Oboe.js

Examples

This page is also available as a PDF.

此页面还提供 PDF 格式。

A simple download 一个简单的下载

It isn't what Oboe was built for but it works fine as a simple AJAX library. This might be a good tactic to drop it into an existing application before iteratively refactoring towards progressive loading. The call style should be familiar to jQuery users.

它不是构建 Oboe 的目的，但它作为一个简单的 AJAX 库工作得很好。在迭代重构为渐进式加载之前，将其放入现有应用程序可能是一个很好的策略。 jQuery 用户应该熟悉这种调用方式。

```
oboe('/myapp/things.json')
   .done(function(things) {

      // we got it
   })
   .fail(function() {

      // we don't got it
   });
```

Extracting objects from the JSON stream 从 JSON 流中提取对象

Say we have a resource called things.json that we need to fetch:

假设我们有一个名为 things.json 的资源需要获取：

```
{
   "foods": [
      {"name":"aubergine",    "colour":"purple"},
      {"name":"apple",        "colour":"red"},
      {"name":"nuts",         "colour":"brown"}
   ],
   "badThings": [
      {"name":"poison",       "colour":"pink"},
      {"name":"broken_glass", "colour":"green"}
   ]
}
```

On the client side we want to download and use this JSON. Running the code below, each item will be logged as soon as it is transferred without waiting for the whole download to complete.

在客户端，我们要下载并使用这个 JSON。运行下面的代码，每个项目将在传输后立即记录，而无需等待整个下载完成。

```
oboe('/myapp/things.json')
   .node('foods.*', function( foodThing ){
```

```
      // This callback will be called everytime a new object is
      // found in the foods array.

      console.log( 'Go eat some', foodThing.name);
   })
   .node('badThings.*', function( badThing ){

      console.log( 'Stay away from', badThing.name);
   })
   .done(function(things){

      console.log(
         'there are', things.foods.length, 'things to eat',
         'and', things.nonFoods.length, 'to avoid');
   });
```

Duck typing

Sometimes it is easier to say *what you are trying to find* than *where you'd like to find it*. Duck typing is provided for these cases.

有时说出你想找什么比你想在哪里找到它更容易。为这些情况提供了鸭子类型。

```
oboe('/myapp/things.json')
   .node('{name colour}', function( thing ) {
      // I'll be called for every object found that
      // has both a name and a colour
      console.log(thing.name, ' is ', thing.colour);
   };
```

Hanging up when we have what we need
当我们有需要的时候挂断电话

If you don't control the data source the service sometimes returns more information than your application actually needs.

如果您不控制数据源，服务有时会返回比您的应用程序实际需要的信息更多的信息。

If we only care about the foods and not the non-foods we can hang up as soon as we have the foods, reducing our precious download footprint.

如果我们只关心食物而不关心非食物，我们就可以在拥有食物后立即挂断，从而减少我们宝贵的下载足迹。

```
oboe('/myapp/things.json')
   .node({
      'foods.*': function( foodObject ){

         alert('go ahead and eat some ' + foodObject.name);
      },
      'foods': function(){
         // We have everything that we need. That's enough.
         this.abort();
      }
   });
```

Detecting strings, numbers 检测字符串、数字

Want to detect strings or numbers instead of objects? Oboe doesn't care about node types so the syntax is the same:

想要检测字符串或数字而不是对象？ Oboe 不关心节点类型，所以语法是一样的：

```
oboe('/myapp/things.json')
   .node({
      'colour': function( colour ){
         // (colour instanceof String) === true
      }
   });
```

Reacting before we get the whole object
在我们得到整个对象之前做出反应

As well as `node` events, you can listen on `path` events which fire when locations in the JSON are found, before we know what will be found there. Here we eagerly create elements before we have their content so that the page updates as soon as possible:

除了 `node` 事件，您还可以监听 `path` 事件，这些事件在找到 JSON 中的位置时触发，然后我们才知道那里会找到什么。在这里，我们在拥有内容之前急切地创建元素，以便页面尽快更新：

```
var currentPersonElement;
oboe('people.json')
   .path('people.*', function(){
      // we don't have the person's details yet but we know we
      // found someone in the json stream. We can eagerly put
      // their div to the page and then fill it with whatever
      // other data we find:
      currentPersonElement = $('<div class="person">');
      $('#people').append(currentPersonElement);
   })
   .node({
      'people.*.name': function( name ){
         // we just found out their name, lets add it
         // to their div:
         currentPersonElement.append(
            '<span class="name">' + name + '</span>');
      },
      'people.*.email': function( email ){
         // we just found out their email, lets add
         // it to their div:
         currentPersonElement.append(
            '<span class="email">' + email + '</span>');
      }
   });
```

Giving some visual feedback as a page is updating
在页面更新时提供一些视觉反馈

Suppose we're using progressive rendering to go to the next 'page' in a dashboard-style single page webapp and want to put some kind of indication on the page as individual modules load.

假设我们正在使用渐进式渲染转到仪表板样式的单页 Web 应用程序中的下一个“页面”，并希望在加载单个模块时在页面上放置某种指示。

We use a spinner to give visual feedback when an area of the page is loading and remove it when we have the data for that area.

我们使用微调器在加载页面区域时提供视觉反馈，并在我们拥有该区域的数据时将其删除。

```
// JSON from the server side:
{
   'progress':[
      'faster loading',
      'maintainable velocity',
      'more beer'
   ],
   'problems':[
      'technical debt',
      'team drunk'
   ]
}
```

```
MyApp.showSpinnerAt('#progress');
MyApp.showSpinnerAt('#problems');

oboe('/agileReport/sprint42')
   .node({
      '!.progress.*': function( itemText ){
         $('#progress')
            .append('<div>')
            .text('We made progress in ' + itemText);
      },
      '!.progress': function(){
         MyApp.hideSpinnerAt('#progress');
      },
      '!.problems.*': function( itemText ){
         $('#problems')
            .append('<div>')
            .text('We had problems with ' + itemText);
      },
      '!.problems': function(){
         MyApp.hideSpinnerAt('#problems');
      }
   });
```

Taking meaning from a node's location
从节点的位置获取意义

Node and path callbacks receive a description of where items are found as an array of strings describing the path from the JSON root. It is sometimes preferable to register a wide-matching pattern and use the item's location to decide programmatically what to do with it.

节点和路径回调接收关于在何处找到项目的描述，作为描述来自 JSON 根的路径的字符串数组。有时最好注册一个广泛匹配的模式，并使用项目的位置以编程方式决定如何处理它。

```
// JSON data for homepage of a social networking site.
// Each top-level object is for a different module on the page.
{  "notifications":{
      "newNotifications": 2,
      "totalNotifications": 8
   },
   "messages": [
      {  "from":"Joe",
         "subject":"Wanna go fishing?",
         "url":"messages/1"
      },
```

```
      {  "from":"Baz",
         "subject":"Hello",
         "url":"messages/2"
      }
   ],
   "photos": {
      "new": [
         {  "title": "Birthday Party",
            "url":"/photos/5",
            "peopleTagged":["Joe","Baz"]
         }
      ]
   }
   // ... other modules ...
}
```

```
oboe('http://mysocialsite.example.com/homepage')
   .node('!.*', function( moduleJson, path ){

      // This callback will be called with every direct child
      // of the root object but not the sub-objects therein.
      // Because we're maching direct children of the root the
      // path argument is a single-element array with the module
      // name; it resembles ['messages'] or ['photos'].
      var moduleName = path[0];

      My.App
         .getModuleCalled(moduleName)
         .showNewData(moduleJson);
   });
```

Deregistering a callback 注销回调

Calling `this.forget()` from inside a callback deregisters that listener.

从回调内部调用 `this.forget()` 会注销该侦听器。

```

// We have a list of items to plot on a map. We want to draw
// the first ten while they're loading. After that we want
// to store the rest in a model to be drawn later.

oboe('/listOfPlaces')
   .node('list.*', function( item, path ){
      var itemIndex = path[path.length-1];

      model.addItemToModel(item);
      view.drawItem(item);

      if( itemIndex == 10 ) {
         this.forget();
      }
   })
   .done(function( fullJson ){
      var undrawnItems = fullJson.list.slice(10);

      model.addItemsToModel(undrawnItems);
   });
```

Css4 style patterns Css4样式图案

Sometimes when downloading an array of items it isn't very useful to be given each element individually. It is easier to integrate with libraries like Angular if you're given an array repeatedly whenever a new element is

concatenated onto it.

有时在下载项目数组时，单独给定每个元素不是很有用。如果每当将新元素连接到数组上时重复给定数组，则与 Angular 等库集成会更容易。

Oboe supports css4-style selectors and gives them much the same meaning as in the proposed css level 4 selector spec.

Oboe 支持 css4 样式的选择器，并赋予它们与提议的 css level 4 选择器规范中相同的含义。

If a term is prefixed with a dollar sign, the node matching that term is explicitly selected, even if the pattern as a whole matches a node further down the tree.

如果术语以美元符号为前缀，则显式选择与该术语匹配的节点，即使模式作为一个整体与树中更下方的节点匹配也是如此。

```
// JSON
{"people": [
   {"name":"Baz", "age":34, "email": "baz@example.com"}
   {"name":"Boz", "age":24}
   {"name":"Bax", "age":98, "email": "bax@example.com"}}
]}
```

```
// we are using Angular and have a controller:
function PeopleListCtrl($scope) {

   oboe('/myapp/things')
      .node('$people[*]', function( peopleLoadedSoFar ){

         // This callback will be called with a 1-length array,
         // a 2-length array, a 3-length array etc until the
         // whole thing is loaded.
         // Putting this array on the scope object under
         // Angular re-renders the list of people.

         $scope.people = peopleLoadedSoFar;
      });
}
```

Like css4 stylesheets, this can also be used to express a 'containing' operator.

与 css4 样式表一样，这也可用于表达"包含"运算符。

```
oboe('/myapp/things')
   .node('people.$*.email', function(personWithAnEmailAddress){

      // here we'll be called back with baz
      // and bax but not Boz.

   });
```

Transforming JSON while it is streaming
在流式传输时转换 JSON

Say we have the JSON below:

假设我们有以下 JSON：

```
[
   {"verb":"VISIT", "noun":"SHOPS"},
   {"verb":"FIND", "noun":"WINE"},
```

```
   {"verb":"MAKE", "noun":"PIZZA"}
]
```

We want to read the JSON but the uppercase is a bit ugly. Oboe can transform the nodes as they stream:

我们想读取 JSON 但大写有点难看。 Oboe 可以在流式传输时转换节点：

```
function toLower(s){
   return s.toLowerCase();
}

oboe('words.json')
   .node('verb', toLower)
   .node('noun', toLower)
   .node('!.*', function(pair){

      console.log('Please', pair.verb, 'me some', pair.noun);
   });
```

The code above logs: 上面的代码记录：

```
Please visit me some shops
Please find me some wine
Please make me some pizza
```

Demarshalling JSON to an OOP model
将 JSON 解组为 OOP 模型

If you are programming in an OO style you probably want to instantiate constructors based on the values you read from the JSON.

如果您使用 OO 风格进行编程，您可能希望根据从 JSON 中读取的值来实例化构造函数。

You can build the OOP model while streaming using Oboe's node replacement feature:

您可以在使用 Oboe 的节点替换功能进行流式传输时构建 OOP 模型：

```
function Person(firstName, lastName) {
   this.firstName = firstName;
   this.lastName = lastName;
}

Person.prototype.getFullName = function() {
   return this.firstName + ' ' + this.lastName;
}

oboe('/myapp/people')
   .node('people.*', function(person){

      // Any value we return from a node callback will replace
      // the parsed object:

      return new Person(person.firstName, person.lastName)
   })
   .done(function(model){

      // We can call the .getFullName() method directly because the
      // model here contains Person instances, not plain JS objects
```

```
      console.log( model.people[1].getFullName() );
   });
```

Loading JSON trees larger than the available RAM
加载大于可用 RAM 的 JSON 树

By default Oboe assembles the full tree of the JSON that it receives. This means that every node is kept in memory for the duration of the parse.

默认情况下，Oboe 会组装它接收到的 JSON 的完整树。这意味着在解析期间每个节点都保存在内存中。

If you are streaming large resources to avoid memory limitations, you can delete any detected node by returning `oboe.drop` from the node event.

如果您流式传输大量资源以避免内存限制，您可以通过从节点事件返回 `oboe.drop` 来删除任何检测到的节点。

```
// we are getting this large JSON
{
   "drinks":[
      {"name":"Orange juice", "ingredients":"Oranges"},
      {"name":"Wine", "ingredients":"Grapes"},
      {"name":"Coffee", "ingredients":"Roasted Beans"}

      // ... lots more records ...
   ]
}
```

```
oboe('drinks.json')
   .node('!.*', function(drink){

      if( available(drink.ingredients) ) {
         startMaking(drink.name);
      }

      // By returning oboe.drop, the parsed JSON object will be freed,
      // allowing it to be garbage collected.
      return oboe.drop;

   }).done(function( finalJson ){

      // most of the nodes have been dropped

      console.log( finalJson );  // logs: {"drinks":[]}
   })
```

There is also a shorthand form of `oboe.drop` if you want to drop nodes without examining them first:

如果您想在不先检查节点的情况下删除节点，还有一个简写形式 `oboe.drop`：

```
oboe('drinks.json')
   // we don't care how the drink is made
   .node('ingredients', oboe.drop)
   .done(function( finalJson ){

      console.log( finalJson.drinks );
      //  [ {"name":"Orange juice"}, {"name":"Wine"}, {"name":"Coffee"} ]}
   })
```

Streaming out HTML from express 从 express 流出 HTML

Generating a streamed HTML response from a streamed JSON data service.

从流式 JSON 数据服务生成流式 HTML 响应。

```
app.get('/foo', function(req, res){
   function writeHtml(err, html){
      res.write(html);
   }

   res.render('pageheader', writeHtml);

   oboe( my_stream )
      .node('items.*', function( item ){
          res.render('item', item, writeHtml);
      })
      .done(function() {
          res.render('pagefooter', writeHtml);
      })
});
```

Using Oboe with d3.js 将双簧管与 d3.js 一起使用

Oboe works very nicely with d3.js to add content to a visualisation while the JSON downloads.

Oboe 与 d3.js 配合得很好，可以在下载 JSON 的同时将内容添加到可视化。

```
// get a (probably empty) d3 selection:
var things = d3.selectAll('rect.thing');

// Start downloading some data.
// Every time we see a new thing in the data stream, use
// d3 to add an element to our visualisation. This pattern
// should work for most d3 based visualistions.
oboe('/data/things')
   .node('$things.*', function( thingsArray ){

      things.data(thingsArray)
         .enter().append('svg:rect')
            .classed('thing', true)
            .attr(x, function(d){ return d.x })
            .attr(y, function(d){ return d.x })
            .attr(width, function(d){ return d.w })
            .attr(height, function(d){ return d.h })

      // no need to handle update or exit set here since
      // downloading is purely additive
   });
```

Reading from Node.js streams 从 Node.js 流中读取

Instead of giving `oboe` a URL you can pass any ReadableStream. To load from a local file you'd do this:

您可以传递任何 ReadableStream 而不是给 `oboe` 一个 URL。要从本地文件加载，您可以这样做：

```
oboe( fs.createReadStream( '/home/me/secretPlans.json' ) )
   .on('node', {
```

```
    'schemes.*': function(scheme){
       console.log('Aha! ' + scheme);
    },
    'plottings.*': function(deviousPlot){
       console.log('Hmmm! ' + deviousPlot);
    }
 })
 .on('done', function(){
    console.log("*twiddles mustache*");
 })
 .on('fail', function(){
    console.log("Drat! Foiled again!");
 });
```

Because explicit loops are replaced with pattern-based declarations, the code is usually about the same length as with JSON.parse:

因为显式循环被基于模式的声明所取代，所以代码的长度通常与 JSON.parse 的长度大致相同：

```
fs.readFile('/home/me/secretPlans.json',
   function(err, plansJson){
      if(err) {
         console.log("Drat! Foiled again!");
         return;
      }
      var plans = JSON.parse(plansJson);

      plans.schemes.forEach(function(scheme){
         console.log('Aha! ' + scheme);
      });
      plans.plottings.forEach(function(deviousPlot){
         console.log('Hmmm! ' + deviousPlot);
      });

      console.log("*twiddles mustache*");
   });
```

Rolling back on error 错误回滚

The fail event notifies when something goes wrong. If you have started putting elements on the page and the connection goes down you have a few options

fail 事件会在出现问题时发出通知。如果您已经开始在页面上放置元素并且连接中断，您有几个选择

- If the new elements you added are useful without the rest, leave them. For example, in a web-based email client it is more useful to show some messages than none. See dropped connections visualisation.

  如果您添加的新元素在没有其他元素的情况下很有用，请保留它们。例如，在基于 Web 的电子邮件客户端中，显示一些消息比不显示消息更有用。查看丢弃的连接可视化。

- If they are useful but you need the rest, make a new request. If the service supports it you need only ask for the missing items.

  如果它们很有用但您还需要其余的，请提出新的请求。如果服务支持，您只需索要丢失的物品。

- Rollback any half-done changes. 回滚任何未完成的更改。

The example below implements rollback.

下面的例子实现了回滚。

```
var currentPersonElement;
oboe('everyone')
   .path('people.*', function(){
      // we don't have the person's details yet but we know we
      // found someone in the json stream, we can use this to
      // eagerly add them to the page:
      currentPersonElement = $('<div class="person">');
      $('#people').append(currentPersonElement);
   })
   .node('people.*.name', function( name ){
      // we just found out that person's name, lets add it to
      // their div:
      var markup = '<span class="name">' + name + '</span>';
      currentPersonElement.append(markup);
   })
   .fail(function(){
      if( currentPersonElement ) {
         // oops, that didn't go so well. instead of leaving
         // this dude half on the page, remove them altogether
         currentPersonElement.remove();
      }
   })
```

Example patterns

`*`	Every object, string, number etc found in the json stream 在 json 流中找到的每个对象、字符串、数字等
`!`	The root object. Fired when the whole response is available, like JSON.parse() 根对象。当整个响应可用时触发，例如 JSON.parse()
`!.foods.colour`	The colours of the foods 食物的颜色
`person.emails[1]`	The first element in the email array for each person 每个人的电子邮件数组中的第一个元素
`{name email}`	Any object with a name and an email property, regardless of where it is in the document 具有名称和电子邮件属性的任何对象，无论它在文档中的什么位置
`person.emails[*]`	Any element in the email array for each person 每个人的电子邮件数组中的任何元素
`person.$emails[*]`	Any element in the email array for each person, but the callback will be passed the array so far rather than the array elements as they are found. 每个人的电子邮件数组中的任何元素，但回调将传递到目前为止的数组而不是找到的数组元素。

`person`	All people in the json, nested at any depth json中的所有人，嵌套在任意深度
`person.friends.*.name`	Detecting friend names in a social network 检测社交网络中的好友名称
`person.friends..{name}`	Detecting friends with names in a social network 在社交网络中检测有名字的朋友
`person..email`	Email addresses anywhere as descendant of a person object 任何地方的电子邮件地址作为个人对象的后代
`person..{email}`	Any object with an email address relating to a person in the stream 任何具有与流中的人相关的电子邮件地址的对象
`$person..email`	Any person in the json stream with an email address json 流中具有电子邮件地址的任何人

Improve this page 改进此页面

Is this page clear enough? Typo-free? Could you improve it?

这个页面够清楚吗？无错字？你能改进一下吗？

Please fork the Oboe.js website repo and make a pull request. The markdown source is here.

请分叉 Oboe.js 网站 repo 并提出拉取请求。降价源在这里。