Яндекс

SVARX:
борьба с большими формами

Макс Ширшин
Руководитель группы разработки интерфейсов
Рекламных Технологий

Я.Субботник, Новосибирск, 19 ноября 2011

Большие формы
и клиентская валидация

Зачем нужна клиентская валидация?

- защита от **случайных** ошибок
- **пояснить** суть ошибки, **указать** способ устранения
- **плюс** перезагрузка — **минус** конверсия :-(


Клиентская валидация
Серверная валидация
Общие правила

Клиентская валидация
часто делается «на коленках»

```
<form onsubmit="if (!this.elements.email.value) return
false; if (this.elements.firstname.value == '' ||
this.elements.secondname.value == '') {alert('Укажите имя
и фамилию'); return false}">
```

```
<script>
$(function(){$('form').eq(1).submit(function() {
  var eml = $(this).find('input[name=email]').val();
  eml = $.trim(eml);
  var emlRe = /[\w\d]+[2,20]\@[\w\d]+[2,20]\.\w+[2,10]/;
  if (!emlRe.test(eml)) return false;
});});
</script>
```

Большие веб-формы — большие ожидания

— менеджер хочет **быстро**
— пользователь хочет **удобно**
— дизайнер просит **красиво**
— коллеги **ненавидят баги**

По́том и кровью выстраданные принципы:

- проверка может включать **более одного поля**
- валидация может зависеть **от внешних условий**
- тексты ошибок **отделяем** от самих правил
- правила **удобно** хранить отдельно
- **препроцессинг** данных — часть валидации
- нужен **блочный** формат

Что такое SVARX?

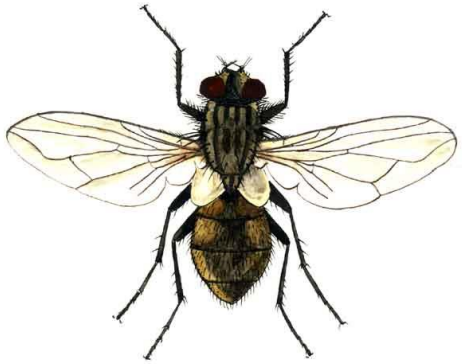
SVARX — это...
Semantical **VA**lidation **R**ulesets in **X**ML

Готовая идеология работы с правилами валидации

Всё отдельно:

- Смысловые (**семантические**) правила
- Код, выполняющий проверки
- Тексты ошибок
- Способ реакции на ошибки

SVARX — это...
Semantical **VA**lidation **R**ulesets in **X**ML



Мухи — отдельно

Котлеты — отдельно

SVARX

- **human-readable** XML-формат
- логика **and**-**or**-**not**, условные блоки **if**-**then**-**else**
- любые правила: встроенные и **ваши собственные**
- валидация по частям **new**
- **препроцессинг** данных

И ещё...

...есть плагин для ***jQuery***, который умеет:

- **читать** SVARX-формат (XML или **JSON bridge**[new])
- **валидировать** веб-формы
- интегрировать любые способы **показа ошибок** *
- добавлять **пользовательские правила** *

* *внешнее API плагина*

Как устроен SVARX-формат

Общая структура

```
<svarx>
    <preprocess>
        <!-- правила
препроцессинга -->
    </preprocess>

    <validate>
        <!-- правила валидации -->
    </validate>
</svarx>
```

Каркас типичного правила

```
<svarx>
   <preprocess>
      <!-- правила препроцессинга -->
   </preprocess>

   <validate>
      <rule
         for = “имя поля формы”
         type = “тип валидации”
         onerror = “идентификатор ошибки”
         ... доп. атрибуты правила
      />

   </validate>
</svarx>
```

Простой пример

```
<preprocess>
    <rule for="age" type="parseint" />
</preprocess>
<validate>
     <rule
          for="age" type="required"
          onerror="age-not-specified" />
      <rule
          for="age" type="range"
              min="14"
              max="99"
          onerror="age-incorrect" />
</validate>
```

Составное правило

```
<block logic="and"
       onerror="epic-fail">
    <rule
        for="user-email"
        type="email" />
    <rule for="age" type="required" />
    <rule for="age" type="range"
        min="14"
        max="99" />
</block>
```

Многоэлементное правило

```
<rule type="eq" comparison="string"
      onerror="fail">
    <!-- сравниваем 2 значения -->
    <el name="password1" />
    <el name="password2" />

    <!-- опционально можем переопределять
         элемент, который будет target'ом
         svarx-ошибки  -->
    <errtarget name="password1" />
</rule>
```

Условные проверки

```
<block logic="if">
    <!-- IF-условие -->
    <rule for="agreed" type="checked" />
    <!-- THEN-условие -->
    <block onerror="need_email">
        <rule for="email" type="required" />
        <rule for="email" type="email" />
    </block>
    <!-- ELSE-условие (опционально) -->
    <block onerror="tel_number_required">
        <rule for="tel" type="required" />
    </block>
</block>
```

Ещё вкусности

```
<!-- отрицание NOT -->
<rule for="address" type="regexp"
      match="[A-Z]"
      inverted="yes"
      onerror="some-error" />

<!-- выборка из одноимённых элементов -->
<rule for="address" type="email"
     item="0"
     onerror="fail" />
```

Валидация по частям

```
<!-- указываем id для правила или блока -->
<rule for="address" type="regexp"
      match="[A-Z]"
      id="unique-id"
      onerror="some-error" />
```

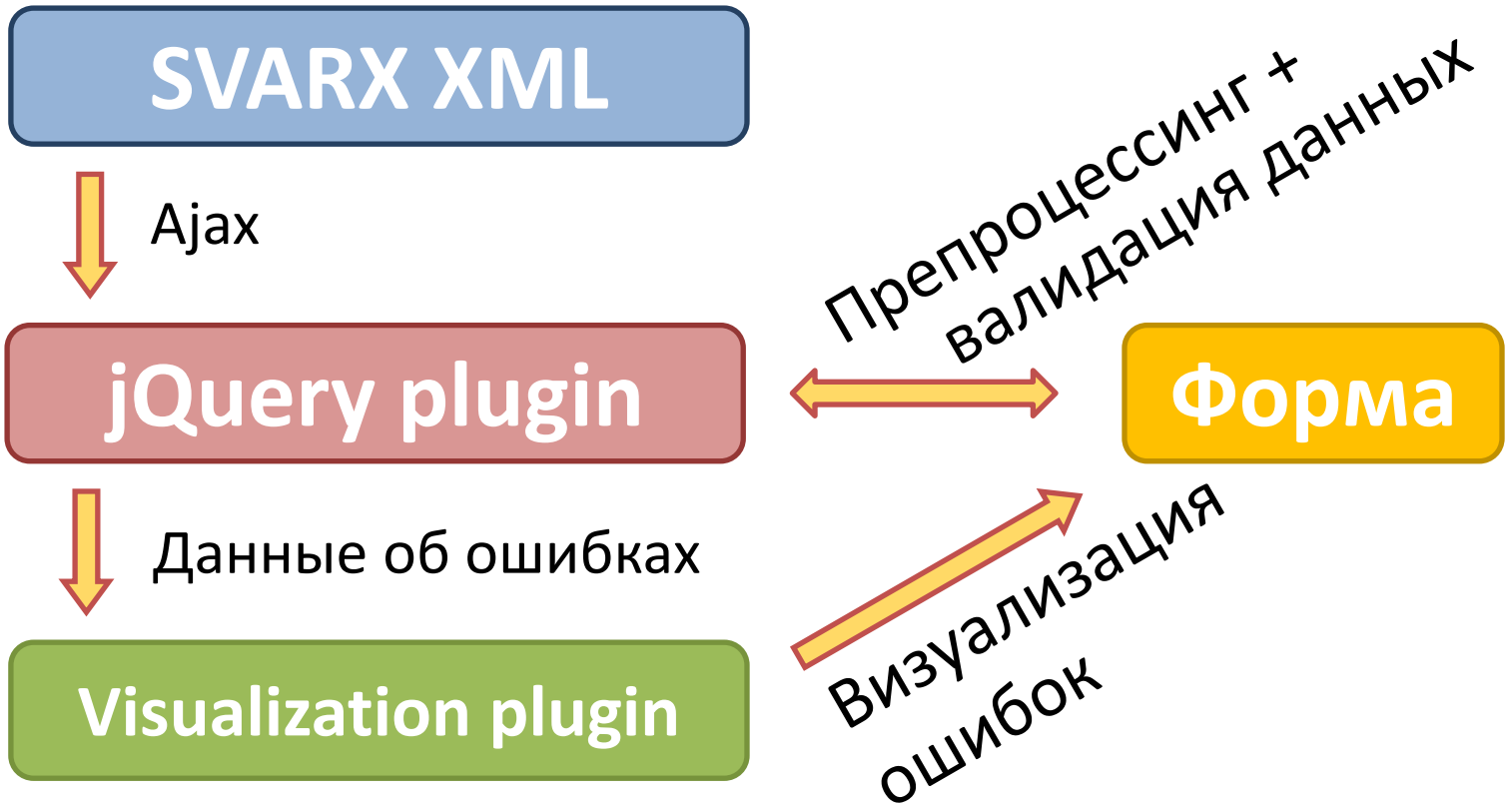
```
// можно однократно вызвать проверку
// одного правила или блока правил
$('form').trigger(
    'myValidationEvent',
    [{validationBlockId: 'unique-id'}]
);
```

Важные договорённости

- пустые элементы дают TRUE на любой проверке...
- ...кроме правила required
- несуществующие элементы дают TRUE на любой проверке
- условные проверки от несуществующих элементов пропускаются целиком (IF-THEN-ELSE)
- ...кроме случая, когда мы явно укажем обратное failifnull=“yes”

SVARX и JavaScript

Архитектура SVARX на клиенте



Подключение на проект

```
<script type="text/javascript"
  src="jquery.svarx.js"></script>
<script type="text/javascript"
  src="svarx.methods.messages.js"></script>
<script type="text/javascript">
$(function() {
  $('form').svarx({
     svarxURL: '/validation/svarx.xml',
     bindTo: 'submit',
     immutable: true
  });
});
</script>
```

Что такое «плагин визуализации»?

```
$.fn.svarx.methods.messages = {
    before: function(e) {},
    after:  function(e, result, eType) {
        if (!result && eType == 'submit') {
            e.preventDefault();
        }
    },
    error: function(e, id) {
        alert('Error '
            + id
            + ' at element '
            + e.target.name);
    }
};
```

Что мы с этого имеем?

Плюсы

- set it and forget it
- перестаём писать JS-валидацию на коленках
- порою — вообще перестаём писать JS-валидацию
- все знания о валидации в одном файле
- правим отображение ошибок независимо от правил

Где уже применяется

http://sprav.yandex.ru

http://passport.yandex.ru

\+ внутренние проекты Яндекса

To Do по состоянию на май 2011

- нет асинхронных проверок
- нет условий, не зависящих от состояния формы
- XML — not the way to go?
- скорость в IE
- нет «компиляции» SVARX XML в JS-код
- мало syntax sugar внутри самого формата

Сделано:

- нет асинхронных проверок
  доступно howto в документации
- нет условий, не зависящих от состояния формы
  поддержаны правила без привязки к элементам
- XML — not the way to go?
  создан JSON bridge для генерации правил из JSON
- скорость в IE
  система внутренних кешей
- нет «компиляции» SVARX XML в JS-код

- мало syntax sugar внутри самого формата

Где взять?

Исходный код:

http://github.com/ingdir/svarx

Документация на русском:

http://github.com/ingdir/svarx/docs

Twitter (анонсы, техподдержка):

@svarx

Техподдержка:

max.shirshin@yandex.ru

Вопросы?

Макс Ширшин
Руководитель группы разработки
интерфейсов Рекламных Технологий

ingdir@yandex-team.ru