14 - Работа с протоколом HTTP.

1. Введение

Протокол передачи гипертекста (Hypertext Transfer Protocol, HTTP) определяет, как веб-браузеры должны запрашивать документы, как они должны передавать информацию веб-серверам и как веб-серверы должны отвечать на эти запросы и передачи. Очевидно, что веб-браузеры очень много работают с протоколом HTTP. Тем не менее, как правило, сценарии не работают с протоколом HTTP, когда пользователь щелкает на ссылке, отправляет форму или вводит URL в адресной строке. Однако JavaScript-код способен работать с протоколом HTTP. HTTP-запросы могут инициироваться, когда сценарий устанавливает значение свойства location объекта Window или вызывает метод submit() объекта Form. В обоих случаях браузер загружает в окно новую страницу. Такого рода взаимодействие с протоколом HTTP может быть вполне оправданным в веб-страницах, но в этой лекции мы будем говорить совсем о другом. Мы рассмотрим такое взаимодействие JavaScript-кода с веб-сервером, при котором веб-браузер не перезагружает содержимое окна.

Ajax – это аббревиатура от Asynchronous JavaScript and XML (асинхронный JavaScript и XML). Термин Ajax описывает архитектуру веб-приложений, отличительной чертой которых является работа с протоколом HTTP. Ключевой особенностью Ajax-приложения является использование протокола HTTP для инициации обмена данными с веб-сервером без необходимости перезагружать страницу. Возможность избежать перезагрузки страницы (что было привычным на первых этапах развития Всемирной паутины) позволяет создавать веб-приложения, близкие по своему поведению к обычным приложениям. Веб-приложение может использовать технологии Ajax для передачи на сервер результатов взаимодействия с пользователем или для ускорения запуска приложения, отображая сначала простую страницу и подгружая дополнительные данные и компоненты страницы по мере необходимости.

Термин Comet описывает похожую архитектуру веб-приложений, также использующих протокол HTTP. В некотором смысле архитектура Comet является обратной по отношению к Ajax: в архитектуре Comet не клиент, а сервер инициирует взаимодействие, асинхронно отсылая сообщения клиенту. Если веб-приложению потребуется отвечать на сообщения, отправляемые сервером, оно сможет использовать приемы Ajax для отправки или запроса данных. В архитектуре Ajax клиент «вытягивает» данные с сервера. В архитектуре Comet сервер «навязывает» данные клиенту.

XMLHttpRequest - определяющий прикладной интерфейс для работы с протоколом HTTP. Этот интерфейс обеспечивает возможность выполнять POST-запросы в дополнение к обычным GET-запросам и может возвращать ответ веб-сервера синхронно или асинхронно, в виде простого текста или в виде объекта Document. Несмотря на свое название, объект XMLHttpRequest не ограничивается использованием XML-документов – он в состоянии принимать любые текстовые документы.

2. Использование объекта XMLHttpRequest

Прикладной интерфейс к протоколу HTTP в браузерах определяется в виде класса XMLHttpRequest. Каждый экземпляр этого класса представляет единственную пару запрос/ответ, а свойства и методы объекта позволяют определять параметры запроса и извлекать данные из ответа. Объект XMLHttpRequest поддерживается веб-браузерами уже довольно давно, также ведутся работы над проектом стандарта «XMLHttpRequest

Level 2». Первое, что обычно необходимо сделать при использовании этого прикладного интерфейса к протоколу HTTP, создать экземпляр объекта XMLHttp-Request:

```
var request = new XMLHttpRequest();
```

Допустимо повторно использовать уже имеющийся экземпляр объекта XMLHttp-Request, но следует иметь в виду, что в этом случае будет прервано выполнение запроса, уже отправленного объектом.

HTTP-запрос состоит из четырех частей:

• метод HTTP-запроса или тип «операции»
• запрашиваемый URL-адрес
• необязательные заголовки запроса, которые могут включать информацию для аутентификации
• необязательное тело запроса

HTTP-ответ, возвращаемый сервером, состоит из трех частей:

• числовое и текстовое значение, определяющее код состояния, свидетельствующий об успехе или об ошибке
• набор заголовков ответа
• тело ответа

3. Выполнение запроса

Следующий этап после создания объекта XMLHttpRequest – определение параметров HTTP-запроса **вызовом метода open()** объекта XMLHttpRequest, которому передаются две обязательные части запроса: метод и URL:

```
request.open("GET", // Запрос типа HTTP GET
"data.csv"); // на получение содержимого по этому URL-адресу
```

Первый аргумент метода open() определяет HTTP-метод или операцию. Методы «GET» и «POST» поддерживаются всеми браузерами. Метод «GET» используется для «обычных» запросов и соответствует случаю, когда URL-адрес полностью определяет запрашиваемый ресурс. Он используется, когда запрос не имеет побочных эффектов. Метод «POST» обычно используется HTML-формами. Он включает в тело запроса дополнительные данные (данные формы), и эти данные часто сохраняются в базе данных на стороне сервера (побочный эффект). В ответ на повторяющиеся запросы POST к одному и тому же URL сервер может возвращать разные ответы.

Помимо запросов «GET» и «POST», спецификация XMLHttpRequest также позволяет передавать методу open() строки «DELETE», «HEAD», «OPTIONS» и «PUT» в первом аргументе.

Вторым аргументом методу open() передается URL-адрес запрашиваемого ресурса. Это относительный URL-адрес документа, содержащего сценарий, в котором вызывается метод open(). Если указать абсолютный адрес, то в общем случае протокол, доменное имя и порт должны совпадать с аналогичными параметрами адреса документа: нарушение политики общего происхождения обычно вызывает ошибку. (Однако

спецификация «XMLHttpRequest Level 2» допускает выполнение запросов к другим серверам, если сервер явно разрешил это.)

Третий этап в выполнении запроса – установка заголовков запроса, если это необходимо. Запросы POST, например, требуют, чтобы был определен заголовок «Content-Type», определяющий MIME-тип тела запроса:

```
request.setRequestHeader("Content-Type", "text/plain");
```

Если вызвать метод setRequestHeader() несколько раз с одним и тем же заголовком, новое значение не заменит прежнее: вместо этого в HTTP-запрос будет вставлено несколько копий заголовка или один заголовок с несколькими значениями.

Нельзя определять собственные заголовки «Content-Length», «Date», «Referer» и «User-Agent»: объект XMLHttpRequest добавляет их автоматически и не позволяет подделывать их. Аналогично объект XMLHttpRequest автоматически обрабатывает cookies и срок поддержки открытого соединения, определяет кодировку символов и выполняет кодирование сообщений.

Последний этап в процедуре выполнения HTTP-запроса с помощью объекта XMLHttpRequest – передача необязательного тела запроса и отправка его серверу. Делается это с помощью **метода send():**

```
request.send(null);
```

GET-запросы не имеют тела, и в этом случае можно передать методу значение null или вообще опустить аргумент. POST-запросы обычно имеют тело, и оно должно соответствовать заголовку «Content-Type», установленному с помощью метода setRequestHeader().

Пример далее демонстрирует использование всех методов объекта XMLHttpRequest, описанных выше. Он отправляет серверу текстовую строку методом POST и игнорирует ответ, возвращаемый сервером.

```
function postMessage(msg) {
    var request = new XMLHttpRequest(); // Новый запрос
    request.open("POST", "/log.php"); // серверному сценарию методом POST

    // Отправить простое текстовое сообщение в теле запроса
    // Тело запроса - простой текст
    request.setRequestHeader("Content-Type", "text/plain;charset=UTF-8");

    request.send(msg); // msg как тело запроса
    // Запрос выполнен. Мы игнорируем возможный ответ или ошибку.
}
```

Обратите внимание, что вызов метода send() в примере инициирует запрос и затем возвращает управление: он не блокируется в ожидании ответа от сервера. HTTP-ответы практически всегда обрабатываются асинхронно.

4. Получение ответа

Полный HTTP-ответ содержит код состояния, набор заголовков ответа и тело ответа. Все это доступно в виде свойств и методов объекта XMLHttpRequest:

• Свойства status и statusText возвращают код состояния HTTP в числовом и текстовом виде. Эти свойства хранят стандартные HTTP-значения, такие как 200 и «OK» в случае успешного выполнения запроса или 404 и «Not Found» при попытке обратиться к ресурсу, отсутствующему на сервере.

• Заголовки ответа можно получить с помощью методов getResponseHeader() и getAllResponseHeaders().

• Тело ответа в текстовом виде доступно через свойство responseText или в виде объекта Document через свойство responseXML. (Выбор такого имени свойства объясняется историческими причинами: фактически оно предназначено для работы с XHTML- и XML-документами, но спецификация «XHR2» определяет, что оно также должно работать с обычными HTML-документами.)

Обычно объект XMLHttpRequest используется в асинхронном режиме и метод send() возвращает управление сразу же после отправки запроса, поэтому методы и свойства, перечисленные выше, не могут использоваться до фактического получения ответа. Чтобы определить момент получения ответа, необходимо обрабатывать **событие «readystatechange»** (или событие «progress», определяемое новой спецификацией «XHR2»), возбуждаемое в объекте XMLHttpRequest. Но, чтобы понять, как обрабатывать это событие, необходимо сначала разобраться со свойством readyState.

Свойство readyState – это целочисленное значение, определяющее код состояния HTTP-запроса; его возможные значения:

UNSENT	0	Метод open() еще не был вызван
OPENED	1	Метод open() был вызван
HEADERS_RECEIVED	2	Были получены заголовки
LOADING	3	Идет прием тела ответа
DONE	4	Прием ответа завершен

Идентификаторы, указанные в первой колонке, – это константы, определяемые конструктором XMLHttpRequest. Эти константы являются частью спецификации XMLHttpRequest.

Теоретически событие «readystatechange» генерируется всякий раз, когда изменяется значение свойства readyState. На практике же событие может не возбуждаться, когда свойство readyState получает значение 0 или 1. Оно часто возбуждается при вызове метода send(), даже при том, что свойство readyState по-прежнему содержит значение OPENED. Некоторые браузеры возбуждают событие множество раз для состояния LOADING, чтобы обеспечить обратную связь. Все браузеры возбуждают событие «readystatechange», когда завершается прием ответа сервера и свойство readyState получает значение 4. Так как это событие может возбуждаться еще до завершения приема ответа, обработчики события «readystatechange» всегда должны проверять значение свойства readyState.

Чтобы обрабатывать события «readystatechange», нужно присвоить функцию обработчика события свойству onreadystatechange объекта XMLHttpRequest. Можно также воспользоваться методом addEventListener(), но обычно для обработки запроса

бывает вполне достаточно одного обработчика, поэтому проще установить свойство onreadystatechange.

Пример ниже определяет функцию getText(), которая демонстрирует особенности обработки событий «readystatechange». Обработчик события сначала проверяет завершение запроса. После этого он проверяет код состояния ответа и убеждается в успешном выполнении. Затем он извлекает заголовок «Content-Type», чтобы убедиться, что получен ответ ожидаемого типа. Если выполняются все три условия, он передает тело ответа (в виде текста) указанной функции обратного вызова.

```
// Выполняет запрос HTTP GET содержимого
указанного URL-адреса.
// После успешного получения ответа
проверяет, содержит ли он простой текст,
// и передает его указанной функции обратного
вызова
function getText(url, callback) {
      var request = new XMLHttpRequest(); // Создать новый
запрос
      request.open("GET", url); // Указать URL-адрес ресурса
      request.onreadystatechange = function() { // Определить
обработчик
            // Если запрос был выполнен успешно
            if (request.readyState === 4 && request.status === 200) {
                  var type = request.getResponseHeader("Content-Type");
                  if (type.match(/^text/)) // Убедиться, что это
текст
                  callback(request.responseText); // Передать
функции
            }
      };
      request.send(null); // Отправить запрос
}
```

5. Получение синхронного ответа

Сама природа HTTP-ответа предполагает их асинхронную обработку. Тем не менее объект XMLHttpRequest поддерживает возможность получения ответов в синхронном режиме. Если в третьем аргументе передать методу open() значение false, выполнение метода send() будет заблокировано до завершения запроса. В этом случае отпадает необходимость использовать обработчик события: после того как метод send() вернет управление, можно будет сразу же проверить свойства status и responseText объекта XMLHttpRequest.

```
      // Выполняет синхронный запрос HTTP GET
      содержимого по указанному URL-адресу.
      // Возвращает текст ответа. Возбуждает
      исключение в случае неудачи
      // или если ответ не является текстом.
      function getTextSync(url) {
            var request = new XMLHttpRequest(); // Создать новый
```

```
запрос
        request.open("GET", url, false); // false - синхронный
режим
        request.send(null); // Отправить запрос

        // Возбудить исключение, если код
состояния не равен 200
        if (request.status !== 200) throw new
Error(request.statusText);
        // Возбудить исключение, если ответ
имеет недопустимый тип
        var type = request.getResponseHeader("Content-Type");

        if (!type.match(/^text/)) {
            throw new Error("Ожидался текстовый
ответ; получен: " + type);
        }
        return request.responseText;
    }
```

Синхронные запросы выглядят весьма заманчиво, однако использовать их нежелательно. Интерпретатор JavaScript на стороне клиента выполняется в единственном потоке, и когда метод send() блокируется, это обычно приводит к зависанию пользовательского интерфейса всего броузера. Если сервер, к которому выполнено подключение, отвечает на вопросы с задержкой, браузер пользователя будет зависать.

6. Запросы с данными в формате JSON

Использование формата представления данных форм в теле POST-запросов является распространенным соглашением, но не является обязательным требованием протокола HTTP. В последние годы в роли формата обмена данными во Всемирной паутине все большей популярностью пользуется формат JSON. Выполнение запроса HTTP POST с данными в формате JSON:

```
function postJSON(url, data, callback) {
    var request = new XMLHttpRequest();
    request.open("POST", url); // Методом POST на указ. url

    request.onreadystatechange = function() { // Простой
обработчик
        if (request.readyState === 4 && callback){// При
получении ответа
            callback(request); // вызвать указанную
функцию
        }
    };

    request.setRequestHeader("Content-Type", "application/json");
    request.send(JSON.stringify(data));
}
```

7. Прерывание запросов и предельное время ожидания

Выполнение HTTP-запроса можно **прерывать вызовом метода abort()** объекта XMLHttpRequest. Метод abort() доступен во всех версиях объекта XMLHttpRequest, и согласно спецификации «XHR2» вызов метода abort() генерирует **событие «abort».**

Основная причина для вызова метода abort() – появление необходимости отменить запрос, превышение предельного времени ожидания или если ответ становится ненужным. Допустим, что объект XMLHttpRequest используется для запроса подсказки в механизме автодополнения для текстового поля ввода. Если пользователь успеет ввести в поле новый символ еще до того, как подсказка будет получена с сервера, надобность в этой подсказке отпадает и запрос можно прервать.

Спецификация «XHR2» определяет **свойство timeout, в котором указывается промежуток времени в миллисекундах**, после которого запрос автоматически будет прерван, а также определяет **событие «timeout»,** которое должно генерироваться (вместо события «abort») по истечении установленного промежутка времени.