Article
Vitaliy Serdtsev · Dec 15, 2017 5m read

$(REST - CSP - (-HyperEvents) + EasyUI + File Upload). Part 4

... but let ’ s first take a look at two relatively small topics, and namely:

Creation of a multi-page REST application

We only have one page for now. How about adding one more? Said - done.

```
...
<Route Url="/upload" Method="POST" Call="SrvUpload"/>
<Route Url="/secondpage" Method="GET" Call="SecondPage"/>
...

ClassMethod SecondPage() As %Status
{
  s %response.ContentType="text/html"

  &html<
<!DOCTYPE html>
<html>
  <head>
    <meta charset="#(..#CHARSET)#">
    <meta http-equiv="X-UA-Compatible" content="IE=edge"/>
    <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-
scale=1, user-scalable=no"/>
    <title>Another page</title>
  </head>
  <body>
    <button onclick="alert('This is a different page')">Click me again</button>
    <a href=".">Go to the main REST page </a>
  </body>
</html>>
  q $$$OK
}
```

Let ’ s add a link to a new page on the main page:

```
...
<input id="fileupload" type="file" name="files[]" data-url="upload" multiple data-
sequential-uploads="false">
<a href="secondpage">Go to another REST page</a>
...
```

You can also add extra parameters to the URL and read this data on the page.

Important:
Since the creation of REST pages is something totally “ new ” compared with csp/cls, you will not yet be able to use some security-related CSP/CLS features, for example: private pages (PRIVATE=1 parameter), transparent validation of input parameters (ENCODED=2 parameter) and the **Link** method, accordingly. Therefore, you will need to manually use the **..Encrypt**/**..Decrypt** methods for now.

Let ’ s take a look at a small example:

MainPage:

```
...
<a href="secondpage">Go to REST page 1</a><br>
<a href="secondpage?a=1&b=test">Go to REST page 2</a><br>

#[s p("parameter")="secondpage.csp",p("<script>")="alert(""<!>"")"]#
<a href="#($replace(..Link("secondpage.csp",.p),".csp","",,1))#"
>Go to REST page 3</a>
...
```

SecondPage:

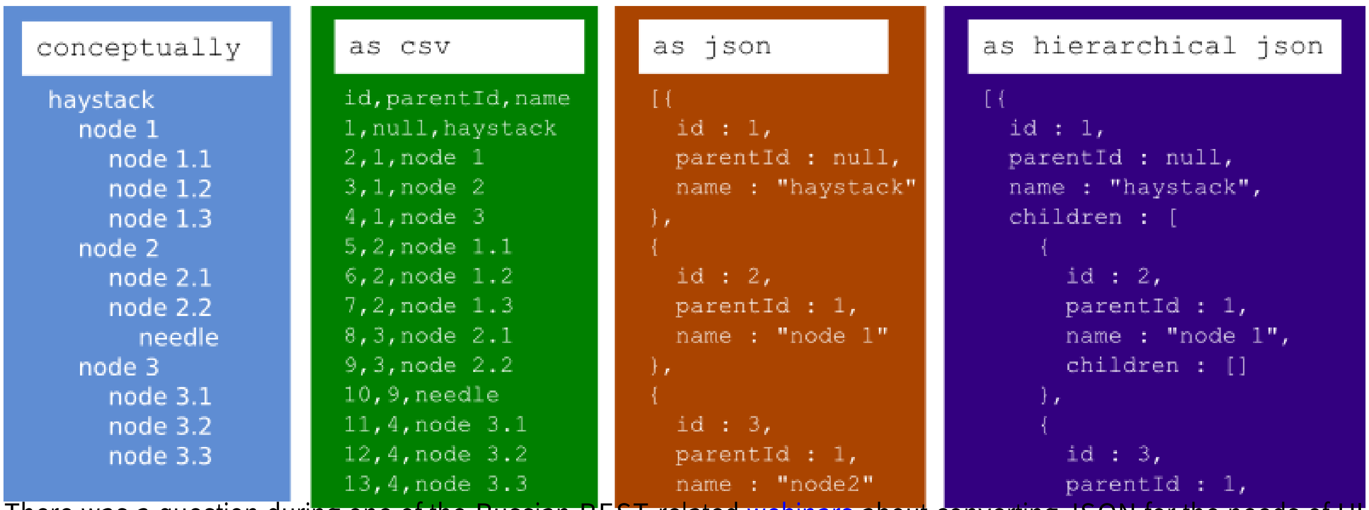
```
...
<body>
  That’s what was passed to us:<br>
  a = #(%request.Data("a",1))#<br>
  b = #(%request.Data("b",1))#<br>
  parameter = #(%request.Data("parameter",1))#<br>
  &lt;script&gt; = #(..EscapeHTML(%request.Data("&lt;script&gt;",1)))#<br>
  <button onclick="alert('This is different page!')">Click me 2</button>
  <a href=".">Go to the main REST page</a>
</body>
...
```

If you have many parameters and they are all complex, it ’ s a better idea to use the **Link** method that will do the job for you. However, it does not yet support page names without extensions, so you ’ ll need to adjust. Alternatively, you can redefine it:

```
ClassMethod Link(
  link As %String,
  ByRef query As %String,
  addQ As %Boolean = 0) As %String [ CodeMode = expression, ServerOnly = 1 ]
{
$replace($$cspLink^%SYS.cspServer(link_".csp",.query,addQ),".csp","",,1)
}
```

In this case, the code above will be somewhat simplified.

Conversion of JSON from the {id:1,parentId:1} format to {id:1,children:[{}]} for tree visualization



```
haystack
  node 1
    node 1.1
    node 1.2
    node 1.3
  node 2
    node 2.1
    node 2.2
      needle
  node 3
    node 3.1
    node 3.2
    node 3.3
```

```
id,parentId,name
1,null,haystack
2,1,node 1
3,1,node 2
4,1,node 3
5,2,node 1.1
6,2,node 1.2
7,2,node 1.3
8,3,node 2.1
9,3,node 2.2
10,9,needle
11,4,node 3.1
12,4,node 3.2
13,4,node 3.3
```

```
[{
  id : 1,
  parentId : null,
  name : "haystack"
},
{
  id : 2,
  parentId : 1,
  name : "node 1"
},
{
  id : 3,
  parentId : 1,
  name : "node2"
```

```
[{
  id : 1,
  parentId : null,
  name : "haystack",
  children : [
    {
      id : 2,
      parentId : 1,
      name : "node 1",
      children : []
    },
    {
      id : 3,
      parentId : 1,
```

There was a question during one of the Russian REST-related webinars about converting JSON for the needs of UI frameworks that are often designed to work with a single format, while backend developers often return data in a different format.

So here comes a question: what can be done ~~and who ’ d to blame~~?

Answer: use what others have already made for you.

Nothing needs to be done on the server side. On the client side, we need to call just one short method. Different kinds of formats may be required and you can hardly make provisions for all of them. Besides, there is no point in overloading the server if you can relatively easily do it on the client side.

The proposed code in JavaScript looks like this:

```
_makeTree: function(options) {
  var children, e, id, o, pid, temp, _i, _len, _ref;
  id = options.id || "id";
  pid = options.parentid || "parentid";
  children = options.children || "children";

  // create the index
  temp = {};
  _ref = options.q;

  for (_i = 0, _len = _ref.length; _i < _len; _i++) {
    e = _ref[_i];
    // predefine the childs array
    e[children] = [];
    // write the id index to find by parentid in second loop
    temp[e[id]] = e;
  }

  // Create the tree
  o = [];
  _ref = options.q;
  for (_i = 0, _len = _ref.length; _i < _len; _i++) {
    e = _ref[_i];
    // This parent should be in the index
    if (temp[e[pid]] != null) {
      // This row is a child
      // Add the child to the parent
      temp[e[pid]][children].push(e);
    } else {
      // Add a root item
      o.push(e);
    }
  }
  return o;
}
```

We will need it for the next part, which will conclude the series of articles devoted to REST applications.

#REST API #UI Development #Frontend #Caché

Source URL:https://community.intersystems.com/post/rest-csp-hyperevents-easyui-file-upload-part-4