YAValidator ≡ Yet Another Validator

Назначение

YAValidator это многофункциональная, простая в использовании (один класс с 5-ю методами) и легковесная (менее 6 кБ) библиотека предназначенная для выполнения ряда типовых операций обработки значений HTML элементов. Хорошо подходит для использования в SPA WEB-приложениях, построения WEB-интерфейса настройки встраиваемых устройств и других похожих применений.

Типовыми задачами, встающими перед разработчиком WEB-интерфейса, являются:

- Присвоение результатов AJAX-запроса значениям HTML элементов `<input>`, `<select>`
- Контроль ввода данных пользователем, валидация данных
- Контроль статуса изменений введённых данных
- Формирование данных для AJAX-запроса записи

Помочь решению этих задач позволяет **YAValidator**.

Совместимость

Библиотека может быть использована для работы в любых браузерах поддерживающих спецификацию стандарта ES6. Данная спецификация поддерживается большинством современных браузеров начиная с 2015-2016 года.

Основные функции

- Присвоение данных значениям HTML элементов
- Валидация введённых данных согласно правилам
- Отслеживание статуса изменения данных
- Фиксация изменений и формирование данных для записи

Установка

Просто скопируйте файл **yavalidator.min.js** в папку со скриптами и добавьте в текст вашего HTML что-то подобное:

```
<script src="/js/yavalidator.min.js"></script>
```

Быстрый старт

Предположим, что вашей HTML-странице имеются три HTML элемента - поля ввода. Легко догадаться об их назначении можно из атрибутов `title`:

```
<input id="edit_hostname" type="text" title="Hostname">
<input id="edit_ip" type="text" title="IP address">
<input id="edit_port" type="number" min="1" max="65535" step="1" title="TCP port">
```

Для того, что-бы начать работу, вам необходимо установить для этих элементов произвольное (но одинаковое) имя класса, которое объединит их в единую группу. Пусть имя класса будет "group1":

```
<input id="edit_hostname" class="group1" type="text" title="Hostname">
<input id="edit_ip" class="group1" type="text" title="IP address">
<input id="edit_port" class="group1" type="number" min="1" max="65535" step="1"
title="TCP port">
```

Для выполнения валидации значений элементов, необходимо установить для каждого элемента правило, в соответствии с которым будет производиться его проверка. Для некоторых типов элементов правило назначается автоматически на основании его типа, для некоторых его необходимо установить вручную.

Ручная установка правил производится путём установки для элемента классов с определёнными именами. Подробное описание приведено разделе "Правила валидации", а пока просто откорректируйте вашу страницу следующим образом:

```
<input id="edit_hostname" class="group1 yav-hostname" type="text" title="Hostname">
<input id="edit_ip" class="group1 yav-ip" type="text" title="IP address">
<input id="edit_port" class="group1" type="number" min="1" max="65535" step="1"
title="TCP port">
```

Вы можете заметить, что для элемента "edit_hostname" добавлен класс "yav-hostname", а для элемента "edit_ip" - класс "yav-ip". Имена этих классов определяют правила, в соответствии с которыми будет выполняться валидация значений. Правило для валидации элемента "edit_port" будет назначено автоматически на основании его типа `number` и значений атрибутов `min`, `max` и `step`.

Наверняка вы захотите, что-бы введённые пользователем некорректные данные были как-то выделены. Для этого в таблицу стилей вашей страницы добавьте новый класс, который будет использован для выделения некорректных данных. Например такой:

```
<style>
  .invalid-element {
    outline: solid red 1px;
    background-color: lightpink;
  }
  ...
</style>
```

Итак, начинается самое интересное: создаём экземпляр класса **YAValidator**. Сделать это необходимо, когда весь HTML документ будет загружен, например в обработчике "onload" HTML страницы.

В качестве первого параметра конструктора класса указываем наименование группы элементов, второй аргумент - имя класса для выделения элементов с некорректными значениями. У конструктора есть и другие необязательные параметры, которые будут рассмотрены ниже.

```
<script>
  function onload() {
    var validator = new YAValidator("group1", "invalid-element");
  }
</script>
<body onload="onload">
```

```
...
</body>
```

Всё готово! Теперь вы определили группу из трёх HTML элементов ввода и можете использовать в своём коде следующие методы класса **YAValidator**:

```
// проверка элементов группы в соответствии с установленными для них правилами
var res = validator.validate();
if(res) alert("Ok"); else alert("Error");
//
// получение текущих значений элементов группы
var data = validator.getData("edit_");
//
// установка значений для элементов группы
validator.setData({
  "hostname": "my hostname",
  "ip": "192.168.1.1",
  "port": 22
}, "edit_");
```

Валидация "на лету"

В приведённом выше примере для валидации элементов необходимо вызывать метод `validate()`. Если вы хотите, что-бы валидация выполнялась по мере ввода данных пользователем, то вам необходимо добавить дополнительный параметр при вызове конструктора:

```
var validator = new YAValidator("group1", "invalid-element", {
  "onedit": true              // проверять при изменениях значений
});
```

Теперь валидация будет выполняться при любом изменении значения любого HTML элемента группы без необходимости вызова метода `validate()`.

По умолчанию класс не выполняет валидацию и не возвращает в методе `getData()` значения HTML элементов у которых установлены атрибуты `disabled` или `readonly`. Также не выполняется проверка невидимых элементов, у которых (а также у их родительских элементов) стиль `display` имеет значение `none`.

Вы можете изменить это, указав при вызове конструктора следующие параметры:

```
var validator = new YAValidator("group1", "invalid-element", {
  "disabled": true,           // проверять элементы с атрибутом 'disabled'
  "readonly": true,           // проверять элементы с атрибутом 'readonly'
  "hidden": true,             // проверять невидимые элементы
  "onedit": true              // проверять при изменениях значений
});
```

Отслеживание изменений

Отслеживание изменений может оказаться полезным для выполнения дополнительной валидации, манипуляций (скрытие, отображение, изменения атрибутов) с зависимыми элементами HTML при изменении данных или формирования разрешения для кнопки выполнения AJAX-запроса записи данных изменённых пользователем.

Для отслеживания изменений значений HTML элементов в вызов конструктора класса необходимо добавить функцию - обработчик изменения значений. Обработчик принимает три параметра:

```
var validator = new YAValidator("group1", "invalid-element", {}, function(changed,
valid, data) {
  // changed {boolean} - true: значения изменились
  // valid   {boolean} - true: значения корректны
  // data    {Object}  - значения всех HTML элементов группы
  //
  document.getElementById("save_button").disabled = !(changed && valid);
});
```

В примере показано управление установкой атрибута `disabled` для кнопки записи изменений.

Для расчёта признака изменений необходимо зафиксировать первоначальное состояние значений HTML элементов с которым в дальнейшем будет производиться сравнение. Такая фиксация выполняется при установке данных путём вызове метода `setData(data)`. Кроме того, текущие значения HTML элементов могут быть зафиксированы в любой момент путём вызова метода `fixData()`.

```
  // фиксация значений
  validator.setData({
    "hostname": "my hostname",
    "ip": "192.168.1.1",
    "port": 22
  }, "edit_");
  //
  // тоже фиксация значений
  validator.fixData();
```

Манипуляции значениями элементов

Выше уже были упомянуты методы класса `getData()`, `setData()` и `fixData()`.

Рассмотрим подробнее работу методов `getData()`, `setData()`.

Метод `getData()` возвращает объект со значениями всех входящим в группу элементов. В качестве ключей возвращаемого объекта используются значения атрибутов "id" элементов за исключением элементов `<input type="radio">`. Для них вместо "id" используется значение атрибута "name".

У метода `getData()` имеется необязательный параметр `id_prefix`. Если этот параметр присутствует (это должна быть строка) и если "id" (ну или "name") начинается с его значения, то это значение будет удалено из начала ключа "id". Получилось немного запутанно, хотя всё на самом деле просто...

Вот пример:

```
<input id="edit_hostname" class="group1 yav-hostname" type="text" title="Hostname">
<input id="edit_ip" class="group1 yav-ip" type="text" title="IP address">
<input id="edit_port" class="group1" type="number" min="1" max="65535" step="1"
title="TCP port">
<input class="group1" type="radio" name="edit_radio" value="0">
<input class="group1" type="radio" name="edit_radio" value="1">
```

Для описанных выше элементов вызов метода `getData()` вернёт следующий объект:

```
// Вызов без параметра
var data = validator.getData();
//
//data: {
//  "edit_hostname": ...,
//  "edit_ip": ...,
//  "edit_port": ...
//  "edit_radio": ...
//}
// Вызов с параметром
var data = validator.getData("edit_");
//
//data: {
//  "hostname": ...,
//  "ip": ...,
//  "port": ...
//  "radio": ...
//}
```

Вызов `getData()` с параметром оказаться полезным при подготовке объекта с данными для выполнения AJAX-запроса записи введённых данных.

Метод `setData()` также имеет необязательный параметр `id_prefix`, который работает аналогично параметру метода `getData()`. При его присутствии ключам объекта будет добавлено его значение при установке значения элементам HTML. Пример:

```
var data = {
    "number": 42,                // значение будет присвоено элементу с "id" =
"edit_number"
    "url":    "www.aaa.com"      // значение будет присвоено элементу с "id" =
"edit_url"
};
validator.setData(data, 'edit_');
```

Правила валидации

Начнём с того, что проверке подвергаются только значения HTML элементов `<input>` с типами `text`, `number`, `url`, `phone` и `email`. Для остальных типов `<input>`, таких как `checkbox`, `radio`,

`range`, `password` и других, а также для элемента `<select>` библиотека может быть использована только для манипуляции данными.

При валидации проверяется наличие у HTML элемента атрибута `required`. Если атрибут не установлен, то пустое значение элемента считается корректным. Если пустое значение элемента не допускается, то вам необходимо добавить к нему этот атрибут. Пример:

```
<!-- для элемента "edit_hostname0" допустимо пустое значение -->
<input id="edit_hostname0" class="group1 yav-hostname" type="text" title="Hostname">
<!-- для элемента "edit_hostname1" пустое значение не допустимо -->
<input id="edit_hostname1" class="group1 yav-hostname" type="text" title="Required
Hostname" required>
```

Полный перечень поддерживаемых правил валидации приведён ниже.

Основным способом установки правила валидации для HTML элемента является присвоение ему класса с определённым именем, определяющим это правило. Элемент должен являться элементом `<input type="text">`. Если у элемента одновременно установлено несколько имён классов, то значение этого элемента является корректным если оно соответствует хотя-бы одному правилу.

В некоторых случаях существует по два варианта установки правила: через установку класса с определённым именем или через указание значение типа HTML элемента `<input type="">`. Все эти случаи описаны ниже. Вы можете выбрать любой из этих двух вариантов.

Целые числа

Для валидации целых чисел можно использовать класс `yav-integer` либо HTML-элемент `<input type="number">`. Во втором случае дополнительно проверяется на соответствие значениям атрибутов `min`, `max` и `step`. Пример:

```
<!-- вариант 1 -->
<input id="edit_int0" class="group1 yav-integer" type="text">
<!-- вариант 2 -->
<input id="edit_int1" class="group1" type="number" min="0" max="100" step="1">
```

Вещественные числа

Аналогично валидации целых чисел для проверки вещественных чисел существует два способа: класс `yav-number` либо HTML-элемент `<input type="number">`. Во втором случае дополнительно проверяется на соответствие значениям атрибутов `min`, `max` и `step`. Пример:

```
<!-- вариант 1 -->
<input id="edit_num0" class="group1 yav-number" type="text">
<!-- вариант 2 -->
<input id="edit_num1" class="group1" type="number" min="0" max="100" step="0.1">
```

Обратите внимание, на то, что во втором варианте значение атрибута `step` является дробным числом.

Имя хоста

Для установки правила валидации имён хостов необходимо использовать класс `yav-hostname`. Проверка выполняется в соответствии с требованиями RFC 952.

```
<input id="edit_hostname" class="group1 yav-hostname" type="text">
```

IP адрес или маска подсети

Для валидации записи IP адреса или маски подсети (IPv4) необходимо использовать класс `yav-ip`.

```
<input id="edit_ip" class="group1 yav-ip" type="text">
```

Строка URL

Проверка значения URL (Uniform Resource Locator) может быть выполнена двумя способами: использование класса `yav-url` или HTML-элемента `<input type="url">`:

```
<!-- вариант 1 -->
<input id="edit_url0" class="group1 yav-url" type="text">
<!-- вариант 2 -->
<input id="edit_url1" class="group1" type="url">
```

Адрес email

Валидация значения адреса электронной почты может быть выполнена двумя способами:

```
<!-- вариант 1 -->
<input id="edit_email0" class="group1 yav-email" type="text">
<!-- вариант 2 -->
<input id="edit_email1" class="group1" type="email">
```

Номер телефона

Валидация значения номера телефона может быть выполнена двумя способами:

```
<!-- вариант 1 -->
<input id="edit_tel0" class="group1 yav-tel" type="text">
<!-- вариант 2 -->
<input id="edit_tel1" class="group1" type="tel">
```

Номер расчётного счёта

Для установки правила валидации номера расчётного счёта необходимо использовать имя класса `yav-acc`:

```
<input id="edit_acc0" class="group1 yav-acc" type="text">
```

Номер БИК

Для установки правила валидации номера БИК (Банковский идентификационный код) необходимо использовать имя класса `yav-bik`:

```
<input id="edit_bik0" class="group1 yav-bik" type="text">
```

Номер ИНН юрлиц, физлиц и ИП

Для установки правила валидации номера ИНН (Индивидуальный номер налогоплательщика) необходимо использовать имя класса `yav-inn`. Поддерживаются 10-ти и 12-ти значные ИНН, валидация включает расчёт и проверку контрольного символа.

```
<input id="edit_inn0" class="group1 yav-inn" type="text">
```

Номер СНИЛС

Для установки правила валидации номера СНИЛС (Страховой номер индивидуального лицевого счета) необходимо использовать имя класса `yav-snils`. Валидация включает расчёт и проверку контрольного символа.

```
<input id="edit_inn0" class="group1 yav-inn" type="text">
```

Пользовательские правила

Если ни одно из описанных выше правил вас не устраивает, то вы можете задать своё. Сделать это можно путём присвоения строки с регулярным выражением значению атрибута `pattern` элемента HTML. Если у элемента установлен атрибут `pattern`, то другие, установленные через имена классов, правила валидации для этого элемента не применяются. Таким образом, установка атрибута `pattern` является приоритетной.

```
<input id="edit_pattern" type="text" pattern="^(Hello)$">
```

Сводная таблица правил валидации

Правило	Используемый класс	Альтернатива
Целые числа	`yav-integer`	`<input type="number" min="0" max="100" step="1">`
Вещественные числа	`yav-number`	`<input type="number" min="0" max="100" step="0.1">`
IP адрес	`yav-ip`	
URL	`yav-url`	`<input type="url">`
Имя хоста	`yav-hostname`	
Адрес email	`yav-email`	`<input type="email">`
Номер телефона	`yav-tel`	`<input type="tel">`

Номер расчётного счёта	`yav-acc`	
Номер БИК	`yav-bik`	
Номер ИНН	`yav-inn`	
Номер СНИЛС	`yav-snils`	
Пользовательское		`<input type="text" pattern="">`

Примеры использования

Примеры использования можно увидеть в index.html.

Полное описание класса

Classes

YAValidator

Класс валидации значений HTML элементов

Typedefs

changeDataCallback : `function`

Обработчик изменения значения элемента в группе.

yavOptions : `Object`

Параметры валидации.

YAValidator

Класс валидации значений HTML элементов

Kind: global class

- YAValidator
  - new YAValidator(selector, [invalidClassName], [options], [changeDataCB])
  - .validate() ⇒ `boolean`
  - .invalidate()
  - .setData(data, [id_prefix], [nofix])
  - .getData([id_prefix]) ⇒ `Object`
  - .fixData()
  - .restoreData()

new YAValidator(selector, [invalidClassName], [options], [changeDataCB])

Param	Type	Description
selector	`string`	Имя класса для выбора проверяемых элементов

[invalidClassName]	`string`	Имя класса для маркировки элементов с некорректными значениемя
[options]	`yavOptions`	Набор параметров
[changeDataCB]	`changeDataCallback`	Обработчик изменения значения элемента в группе

yaValidator.validate() ⇒ `boolean`

Валидация значений всех элементов группы. Если валидация значения хотя-бы одного одного элемента не прошла, то возвращает false. Для элементов не прошедших валидацию назначается класс "invalidClassName".

Kind: instance method of `YAValidator`
Returns: `boolean` - Результат валидации

yaValidator.invalidate()

Удалить класс "invalidClassName" у всех элементов

Kind: instance method of `YAValidator`

yaValidator.setData(data, [id_prefix], [nofix])

Устанавливает значения элементам входящим в группу. Объект "data" должен содержать записи соотвтетствующие значениям атрибутов "id" элементов группы. Для элементов "input[type=radio]" вместо "id" используется значение атрибута "name". Значение параметра "id_prefix" аналогично использованию в методе "getData()".

Kind: instance method of `YAValidator`

Param	Type	Description
data	`Object`	Объект со значениями элементов группы
[id_prefix]	`string`	Добавляемый префикс для ключей объекта со значениями
[nofix]	`boolean`	Не фиксировать изменения

yaValidator.getData([id_prefix]) ⇒ `Object`

Возвращает объект со значениями всех элементов входящим в группу. В качестве ключей возвращаемого объекта используются значения атрибутов "id" элементов за исключением элементов "input[type=radio]". Для них вместо "id" используется значение атрибута "name". Если ключ начинается с префикса "id_prefix", то он будет удалён.

Kind: instance method of `YAValidator`
Returns: `Object` - Значения всех элементов

Param	Type	Description
[id_prefix]	`string`	Удаляемый префикс для ключей объекта со значениями

yaValidator.fixData()

Фиксация текущих значений элементов. Зафиксированное значение используется для сравнения с текущими значениями элементов при формировании параметра "changed" обработчика изменений "changeDataCB". Если установлен параметр "onedit", то дополнительно выполняется валидация всех элементов.

Kind: instance method of `YAValidator`

yaValidator.restoreData()

Метод восстанавливает предварительно зафиксированные значение элементов. Если данные не были зафиксированы, то метод ничего не делает.

Kind: instance method of `YAValidator`

changeDataCallback : `function`

Обработчик изменения значения элемента в группе.

Kind: global typedef

Param	Type	Description
changed	`boolean`	Значения элементов группы изменились по сравнению с ранее зафиксированным значением
valid	`boolean`	Значения элементов группы корректны
data	`Object`	Текущие значения элементов группы

yavOptions : `Object`

Параметры валидации.

Kind: global typedef
Properties

Name	Type	Default	Description
[disabled]	`boolean`	`false`	Проверять элементы с атрибутом "disabled"
[readonly]	`boolean`	`false`	Проверять элементы с атрибутом "readonly"
[hidden]	`boolean`	`false`	Проверять невидимые (display=none) элементы
[onedit]	`boolean`	`false`	Проверять в ходе редактирования

Обратная связь

Aleksey Lutovinin crossrw1@gmail.ru

Лицензия