HTML5 输入类型

在上一篇文章中，我们研究了元素，涵盖了自 HTML 早期以来可用的属性 `<input>` 的原始值。 `type` 现在我们将详细了解较新的表单控件的功能，包括一些新的输入类型，它们是在 HTML5 中添加的以允许收集特定类型的数据。

先决条件：	基本的计算机知识、以及对 HTML 的 基本 理解。
客观的：	了解可用于创建本机表单控件的较新输入类型值，以及如何使用 HTML 实现它们。

注意： 本文中讨论的大多数功能在浏览器中得到广泛支持。我们会记录任何例外情况。如果您想了解有关浏览器支持的更多详细信息，您应该查阅我们的HTML 表单元素参考，尤其是我们广泛的<input> 类型参考。

由于 HTML 表单控件的外观可能与设计者的规格有很大不同，因此 Web 开发人员有时会构建自己的自定义表单控件。我们在高级教程中对此进行了介绍：如何构建自定义表单小部件。

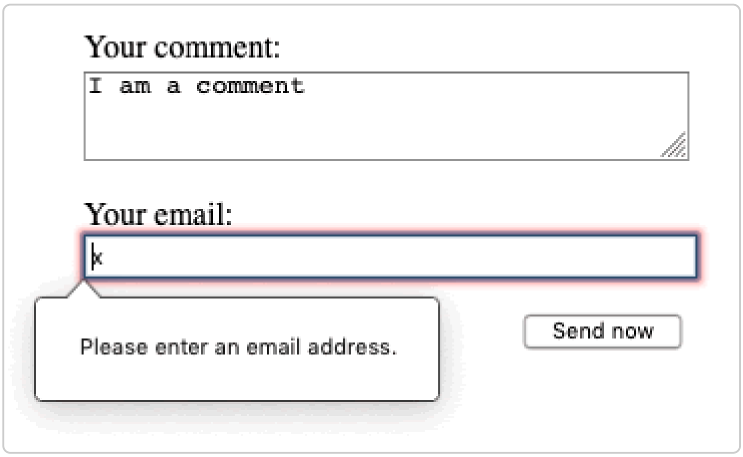
电子邮件地址字段

这种类型的字段是使用属性值设置 `email` 的 `type` ：

```
<input type="email" id="email" name="email" />
```

使用此 `type` 功能时，用户需要在字段中输入有效的电子邮件地址。任何其他内容都会导致浏览器在提交表单时显示错误。您可以在下面的屏幕截图中

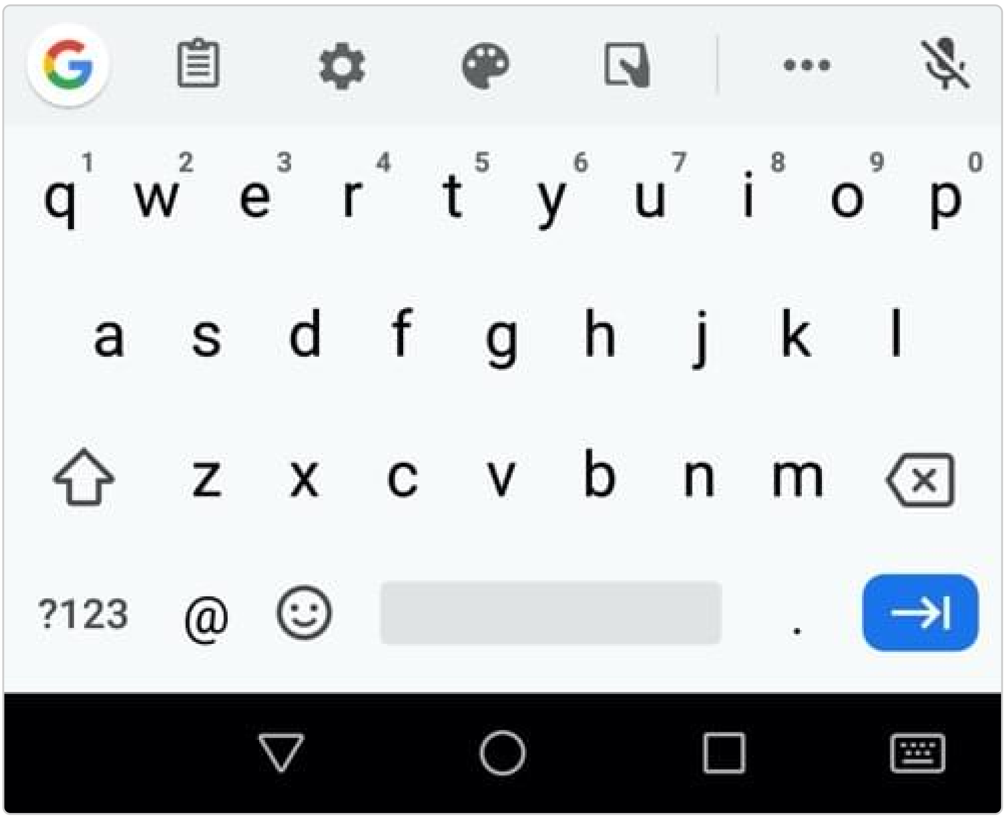
看到这一点。



multiple 您还可以将该属性与输入类型结合使用 email ，以允许在同一输入中输入多个电子邮件地址（以逗号分隔）：

```
<input type="email" id="email" name="email" multiple />
```

在某些设备上——尤其是智能手机等带有动态键盘的触摸设备——可能会出现一个不同的虚拟键盘，它更适合输入电子邮件地址，包括密钥 @ 。有关示例，请参见下面的 Firefox for Android 键盘屏幕截图：



> **注意：您可以在基本输入示例** 中找到基本文本输入类型的示例（另请参阅源代码）。

这是使用这些较新的输入类型的另一个很好的理由，可以改善这些设备用户的用户体验。

客户端验证

正如您在上面看到的， `email` 与其他较新的 `input` 类型一起，提供了内置的*客户端*错误验证，由浏览器在数据发送到服务器之前执行。它*有助于*指导用户准确填写表格，并且可以节省时间：立即知道您的数据不正确是很有用的，而不必等待往返服务器。

但*不应将其视为*详尽的安全措施！*您的应用程序应始终对服务器端*和客户端的任何表单提交数据执行安全检查，因为客户端验证太容易关闭，因此恶意用户仍然可以轻松地将不良数据发送到您的服务器。阅读网站安全以了解*可能*发生的情况；实现服务器端验证有点超出了本模块的范围，但您应该牢记这一点。

a@b 请注意，根据默认提供的约束，这是一个有效的电子邮件地址。这是因为 `email` 输入类型默认允许 Intranet 电子邮件地址。实现不同的验证行为，可以使用 `pattern` 属性，也可以自定义错误信息；我们将在稍后的客户端表单验证文章中讨论如何使用这些功能。

> **注意：** 如果输入的数据不是邮箱地址， `:invalid` 伪类会匹配， `validityState.typeMismatch` 属性会返回 `true` 。

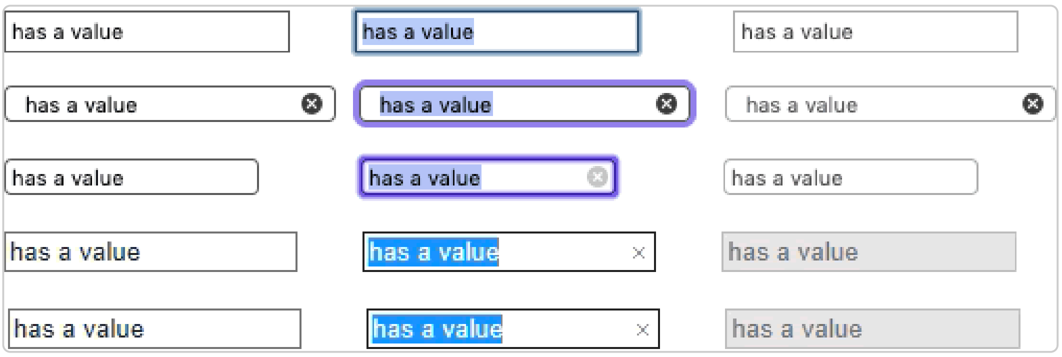
搜索字段

搜索字段旨在用于在页面和应用程序上创建搜索框。 `search` 这种类型的字段是通过使用属性的值来设置的 `type` ：

```
<input type="search" id="search" name="search" />
```

`text` 字段和字段之间的主要区别 `search` 在于浏览器如何设置其外观样式。通常， `search` 字段呈现为圆角；他们有时还会显示一个“Ⓧ”，单击时会清除任何值的字段。此外，在带有动态键盘的设备上，键盘的回车键可能显示为“**搜索**”，或显示放大镜图标。

下面的屏幕截图显示了 Firefox 71、Safari 13 和 macOS 上的 Chrome 79 以及 Windows 10 上的 Edge 18 和 Chrome 79 中的非空搜索字段。请注意，清除图标仅在该字段有值时出现，除此之外在 Safari 中，它仅在字段获得焦点时显示。



另一个值得注意的功能是字段的值 `search` 可以自动保存并重复使用，以提供跨同一网站的多个页面的自动完成；在大多数现代浏览器中，这往往会自动发生。

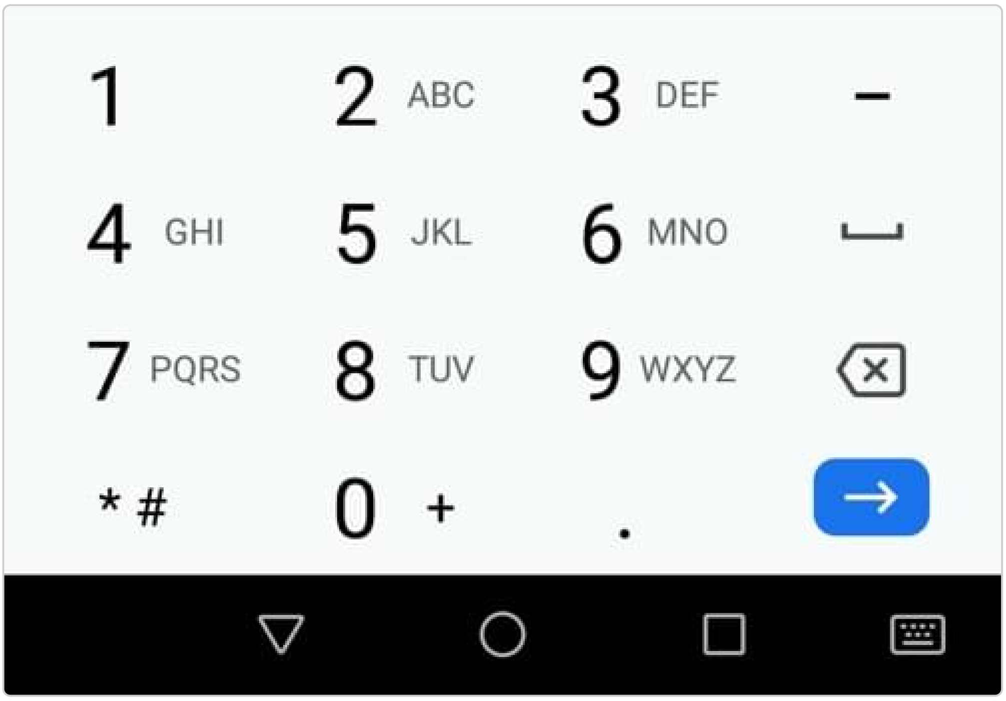
电话号码字段

`tel` 可以使用as 属性的值创建一个用于填写电话号码的特殊字段 [`type`]：

```
<input type="tel" id="tel" name="tel" />
```

当通过带有动态键盘的触摸设备访问时，大多数设备在遇到时都会显示数字

以下 Firefox for Android 键盘截图提供了一个示例：



由于世界各地的电话号码格式多种多样，这种类型的字段不会对用户输入的值施加任何限制（这意味着它可能包括字母等）。

正如我们前面提到的，该 `pattern` 属性可用于强制执行约束，您将在客户端表单验证中了解这一点。

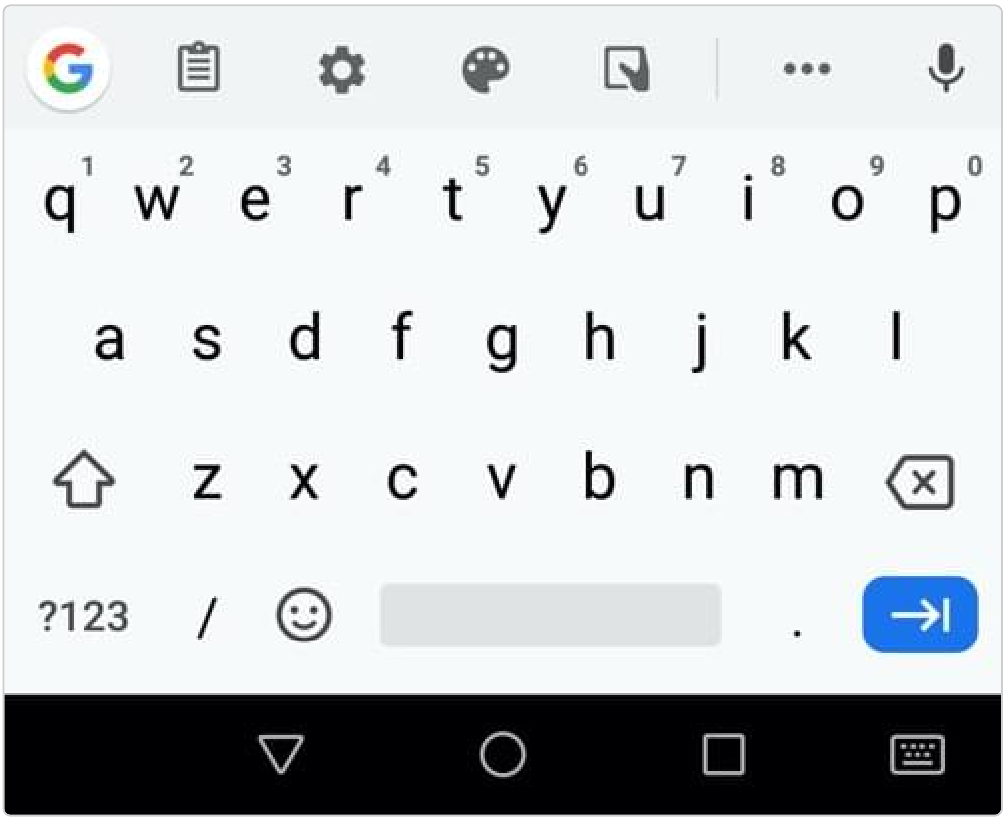
网址字段

`url` 可以使用属性值创建用于输入 URL 的特殊类型的字段 `type`：

```
<input type="url" id="url" name="url" />
```

它向该字段添加了特殊的验证约束。 `http:` 如果未输入任何协议（例如），或者 URL 格式不正确，浏览器将报告错误。在具有动态键盘的设备上，默认键盘通常会将部分或全部冒号、句号和正斜杠显示为默认键。

请参阅下面的示例（在 Firefox for Android 上拍摄）：

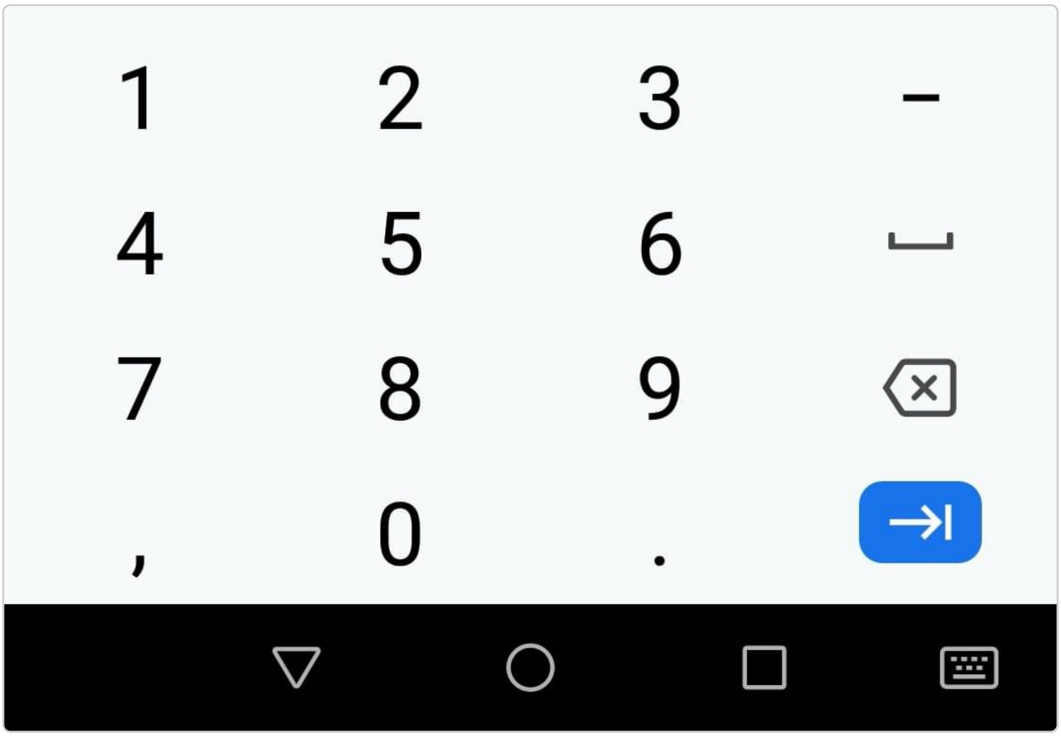


注意：仅仅因为 URL 格式正确并不一定意味着它指的是实际存在的位置！

数字字段

可以使用of创建用于输入数字的控件。这个控件看起来像一个文本字段，但只允许浮点数，并且通常以微调器的形式提供按钮来增加和减少控件的值。在具有动态键盘的设备上，通常会显示数字键盘。 `<input>` `type` `number`

以下屏幕截图（来自 Firefox for Android）提供了一个示例：



对于输入类型，您可以通过设置和属性 `number` 来限制允许的最小值和最大值。 [`min`](#) [`max`](#)

You can also use the `step` attribute to set the increment increase and decrease caused by pressing the spinner buttons. By default, the number input type only validates if the number is an integer. To allow float numbers, specify `step="any"`. If omitted, the `step` value defaults to `1`, meaning only whole numbers are valid.

Let's look at some examples. The first one below creates a number control whose value is restricted to any value between `1` and `10`, and whose increase and decrease buttons change its value by `2`.

```
<input type="number" name="age" id="age" min="1" max="10"
step="2" />
```

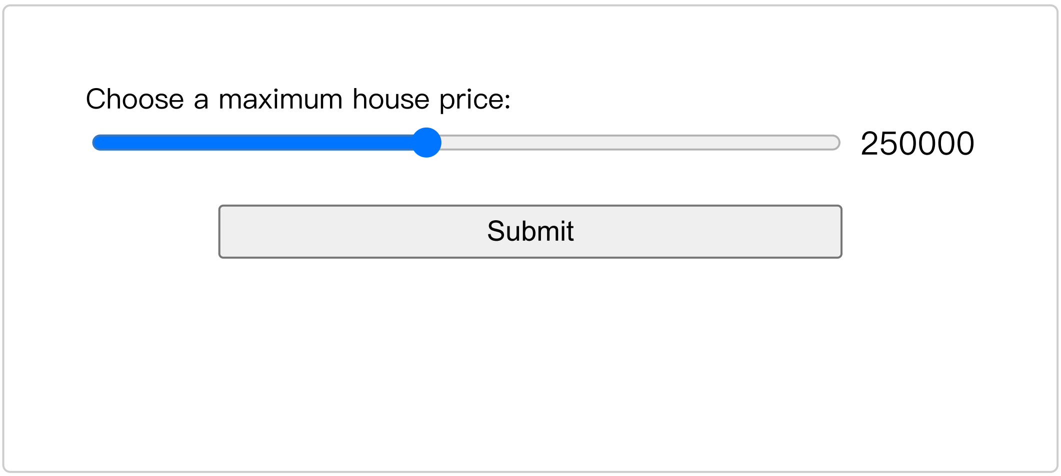
The second one creates a number control whose value is restricted to any value between `0` and `1` inclusive, and whose increase and decrease buttons change its value by `0.01`.

```html
<input type="number" name="change" id="pennies" min="0" max="1"
step="0.01" />
```

The `number` input type makes sense when the range of valid values is limited, for example a person's age or height. If the range is too large for incremental increases to make sense (such as USA ZIP codes, which range from `00001` to `99999`), the `tel` type might be a better option; it provides the numeric keypad while forgoing the number's spinner UI feature.

Slider controls

Another way to pick a number is to use a **slider**. You see these quite often on sites like house-buying sites where you want to set a maximum property price to filter by. Let's look at a live example to illustrate this:



Usage-wise, sliders are less accurate than text fields. Therefore, they are used to pick a number whose *precise* value is not necessarily important.

A slider is created using the `<input>` with its `type` attribute set to the value `range`. The slider-thumb can be moved via mouse or touch, or with the arrows of the keypad.

It's important to properly configure your slider. To that end, it's highly recommended that you set the `min`, `max`, and `step` attributes which set the minimum, maximum, and increment values, respectively.

Let's look at the code behind the above example, so you can see how it's done. First of all, the basic HTML:

```
<label for="price">Choose a maximum house price: </label>
<input
  type="range"
  name="price"
  id="price"
  min="50000"
  max="500000"
  step="100"
  value="250000" />
<output class="price-output" for="price"></output>
```

This example creates a slider whose value may range between `50000` and `500000`, which increments/decrements by 100 at a time. We've given it a default value of `250000`, using the `value` attribute.

One problem with sliders is that they don't offer any kind of visual feedback as to what the current value is. This is why we've included an `<output>` element to contain the current value. You could display an input value or the output of a calculation inside any element, but `<output>` is special — like `<label>` — and it can take a `for` attribute that allows you to associate it with the element or elements that the output value came from.

To actually display the current value, and update it as it changed, you must use JavaScript, but this is relatively easy to do:

```
const price = document.querySelector("#price");
const output = document.querySelector(".price-output");

output.textContent = price.value;

price.addEventListener("input", () => {
  output.textContent = price.value;
});
```

Here we store references to the `range` input and the `output` in two variables. Then we immediately set the `output`'s `textContent` to the current `value` of the input. Finally, an event listener is set to ensure that whenever the range slider is moved, the `output`'s `textContent` is updated to the new value.

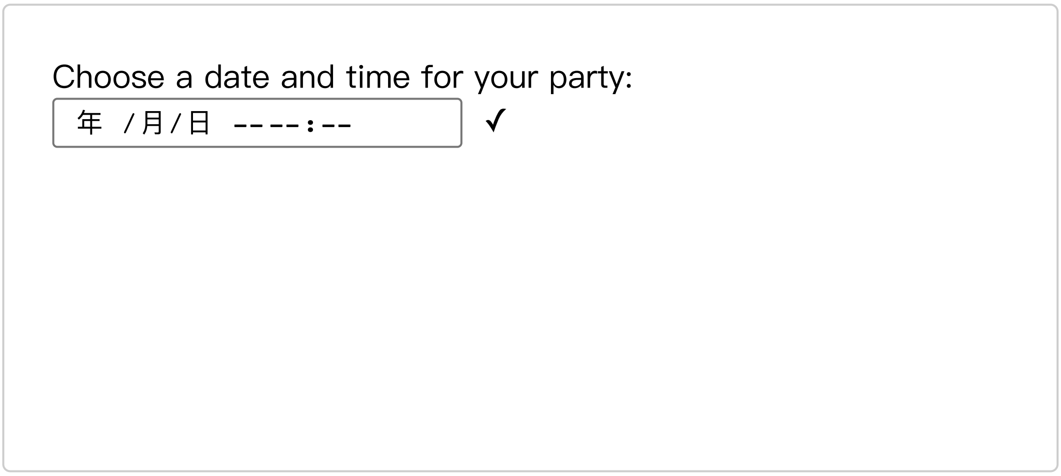
> **Note:** There is a nice tutorial covering this subject on CSS Tricks: The Output Element .

Date and time pickers

Gathering date and time values has traditionally been a nightmare for web developers. For a good user experience, it is important to provide a calendar selection UI, enabling users to select dates without necessitating context switching to a native calendar application or potentially entering them in differing formats that are hard to parse. The last minute of the previous millennium can be expressed in the following different ways, for example: 1999/12/31, 23:59 or 12/31/99T11:59PM.

HTML date controls are available to handle this specific kind of data, providing calendar widgets and making the data uniform.

A date and time control is created using the `<input>` element and an appropriate value for the `type` attribute, depending on whether you wish to collect dates, times, or both. Here's a live example that falls back to `<select>` elements in non-supporting browsers:



Let's look at the different available types in brief. Note that the usage of these types is quite complex, especially considering browser support (see below); to find out the full details, follow the links below to the reference pages for each type, including detailed examples.

`datetime-local`

`<input type="datetime-local">` creates a widget to display and pick a date with time with no specific time zone information.

```
<input type="datetime-local" name="datetime" id="datetime" />
```

`month`

`<input type="month">` creates a widget to display and pick a month with a year.

```
<input type="month" name="month" id="month" />
```

`time`

`<input type="time">` creates a widget to display and pick a time value. While time may *display* in 12-hour format, the *value returned* is in 24-hour format.

```
<input type="time" name="time" id="time" />
```

`week`

`<input type="week">` creates a widget to display and pick a week number and its year.

Weeks start on Monday and run to Sunday. Additionally, the first week 1 of each year contains the first Thursday of that year — which may not include the first day of the year, or may include the last few days of the previous year.

```
<input type="week" name="week" id="week" />
```

Constraining date/time values

All date and time controls can be constrained using the `min` and `max` attributes, with further constraining possible via the `step` attribute (whose value varies according to input type).

```
<label for="myDate">When are you available this summer?</label>
<input
  type="date"
  name="myDate"
  min="2013-06-01"
  max="2013-08-31"
  step="7"
  id="myDate" />
```

Color picker control

Colors are always a bit difficult to handle. There are many ways to express them: RGB values (decimal or hexadecimal), HSL values, keywords, and so on.

A `color` control can be created using the `<input>` element with its `type` attribute set to the value `color`:

```
<input type="color" name="color" id="color" />
```

Clicking a color control generally displays the operating system's default color-picking functionality for you to choose.

Here is a live example for you to try out:

The value returned is always a lowercase 6-value hexadecimal color.

Test your skills!

You've reached the end of this article, but can you remember the most important information? You can find some further tests to verify that you've retained this information before you move on — see Test your skills: HTML5 controls.

Summary

That brings us to the end of our tour of the HTML5 form input types. There are a few other control types that cannot be easily grouped together due to their very specific behaviors, but which are still essential to know about. We cover those in the next article.

Advanced Topics

- How to build custom form controls
- Sending forms through JavaScript
- Property compatibility table for form widgets

This page was last modified on Mar 12, 2023 by MDN contributors.