12th November 2019

Focusing: focus/blur

An element receives a focus when the user either clicks on it or uses the `Tab` key on the keyboard. There's also an `autofocus` HTML attribute that puts the focus into an element by default when a page loads and other means of getting a focus.

Focusing on an element generally means: "prepare to accept the data here", so that's the moment when we can run the code to initialize the required functionality.

The moment of losing the focus ("blur") can be even more important. That's when a user clicks somewhere else or presses `Tab` to go to the next form field, or there are other means as well.

Losing the focus generally means: "the data has been entered", so we can run the code to check it or even to save it to the server and so on.

There are important peculiarities when working with focus events. We'll do the best to cover them further on.

Events focus/blur

The `focus` event is called on focusing, and `blur` – when the element loses the focus.

Let's use them for validation of an input field.

In the example below:

- The `blur` handler checks if the field the email is entered, and if not – shows an error.
- The `focus` handler hides the error message (on `blur` it will be checked again):

```
<style>
  .invalid { border-color: red; }
  #error { color: red }
</style>

Your email please: <input type="email" id="input">

<div id="error"></div>

<script>
input.onblur = function() {
  if (!input.value.includes('@')) { // not email
    input.classList.add('invalid');
    error.innerHTML = 'Please enter a correct email.'
  }
};

input.onfocus = function() {
  if (this.classList.contains('invalid')) {
```

```
    // remove the "error" indication, because the user wants to re-enter some
    this.classList.remove('invalid');
    error.innerHTML = "";
  }
};
</script>
```

Your email please:

Modern HTML allows us to do many validations using input attributes: `required`, `pattern` and so on. And sometimes they are just what we need. JavaScript can be used when we want more flexibility. Also we could automatically send the changed value to the server if it's correct.

Methods focus/blur

Methods `elem.focus()` and `elem.blur()` set/unset the focus on the element.

For instance, let's make the visitor unable to leave the input if the value is invalid:

```
<style>
  .error {
    background: red;
  }
</style>

Your email please: <input type="email" id="input">
<input type="text" style="width:220px" placeholder="make email invalid and tr

<script>
  input.onblur = function() {
    if (!this.value.includes('@')) { // not email
      // show the error
      this.classList.add("error");
      // ...and put the focus back
      input.focus();
    } else {
      this.classList.remove("error");
    }
  };
</script>
```

Your email please: make email invalid and try to focus he

It works in all browsers except Firefox (bug).

If we enter something into the input and then try to use `Tab` or click away from the `<input>`, then `onblur` returns the focus back.

Please note that we can't "prevent losing focus" by calling `event.preventDefault()` in `onblur`, because `onblur` works *after* the element lost the focus.

JavaScript-initiated focus loss

A focus loss can occur for many reasons.

One of them is when the visitor clicks somewhere else. But also JavaScript itself may cause it, for instance:

- An `alert` moves focus to itself, so it causes the focus loss at the element (`blur` event), and when the `alert` is dismissed, the focus comes back (`focus` event).
- If an element is removed from DOM, then it also causes the focus loss. If it is reinserted later, then the focus doesn't return.

These features sometimes cause `focus/blur` handlers to misbehave – to trigger when they are not needed.

The best recipe is to be careful when using these events. If we want to track user-initiated focus-loss, then we should avoid causing it ourselves.

Allow focusing on any element: tabindex

By default many elements do not support focusing.

The list varies a bit between browsers, but one thing is always correct: `focus/blur` support is guaranteed for elements that a visitor can interact with: `<button>` , `<input>` , `<select>` , `<a>` and so on.

From the other hand, elements that exist to format something, such as `<div>` , `<span>` , `<table>` – are unfocusable by default. The method `elem.focus()` doesn't work on them, and `focus/blur` events are never triggered.

This can be changed using HTML-attribute `tabindex` .

Any element becomes focusable if it has `tabindex` . The value of the attribute is the order number of the element when `Tab` (or something like that) is used to switch between them.

That is: if we have two elements, the first has `tabindex="1"` , and the second has `tabindex="2"` , then pressing `Tab` while in the first element – moves the focus into the second one.

The switch order is: elements with `tabindex` from `1` and above go first (in the `tabindex` order), and then elements without `tabindex` (e.g. a regular `<input>`).

Elements with matching `tabindex` are switched in the document source order (the default order).

There are two special values:

- `tabindex="0"` puts an element among those without `tabindex` . That is, when we switch elements, elements with `tabindex=0` go after elements with `tabindex ≥ 1` .

  Usually it's used to make an element focusable, but keep the default switching order. To make an element a part of the form on par with `<input>` .

- `tabindex="-1"` allows only programmatic focusing on an element. The `Tab` key ignores such elements, but method `elem.focus()` works.

For instance, here's a list. Click the first item and press `Tab` :

```
Click the first item and press Tab. Keep track of the order. Please note that
<ul>
  <li tabindex="1">One</li>
  <li tabindex="0">Zero</li>
  <li tabindex="2">Two</li>
  <li tabindex="-1">Minus one</li>
</ul>

<style>
  li { cursor: pointer; }
  :focus { outline: 1px dashed green; }
</style>
```

Click the first item and press Tab. Keep track of the order. Please note that many subsequent Tabs can move the focus out of the iframe with the example.

- One
- Zero
- Two
- Minus one

The order is like this: `1 - 2 - 0`. Normally, `<li>` does not support focusing, but `tabindex` full enables it, along with events and styling with `:focus`.

The property `elem.tabIndex` works too

We can add `tabindex` from JavaScript by using the `elem.tabIndex` property. That has the same effect.

Delegation: focusin/focusout

Events `focus` and `blur` do not bubble.

For instance, we can't put `onfocus` on the `<form>` to highlight it, like this:

```
<!-- on focusing in the form -- add the class -->
<form onfocus="this.className='focused'">
  <input type="text" name="name" value="Name">
  <input type="text" name="surname" value="Surname">
</form>

<style> .focused { outline: 1px solid red; } </style>
```

Name Surname

The example above doesn't work, because when user focuses on an `<input>`, the `focus` event triggers on that input only. It doesn't bubble up. So `form.onfocus` never triggers.

There are two solutions.

First, there's a funny historical feature: `focus/blur` do not bubble up, but propagate down on the capturing phase.

This will work:

```
<form id="form">
  <input type="text" name="name" value="Name">
  <input type="text" name="surname" value="Surname">
</form>

<style> .focused { outline: 1px solid red; } </style>

<script>
  // put the handler on capturing phase (last argument true)
  form.addEventListener("focus", () => form.classList.add('focused'), true);
  form.addEventListener("blur", () => form.classList.remove('focused'), true)
</script>
```

Name Surname

Second, there are `focusin` and `focusout` events – exactly the same as `focus/blur`, but they bubble.

Note that they must be assigned using `elem.addEventListener`, not `on<event>`.

So here's another working variant:

```
<form id="form">
  <input type="text" name="name" value="Name">
  <input type="text" name="surname" value="Surname">
</form>

<style> .focused { outline: 1px solid red; } </style>

<script>
  form.addEventListener("focusin", () => form.classList.add('focused'));
  form.addEventListener("focusout", () => form.classList.remove('focused'));
</script>
```

Name Surname

Summary

Events `focus` and `blur` trigger on focusing/losing focus on the element.

Their specials are:

- They do not bubble. Can use capturing state instead or `focusin/focusout`.
- Most elements do not support focus by default. Use `tabindex` to make anything focusable.

The current focused element is available as `document.activeElement`.

Tasks

Editable div

importance: 5

Create a `<div>` that turns into `<textarea>` when clicked.

The textarea allows to edit the HTML in the `<div>`.

When the user presses `Enter` or it loses focus, the `<textarea>` turns back into `<div>`, and its content becomes HTML in `<div>`.

Demo in new window

Open a sandbox for the task.

solution

Edit TD on click

importance: 5

Make table cells editable on click.

- On click – the cell should became “editable” (textarea appears inside), we can change HTML. There should be no resize, all geometry should remain the same.
- Buttons OK and CANCEL appear below the cell to finish/cancel the editing.
- Only one cell may be editable at a moment. While a `<td>` is in “edit mode”, clicks on other cells are ignored.
- The table may have many cells. Use event delegation.

The demo:

Click on a table cell to edit it. Press OK or CANCEL when you finish.

***Bagua* Chart: Direction, Element, Color, Meaning**

Northwest Metal Silver Elders	**North** Water Blue Change	**Northeast** Earth Yellow Direction
West Metal Gold Youth	**Center** All Purple Harmony	**East** Wood Blue Future
Southwest Earth Brown Tranquility	**South** Fire Orange Fame	**Southeast** Wood Green Romance

Open a sandbox for the task.

solution

Keyboard-driven mouse

importance: 4

Focus on the mouse. Then use arrow keys to move it:

Demo in new window

P.S. Don't put event handlers anywhere except the `#mouse` element. P.P.S. Don't modify HTML/CSS, the approach should be generic and work with any element.

Open a sandbox for the task.

solution

Previous lesson Next lesson

Share

Tutorial map

Comments

- If you have suggestions what to improve - please submit a GitHub issue or a pull request instead of commenting.
- If you can't understand something in the article – please elaborate.
- To insert a few words of code, use the `<code>` tag, for several lines – use `<pre>`, for more than 10 lines – use a sandbox (plnkr, JSBin, codepen…)