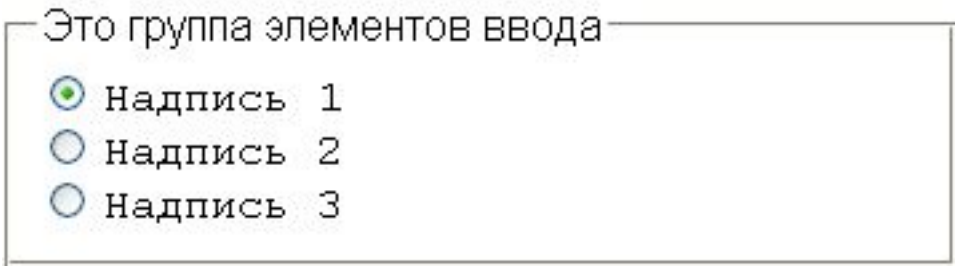
Как выглядят все эти элементы ввода можно посмотреть в примере

input.html

Визуальная группировка элементов UI

Элементы ввода можно визуально сгруппировать и добавить надпись к группе:

```
<fieldset style="width: 40%">
   <legend>Это группа элементов ввода</legend>
   <input type="radio" checked="true" name="radio-group"
          id="radio-1"/>
   <label for="radio-1">Надпись 1</label>
   <input type="radio" name="radio-group" id="radio-2"/>
   <label for="radio-2">Надпись 2</label>
   <input type="radio" name="radio-group" id="radio-3"/>
   <label for="radio-3">Надпись 3</label>
</fieldset>
```



Формы

Форма – это средство для группировки элементов ввода с целью последующей отправки введенной информации на сервер.

Отправка информации производится с помощью запроса типа GET или POST с параметрами.

```
<form action="http://www.google.com/search" method="GET">
   <label>Введите запрос: <input type="text" name="q"/></label>
   <input type="submit"/>
</form>
```

Атрибут *action* задает URL сервера, на который отправляется запрос.

Атрибут *method* задает используемую команду – GET или POST.

В приведенном примере осуществляется запрос GET к поисковой машине Google с параметром q и значением введенного запроса.

Нажатие кнопки submit (отправить запрос) эквивалентно в данном случае выдаче запроса http://www.google.com/search?q=value, где value – содержимое текстового поля ввода.

simple-form.html

Более сложный пример формы

Важный атрибут каждого поля ввода – name. Он задает имя аргумента при запросе к серверу. Пример формы:

```
<form action="http://jobserver.ru/anketa.php" method="get">
  <fieldset>
    <legend>Введите анкетные данные</legend>
    <label>Ваше имя: <input type="text" name="name"/></label>
    <label>Запрашиваемая должность:
      <select name="job">
        <option value="core">Программист</option>
        <option value="web">Web-дизайнер</option>
        <option value="lamer">Крутой спец</option>
      </select></label>
    <label>Квалификация:</label>
    <label><input type="radio" value="0" name="qual">Новичок</label>
    <label><input type="radio" value="1" name="qual">Профи</label>
    <label><input type="radio" value="5" name="qual">Эксперт</label>
    <label>Хочу получать от <input type="text" name="min"/></label>
    <input type="submit"/>
  </fieldset>
</form>
```

complex-form.html

Методы GET и POST

Метод GET отправляет все значения в строке запроса с помощью кодирования типа `http://myurl.com/page?name1=val1&name2=val2...`

- вся строка видна в окне браузера;
- ее можно запомнить в «закладках»;
- легко вернуться назад на эту же страницу;
- число и размер параметров ограничены длиной URL.

Метод POST отправляет все значения в заголовке запроса с помощью задания пар `name=value`

- отправляемые значения не так легко доступны;
- URL запроса бесполезно запоминать: параметров в нем нет;
- при возврате назад на эту же страницу чаще всего возникает ошибка POSTDATA;
- практически нет ограничений на число и длину параметров.

Специальные кнопки Submit и Reset

Запрос отправляется с помощью специального элемента ввода, имеющего вид кнопки:

```
<input type="submit"/>
```

Надпись на кнопке можно специфицировать с помощью атрибута value:

```
<input type="submit" value="Заказать"/>
```

Если надпись не специфицирована, то используется системное значение, зависящее от локализации системы ("Submit" для английского языка, "Отправить запрос" для русского).

Кнопка Reset используется для восстановления начальных значений элементов ввода в форме:

```
<input type="reset" value="Вернуться к началу"/>
```

Имитировать действие этих кнопок можно и программным путем.

```
var form = ... // DOM-узел для формы
form.submit();    ...    form.reset();
```

Использование CSS для элементов ввода

Поскольку один и тот же тег `input` используется для элементов самого разного вида, то задавать стили элементов в виде

```
input {
  color: blue;
  font-style: italic;
  font-size: large;
}
```

хотя и допустимо, но часто неудобно. Задавать стили можно отдельно для каждого типа элемента, например:

```
input[type="text"] {
  color: blue;
  font-style: italic;
  font-size: large;
}
```

Проверка корректности вводимых данных

Корректность введенных данных можно осуществлять

- перед отсылкой запроса (на клиенте с помощью Javascript);
- при обработке запроса на сервере (php, сервлеты).

Вот как может выглядеть схема программы на Javascript для проверки корректности введенных данных.

```
function onLoad() {
  var submit = document.getElementById('mySubmitButton');
  submit.addEventListener('click', onSubmit, false);
}

function onSubmit(event) {
  if (!checkInputValues()) {
    event.preventDefault();
    showErrors();
  }
}
```

В IE вместо `event.preventDefault()` используется `return false;`