Формат подборка, статья, руководство, истории, интервью, колонка (внешний эксперт)	статья (колонка внешнего эксперта)
Автор	Редакция
Раздел	Код
SEO Title	HTML-тег <input>: что это, зачем он нужен, как создать поле ввода и настроить атрибуты
SEO Description	Рассказываем, как работает тег <input> в HTML, какие бывают типы полей ввода и атрибуты, как создать форму для ввода текста, пароля, email и других данных.
Ключи	input, input html, инпут, type input, input css, атрибуты input html
Рекламный тег	Фронтенд-разработка
Курс (название, ссылка)	Профессия Фронтенд-разработчик

Тег <input> в HTML

> Рассказываем, как правильно использовать тег для создания форм на странице.

Тег `<input>` используют, чтобы создать поля ввода в HTML-формах. С его помощью можно добавить на страницу текстовое поле, поле для пароля, выбора даты или загрузки файла. `<input>` показывает пользователю элемент, с помощью которого он может ввести или выбрать данные. Потом эти данные можно отправить на сервер или обработать с использованием JavaScript.

Тег поддерживает разные типы полей и множество атрибутов, которые помогают контролировать его поведение и внешний вид. В статье мы подробно разберём, как с ним работать.

Содержание

1Как пишется тег <input>

<input> — это одиночный тег. У него нет закрывающего парного тега, поэтому запись всегда выглядит как одна строка. Чтобы тег работал, ему задают атрибуты. Единственный обязательный атрибут — type, который определяет, какое именно поле появится на странице.

Простейший пример:

```
<input type="text">
```

В этом случае браузер создаст текстовое поле, в которое можно ввести строку.

Если изменить значение атрибута type, то изменятся свойства поля. Например:

```
<input type="password">
```

Это поле скрывает введённые символы точками или звёздочками.

2Основные типы полей <input>

Атрибут `type` определяет, какое именно поле появится на странице. Вот основные варианты:

Текстовое поле

```
<input type="text" placeholder="Ваше имя">
```

Используется для ввода обычного текста: имени, названия города, комментария.

Поле для пароля

```
<input type="password" placeholder="Пароль">
```

Похоже на текстовое, но введённые символы скрываются.

Поле для электронной почты

```
<input type="email" placeholder="example@mail.com">
```

Проверяет, похоже ли введённое значение на адрес почты.

Поле для числа

```
<input type="number" placeholder="0">
```

Позволяет вводить только цифры. Можно задать минимальное и максимальное значение, чтобы ограничить ввод.

Флажок (checkbox)

```
<input type="checkbox"> Подписаться на новости
```

Даёт возможность выбрать «да» или «нет». Можно поставить несколько галочек в одной форме.

Переключатель (radio)

```
<input type="radio" name="gender" value="male"> Мужской
<input type="radio" name="gender" value="female"> Женский
```

Позволяет выбрать только один вариант из группы.

Поле для выбора файла

```
<input type="file">
```

Открывает диалог выбора файла на компьютере.

Выбор даты

```
<input type="date">
```

Показывает календарь, где можно выбрать дату. Часто используется в формах бронирования и регистрации.

Скрытое поле

```
<input type="hidden" name="id" value="123">
```

Не отображается на странице, но отправляется вместе с формой. Полезно для передачи технических данных, которые пользователь не должен изменять.

3Основные атрибуты тега input

Тег `<input>` работает только с атрибутами — они задают его поведение и внешний вид. Вот основные:

`name`

```
<input type="text" name="username">
```

Имя поля. Именно по этому имени сервер понимает, к каким данным относится введённое значение.

`value`

```
<input type="text" value="Гость">
```

Значение по умолчанию: оно будет стоять в поле ещё до того, как пользователь что-то введёт. Такие поля используют, например, в формах входа: туда автоматически подставляется логин, который пользователь вводил в прошлый раз.

`placeholder`

```
<input type="email" placeholder="example@mail.com">
```

Текст-подсказка внутри поля. Она исчезает, когда пользователь начинает вводить данные.

`required`

```
<input type="password" required>
```

Делает поле обязательным для заполнения. Без него форма не отправится.

`readonly`

```
<input type="text" value="Только для чтения" readonly>
```

Поле видно, но редактировать его нельзя. Здесь может быть, например, номер заказа или системная информация.

`disabled`

```
<input type="text" value="Отключено" disabled>
```

Поле полностью выключено: пользователь не может ввести данные, такое поле не отправится вместе с формой.

`maxlength`

```
<input type="text" maxlength="10">
```

Ограничивает количество вводимых символов. Например, не больше 10.

`pattern`

```
<input type="text" pattern="[0-9]{3}">
```

Задаёт шаблон для проверки ввода. В этом примере можно ввести только три цифры. Для создания шаблона используются регулярные выражения (regex).

Например, квадратные скобки `[]` задают набор допустимых символов.

- `[0-9]` — любая цифра.
- `[A-Za-z]` — любая буква латинского алфавита.

А фигурные скобки `{}` — указывают на количество символов.

- `{3}` — ровно три символа.
- `{2,5}` — от двух до пяти символов.

Читайте также: Регулярные выражения в JavaScript

4Как организовать валидацию данных

Чтобы форма работала правильно, важно убедиться, что пользователь ввёл данные в нужном формате. В HTML есть встроенные инструменты для этого, а при необходимости можно добавить собственные проверки через JavaScript.

Встроенная проверка

Многие атрибуты автоматически проверяют ввод. Например:

```
<input type="email" name="user_email" required>
```

- `type="email"` заставит браузер проверить, похож ли введённый адрес на email (например, содержит ли он `@`).
- `required` не даст отправить форму, если поле пустое.

Если данные не подходят, браузер покажет предупреждение без дополнительного кода.

Проверка по шаблону

Можно задать правило с помощью атрибута `pattern`, о котором мы подробно говорили выше.

Своя логика на JavaScript

Если встроенных правил недостаточно, можно добавить JavaScript:

```
<form>
  <input type="text" id="code" placeholder="Введите код">
```

```
  <button type="submit">Отправить</button>
</form>

<script>
  // Находим форму на странице
  const form = document.querySelector("form");

  // Находим поле ввода по его id
  const code = document.getElementById("code");

  // Добавляем обработчик события submit (отправка формы)
  form.addEventListener("submit", function(event) {

    // Проверяем, равняется ли значение поля "123"
    if (code.value !== "123") {

      // Если условие не выполнено, отменяем отправку формы
      event.preventDefault();

      // Показываем сообщение пользователю
      alert("Код введён неправильно");
    }
  });
</script>
```

Читайте также: Как запустить JavaScript-код: в браузере, консоли и редакторе кода

5Как использовать <input> в составе формы и отправлять данные на сервер

Форма — это контейнер, который объединяет поля `<input>` и сообщает браузеру, куда и как отправлять введённые пользователем данные.

Простейшая форма

```
<form action="/submit" method="post">
  <label for="name">Имя</label>
  <input id="name" name="username" type="text">

  <label for="email">Email</label>
  <input id="email" name="user_email" type="email" required>

  <input type="submit" value="Отправить">
</form>
```

Что здесь происходит:

- `form` — это сам контейнер. Атрибут `action` указывает адрес, на который браузер отправит данные. `method` задаёт способ отправки (здесь `post` — данные будут в теле запроса).
- Для каждого поля важно задать `name`. При отправке сервер получит пары «имя = значение» и сможет по `name` найти нужное значение.
- `required` запрещает браузеру отправку пустого поля.
- Кнопка `type="submit"` запускает процесс отправки формы.

Что отправляется и куда браузер кладёт данные

Если указан метод `get`, браузер упаковывает все видимые поля в строку запроса (query string) и добавляет её к URL: `/search?q=чай&category=напитки` — это видимая часть адреса. Такой способ годится для поиска и фильтров, потому что URL можно скопировать и отправить.

Если `method="post"`, данные идут в теле HTTP-запроса. По умолчанию браузер использует формат `application/x-www-form-urlencoded`, то есть пары вида `username=ivan&age=30`. Этот способ подходит для отправки конфиденциальных или больших данных.

Если вы не указали `method`, браузер по умолчанию использует `GET`. Если не указали `action`, форма отправится на тот же адрес, где находится страница.

Отправка файлов

Для передачи файлов нужны две вещи: поле `type="file"` с `name` и атрибут `enctype="multipart/form-data"` у формы :

```
<form action="/upload" method="post"
enctype="multipart/form-data">
  <label for="avatar">Аватар</label>
  <input id="avatar" name="avatar" type="file">
  <input type="submit" value="Загрузить">
</form>
```

`multipart/form-data` разбивает запрос на части и отправляет файл вместе с другими полями. Без такого `enctype` файл не дойдёт корректно.

Что не отправляется

- Поля без `name` не включаются в отправку. Если вы забыли `name`, сервер не увидит значение.
- Поля с `disabled` не отправляются. Поля `readonly` отправляются (их нельзя изменить, но их значение включается в запрос).

Пример: что видит сервер

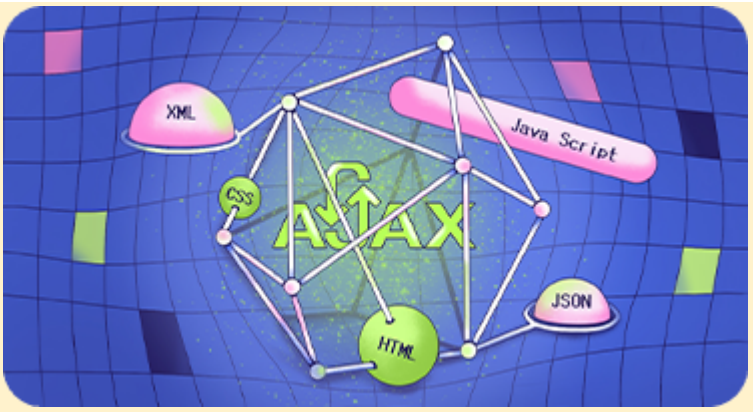
- При `GET` и строке запроса `?q=book&page=2` сервер получит `q=book` и `page=2`.
- При `POST` с `application/x-www-form-urlencoded` тело запроса будет `q=book&page=2`.
- При `multipart/form-data` сервер получит части: текстовые поля и отдельную часть с содержимым файла.

Отправка формы через JavaScript (AJAX/fetch)

Иногда нужно отправлять форму без перезагрузки страницы. Для этого перехватывают событие `submit`, собирают данные и отправляют через `fetch`.

```
<form id="myForm" action="/submit" method="post">
  <input name="username" type="text" value="Гость">
  <input name="avatar" type="file">
  <button type="submit">Отправить</button>
</form>

<script>
  const form = document.getElementById('myForm');

  form.addEventListener('submit', function (event) {
    event.preventDefault(); // Не отправляем форму стандартным способом

    // FormData автоматически соберёт все поля формы, включая файлы
    const data = new FormData(form);

    // Отправляем на сервер методом POST
    fetch(form.action, {
      method: 'POST',
      body: data
      // При использовании FormData заголовок Content-Type устанавливается автоматически
    })
    .then(response => response.ok ? response.text() : Promise.reject(response))
    .then(text => {
      // Здесь можно показать сообщение об успехе пользователю
      console.log('Ответ сервера:', text);
    })
    .catch(err => {
      // Обработка ошибок отправки
      console.error('Ошибка отправки:', err);
    });
  });
</script>
```

Важно: при таком подходе сервер должен уметь принимать формат, в котором вы отправляете данные. `FormData` по умолчанию отправляет `multipart/form-data`.



Читайте также: Что такое AJAX и как он помогает обновлять контент на странице

6Доступность полей ввода

Форма должна быть удобной для всех пользователей, включая людей с нарушениями зрения, моторики или тех, кто использует клавиатуру вместо мыши. Для этого необходимо правильно оформлять поля `<input>` и использовать специальные атрибуты и элементы.

Подписи к полям: тег <label>

Каждое поле ввода должно сопровождаться текстовой подписью, которая объясняет, что именно нужно ввести. Подпись связывается с полем с помощью атрибута `for`, значение которого должно совпадать с `id` поля.

Пример:

```
<label for="username">Имя:</label>
<input id="username" name="username" type="text" required />
```

Если указать тег label, при клике на текст подписи курсор автоматически попадает в поле, а скринридеры — программы для чтения текста с экрана — озвучивают подпись вместе с полем. Всё это упрощает навигацию с помощью клавиатуры.

Разница между label и placeholder

- `<label>` — эта подпись всегда видна, она остаётся на экране.
- `placeholder` — временная подсказка, исчезает при вводе.

С точки зрения доступности `label` и `placeholder` не взаимозаменяемы. После начала ввода пользователь может забыть, что именно требовалось, поэтому ему нужна постоянная подсказка.

Пример:

```
<label for="email">Email:</label>
<input id="email" type="email" placeholder="example@mail.com" />
```

Атрибуты ARIA

ARIA-атрибуты помогают скринридерам правильно озвучивать поля ввода.

aria-label

```
<input type="search" aria-label="Поиск по сайту" />
```

Скринридер озвучит «Поиск по сайту» как описание поля, даже если визуально подпись отсутствует. Используется, если невозможно применить `<label>`.

aria-describedby

Указывает элемент, содержащий дополнительное описание поля. Скринридер озвучивает это описание после основной метки.

```
<input type="text" aria-describedby="hint" />
<span id="hint">Введите номер заказа без пробелов</span>
```

aria-required

Обозначает, что поле обязательно для заполнения. Используется в дополнение к `required`, чтобы скринридер озвучил это требование.

```
<input type="text" aria-required="true" />
```

aria-invalid

Показывает, что введённые данные некорректны. Скринридер озвучит, что поле содержит ошибку.

```
<input type="email" aria-invalid="true" />
```

Атрибут можно устанавливать динамически при проверке формы.

Навигация с клавиатуры

Пользователь должен иметь возможность переходить между полями с помощью клавиши `Tab`. Это работает автоматически, если:

- поля имеют корректную разметку;
- не нарушена последовательность в DOM;
- не используются нестандартные элементы без `tabindex`.

7Советы и лайфхаки

Настройте поведение ввода под пользователя

Чтобы формы были удобнее, используйте два атрибута, которые напрямую влияют на то, как человек вводит данные.

`autocomplete` подсказывает браузеру, что именно ожидается в поле. Так он может заранее предложить имя, адрес или email, которые пользователь уже вводил. Это экономит время и снижает количество ошибок.

- `autocomplete="name"` — значит, здесь имя человека, можно подставить сохранённое.
- `autocomplete="email"` — поле для адреса почты.
- `autocomplete="off"` — вообще не предлагать подсказок.

Атрибут управляет автоподстановкой и экономит пользователю время, если задан верно.

`autocapitalize` управляет виртуальной клавиатурой на телефонах и планшетах. С его помощью можно автоматически начинать каждое слово с заглавной буквы (подходит для имён), включить верхний регистр для кодов или совсем отключить автоматические заглавные буквы.

`autocapitalize` — это настройка виртуальной клавиатуры (в основном на мобильных устройствах). Она указывает, как обрабатывать буквы при вводе:

- `autocapitalize="words"` — каждое слово начинается с заглавной буквы (удобно для имён, названий городов).

- `autocapitalize="characters"` — все буквы становятся заглавными (полезно для кодов, аббревиатур).
- `autocapitalize="off"` — никаких автоматических заглавных, всё пишется как есть.

Оба атрибута не меняют содержимое формы сами по себе, но делают ввод естественнее и быстрее, если настроить их правильно.

Подсказка для клавиатуры

Атрибут `inputmode` не меняет сам `<input>`, но управляет тем, какую клавиатуру увидит человек на телефоне или планшете: обычную, цифровую или спецсимволы.

```
<input type="tel" inputmode="numeric" placeholder="+7 (___)
___-__-__">
```

Здесь мы не только явно указываем, что это телефон (`type="tel"`), но и просим устройство показать цифры (`inputmode="numeric"`). В итоге человек сразу набирает номер, не отвлекаясь.

Точно так же можно сделать поле для PIN-кода:

```
<input type="text" inputmode="numeric" maxlength="4"
placeholder="Введите PIN">
```

Для email лучше подсказать:

```
<input type="email" inputmode="email"
placeholder="example@mail.com">
```

На клавиатуре появится значок `@`, а иногда и `.com`.

Поиск тоже можно сделать удобнее:

```
<input type="search" inputmode="search" placeholder="Поиск по
сайту">
```

Кнопка на клавиатуре превратится в «Поиск» вместо обычного Enter.

Больше интересного про код и технологии — в нашем телеграм-канале. Подписывайтесь!

Читайте также:

- Что такое header в HTML
- Тег <footer> в HTML: как сделать подвал страницы
- Как использовать заголовки <h1> — <h6> и тег <p> в HTML