1. **Collection+JSON Clients**

人类与较小的灵长类动物的不同在于他对列表的热衷—H. Allen Smith

**Bob and Carol**



你知道吗，Carol，我已经回顾了Collection+JSON的超媒体格式，并且想知道它是否能在追求更加适用的API客户端上给我们帮助

很有趣你会提到Cj，Bob。我的团队正考虑把它作为下一个尝试。他们告诉我它会很有希望。





是的，我注意到Cj看起来与Siren和HAL很像，但是在文档中也有一些关于客户端如何使用CRUD模式的东西，这些东西与我们的普通JSON客户端所做的方式是相似的。

没错，除了支持CRUD，Cj提供了与Siren中的action元素相似的对查询模板的支持。





但是，我有点怀疑Cj发送Items集合中的数据的方式，这似乎有点奇怪。

嗯，那是昨天我们主要讨论的对象。Cj对每个领域模型需要额外的元数据，并且那意味着它很难像我们在Siren和HAL中做的那样简单地“反序列化”我们的领域模型到响应中。





哈，那这样对于服务器端表述转换器是更多工作，客户端是更多载荷，对吗？我想知道，我是否需要全部的信息。

嗯，据我们所知，Cj载荷的大小并不比我们标准的HTML载荷要大，并且我们总是发送HTML。





嗯，有意思，我想知道是否元数据提供一些我们看不到的重要信息？

对，我客户端团队中的一些人认为，随着时间的推移，额外的元数据会提升适应性。





真的吗？好吧，让我们试试看，我的团队可以很快地在服务器端建立一个Cj表述转换器。你的团队应该能有没有太多麻烦地创建一个Cj客户端，对吧Carol？

是的，Bob。让我们看看我们的团队想出了什么，一周后见。





好的，Carol。开始吧！

我们将在本书中回顾的最后一个超媒体格式是Collection+JSON格式。它跟HAL和Siren有相似之处，但是也有一种相当独特的方式。Cj从一开始就被作为一种列表风格的格式设计出来——这意味着要返回记录的列表。正如我们将看到的，当我们在下一节中回顾这种设计的时候，这种格式还有更多的东西而不仅仅是列表，但是列表是Cj的全部。

Cj也从现今大多数Web API客户端使用的传统的增删改查（CRUD）模式中获得启发。我们在XX章中讨论了这种风格，并且在那一章，从我们创建的客户端中获得的一些经验教训将适用于Cj客户端。

在这一章，我们将看一下格式的设计、Cj表述转换器代码和Cj通用客户端。然后，正如我们对其它客户端的实现，我们将引入一些变化来看，随着后端API做向后兼容改动，API客户端是如何很好地应对的。

最后，我们将确保检查在沿着这条路线的进展上满足OAA（OBJECTS、ADDRESSES、ACTIONS）的要求。虽然HAL擅长处理ADDRESSES，Siren对ACTIONS有很好的支持，但是Cj是被设计用于满足OBJECTS要求的——在运行时共享领域模型相关的元数据的能力。并且，正如我们会在回顾中所看到的，Cj用一种新颖的解决方案满足OBJECTS要求——通过使它们对客户端-服务器的实践变得无关紧要。

**Collection+JSON格式**

2011年，我设计并发布Collection+JSON的格式——同年，Mike Kelly发布了他的HAL规范。Collection+JSON（又名Cj）是被设计来让管理数据列表，例如：博客帖子、客户、产品，用户等等变得简单。Cj规范页上的描述说：

“[Cj]类似于Atom联合格式（RFC4287）和Atom发布协议（RFC5023）。但是，Collection+JSON在单个媒体类型中定义了格式和协议语义。[Cj]也包含对查询模板的支持并通过写模板（Write Template）的使用扩展了写支持（write support）。”

——Collection+JSON规范

本质上，Cj是Atom以带有表单的JSON表示的形式。好消息是Cj遵循Atom对增删改查（CRUD）模式的支持。这意味着大多数开发者可以很容易地理解Cj的读/写语义。Cj额外的好处是它有用于描述类似HTML表单、过滤数据（用Cj的queries元素）和更新服务器上（通过template元素）的内容的元素。然而，正如我们会在随后的回顾中所看到的，template元素使用的方式会有点挑战性。

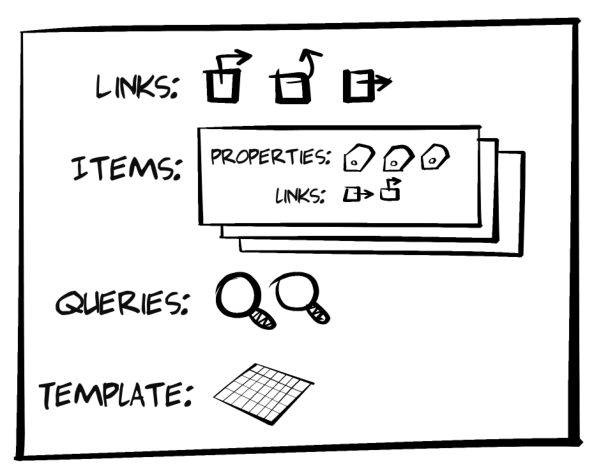


图5-1. Collection+JSON文档模型

你可以通过查看在线的Cj文档来更好地理解Collection+JSON媒体类型。也有Cj的讨论列表和额外的信息共享的github组织。查看本章参考资料部分了解详情。

每个Cj报文的基本元素有：

* Links：一个或多个link元素的集合。它们与HAL和Siren的link元素很相似。
* Items：一个或多个数据项——本质上是APIs领域模型。HAL和Siren的properties与Cj的items很相似。
* Queries：这些本质上是HTML中GET方法的表单。Cj的queries就像HAL中的模板链接、Siren中的action元素（method设为“GET”）。
* Template：在Cj中，所有的写操作（HTTP POST和PUT）都是使用template元素完成。它包括一个或多个data对象——每一个就像HTML的input元素。此外，它就像Siren的action元素。HAL没有什么可以与Cj的template元素相比。

Cj还有一个error元素用于返回错误信息以及一个content元素用于返回自由格式的文本和标记。今天我们在这里将不涉及到这些。你可以在参考资料部分中提到的Cj的文档中仔细研究它们。

这里有一个简单的Collection+JSON报文的例子，它展示了一个Cj文档的主要部分，包括：links(#1)，items(#2)，queries(#3)和template(#4)元素。

{

"collection": {

"version": "1.0",

"href": "http://orm-hyper-todo.herokuapp.com", #5

"title": "ORM Hyper-Tasks",

"links": [ #1

{

"href": "http://orm-hyper-todo.herokuapp.com/",

"rel": "collection",

"prompt": "All task"

}

],

"items": [ #2

{

"rel": "item",

"href": "http://orm-hyper-todo.herokuapp.com/1sv697h2yij",

"data": [

{"name": "id", "value": "1sv697h2yij", "prompt": "id"},

{"name": "title", "value": "Marina", "prompt": "title"},

{"name": "completed", "value": "false", "prompt": "completed"}

]

},

{

"rel": "item",

"href": "http://orm-hyper-todo.herokuapp.com/25ogsjhqtk7",

"data": [

{"name": "id", "value": "25ogsjhqtk7", "prompt": "id"},

{"name": "title", "value": "new stuff", "prompt": "title"},

{"name": "completed", "value": "true", "prompt": "completed"}

]

}

],

"queries": [ #3

{

"rel": "search",

"href": "http://orm-hyper-todo.herokuapp.com/",

"prompt": "Search tasks",

"data": [

{"name": "title", "value": "", "prompt": "Title"}

]

}

],

"template": { #4

"prompt": "Add task",

"rel": "create-form",

"data": [

{"name": "title", "value": "", "prompt": "Title"},

{"name": "completed", "value": "false", "prompt": "Complete"}

]

}

}

}

一个Cj文档中另外一个重要的属性就是根级别的href（看插图编号#5）。当添加新的记录到items集合中时就需要用到href的值。当我们在下面涉及到template元素的时候，我们将更多地讨论这个属性。

**Links**

Cj文档中的links元素始终是一个有效的JSON数组，它包含一个或多个link对象，重要的link元素属性包括：href、rel和prompt属性。它们和HTML <a>...</a>标签的作用相似——用于HTTP GET动作的静态URL。

在Collection+JSON中表示Links

"links": [

{

"href": "http://rwcbook12.herokuapp.com/home/",

"rel": "home collection",

"prompt": "Home"

},

{

"href": "http://rwcbook12.herokuapp.com/task/",

"rel": "self task collection",

"prompt": "Tasks"

},

{

"href": "http://rwcbook12.herokuapp.com/user/",

"rel": "user collection",

"prompt": "Users"

}

]

在Cj中，links部分典型地用于保存与当前文档相关的，或者在以人为中心的UI中，与当前的屏幕或者页面相关的链接。与应用程序（在上面的例子中）重要的导航链接一样，links部分可能包含页面级导航（first，previous，next，last）或者其它相似的链接。

在Cj link对象中另外一个方便的属性是render属性。它告诉消费者应用程序如何对待链接。例如：如果render的值设为none，客户端应用程序将不会显示该链接。当传送用于CSS样式表、概要文件的URL或者其它类型信息的link元素时，这样很方便。

在Collection+JSON中表示Links

"links": [

{

"href": "http://api.example.org/profiles/task-management",

"rel": "profile",

"render" : "none"

}

]

**Items**

在Cj文档中，最独特的元素可能就是item部分。Items部分与HAL根级别的properties、Siren的properties对象相似，Cj item包含响应中的领域对象，如：用户、顾客、产品等等。然而，与HAL、Siren表示领域对象的方式不同的是，Cj拥有一种高度结构化的方式。HAL和Siren要么以简单的名-值对，要么就Siren而言以subentities，表示它们的领域对象。HAL和Siren都支持发送嵌套的JSON对象作为属性。但是Cj不像那样做并且这可能是挫折和自由的来源。（？？？？）

这是一个用户对象表示为一个Cj item的例子

{

"rel": "item http://api.example.org/rels/user",

"href": "http://api.example.org/user/alice", #1

"data": [ #2

{"name": "id", "value": "alice", "prompt": "ID", "render":"none"},

{"name": "nick", "value": "alice", "prompt": "Nickname"},

{"name": "email", "value": "alice-ted@example.org", "prompt": "Email"},

{"name": "name", "value": "Alice Teddington, Jr.", "prompt": "Full Name"}

],

"links": [ #3

{

"prompt": "Change Password",

"rel": "edit-form http://api.example.org/rels/changePW",

"href": "http://api.example.org/user/pass/alice"

},

{

"prompt": "Assigned Tasks",

"rel": "collection http://api.example.org/rels/filterByUser",

"href": "http://api.example.org/task/?assignedUser=alice"

}

]

}

正如你所看到的在上面的例子中，一个Cj item包含rel和href（#1）、data元素的列表（#2）、也可能包含一个或多个用于只读且与item相关的行为的link元素。Cj表示item属性（id，nick，email和name）的方式在本书中提到的格式中是独特的。Cj文档不仅仅返回属性标识符和值（例如：”id”:”alice”），还有一个暗示用的prompt属性。Cj也支持其他属性包括render，用于帮助客户端决定是否显示屏幕上的属性。这种高度结构化的格式使发送领域数据和每个属性以及对象的元数据成为可能。当我们开始在Cj客户端应用上面工作时，我们会看到，当创建以人为中心的界面时，数据的添加很方便。

在每个Cj item中的links集合包含一个或多个静态安全的链接（像那些在根级别的links集合）。这个空间（？？？？）可以用来在Cj响应中传递item级别的链接。例如：在上面的代码片段中，你可以看到一个链接指出了一种用于更新用户密码的形式、一个链接指向一个过滤后的与用户对象相关的任务列表。Item级别的links部分是可选的，任何在集合中出现的链接必须被视为一个安全的链接（例如：非关联的使用HTTP GET（？？？？？））。

**Queries**

Collection+JSON中的queries元素用于保存用于一个或多个参数的安全请求（例如：HTTP GET）。这些类似于method属性设为GET的HTML表单。一个Cj文档中的queries部分是一个或多个query对象的数组。除了能有一个相关的data数组之外，它们看上去也类似于Cj的link对象。

这里有一个例子

{

"rel": "search",

"name" : "usersByEmai",

"href": "http://api.example.org/user/",

"prompt": "Search By Email",

"data": [

{

"name": "email",

"value": "",

"prompt": "Email",

"required": "true"

}

]

}

正如你从上面例子看到的，一个Cj query对象有rel、name、href和prompt属性。然后是一个或多个data元素。Data元素类似于HTML input元素。除了name、value和prompt属性，data元素还可以有required、（上面没有显示）readOnly和pattern属性。后面的这些属性帮助服务发送客户端用于查询的额外的相关参数的元数据。

注意，当执行查询的时候，Cj query对象没有一个用于显示使用哪个HTTP方法的属性。这是因为Cj queries始终使用HTTP GET方法。

还有另外一个类似于HTTP形式的Cj元素：template元素。

**Template**

Cj的template元素看上去类似于Cj的queries元素——但是更小。它只有一个包含一个或多个data对象的集合。这些代表一个写动作（例如：HTTP POST或PUT）的输入参数。这是一个Cj template的例子

"template": {

"prompt": "Add Task",

"data": [

{"name": "title", "value": "", "prompt": "Title", "required": "true"},

{"name": "tags", "value": "", "prompt": "Tags"},

{"name": "completeFlag", "value": "false", "prompt": "Complete",

"patttern": "true|false"}

]

}

Template元素可以有一个可选的prompt，但是template最重要的部分是描述用于写操作的可能的输入参数的data数组。像出现在Cj queries和items中的data元素一样，template的data除了一个prompt属性，还包含name和value属性。并且，就像queries版本的data元素一样，它可以有额外的元数据属性，包括readOnly、required和pattern。Pattern元素和HTML pattern属性的工作方式一样。

写操作有两个重要的方面，是Cj template中缺失的：1）目标URL，2）HTTP方法。那是因为，在Cj中，template应用于CRUD模型的两个不同部分：新建和更新。请求如何执行取决于客户端应用想要做什么。

**使用Cj Templates创建新的资源**

当用于创建集合的一个新成员时，客户端应用填充template，然后使用HTTP POST作为方法，Cj文档的href的值作为目标URL。

例如，使用Cj文档表示这章的开始，客户端应用可以收集来自用户的输入并发送一个POST请求来添加一个新的任务记录。HTTP请求是这样的

\*\*\* REQUEST \*\*\*

POST / HTTP/1.1

Host: http://orm-hyper-todo.herokuapp.com

Content-Type: application/vnd.collection+json

...

"template": {

"data": [

{"name": "title", "value": "adding a new record"},

{"name": "tags", "value": "testing adding"},

{"name": "completeFlag", "value": "false"}

]

}

Cj规范说客户端可以发送template块（如上所示）或者只发送data对象数组，服务端应该两者都接受。并且，服务端应该接受带有包含prompts和其它属性的data对象的载荷，只是忽略这些属性。

在上面的例子中可以看到，Cj文档中href的URL与HTTP POST方法用于添加一个新资源到Cj集合中。

**使用Cj Templates 更新已有的资源**

当客户端应用想更新一个已存在的资源时，它们使用HTTP PUT方法和item的href属性来更新。通常情况下，客户端应用将会使用已存在item的值自动填充template.data数组，并允许用户修改数据，然后执行PUT请求发送更新信息给服务器。

\*\*\* REQUEST \*\*\*

PUT /1sv697h2yij HTTP/1.1 #1

Host: http://orm-hyper-todo.herokuapp.com

Content-Type: application/vnd.collection+json

...

"template": {

"data": [

{"name": "id", "value": "1sv697h2yij"},

{"name": "title", "value": "Marina Del Ray"},

{"name": "completed", "value": "true"}

]

}

注意（在插图编号#1）从item的href属性中得到的URL与HTTP PUT方法一起使用。这就是Cj客户端如何使用template来更新已存在的item。

因此，一个template，两种使用它的方式。这就是Cj如何描述写操作的。

**Error**

Collection+JSON的设计还包括error元素。它用于传递从服务器到客户端的特定领域的错误信息。例如：如果一个资源无法找到或者试图更新现有记录失败，服务器可以返回error元素而不只是HTTP 404或400。（？？？？）它可以返回一段问题的文本描述，甚至包括如何解决问题的意见。

例如，如果有人试图分配给一个不存在的用户一个任务处理系统的任务，服务器会这样回应：

{

"collection": {

"version": "1.0",

"href": "//rwcbook12.herokuapp.com/error/",

"title": "TPS - Task Processing System",

"error": {

"code": 400,

"title": "Error",

"message": "Assigned user not found (filbert). Please try again.",

"url": "http://rwcbook12.herokuapp.com/task/assign/1l9fz7bhaho"

}

}

}

正如前面所提到的，有一些额外的Cj文档元素和属性我不会在这里涉及到。你可以在本章末尾的参考资料部分列举的网址查看完整的规范。

**简单的总结**

到目前为止，我们可以看到，三个特色的超媒体类型（HAL、Siren和Cj）有几个共同点。像HAL和Siren一样，Cj有一个传递链接或地址的元素（links）。与Siren一样，Cj的queries和template元素在响应中传递动作元数据。三者都有一种方式来传递特定领域的对象（HAL的根级别属性，Siren的properties对象以及Cj的items集合）。Cj的items集合是独特的，因为他包括了领域对象中每个属性的元数据（例如：prompt和render）。这提升了Cj处理开放应用架构（OAA）要求的OBJECT方面的能力。当我们创建Cj客户端应用的时候，我们会再讨论这个。

现在，我们有足够的背景来回顾Cj表述器，然后演练我们Cj客户端单页面应用的代码。

**Collection+JSON 表述器**

和其它格式一样，编码一个Cj表述器的过程是一件转换我们内部的资源表述形式（以WeSTL对象的形式）为一个有效的Collection+JSON文档的事情。就像其它表述器一样，只用大约300行NodeJS代码来创建一个用于生成有效Cj响应的功能完整的模块。

Cj表述器的源代码可以在相应的github仓库https://github.com/RWCBook/cj-client中找到。本章描述的应用的一个运行的版本可以在这里找到：<http://rwcbook12.herokuapp.com/。>

下面是一个Cj表述器重点代码的快速演练。

**顶层处理循环**

我的Cj表述器的顶层处理循坏是非常简单的。它首先初始化一个空的collection对象（用来表示一个JSON格式的Cj文档），然后用每个主要的Cj元素填充这个对象：

• Links

• Items

• Queries

• Template

• Error (如果需要的话)

这是函数的样子：

function cj(object, root) {

var rtn;

rtn = {};

rtn.collection = {}; #1

rtn.collection.version = "1.0";

for(var o in object) {

rtn.collection.href = root+"/"+o+"/"; #2

rtn.collection.title = getTitle(object[o]); #3

rtn.collection.links = getLinks(object[o].actions);

rtn.collection.items = getItems(object[o],root);

rtn.collection.queries = getQueries(object[o].actions);

rtn.collection.template = getTemplate(object[o].actions);

// handle any error

if(object.error) { #4

rtn.collection.error = getError(object.error);

}

}

// send results to caller

return JSON.stringify(rtn, null, 2); #5

}

上面的代码有几个有趣的项。初始化一个collection文档（#1）之后，建立文档级别的href（#2），代码演练被传入的WeSTL对象树（#3）并构造Cj的title、links、items、queries和template元素。接着，如果当前对象的error属性存在（？？？？），Cj error元素将被构造（#4）。最后，完成后的Cj文档被返回（#5）给调用方。

现在，让我们看看每个用于建立Cj文档的主要过程（？？？？）。

**Links**

Cj中的links元素保存文档的所有顶级链接。Cj表述器代码审查传入的用于任何action元素的WesTL对象，action元素具备合格条件，如果需要，在添加链接到集合之前确定所有的URI模板（？？？？）。

这是代码：

// get top-level links

function getLinks(obj, root, tvars) {

var link, rtn, i, x, tpl, url;

rtn = [];

if(Array.isArray(obj)!==false) {

for(i=0,x=obj.length;i<x;i++) { #1

link = obj[i];

if(link.type==="safe" &&

link.target.indexOf("app")!==-1 &&

link.target.indexOf("cj")!==-1) #2

{

if(!link.inputs) {

tpl = urit.parse(link.href);

url = tpl.expand(tvars); #3

rtn.push({ #4

href: url,

rel: link.rel.join(" ")||"",

prompt: link.prompt||""

});

}

}

}

}

return rtn; #5

}

这是getLinks函数中最有意思的部分：

1. 如果我们有操作对象，遍历它们
2. 首先，检查看看当前链接是否符合在Cj文档中最为顶级链接的标准
3. 如果成立，用传入的tvars集合（模板变量）来确定所有的URI模板
4. 然后把结果添加到链接集合
5. 最后，返回填充过的集合给调用方

**Items**

下一个有趣的函数是处理items的。这是最Cj表述器中最复杂的程序。那是因为Cj提供相当多的有关每个被传到客户端应用的领域模型的数据和元数据。

这是代码：

// get list of items

function getItems(obj, root) {

var coll, temp, item, data, links, rtn, i, x, j, y;

rtn = [];

coll = obj.data;

if(coll && Array.isArray(coll)!==false) {

for(i=0,x=coll.length;i<x;i++) {

temp = coll[i];

// create item & link

item = {}; #1

link = getItemLink(obj.actions);

if(link) {

item.rel = (Array.isArray(link.rel)?link.rel.join(" "):link.rel);

item.href = link.href;

if(link.readOnly===true) {

item.readOnly="true";

}

}

// add item properties

tvars = {}

data = [];

for(var d in temp) { #2

data.push(

{

name : d,

value : temp[d],

prompt : (g.profile[d].prompt||d),

render:(g.profile[d].display.toString()||"true")

}

);

tvars[d] = temp[d];

}

item.data = data;

// resolve URL template #3

tpl = urit.parse(link.href);

url = tpl.expand(tvars);

item.href = url;

// add any item-level links #4

links = getItemLinks(obj.actions, tvars);

if(Array.isArray(links) && links.length!==0) {

item.links = links;

}

rtn.push(item); #5

}

}

return rtn; #6

}

getItems程序是在Cj表述器中最大的。它实际上处理三个关键的事情，item的URL、item的数据属性和与item相关的所有链接。下面是详细内容：

1. 对于列表中的每个数据项，首先设置href属性
2. 然后遍历领域模型的全部属性，构造Cj data元素
3. 在收集完数据值之后，使用那个集合来确定所有在item的href中的URL模板
4. 接下来，去收集（和确定）所有用于这个item的Cj link对象
5. 一旦全部完成，把结果添加到内部的item集合
6. 最后，返回已完成的集合给调用程序

生成的item集合看起来像这样：

"items": [

{

"rel": "item",

"href": "http://rwcbook12.herokuapp.com/task/1l9fz7bhaho",

"data": [

{"name":"id","value":"1l9fz7bhaho","prompt":"ID","render":"true"},

{"name":"title","value":"extensions","prompt":"Title","render":"true"},

{"name":"tags","value":"forms testing","prompt":"Tags","render":"true"},

{"name":"completeFlag","value":"true","prompt":"Complete Flag",

"render":"true"},

{"name":"assignedUser","value":"carol","prompt":"Asigned User",

"render":"true"},

{"name":"dateCreated","value":"2016-02-01T01:08:15.205Z",

"prompt":"Created","render":"false"}

],

"links": [

{

"prompt": "Assign User",

"rel": "assignUser edit-form",

"href": "http://rwcbook12.herokuapp.com/task/assign/1l9fz7bhaho"

},

{

"prompt": "Mark Active",

"rel": "markActive edit-form",

"href": "http://rwcbook12.herokuapp.com/task/active/1l9fz7bhaho"

}

]

}

... more items here ...

]

**Queries**

getQueries是生成安全的参数化查询的程序——本质上是HTML GET表单。这意味着，连同URL，有一个或多个参数描述的列表。这些是一个表单的HTML input元素。代码生成Cj queries很明确，看起来像这样

// get query templates

function getQueries(obj) {

var data, d, query, q, rtn, i, x, j, y;

rtn = [];

if(Array.isArray(obj)!==false) {

for(i=0,x=obj.length;i<x;i++) { #1

query = obj[i];

if(query.type==="safe" && #2

query.target.indexOf("list")!==-1 &&

query.target.indexOf("cj") !==-1)

{

q = {}; #3

q.rel = query.rel.join(" ");

q.href = query.href||"#";

q.prompt = query.prompt||"";

data = [];

for(j=0,y=query.inputs.length;j<y;j++) { #4

d = query.inputs[j];

data.push(

{

name:d.name||"input"+j,

value:d.value||"",

prompt:d.prompt||d.name,

required:d.required||false,

readOnly:d.readOnly||false,

patttern:d.pattern||""

}

);

}

q.data = data;

rtn.push(q); #5

}

}

}

return rtn; #6

}

解释非常简单：

1. 遍历WeSTL文档中所有的转换
2. 找到对Cj queries集合有效的转换
3. 创建一个空的查询对象并设置href和rel属性
4. 遍历WeSTL input元素来创建查询的Cj data元素
5. 天假已完成的查询到集合
6. 最后，返回集合给调用程序

再说一遍，没有用于每个查询的HTTP方法，因为规范说所有的Cj queries应该用HTTP GET执行。

以上涉及到Cj中“读”的形式。接下来的工作是处理“写”的形式——Cj template。

**Template**

Cj中，“写”的形式在template元素中表示。在我们Cj表述器中的getTemplate程序处理生成template元素，代码看起来像这样：

// get the add template

function getTemplate(obj) {

var data, temp, field, rtn, tpl, url, d, i, x, j, y;

rtn = {};

data = [];

if(Array.isArray(obj)!==false) {

for(i=0,x=obj.length;i<x;i++) {

if(obj[i].target.indexOf("cj-template")!==-1) { #1

temp = obj[i];

// emit data elements

data = [];

for(j=0,y=temp.inputs.length;j<y;j++) { #2

d = temp.inputs[j];

field = { #3

name:d.name||"input"+j,

value:(d.value||"",

prompt:d.prompt||d.name,

required:d.required||false,

readOnly:d.readOnly||false,

patttern:d.pattern||""

};

data.push(field); #4

}

}

}

}

rtn.data = data;

return rtn; #5

}

getTemplate程序没有多少东西，所以重点部分有点乏味：

1. 遍历WeSTL转换并找到一个Cj template有效的
2. 然后遍历转换的input集合
3. 使用input集合中元素的信息创建Cj data元素
4. 添加创建的data元素到用于这个template的data元素集合
5. 最后，添加完成的data集合到template对象之后，返回结果给调用者

提醒一下，Cj templates没有href属性或HTTP方法。要使用的URL和方法由客户端根据客户端是否尝试创建或更新操作在运行时决定。

只剩下一个小对象要回顾：Cj error元素。

**Error**

不像HAL和Siren，Cj有专门用于响应的error元素。这使客户端很容易识别和提供任何在服务器响应中特定领域的错误信息。Cj error对象只有四个定义的字段：title、message、code和url。getError函数很小，看起来像这样：

// get any error info

function getError(obj) {

var rtn = {};

rtn.title = "Error";

rtn.message = (obj.message||"");

rtn.code = (obj.code||"");

rtn.url = (obj.url||"");

return rtn;

}

这里真的没有什么可说的，因为程序就是这么简单。值得指出的是Cj响应可以包含错误信息和links、items、queries及template元素里的内容。这使得返回完全填充的Cj文档和一些帮助用户解决任何问题的错误数据成为可能。

error对象使用url属性传递与这个错误相关的URL。在所有其他的Cj元素中，URL以href属性传递。这是设计中的一个矛盾，总有一天我需要修复它。但是，按照支持只向后兼容的改变，这意味着error元素将来很可能会包括url和href属性。

随着Cj表述器演练的完成，是时候回顾下Cj客户端单页面应用了。

**Collection+JSON单页面客户端**

好了，现在我们可以回顾Collection+JSON单页面应用了。这个Cj客户端支持Cj所有主要的特性，包括links、items、queries和template。它也支持其它Cj元素，包括title、content和error元素。

Cj表述器的源代码可以在相应的github仓库：[https://github.com/RWCBook/cj-client中找到。本章中描述的应用程序的一个正在运行的版本可以在http://rwcbook12.herokuapp.com/files/cj.client.html找到。](https://github.com/RWCBook/cj-client中找到。本章中描述的应用程序的一个正在运行的版本可以在http:/rwcbook12.herokuapp.com/files/cj.client.html找到。)

像我们对JSON、HAL和Siren单页面应用所做的一样，我们将先以HTML容器的回顾开始，然后转到回顾顶级解析程序和解析关键Cj文档部分来创建普通Cj客户端的剩余部分的主要函数（？？？？）。

**HTML容器**

本书中，所有的单页面应用程序以HTML容器开始，这次也不例外。下面是用于保持由服务器发送的Cj文档的静态HTML。

<!DOCTYPE html>

<html>

<head>

<title>Cj</title>

<link href="./semantic.min.css" rel="stylesheet" />

</head>

<body>

<div id="links"></div> #1

<div style="margin: 5em 1em">

<h1 id="title" class="ui page header"></h1> #2

<div id="content" style="margin-bottom: 1em"></div> #3

<div class="ui mobile reversed two column stackable grid">

<div class="column">

<div id="items" class="ui segments"></div> #4

</div>

<div class="column">

<div id="edit" class="ui green segment"></div>

<div id="template" class="ui green segment"></div> #5

<div id="error"></div> #6

<div id="queries-wrapper">

<h1 class="ui dividing header">

Queries

</h1>

<div id="queries"></div> #7

</div>

</div>

</div>

<div>

<pre id="dump"></pre>

</div>

</div>

</body>

<script src="dom-help.js">//na </script>

<script src="cj-client.js">//na </script> #8

<script>

window.onload = function() {

var pg = cj();

pg.init("/", "TPS - Task Processing System");

}

</script>

</html>

上面所示的大量HTML是为了支撑CSS库的布局。但是你仍然可以在页面中找到以<div>标签表示的所有主要的Cj文档元素。它们是：

1. links集合
2. Title元素
3. Content元素
4. Items元素
5. Template元素
6. Error元素
7. Queries元素

Cj解析脚本在标注#8处被加载，在所有内容加载完之后，初始化请求在标注#9处开始，那一行调用了Cj库的顶级解析循环。

**顶级解析循环**

在Cj客户端中，每次用户在用户界面中作出选择，顶级解析循环就被调用，这遵循我在第XX章中提到的请求-解析-等待（RPW）模式。这证明Cj的解析循环比JSON、HAL、Siren客户端的都要简单一点。

// init library and start

function init(url) {

if(!url || url==='') {

alert('\*\*\* ERROR:\n\nMUST pass starting URL to the Cj library');

}

else {

g.url = url;

req(g.url,"get"); #1

}

}

// primary loop

function parseCj() { #2

dump();

title();

content();

links();

items();

queries();

template();

error();

cjClearEdit();

}

到目前为止，上面代码看起来很相似。在作出初始请求（标注#1）之后，parseCj程序被调用并且它遍历Collection+JSON文档的所有主要元素。在这个代码片段中唯一有趣的元素是内部的程序。首先，dump()方法在循环开始的地方调用——这只是为了屏幕上调试的帮助——然后，在程序的末端，cjClearEdit()被调用来清理用于显示用户界当前编辑形式的HTML div。

我将在这里跳过对title和content程序的讨论——你可以自己在源代码中查看他们。下面是用来处理Cj响应的其它主要程序的演练。

**Links**

处理解析和渲染Cj links的程序很简单。但是，它有点弯曲。代码检查相关链接特定领域的元数据。例如，一些链接不在屏幕上渲染出来（例如：HTML样式表，互联网编号分配机构（IANA）配置文件标识符等等）。一些其他的链接实际上应该被渲染成嵌入的图片而不是导航链接。Cj的设计允许服务器在消息中表明链接元数据这一级别——HAL和Siren客户端在它们的设计中不支持。

这是links()函数的代码

// handle link collection

function links() {

var elm, coll, menu, item, a, img, head, lnk;

elm = d.find("links");

d.clear(elm);

if(g.cj.collection.links) { #1

coll = g.cj.collection.links;

menu = d.node("div");

menu.className = "ui blue fixed top menu";

menu.onclick = httpGet;

for(var link of coll) { #2

// stuff render=none Cj link elements in HTML.HEAD #3

if(isHiddenLink(link)===true) {

head = d.tags("head")[0];

lnk = d.link({rel:link.rel,href:link.href,title:link.prompt});

d.push(lnk,head);

continue;

}

// render embedded images, if asked #4

if(isImage(link)===true) {

item = d.node("div");

item.className = "item";

img = d.image({href:link.href,className:link.rel});

d.push(img, item);

d.push(item, menu);

}

else {

a = d.anchor({rel:link.rel,href:link.href,text:link.prompt, #5

className: "item"});

d.push(a, menu);

}

}

d.push(menu, elm); #6

}

}

尽管这里有很多行代码，但它们都很直接。重点是：

1. 在确定要处理的Cj links之后，设置一些布局来容纳它们
2. 现在开始遍历links集合
3. 如果link元素应该现在被渲染，就把它放进页面的HTML <head>部分
4. 如果link元素应该被渲染成一张图片，那就正确地处理它
5. 否则，把它当作简单的<a>标签并加入到布局中
6. 最后，把结果放进可视屏幕的对象

这是一个在运行时渲染Cj links的例子：

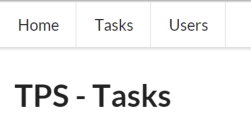


图 5-2. 在运行时渲染Cj links

事实证明，没有显示的link元素（标注#3）或者link是一个图片（标注#4）比link元素只是一个导航元素（标注#5）的情况需要更多代码。当我们解析items集合的时候，我们会看到更多的这种代码。

**Items**

Items()函数是Cj库中最复杂的程序。125行，也是最长的。那是因为，当回顾Cj表述器中items处理的时候，正如我们看到的，items元素是整个Cj文档设计中最复杂的。我不会覆盖这整个程序，但会展示这个程序中的关键步骤。你可以在本章相关的源码仓库中找到完整的代码。

我将把items()程序的代码评审分散成三个部分：

* 渲染Cj item编辑链接
* 渲染Cj item links
* 渲染Cj item data属性

首先，代码处理每个item的读取-更新-删除链接——最后三个增删改查（CRUD）模式的元素。每个Cj item有一个href属性和一个可选的readOnly属性。使用这些信息作为参考，Cj客户端负责用于读取、更新、删除链接的渲染支持。你可以在下面的代码中看到。在标注#1，读取链接被创建。更新链接在#2处创建，删除链接在#3处创建。注意客户端readOnly状态和template是否能在Cj文档中被找到的检查。这些值被用于决定item的哪些链接（更新和删除）被渲染。

// item link

a1 = d.anchor( #1

{

href:item.href,

rel:item.rel,

className:"item link ui basic blue button",

text:item.rel

}

);

a1.onclick = httpGet;

d.push(a1,buttons);

// edit link

if(isReadOnly(item)===false && hasTemplate(g.cj.collection)===true) {

a2 = d.anchor( #2

{

href:item.href,

rel:"edit",

className:"item action ui positive button",

text:"Edit"

}

);

a2.onclick = cjEdit;

d.push(a2, buttons);

}

// delete link

if(isReadOnly(item)===false) {

a3 = d.anchor( #3

{

href:item.href,

className:"item action ui negative button",

rel:"delete",

text:"Delete"

}

);

a3.onclick = httpDelete;

d.push(a3,buttons);

}

在items()程序中下一个重要的片段是处理所有item级别的links。因此，在代码中你可以看到（在标注#1处）是否有这个item的links，每个链接被检查看是否应该被渲染成图片（标注#2），如果不是，它就可以被渲染成导航链接（标注#3）。最后，链接被处理完后，结果被添加到item显示中（标注#4）。

if(item.links) { #1

for(var link of item.links) {

// render as images, if asked

if(isImage(link)===true) { #2

p = d.node("p");

p.className = "ui basic button";

img = d.image(

{

className:"image "+link.rel,

rel:link.rel,

href:link.href

}

);

d.push(img, p);

d.push(p,secondary\_buttons);

}

else {

a = d.anchor( #3

{

className:"ui basic blue button",

href:link.href,

rel:link.rel,

text:link.prompt

}

);

a.onclick = httpGet;

d.push(a,secondary\_buttons);

}

}

d.push(secondary\_buttons,segment); #4

}

在item()程序中最后一个审查的片段是处理所有真实的item data属性。在这个客户端中，它们被一个个作为用户界面表格显示的部分来渲染。代码（看下面）不是很复杂。

for(var data of item.data) {

if(data.display==="true") {

tr = d.data\_row(

{

className:"item "+data.name,

text:data.prompt+"&nbsp;",

value:data.value+"&nbsp;"

}

);

d.push(tr,table);

}

}

这是items()程序的全部。接下来是处理Cj文档queries元素的程序。这是被生成用于Cj items的用户界面的例子。

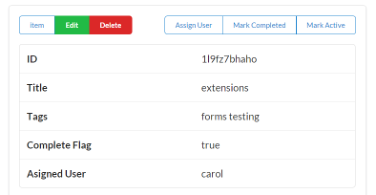


图 5-3. 生成的Cj Items

**Queries**

Queries()程序处理在Cj queries集合中的所有元素并将它们转换成HTML “GET” 表单。代码不是很复杂，但是有点冗长。生成HTML表单用了相当多的行数！我们Cj客户端用来生成Cj queries的用户界面的代码如下。

// handle query collection

function queries() {

var elm, coll;

var segment;

var form, fs, header, p, lbl, inp;

elm = d.find("queries");

d.clear(elm);

if(g.cj.collection.queries) { #1

coll = g.cj.collection.queries;

for(var query of coll) { #2

segment = d.node("div");

segment.className = "ui segment";

form = d.node("form"); #3

form.action = query.href;

form.className = query.rel;

form.method = "get";

form.onsubmit = httpQuery;

fs = d.node("div");

fs.className = "ui form";

header = d.node("div");

header.innerHTML = query.prompt + "&nbsp;";

header.className = "ui dividing header";

d.push(header,fs);

for(var data of query.data) { #4

p = d.input({prompt:data.prompt,name:data.name,value:data.value});

d.push(p,fs);

}

p = d.node("p"); #5

inp = d.node("input");

inp.type = "submit";

inp.className = "ui mini submit button";

d.push(inp,p);

d.push(p,fs);

d.push(fs,form);

d.push(form,segment);

d.push(segment,elm); #6

}

}

}

Queries程序有几个有趣的点：

1. 首先，看看在响应中是否有任何queries要处理
2. 如果有，就遍历它们来创建查询表单
3. 创建HTML <form>元素并填充适当的细节
4. 访问每个data元素来创建需要的HTML <input>
5. 然后，添加submit按钮到表单
6. 最后，将生成的标记添加到页面上用于渲染的用户界面

这就是Cj客户端如何处理生成所有“安全的”查询表单（例如：HTTP “GET”）。有几个处理HTML布局的部分在这里被忽略了，但是你可以看到queries()程序的重要方面。这是在我们Cj客户端应用中生成查询表单的一个例子。

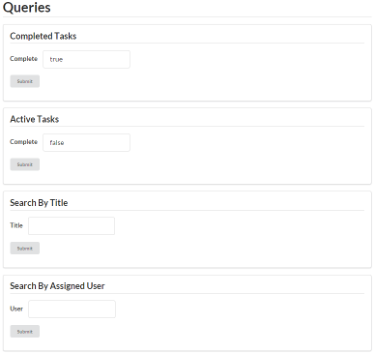


图 5-4. 生成的Cj查询表单

**Template**

正如Cj queries描述安全的操作（例如：HTTP GET），Cj template描述不安全的操作（例如：HTTP POST和PUT）。代码和生成Cj queries的看上去非常相似。

// handle template object

function template() {

var elm, coll;

var form, fs, header, p, lbl, inp;

elm = d.find("template");

d.clear(elm);

if(hasTemplate(g.cj.collection)===true) { #1

coll = g.cj.collection.template.data;

form = d.node("form"); #2

form.action = g.cj.collection.href;

form.method = "post";

form.className = "add";

form.onsubmit = httpPost;

fs = d.node("div");

fs.className = "ui form";

header = d.node("div");

header.className = "ui dividing header";

header.innerHTML = g.cj.collection.template.prompt||"Add";

d.push(header,fs);

for(var data of coll) { #3

p = d.input(

{

prompt:data.prompt+"&nbsp;",

name:data.name,

value:data.value,

required:data.required,

readOnly:data.readOnly,

pattern:data.pattern

}

);

d.push(p,fs);

}

p = d.node("p"); #4

inp = d.node("input");

inp.className = "ui positive mini submit button";

inp.type = "submit";

d.push(inp,p);

d.push(p,fs);

d.push(fs,form);

d.push(form, elm); #5

}

}

这是template程序的要点：

1. 确认在加载的Cj文档中有template元素
2. 如果有，就开始构建和填充HTML <form>
3. 使用template的data属性，创建一个或多个HTML <input>元素
4. 所有输入框创建完后，添加一个HTML提交按钮
5. 最后，添加完成的HTML表单到用户界面

你也会注意到在上面的代码中，HTML <form>元素被设置为使用POST方法。这解决了Cj template CREATE的用例。对于UPDATE的用例，在Cj客户端中有一个叫作cjEdit()的程序（？？？？）。当用户按下为每个item生成的“Edit”按钮时，它被调用。我不会在这里回顾cjEdit()的代码（你可以自己去查看源代码），但是会提一下，除了有关HTTP PUT用例的一些改变，它看起来几乎是一样的。

这是一个被渲染用作CREATE用例的Cj template的例子。

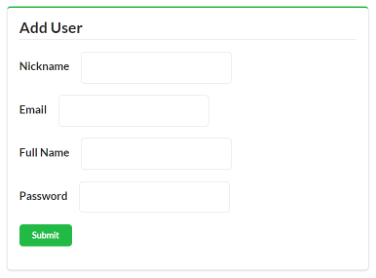


图 5-5. 生成Cj CREATE用户界面

Cj客户端中唯一剩下要回顾的是处理响应中所有error元素的代码。

**Error**

Cj是本书中唯一有特色的超媒体设计，拥有发送特定领域错误信息的内置支持。Cj error元素非常简单。它只有四个属性：title、message、code和url。所以用于呈现错误的客户端程序也很简单。

下面的代码显示Cj客户端应用只是直接回显Cj响应中error元素的属性到屏幕上。

// handle error object

function error() {

var elm, obj;

elm = d.find("error");

d.clear(elm);

if(g.cj.collection.error) {

obj = g.cj.collection.error;

p = d.para({className:"title",text:obj.title});

d.push(p,elm);

p = d.para({className:"message",text:obj.message});

d.push(p,elm);

p = d.para({className:"code",text:obj.code});

d.push(p,elm);

p = d.para({className:"url",text:obj.url});

d.push(p,elm);

}

}

**快速小结**

Cj客户端与我们在本书更早前回顾的HAL和Siren客户端有一些不同。最重要的是在Cj中处理领域模型的方式。Collection+JSON文档只支持返回扁平的items列表，而不是仅仅重复一组名-值对，甚至嵌套的JSON对象图。每个item表示的不只是一个领域模型的属性。它还包括相关领域模型的元数据（prompt和render信息）和一个或多个与领域模型有关联的link元素的集合。

安全和不安全操作的表达方式在Cj中也是独特的。Cj设计上支持两种不同的操作元素：queries和template元素，而不是由源代码决定（像HAL）或者依靠一个所有操作通用的模型（像Siren）。Cj queries用于安全操作（例如：HTML GET），template用于不安全操作（例如：HTTP POST和PUT）。

Cj文档中另一个的主要元素是links集合，它与HAL和Siren表示连接的方式也非常相似。

既然我们有了一个功能完整的Cj通用客户端，让我们介绍一些对后端TPS API的修改并看看它是如何处理向后兼容的变化。