**ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ**

**ОРДЕНА ТРУДОВОГО КРАСНОГО ЗНАМЕНИ**

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ БЮДЖЕТНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

**Московский Технический Университет Связи и Информатики**

**Доклад на тему:**

«Библиотека Prototype»

**Выполнила:**

Райкова Ксения Антоновна
Студентка 4 курса
Группа БПМ1301

**Проверила:**

Смирнова Надежда Ивановна

Москва, 2016 г

**Библиотека Prototype**

Prototype — JavaScript фреймворк, упрощающий работу с Ajax и некоторыми другими функциями. Несмотря на его доступность в виде отдельной библиотеки, он обычно используется программистами вместе с Ruby on Rails, Tapestry, script.aculo.us и Rico. Эта библиотека была написана Сэмом Стефенсеном (Sam Stephenson).

Данный фреймворк совместим со следующими браузерами: Internet Explorer (Windows) 6.0+, Mozilla Firefox 1.5+, Apple Safari 2.0.4+ и Opera 9.25+, Google Chrome 1.0+. Поддержка данных браузеров подразумевает, что он также совместим с Camino, Konqueror, IceWeasel, Netscape 7+, SeaMonkey, Яндекс.Браузер и др., которые принадлежат этим же семействам.

**Вспомогательные функции**

В Prototype присутствуют самые разные способы упрощения создания JavaScript приложений, от сокращённого вызова некоторых функций языка до сложных методов обращения к XMLHttpRequest. Очевидная польза этих функций в сокращении необходимости набирать часто повторящиеся куски кода. Ниже будут приведены некоторые примеры.

**Функция $()**

Для обращения к DOM элементу HTML страницы обычно используется функция document.getElementById:

* *document.getElementById("id\_of\_element ")*

Функция $() уменьшает код до:

* *$("id\_of\_element")*

Но в отличие от функции DOM, функции $() можно передавать более одного аргумента и функция вернет массив (Array) объектов со всеми соответствующими элементами:

* ***var*** *ar = $('id\_1', 'id\_2', 'id\_3');*

 ***for*** *(i=0; i<ar.length; i++)*

 *{ alert(ar[i].innerHTML); }*

Например, для указания цвета текста можно использовать следующий код:

* *$("id\_of\_element").style.color = "#ffffff";*

Или, используя расширенные функции Prototype:

* *$("id\_of\_element").setStyle({color: '#ffffff'});*

Код для версий ниже 1.5:

* Element.setStyle("id\_of\_element", {color: "#ffffff"});

**Функция $$()**

Функция $$() будет полезна тем, кто часто разделяет CSS от контента. Она разбивает один или несколько CSS фильтров, которые поступают на вход в виде выражения подобного регулярному выражению и возвращает элементы которые соответствуют этим фильтрам. Например при выполнении данного скрипта:

* ***var*** *f = $$('div#block .inp');*

получим массив, содержащий все элементы с классом .inp, которые находятся в контейнере div с идентификатором id="block".

Замечание: в данный момент (в версии 1.5.0) реализация функции $$() в prototype.js не очень эффективна. Если вы планируете использовать данную функцию часто в работе с объёмными и сложными HTML документами, можете рассмотреть другие реализации и просто заменить саму функцию.

**Функция $F()**

Похожая на $(), функция $F() возвращает значение определённого элемента HTML формы. Для текстового поля функция будет возвращать данные, содержащиеся в элементе. Для элемента 'select' функция возвратит выбранное в текущий момент значение.

* *$F("id\_of\_input\_element")*

Замечание: знак доллара $ — нормальный разрешённый символ для идентификаторов JavaScript; он был специально добавлен в язык одновременно с поддержкой регулярных выражений для возможности использования Perl-совместимых метасимволов, таких как $` и $'.

**Функция $A()**

Функция $A() преобразует один аргумент, который она получает, в объект Array. Эта функция, в сочетании с расширениями для класса Array, облегчает конвертирование или копирование любых перечислимых списков в объект Array. Один из вариантов использования заключается в том, чтобы преобразовать DOM NodeLists в регулярные массивы, которые могут быть более эффективно использованы.

* *<select id="list" >*

 *<option value="3">Test 1</option>*

 *<option value="2">Test 2</option>*

 *<option value="4">Test 3</option>*

 *</select>* ***var*** *someNodeList = $('list').getElementsByTagName('option');*

 ***var*** *nodes = $A(someNodeList);*

 *//Теперь вместо массива работаем с объектом*

 *nodes.each(****function****(node){*

 *alert(node.nodeName + ': ' + node.innerHTML);*

 *});*
Получаем
3: Test 1
2: Test 2
4: Test 3

**Функция $H()**

Функция $H() преобразовывает объекты в перечислимые Hash-объекты, которые похожи на ассоциативный массив.
Каждый элемент hash-объекта — это массив из двух элементов: ключ и значение. Помимо этого, объект обладает 5-ю методами:
**keys()** — возвращает массив из всех ключей
**values()** — возвращает массив из всех значений
**merge(Hashes)** — принимает объекты типа Hash, возвращает только один объект, результат их объединения
**toQueryString()** — преобраовывает объект в строку query-string. Т.е строку вида «key1=value1&key2=value2&key3=value3»
**inspect()** — возвращает объект в формате массива, вида «ключ: значение»

* *//Создаем объект*

 **var** a = { first: 10, second: 20, third: 30 };

 *//получаем hash*

 **var** h = $H(a);

 alert(h.toQueryString()); *//выведет "first=10&second=20&third=30"*

**Объект Ajax**

Объект Ajax предоставляет простые методы инициализации и работы с функцией XMLHttpRequest, без необходимости подстраивать код под нужный браузер. Существует два способа вызова объекта: Ajax.Request возвращает XML вывод AJAX-запроса, в то время как Ajax.Updater помещает ответ сервера в выбранную ветвь DOM.

Ajax.Request в примере ниже находит значения двух полей ввода, запрашивает страницу с сервера, используя значения в виде POST-запроса, а после завершения выполняет пользовательскую функцию showResponse():

* ***var*** *val1 = escape($F("name\_of\_id\_1"));*

***var*** *val2 = escape($F("name\_of\_id\_2"));*

 ***var*** *url = "http://yourserver/path/server\_script";*

***var*** *pars = {value1: val1, value2: val2};*

 ***var*** *myAjax =* ***new*** *Ajax.Request(*

 *url, { method: "post",*

 *parameters: pars,*

 *onComplete: showResponse });*

**Класс Еlement**

Методы класса предназначены для работы с элементами HTML. Для создания HTML элемента используется конструктор класса:

* ***new*** *Element(tagName[,{attributes}])*

В конструктор передаётся HTML-тег в строковом виде и, если необходимо, атрибуты тега.

Пример создания нового элемента:

* *// Создаём элемент <div> в памяти и указываем атрибуты id, class.*

 ***var*** *newElement =* ***new*** *Element('div',{id: 'childDiv',* ***class****: 'divStyle'});*

 *// Включаем созданный элемент в DOM браузера, а именно в существующий <div>,*

 *// с помощью метода Element.insert*

 *Element.insert($('parrentDiv'),newElement);*

**Объектно-ориентированное программирование**

Prototype также добавляет поддержку более традиционного объектно-ориентированного программирования.

Для создания нового класса используется метод Class.create(). Классу присваивается прототип prototype, который выступает в качестве основы для каждого экземпляра класса. Старые классы могут быть расширены новыми с помощью Object.extend.

* *// создания нового класса в стиле prototype 1.6*

 ***var*** *FirstClass = Class.create({*

 *// Инициализация конструктора*

 *initialize:* ***function*** *() {* ***this****.data = "Hello World"; }*

 *});*

 *// создания нового класса в стиле prototype 1.5*

 ***var*** *DataWriter = Class.create();*

 *DataWriter.prototype = {*

 *printData:* ***function*** *() { document.write(****this****.data); }*

 *};*

 *Object.extend(DataWriter, FirstClass);*