Клавиатура: keydown и keyup

Прежде чем перейти к клавиатуре, обратите внимание, что на современных устройствах есть и другие способы «ввести что-то». Например, распознавание речи (это особенно актуально на мобильных устройствах) или Копировать/Вставить с помощью мыши.

Поэтому, если мы хотим корректно отслеживать ввод в поле `<input>`, то одних клавиатурных событий недостаточно. Существует специальное событие `input`, чтобы отслеживать любые изменения в поле `<input>`. И оно справляется с такой задачей намного лучше. Мы рассмотрим его позже в главе События: change, input, cut, copy, paste.

События клавиатуры же должны использоваться, если мы хотим обрабатывать взаимодействие пользователя именно с клавиатурой (в том числе виртуальной). К примеру, если нам нужно реагировать на стрелочные клавиши `Up` и `Down` или горячие клавиши (включая комбинации клавиш).

Тестовый стенд

Для того, чтобы лучше понять, как работают события клавиатуры, можно использовать тестовый стенд ниже.

Сфокусируйтесь на поле и нажмите какую-нибудь клавишу.

Результат script.js style.css index.html

Предотвратить действие по умолчанию для: keydown keyup

Игнорировать: keydown keyup

Сфокусируйтесь на поле и нажмите какую-нибудь клавишу.

Клавишу нажимать тут

Очистить

События keydown и keyup

Событие `keydown` происходит при нажатии клавиши, а `keyup` – при отпускании.

event.code и event.key

Свойство `key` объекта события позволяет получить символ, а свойство `code` – «физический код клавиши».

К примеру, одну и ту же клавишу `Z` можно нажать с клавишей `Shift` и без неё. В результате получится два разных символа: `z` в нижнем регистре и `Z` в верхнем регистре.

Свойство `event.key` – это непосредственно символ, и он может различаться. Но `event.code` всегда будет тот же:

Клавиша	event.key	event.code
`Z`	`z` (нижний регистр)	`KeyZ`
`Shift+Z`	`Z` (Верхний регистр)	`KeyZ`

Если пользователь работает с разными языками, то при переключении на другой язык символ изменится с `"Z"` на совершенно другой. Получившееся станет новым значением `event.key`, тогда как `event.code` останется тем же: `"KeyZ"`.

«KeyZ» и другие клавишные коды

У каждой клавиши есть код, который зависит от её расположения на клавиатуре. Подробно о клавишных кодах можно прочитать в спецификации о кодах событий UI.

Например:

- Буквенные клавиши имеют коды по типу `"Key<буква>"`: `"KeyA"`, `"KeyB"` и т.д.
- Коды числовых клавиш строятся по принципу: `"Digit<число>"`: `"Digit0"`, `"Digit1"` и т.д.
- Код специальных клавиш – это их имя: `"Enter"`, `"Backspace"`, `"Tab"` и т.д.

Существует несколько широко распространённых раскладок клавиатуры, и в спецификации приведены клавишные коды к каждой из них.

Можно их прочитать в разделе спецификации, посвящённом буквенно-цифровым клавишам или просто нажмите нужную клавишу на тестовом стенде выше и посмотрите.

Регистр важен: `"KeyZ"`, а не `"keyZ"`

Выглядит очевидно, но многие всё равно ошибаются.

Пожалуйста, избегайте опечаток: правильно `KeyZ`, а не `keyZ`. Условие `event.code=="keyZ"` работать не будет: первая буква в слове `"Key"` должна быть заглавная.

А что, если клавиша не буквенно-цифровая? Например, `Shift` или `F1`, или какая-либо другая специальная клавиша? В таких случаях значение свойства `event.key` примерно тоже, что и у `event.code`:

Клавиша	event.key	event.code
`F1`	`F1`	`F1`
`Backspace`	`Backspace`	`Backspace`
`Shift`	`Shift`	`ShiftRight` или `ShiftLeft`

Обратите внимание, что `event.code` точно указывает, какая именно клавиша нажата. Так, большинство клавиатур имеют по две клавиши `Shift`: слева и справа. `event.code` уточняет, какая именно из них была нажата, в то время как `event.key` сообщает о «смысле» клавиши: что вообще было нажато (`Shift`).

Допустим, мы хотим обработать горячую клавишу `Ctrl+Z` (или `Cmd+Z` для Mac). Большинство текстовых редакторов к этой комбинации подключают действие «Отменить». Мы можем поставить обработчик событий на `keydown` и проверять, какая клавиша была нажата.

Здесь возникает дилемма: в нашем обработчике стоит проверять значение `event.key` или `event.code`?

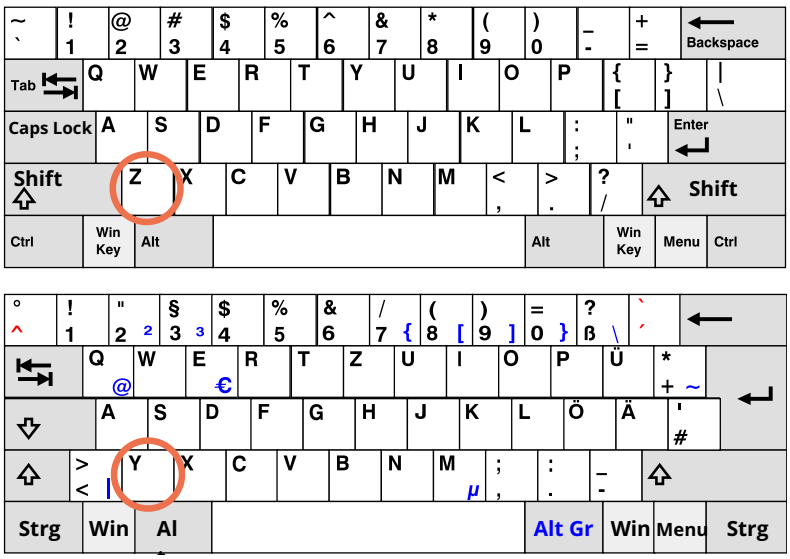
С одной стороны, значение `event.key` – это символ, он изменяется в зависимости от языка, и если у пользователя установлено в ОС несколько языков, и он переключается между ними, нажатие на одну и ту же клавишу будет давать разные символы. Так что имеет смысл проверять `event.code`, ведь его значение всегда одно и тоже.

Вот пример кода:

```
document.addEventListener('keydown', function(event) {
  if (event.code == 'KeyZ' && (event.ctrlKey || event.m
    alert('Отменить!')
  }
});
```

С другой стороны, с `event.code` тоже есть проблемы. На разных раскладках к одной и той же клавише могут быть привязаны разные символы.

Например, вот схема стандартной (US) раскладки («QWERTY») и под ней немецкой («QWERTZ») раскладки (из Википедии):



Для одной и той же клавиши в американской раскладке значение `event.code` равно «Z», в то время как в немецкой «Y».

Буквально, для пользователей с немецкой раскладкой `event.code` при нажатии на `Y` будет равен `KeyZ`.

Если мы будем проверять в нашем коде `event.code == 'KeyZ'`, то для людей с немецкой раскладкой такая проверка сработает, когда они нажимают `Y`.

Звучит очень странно, но это и в самом деле так. В спецификации прямо упоминается такое поведение.

Так что `event.code` может содержать неправильный символ при неожиданной раскладке. Одни и те же буквы на разных раскладках могут сопоставляться с разными физическими клавишами, что приводит к разным кодам. К счастью, это происходит не со всеми кодами, а с несколькими, например `KeyA`, `KeyQ`, `KeyZ` (как мы уже видели), и не происходит со специальными клавишами, такими как `Shift`. Вы можете найти полный список проблемных кодов в спецификации.

Чтобы отслеживать символы, зависящие от раскладки, `event.key` надёжнее.

С другой стороны, преимущество `event.code` заключается в том, что его значение всегда остаётся неизменным, будучи привязанным к физическому местоположению клавиши, даже если пользователь меняет язык. Так что горячие клавиши, использующие это свойство, будут работать даже в случае переключения языка.

Хотим поддерживать клавиши, меняющиеся при раскладке? Тогда `event.key` – верный выбор.

Или мы хотим, чтобы горячая клавиша срабатывала даже после переключения на другой язык? Тогда `event.code` может быть лучше.

Автоповтор

При долгом нажатии клавиши возникает автоповтор: `keydown` срабатывает снова и снова, и когда клавишу отпускают, то отрабатывает `keyup`. Так что ситуация, когда много `keydown` и один `keyup`, абсолютно нормальна.

Для событий, вызванных автоповтором, у объекта события свойство `event.repeat` равно `true`.

Действия по умолчанию

Действия по умолчанию весьма разнообразны, много чего можно инициировать нажатием на клавиатуре.

Для примера:

- Появление символа (самое очевидное).
- Удаление символа (клавиша `Delete`).
- Прокрутка страницы (клавиша `PageDown`).
- Открытие диалогового окна браузера «Сохранить» (`Ctrl+S`)
- ...и так далее.

Предотвращение стандартного действия с помощью `event.preventDefault()` работает практически во всех сценариях, кроме тех, которые происходят на уровне операционной системы. Например, комбинация `Alt+F4` инициирует закрытие браузера в Windows, что бы мы ни делали в JavaScript.

Для примера, `<input>` ниже ожидает телефонный номер, так что ничего кроме чисел, `+`, `()` или `-` принято не будет:

```
<script>
function checkPhoneKey(key) {
  return (key >= '0' && key <= '9') || key == '+' || ke
}
</script>
<input onkeydown="return checkPhoneKey(event.key)" plac
```

Введите телефон

Заметьте, что специальные клавиши, такие как `Backspace`, `Left`, `Right`, `Ctrl+V`, в этом поле для ввода не работают. Это побочный эффект чересчур жёсткого фильтра `checkPhoneKey`.

Добавим ему немного больше свободы:

```
<script>
function checkPhoneKey(key) {
  return (key >= '0' && key <= '9') || key == '+' || ke
    key == 'ArrowLeft' || key == 'ArrowRight' || key ==
}
</script>
<input onkeydown="return checkPhoneKey(event.key)" plac
```

Введите телефон

Теперь стрелочки и удаление прекрасно работают.

...Впрочем, мы всё равно можем ввести в `<input>` что угодно с помощью правого клика мыши и пункта «Вставить» контекстного меню. Так что такой фильтр не обладает 100% надёжностью. Мы можем просто оставить всё как есть, потому что в большинстве случаев это работает. Альтернатива – отслеживать событие `input`, оно генерируется после любых изменений в

поле `<input>`, и мы можем проверять новое значение и подчёркивать/изменять его, если оно не подходит.

«Дела минувших дней»

В прошлом существовало также событие `keypress`, а также свойства `keyCode`, `charCode`, `which` у объекта события.

Но количество браузерных несовместимостей при работе с ними было столь велико, что у разработчиков спецификации не было другого выхода, кроме как объявить их устаревшими и создать новые, современные события (которые и описываются в этой главе). Старый код ещё работает, так как браузеры продолжают поддерживать и `keypress`, и `keyCode` с `charCode`, и `which`, но более нет никакой необходимости в их использовании.

Итого

Нажатие клавиши всегда генерирует клавиатурное событие, будь то буквенно-цифровая клавиша или специальная типа `Shift` или `Ctrl` и т.д. Единственным исключением является клавиша `Fn`, которая присутствует на клавиатуре некоторых ноутбуков. События на клавиатуре для неё нет, потому что она обычно работает на уровне более низком, чем даже ОС.

События клавиатуры:

- `keydown` – при нажатии на клавишу (если клавиша остаётся нажатой, происходит автоповтор),
- `keyup` – при отпускании клавиши.

Главные свойства для работы с клавиатурными событиями:

- `code` – «код клавиши» (`"KeyA"`, `"ArrowLeft"` и так далее), особый код, привязанный к физическому расположению клавиши на клавиатуре.
- `key` – символ (`"A"`, `"a"` и так далее), для не буквенно-цифровых групп клавиш (таких как `Esc`) обычно имеет то же значение, что и `code`.

В прошлом события клавиатуры иногда использовались для отслеживания ввода данных пользователем в полях формы. Это ненадёжно, потому как ввод данных не обязательно может осуществляться с помощью клавиатуры. Существуют события `input` и `change` специально для обработки ввода (рассмотренные позже в главе События: change, input, cut, copy, paste). Они срабатывают в результате любого ввода, включая Копировать/Вставить мышью и распознавание речи.

События клавиатуры же должны использоваться только по назначению – для клавиатуры. Например, чтобы реагировать на горячие или специальные клавиши.

Задачи

Отследить одновременное нажатие

важность: 5

Создайте функцию `runOnKeys(func, code1, code2, ... code_n)`, которая запускает `func` при одновременном нажатии клавиш с кодами `code1`, `code2`, ..., `code_n`.

Например, код ниже выведет `alert` при одновременном нажатии клавиш `"Q"` и `"W"` (в любом регистре, в любой раскладке)

```
runOnKeys(
  () => alert("Привет!"),
  "KeyQ",
  "KeyW"
);
```

Демо в новом окне

решение

Комментарии

перед тем как писать...