13 - Обработка событий.

1. Введение

http://www.webdevout.net/browser-support-dom - поддержка DOM

Клиентские программы на языке JavaScript основаны на модели программирования, когда выполнение программы управляется событиями. При таком стиле программирования веб-браузер генерирует событие, когда с документом или некоторым его элементом что-то происходит. Например, веб-браузер генерирует событие, когда завершает загрузку документа, когда пользователь наводит указатель мыши на гиперссылку или нажимает клавишу на клавиатуре. Если JavaScript-приложение интересует определенный тип события для определенного элемента документа, оно может **зарегистрировать одну или более функций**, которая будет вызываться при возникновении этого события.

Обратите внимание, что слово событие не является техническим термином, требующим строгого определения.

События – это просто некоторые происшествия, о которых броузер извещает программу.

Тип события – это строка, определяющая тип происшествия. Тип «mousemove», например, означает, что пользователь переместил указатель мыши. Тип «keydown» означает, что была нажата клавиша на клавиатуре. А тип «load» означает, что завершилась загрузка документа (или какого-то другого ресурса) из сети. Поскольку тип события – это просто строка, его иногда называют именем события.

Цель события – это объект, в котором возникло событие или с которым это событие связано. Когда говорят о событии, обычно упоминают тип и цель события. Например: событие «load» объекта Window или событие «click» элемента <button>. Самыми типичными целями событий в клиентских приложениях на языке JavaScript являются объекты Window, Document и Element, но некоторые типы событий могут происходить и в других типах объектов.

Обработчик события – это функция, которая обрабатывает, или откликается на событие. Приложения должны зарегистрировать свои функции обработчиков событий в веб-браузере, указав тип события и цель. Когда в указанном целевом объекте возникнет событие указанного типа, броузер вызовет обработчик. Когда обработчики событий вызываются для какого-то объекта, мы иногда говорим, что броузер «возбудил» или «сгенерировал» событие.

Объект события (Event) – это объект, связанный с определенным событием и содержащий информацию об этом событии. Объекты событий передаются функции обработчика события в виде аргумента Все объекты событий имеют свойство type, определяющее тип события, и свойство target, определяющее цель события. Для каждого типа события в связанном объекте события определяется набор свойств. Например, объект, связанный с событиями от мыши, включает координаты указателя мыши, а объект, связанный с событиями от клавиатуры, содержит информацию о нажатой клавише и о нажатых клавишах-модификаторах.

Распространение события – это процесс, в ходе которого броузер решает, в каких объектах следует вызвать обработчики событий. В случае событий,

предназначенных для единственного объекта (таких как событие «load» объекта Window), надобность в их распространении отсутствует. Однако, когда некоторое событие возникает в элементе документа, **оно распространяется, или «всплывает»**, вверх по дереву документа. Иногда удобнее бывает зарегистрировать единственный обработчик события в объекте Document или в другом контейнерном элементе, чем выполнять регистрацию во всех интересующих нас элементах. Обработчик события может прервать дальнейшее распространение события, чтобы оно прекратило всплытие и не привело к вызову обработчиков вмещающих элементов. Делается это вызовом метода или установкой свойства объекта события.

Еще одна форма распространения событий, **которая называется перехватом со бытия**, позволяет специально зарегистрированным обработчикам или контейнерным элементам «перехватывать» события до того, как они достигнут фактической цели. Возможность перехватывать события от мыши совершенно необходима при обработке событий буксировки объектов мышью.

Для некоторых событий предусматриваются связанные с ними действия по умолчанию. Например, для события «click», возникающего в гиперссылке, по умолчанию предусматривается операция перехода по ссылке и загрузки новой страницы. Обработчики могут **предотвратить выполнение действий** по умолчанию, вернув соответствующее значение, вызвав метод или установив свойство объекта события.

2. Типы событий

События, которые вам чаще всего придется использовать в своих веб-приложениях, обычно будут относиться к категории давно существующих и поддерживаемых всеми браузерами: это события для работы с мышью, с клавиатурой, с HTML-формами и с объектом Window. События могут быть сгруппированы в некоторые обобщенные категории, знание которых поможет вам понять длинный перечень событий, следующий далее:

Аппаратно-зависимые события ввода. События из этой категории непосредственно связаны с конкретными устройствами ввода, такими как мышь или клавиатура. В эту категорию входят такие старые события, как «mousedown», «mousemove», «mouseup», «keydown», «keypress» и «keyup», а также новые события, имеющие отношение к сенсорным устройствам, такие как «touchmove» и «gesturechange».

Аппаратно-независимые события ввода. Эти события ввода не связаны непосредственно с каким-то определенным устройством. Например, событие «click» указывает на то, что была активирована ссылка или кнопка (или другой элемент документа). Это событие часто порождается щелчком мыши, но его причиной также может быть нажатие клавиши на клавиатуре или (для сенсорных устройств) некоторое перемещение по экрану. Событие «textinput» является аппаратно-независимой альтернативой событию «keypress» и поддерживает не только ввод с клавиатуры, но и такие альтернативы, как вставка из буфера обмена и ввод рукописного текста.

События пользовательского интерфейса – это высокоуровневые события, которые часто возникают в элементах HTML-форм, составляющих пользовательский интерфейс веб-приложения. В эту категорию входит событие «focus» (возникающее, когда текстовое поле получает фокус ввода), событие «change» (возникающее, когда

пользователь изменяет значение, отображаемое элементом формы) и событие «submit» (возникающее, когда пользователь щелкает на кнопке отправки формы).

События изменения состояния. Некоторые события не связаны непосредственно с деятельностью пользователя, но имеют отношение к выполнению сетевых операций браузером и указывают на переход к другому этапу операции или на изменение состояния. Событие «load», которое возбуждается в объекте Window по окончании загрузки документа, является, пожалуй, наиболее часто используемым типом событий из этой категории. Еще одним представителем из этой категории является событие «DOMContentLoaded». Механизм управления историей посещений, определяемый стандартом HTML5, возбуждает событие «popstate» в ответ на нажатие клавиши Back (Назад) броузера. Прикладной интерфейс автономных веб-приложений, описываемый стандартом HTML5, включает события «online» и «offline». Позже мы изучим, как пользоваться событием «readystatechange», сообщающем о получении данных с сервера. Аналогично новый API чтения локальных файлов, выбранных пользователем, использует события, такие как «loadstart», «progress» и «loadend», для отправки асинхронных извещений о ходе выполнения операций ввода-вывода.

Прикладные события. Некоторые прикладные интерфейсы, определяемые стандартом HTML5 и связанными с ним спецификациями, включают собственные типы событий. Интерфейс drag-and-drop определяет такие события, как «dragstart», «dragenter», «dragover» и «drop». Приложения, обеспечивающие поддержку буксировки элементов мышью, должны реализовать обработку некоторых из этих событий. Элементы <video> и <audio>, определяемые стандартом HTML5, добавляют длинный список связанных с ними типов событий, таких как «waiting», «playing», «seeking», «volumechange» и т.д. Эти события обычно представляют интерес только для веб-приложений, определяющих собственные элементы управления проигрыванием аудио- и видеороликов.

Обработчики ошибок и событий от таймеров. Обработчики ошибок и событий от таймеров являются частью асинхронной модели программирования в клиентском JavaScript и похожи на обработчики обычных событий.

3. События форм

Формы и гиперссылки стали первыми элементами веб-страниц, возможность управления которыми была реализована в начале развития Всемирной паутины и JavaScript. Это означает, что события форм являются наиболее устойчивыми и хорошо поддерживаемыми из всех типов событий. Элементы <form> возбуждают события **«submit»**, при отправке формы, и **«reset»,** перед сбросом формы в исходное состояние. Элементы форм, внешним видом напоминающие кнопки (включая радиокнопки и флажки), возбуждают события **«click»**, когда пользователь взаимодействуют с ними. Элементы форм, позволяющие вводить некоторую информацию, обычно возбуждают события **«change»**, когда пользователь изменяет их состояние, вводя текст, выбирая элемент списка или отмечая флажок. Элементы форм откликаются на изменение фокуса ввода, возбуждая события **«focus» и «blur»** при получении и утере фокуса ввода. Для событий «submit» и «reset» предусматриваются действия по умолчанию, выполнение которых можно отменить в обработчиках событий, как и в случае некоторых событий «click». Все события форм всплывают, кроме событий «focus» и «blur».

4. События объекта Window

События объекта Window представляют происшествия, имеющие отношение к самому окну браузера, а не к определенному содержимому документа, отображаемому в окне. (Однако некоторые из этих событий имеют имена, совпадающие с именами событий, возбуждаемых для элементов документа.) Самым важным из этих событий является событие **«load»**: оно возбуждается сразу после того, как будут загружены и отображены документ и все внешние ресурсы (такие как изображения). Альтернативами событию «load» являются события «DOMContentLoaded» и «readystatechange»: они возбуждаются сразу же, как только документ и его элементы будут готовы к выполнению операций, но до того, как полностью будут загружены внешние ресурсы.

Событие «unload» является противоположностью событию «load»: оно возбуждается, когда пользователь покидает документ. Обработчик события **«unload»** можно использовать, чтобы сохранить информацию о состоянии, но в нем нельзя отменить переход на другую страницу. Событие **«beforeunload»** похоже на событие «unload», но оно дает возможность узнать у пользователя, действительно ли он желает покинуть вашу веб-страницу. Если обработчик события «beforeunload» вернет строку, эта строка будет выведена в диалоге подтверждения перед тем, как будет загружена новая страница; этот диалог даст пользователю возможность отменить переход и остаться на текущей странице.

События «focus» и «blur», описанные выше вместе с другими событиями элементов форм, также поддерживаются объектом Window: они возбуждаются, когда текущее окно браузера получает или теряет фокус ввода.

Наконец, **события «resize» и «scroll»** возбуждаются в объекте Window, когда выполняется изменение размеров или прокрутка окна браузера. События «scroll» могут также возбуждать все прокручиваемые элементы документа.

5. События мыши

События от мыши возбуждаются, когда пользователь перемещает указатель мыши или выполняет щелчок. Эти события генерируются в наиболее глубоко вложенных элементах, над которыми находится указатель мыши, но они всплывают вверх по дереву документа. Объект события, передаваемый обработчикам событий от мыши, имеет свойства, позволяющие узнать координаты указателя, состояние кнопок мыши, а также состояние клавиш-модификаторов на момент возникновения события. Свойства clientX и clientY определяют положение указателя мыши в системе координат окна. Свойства button и which позволяют узнать, какая кнопка была нажата. Свойства altKey, ctrlKey, metaKey и shiftKey получают значение true, если в момент возникновения события удерживалась нажатой соответствующая клавиша-модификатор. А свойство detail для события «click» указывает, был ли выполнен одинарный, двойной или тройной щелчок.

Событие «mousemove» генерируется всякий раз, когда пользователь перемещает указатель мыши. Это событие возбуждается очень часто, поэтому его обработчики не должны выполнять тяжелые вычисления. События **«mousedown»** и **«mouseup»** генерируются, когда пользователь нажимает и отпускает кнопку мыши. Зарегистрировав обработчик события «mousedown», который регистрирует обработчик события **«mousemove»,** можно организовать определение и обработку ситуаций буксировки элементов мышью. При этом необходимо обеспечить возможность перехватывать события от мыши, чтобы продолжать получать события «mousemove», даже когда указатель мыши выходит за пределы элемента, где была начата буксировка.

После последовательности событий «mousedown» и «mouseup» броузер генерирует событие **«click».** Событие «click» было описано выше, как аппаратно-независимое событие форм, но на самом деле оно может генерироваться для любого элемента документа, а не только для элементов форм, и вместе с ним обработчику передается объект события со всеми дополнительными полями, описанными выше. Если вы дважды щелкнете мышью на строке (в течение достаточно короткого промежутка времени), второй щелчок сгенерирует событие **«dblclick»**. Когда щелчок выполняется правой кнопкой мыши, браузеры часто выводят контекстное меню. Вообще, прежде чем вывести контекстное меню, они возбуждают событие «contextmenu», и, если отменить это событие в обработчике, можно предотвратить появление меню. Кроме того, это наиболее простой способ организовать обработку щелчка правой кнопкой мыши.

Когда пользователь перемещает указатель мыши так, что он оказывается над другим элементом, броузер возбуждает событие **«mouseover»** для этого элемента. Когда указатель мыши покидает границы элемента, браузер генерирует для него событие **«mouseout»**. Объект события для этих событий будет иметь свойство relatedTarget, определяющее другой элемент, вовлеченный в переход. События «mouseover» и «mouseout» всплывают, как и все остальные описанные здесь события от мыши. Иногда это оказывается неудобно, потому что в обработчике события «mouseout» приходится проверять, действительно ли указатель мыши покинул данный элемент или он просто переместился из одного дочернего элемента в другой. По этой причине поддерживается невсплывающие версии этих событий, известные как «mouseenter» и «mouseleave», эти события стандартизованы в спецификации «DOM Level 3 Events».

Когда пользователь вращает колесико мыши, браузеры генерируют событие «mousewheel». Объект события, передаваемый вместе с этими событиями, включает свойства, позволяющие узнать, на какое расстояние и в каком направлении было повернуто колесико.

6. События клавиатуры

Когда веб-браузер получает фокус ввода, он начинает генерировать события всякий раз, когда пользователь нажимает и отпускает клавиши на клавиатуре. Нажатия горячих комбинаций, имеющих значение для операционной системы или самого браузера, часто «съедаются» операционной системой или браузером и не передаются обработчикам событий на JavaScript. События от клавиатуры генерируются в любом элементе документа, обладающем фокусом ввода, и всплывают вверх до объектов документа и окна. Если ни один элемент не обладает фокусом ввода, события возбуждаются непосредственно в объекте документа. Обработчикам событий от клавиатуры передается объект события, имеющий свойство keyCode, позволяющее узнать, какая клавиша была нажата или отпущена. В дополнение к свойству keyCode объект события от клавиатуры также имеет свойства altKey, ctrlKey, metaKey и shiftKey, описывающие состояние клавиш-модификаторов.

События **«keydown» и «keyup»** являются низкоуровневыми событиями от клавиатуры: они генерируются, когда производится нажатие или отпускание клавиши (даже если это клавиша-модификатор). Когда событие «keydown» генерируется нажатием клавиши, соответствующей печатаемому символу, после события «keydown», но перед событием «keyup» дополнительно генерируется событие **«keypress».** (В случае если

клавиша удерживается в нажатом состоянии настолько долго, что начинается автоповтор символа, перед событием «keyup» будет сгенерировано множество событий «keypress».)

7. События, генерируемые сенсорными экранами мобильных устройств.

Широкое распространение мощных мобильных устройств, особенно устройств с сенсорными экранами, потребовало создания новых категорий событий. Во многих случаях события от сенсорных экранов отображаются на традиционные типы событий, такие как «click» и «scroll». Но не все виды взаимодействий с пользовательским интерфейсом через сенсорный экран можно имитировать с помощью мыши, и не все прикосновения к такому экрану можно интерпретировать, как события от мыши.

Браузер Safari генерирует события для жестов масштабирования и вращения из двух пальцев. Событие «gesturestart» возбуждается, когда начинается выполнение жеста, а событие «gestureend» по его окончании. Между этими двумя событиями генерируется последовательность событий «gesturechange», позволяющих отслеживать выполнение жеста. Объект события, передаваемый вместе с этими событиями, имеет числовые свойства scale и rotation. Когда палец касается экрана, генерируется событие **«touchstart».** Когда палец перемещается, генерируется событие **«touchmove».** А когда палец отнимается от экрана, генерируется событие **«touchend».** В отличие от событий мыши, события прикосновений не несут непосредственной информации о координатах прикосновения. Вместо этого в объекте события, который поставляется вместе с событием прикосновения, имеется свойство changedTouches. Это свойство хранит объект, подобный массиву, каждый элемент которого описывает позицию прикосновения. Событие **«orientationchanged»** генерируется в объекте Window устройствами, позволяющими пользователям поворачивать экран для перехода из книжной ориентации в альбомную.

8. Регистрация обработчиков событий

Существует два основных способа регистрации обработчиков событий. Первый, появившийся на раннем этапе развития Всемирной паутины, заключается в **установке свойства объекта** или элемента документа, являющегося целью события. Второй способ, более новый и более универсальный, заключается в передаче **обработчика методу объекта** или элемента.

Установка свойств обработчиков событий. Самый простой способ зарегистрировать обработчик события заключается в том, чтобы присвоить свойству целевого объекта события желаемую функцию обработчика. По соглашению свойства обработчиков событий имеют имена, состоящие из слова «on», за которым следует имя события: onclick, onchange, onload, onmouseover и т.д.

```
// Присвоить функцию свойству onload объекта Window.
// Функция - обработчик события: она вызывается, когда документ будет загружен.
window.onload = function() {
    // Отыскать элемент <form>
    var elt = document.getElementById("shipping_address");

    // Зарегистрировать обработчик события, который будет вызываться
    // непосредственно перед отправкой формы.
```

```
        elt.onsubmit = function() { return validate(this); }
    }
```

Такой способ регистрации обработчиков событий поддерживается во всех браузерах для всех часто используемых типов событий. Вообще говоря, все прикладные интерфейсы, получившие широкую поддержку, которые определяют свои события, позволяют регистрировать обработчики установкой свойств обработчиков событий. Недостаток использования свойств обработчиков событий состоит в том, что они проектировались в предположении, что цели событий будут иметь не более одного обработчика для каждого типа событий.

Установка атрибутов обработчиков событий. Свойства обработчиков событий в элементах документа можно также устанавливать, определяя значения атрибутов в соответствующих HTML-тегах. В этом случае значение атрибута должно быть строкой программного кода на языке JavaScript. Этот программный код должен быть не полным объявлением функции обработчика события, а только ее телом. То есть реализация обработчика события в разметке HTML не должна заключаться в фигурные скобки и предваряться ключевым словом function. Например:

```
<button onclick="alert('Спасибо');">Щелкните здесь</button>
```

Если значение HTML-атрибута обработчика события состоит из нескольких Java-Script-инструкций, они должны отделяться точками с запятой либо значение атрибута должно располагаться в нескольких строках.

Некоторые типы событий предназначены для броузера в целом, а не для какогото конкретного элемента документа. Обработчики таких событий в языке JavaScript регистрируются в объекте Window. В разметке HTML они должны помещаться в тег <body>, но броузер зарегистрирует их в объекте Window.

addEventListener(). В стандартной модели событий, поддерживаемой всеми браузерами, целью события может быть любой объект – включая объекты Window и Document и все объекты Elements элементов документа – определяющий метод с именем addEventListener(), с помощью которого можно регистрировать обработчики событий для этой цели. Метод addEventListener() принимает три аргумента.

Первый – тип события, для которого регистрируется обработчик. Тип(или имя) события должен быть строкой и не должен включать префикс «on», используемый при установке свойств обработчиков событий. Вторым аргументом методу addEventListener() передается функция, которая должна вызываться при возникновении события указанного типа. В последнем аргументе методу addEventListener() передается логическое значение. Обычно в этом аргументе передается значение false. Если передать в нем значение true, функция будет зарегистрирована как перехватывающий обработчик и будет вызываться в другой фазе распространения события. Следующий фрагмент регистрирует два обработчика события «click» в элементе <button>. Обратите внимание на различия двух используемых приемов:

```
<button id="mybutton">Щелкни на мне</button>
<script>
    var b = document.getElementById("mybutton");
    b.onclick = function() { alert("Спасибо, что
щелкнули на мне!"); };
```

```
        b.addEventListener("click", function() { alert("Еще раз спасибо!"); }, false);
    </script>
```

Вызов метода addEventListener() со строкой «click» в первом аргументе никак не влияет на значение свойства onclick. Во фрагменте, приведенном выше, щелчок на кнопке приведет к выводу двух диалогов alert(). Но важнее то, что метод addEventListener() можно вызвать несколько раз и зарегистрировать с его помощью несколько функций-обработчиков для одного и того же типа события в том же самом объекте. При появлении события в объекте будут вызваны все обработчики, зарегистрированные для этого типа события, в порядке их регистрации.

Парным к методу addEventListener() является метод **removeEventListener()**, который принимает те же три аргумента, но не добавляет, а удаляет функцию-обработчик из объекта. Это часто бывает удобно, когда необходимо зарегистрировать временный обработчик события, а затем удалить его в какой-то момент. Например, при получении события «mousedown» может потребоваться зарегистрировать временный перехватывающий обработчик событий «mousemove» и «mouseup», чтобы можно было наблюдать за тем, как пользователь выполняет буксировку объектов мышью, а по событию «mouseup» эти обработчики могут удаляться. В такой ситуации реализация удаления обработчиков событий может иметь вид, как показано ниже:

```
document.removeEventListener("mousemove", handleMouseMove, true);
document.removeEventListener("mouseup", handleMouseUp, true);
```

9. Аргумент обработчика событий.

При вызове обработчика событий ему передается объект события в виде единственного аргумента. Свойства объекта события содержат дополнительную информацию о событии. Свойство type, например, определяет тип возникшего события.

10. Контекст обработчиков событий

Когда обработчик событий регистрируется установкой свойства, это выглядит как определение нового метода элемента документа:

```
e.onclick = function() { /* реализация обработчика */ };
```

Поэтому нет ничего удивительного, что обработчики событий вызываются как методы объектов, в которых они определены. То есть в теле обработчика событий ключевое слово this ссылается на цель события. В обработчиках ключевое слово this ссылается на целевой объект, даже когда они были зарегистрированы с помощью метода addEventListener().

11. Область видимости обработчика событий

Подобно всем функциям в языке JavaScript, обработчики событий имеют лексическую область видимости. Они выполняются в той области видимости, в какой были определены, а не в той, где они были вызваны, и имеют доступ ко всем локальным переменным в этой области видимости.

12. Возвращаемые значения обработчиков

Значение, возвращаемое обработчиком события, зарегистрированным установкой свойства объекта или с помощью HTML-атрибута, следует учитывать. Обычно возвращаемое значение false сообщает браузеру, что он не должен выполнять действия, предусмотренные для этого события по умолчанию. Например, обработчик onclick кнопки отправки формы может вернуть false, чтобы предотвратить отправку формы браузером. (Это может пригодиться, если ввод пользователя не прошел проверку на стороне клиента.) Аналогично обработчик события onkeypress поля ввода может фильтровать ввод с клавиатуры, возвращая false при вводе недопустимых символов.

Также важно значение, возвращаемое обработчиком onbeforeunload объекта Window. Это событие генерируется, когда браузер выполняет переход на другую страницу. Если этот обработчик вернет строку, она будет выведена в модальном диалоге, предлагающем пользователю подтвердить свое желание покинуть страницу.

Важно понимать, что учитываются значения, возвращаемые обработчиками событий, только если обработчики зарегистрированы посредством установки свойств. Далее мы увидим, что обработчики, зарегистрированные с помощью addEventListener() вместо этого должны вызывать метод preventDefault() или устанавливать свойство returnValue объекта события.

13. Порядок вызова

Для элемента документа или другого объекта можно зарегистрировать более одного обработчика одного и того же типа события. При возникновении этого события броузер вызовет все обработчики в порядке, определяемом следующими правилами:

• В первую очередь вызываются обработчики, зарегистрированные установкой свойства объекта или с помощью HTML-атрибута, если таковые имеются.

• Затем вызываются обработчики, зарегистрированные с помощью метода addEventListener(), в порядке их регистрации.

14. Распространение событий

Когда целью события является объект Window или какой-то другой самостоятельный объект (такой как XMLHttpRequest), броузер откликается на событие простым вызовом соответствующего обработчика в этом объекте. Однако когда целью события является объект Document или элемент Element документа, ситуация несколько осложняется.

После вызова обработчиков событий, зарегистрированных в целевом элементе, большинство событий «всплывают» вверх по дереву DOM. В результате вызываются обработчики в родителе целевого элемента. Затем вызываются обработчики, зарегистрированные в родителе родителя целевого элемента. Так продолжается, пока не будет достигнут объект Document и затем объект Window. Способность событий всплывать обеспечивает возможность реализации альтернативы множеству обработчиков, зарегистрированных в отдельных элементах документа: можно зарегистрировать единственный обработчик в общем элементе-предке и обрабатывать события в нем (**“Делегирование”**). Например, вместо того чтобы регистрировать обработчик события «change» в каждом элементе формы, его можно зарегистрировать в единственном элементе <form>.

Способностью всплывать обладает большинство событий, возникающих в элементах документа. Заметным исключением являются события «focus», «blur» и «scroll».

Всплытие – это третья «фаза» распространения события. **Вызов обработчика события в целевом объекте** – это вторая фаза. Первая фаза протекает еще до вызова обработчиков целевого объекта и называется **фазой «перехвата»**. Напомню, что метод addEventListener() имеет третий аргумент, в котором принимает логическое значение. Если передать в этом аргументе значение true, обработчик события будет зарегистрирован как перехватывающий обработчик для вызова в первой фазе распространения события. Фаза всплытия событий реализована во всех браузерах, включая IE, и в ней участвуют все обработчики, независимо от того, как они были зарегистрированы (если только они не были зарегистрированы как перехватывающие обработчики). В фазе перехвата, напротив, участвуют только обработчики, зарегистрированные с помощью метода addEventListener(), когда в третьем аргументе ему было передано значение true.

Фаза перехвата напоминает фазу всплытия, только событие распространяется в обратном направлении. В первую очередь вызываются перехватывающие обработчики объекта Window, затем вызываются перехватывающие обработчики объекта Document, затем обработчики объекта body и так далее, вниз по дереву DOM, пока не будут вызваны перехватывающие обработчики родителя целевого объекта. Перехватывающие обработчики, зарегистрированные в самом целевом объекте, не вызываются.

Наличие фазы перехвата позволяет обнаруживать события еще до того, как они достигнут своей цели. Перехватывающий обработчик может использоваться для отладки или для фильтрации событий, чтобы в комплексе с приемом отмены события, описываемом ниже, предотвратить вызов обработчиков в целевом объекте. Одной из типичных областей применения фазы перехвата является реализация буксировки элементов мышью, когда событие перемещения указателя мыши должен обрабатывать буксируемый объект, а не документ, в пределах которого осуществляется буксировка.

15. Отмена событий.

В браузерах, поддерживающих метод addEventListener(), отменить выполнение действий по умолчанию можно также вызовом метода **preventDefault()** объекта события. В следующем фрагменте демонстрируется обработчик вымышленного события, который использует отмену события:

```
function cancelHandler(event) {

    if (event.preventDefault) event.preventDefault(); // Стандартный прием
    return false; // Для обработчиков, зарегистрированных как свойства

}
```

Текущий проект модуля «DOM Events» определяет в объекте Event свойство с именем defaultPrevented. Оно пока поддерживается не всеми браузерами, но суть его в том, что при обычных условиях оно имеет значение false и принимает значение true только в случае вызова метода preventDefault().

Отмена действий, по умолчанию связанных с событием, – это лишь одна из разновидностей отмены события. Имеется также возможность остановить распространение события. В браузерах, поддерживающих метод addEventListener(), объект события имеет метод **stopPropagation(),** вызов которого прерывает дальнейшее распространение события. Если в том же целевом объекте будут зарегистрированы другие обработчики этого события, то остальные обработчики все равно будут вызваны, но никакие другие обработчики событий в других объекта не будут вызваны после вызова метода stopPropagation(). Метод stopPropagation() можно вызвать в любой момент в ходе распространения события. Он действует и в фазе перехвата, и в вызове обработчика целевого объекта, и в фазе всплытия.

Текущий проект спецификации «DOM Events» определяет в объекте Event еще один метод – метод с именем **stopImmediatePropagation()**. Подобно методу stopPropagation(), он предотвращает распространение события по любым другим объектам. Но кроме того, он также предотвращает вызов любых других обработчиков событий, зарегистрированных в том же объекте.