Web Services and Web Data

XJCO3011

Session 5. HyperMedia for RESTful API's

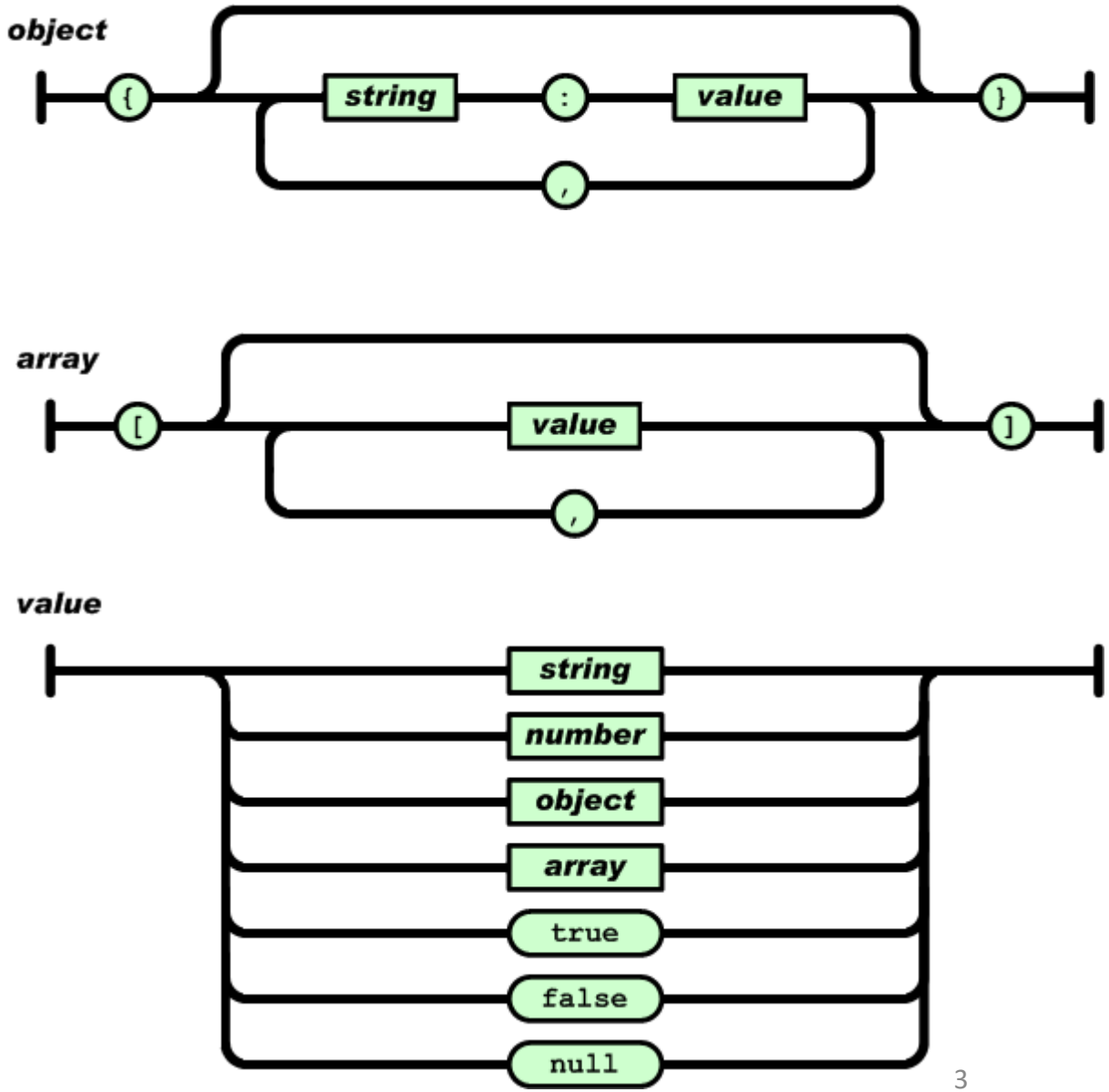
Why HyperMedia

- Hypermedia is a method of creating and representing resources and their relationships on the web. It not only focuses on how to obtain resources, but also emphasizes how to navigate and interact with resources. Through hypermedia, clients can automatically explore and interact with resources by following links and relationships between them.
- RESTful is an architectural style for designing web services that emphasizes using the semantics of the HTTP protocol to describe resources and their relationships. RESTful services typically use HTTP verbs (such as GET, POST, PUT, DELETE, etc.) to manipulate resources and use HTTP status codes to represent the results of operations. In RESTful, hypermedia is an important design principle used to represent relationships between resources and to help clients discover and interact with the service.
- hypermedia can be seen as an extension of RESTful that emphasizes relationships and navigation between resources, making web services more flexible and discoverable.
- In practice, hypermedia can be represented using different formats, such as HTML, JSON, XML, etc.)

JSON

JSON is built on two structures:

- A collection of name-value pairs. In various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array.
- An ordered list of values. In most languages, this is realized as an array, vector, list, or sequence.
- These are universal data structures. Virtually all modern programming languages support them in one form or another. It makes sense that a data format that is interchangeable with programming languages also be based on these structures.



A JSON Example

A person record

Object structures and array structures can be nested inside each other, allowing for more complex data structures

```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 27,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    },
    {
      "type": "mobile",
      "number": "123 456-7890"
    }
  ],
  "children": [],
  "spouse": null
}
```

Collection+JSON

- Collection+JSON is one of several standards designed not to represent one specific problem domain (the Maze+XML does) but to fit a pattern—the collection—that shows up over and over again, in all sorts of domains.

- A collection such as microblog posts, goods in a shopping cart or a collection of readings from a weather sensor would look pretty much the same and have the same protocol semantics.

- A collection is a special kind of resource. Recall that a resource is anything important enough to have been given its own URL. Recall from Chapter 3 that a resource is anything important enough to have been given its own URL. A resource can be a piece of data, a physical object, or an abstract concept—anything at all. All that matters is that it has a URL and the representation—the document the client receives when it sends a GET request to the URL.

- A collection resource is a little more specific than that. It exists mainly to group other resources together. Its representation focuses on links to other resources, though it may also include snippets from the representations of those other resources. (Or even the full representations!)

Collection+JSON Standard

- The Collection+JSON standard defines a representation format based on JSON.
- It also defines the protocol semantics for the HTTP resources that serve that format in response to GET requests.
- It's basically an object with five special properties, predefined slots for application specific data:
    1. href: A permanent link to the collection itself.
    2. items: Links to the members of the collection, and partial representations of them.
    3. links: Links to other resources related to the collection.
    4. queries: Hypermedia controls for searching the collection.
    5. template: A hypermedia control for adding a new item to the collection.

Representing the Items

Let's zoom in on items, the most important field in a Collection+JSON representation:

The href attribute :A permanent link to the item as a standalone resource.

data: Any other information that's an important part of the item's representation.

links
Hypermedia links to other resources related to the item.

```
"items" : [
 { "href" : "/api/airlines/ba",
   "data" : [
        { "name" : "company_name", "value" : "British Airways" },
        { "name" : "date_started", "value" : "1974-03-31T05:33:58.930Z" }
   ],
   "links" : []
 },

 { "href" : "/api/airlines/af",
   "data" : [
      { "name" : " company_name ", "value" : "Air France" },
      { "name" : " date_started ", "value" : "1933-10-07T12:55:59.685Z" }
   ],
  "links" : []
 }
]
```

An item's permanent link

A member's href attribute is a link to the resource outside the context of its collection. If you GET the URL mentioned in the href attribute, the server will send you a Collection+JSON representation of a single item.


```
{ "collection":
  {
    "version" : "1.0",
    "href" : "http://airlines.pythonanywhere.com/api/",
    "items" : [
        { "href" : "/api/airlines/ba",
          "data": [
                { "name" : "company_name", "value" : "British Airways" },
                { "name" : "date_started", "value" : "1974-03-31T05:33:58.930Z" }
            ],
         "links" : []
       }
    ]
  }
}
```

An item's links

An item's representation may also contain a list called links. This contains any number of other hypermedia links to related resources. Here's a link you might see in the representation of a "book" resource:

```
"links" :
[
    {
      "name" : "owner",
      "rel" : "owner",
      "prompt" : "Owner of the company",
       "href" : "/owners/441",
        "render" : "link"
    }
]
```

The rel attribute is a slot for a link relation, just like the rel attribute in Maze+XML

The prompt attribute is a place to put a human-readable description

Setting render to "link" tells a Collection+JSON client to present the link as an outbound link like an HTML <a> tag. Setting render to "image" tells the client to present the link as an embedded image, like HTML's <img> tag

The Write Template

Suppose you want to add a new item to a collection. What HTTP request should you make? To answer this question, you need to look at the collection's write template. Here's the write template for our airline directory API:

```
"template": {
  "data": [
    {"prompt" : "Name of company", "name" : "company_name", "value" :     ""}
  ]
}
```

Interpreting this template according to the Collection+JSON standard tells you it's OK to fill in the blanks and submit a document that looks like this:

```
{ "template" :
  {
    "data" : [
       {"prompt" : " Name of company ", " name " : " company_name", "value" : "Air France"}
    ]
  }
}
```

Where does that request go? The Collection+JSON standard says you add an item to a collection by sending a POST request to the collection (i.e., to its href attribute):
"href" : "http://airlines.pythonanywhere.com/api/"

How a (Generic) Collection Works

- There's not much more to Collection+JSON than what we have just shown. It was designed without any real application semantics, so that it can be used in many different applications.
- Because it's so general, it does a good job illustrating the common features of the collection pattern.
- Before moving on to other collection types, let us go up a level and lay out the pattern itself by describing the behaviour of a generic "collection" resource under HTTP.
- Different collection types - such as Collection+JSON , AtomPub and Odata - take different approaches to collections, but they all have more or less the same protocol semantics.

GET

- Like most resources, a collection responds to GET by serving a representation. Although collection standards (Collection+Json, AtomPub, and OData) don't say much about what an item should look like within a collection, they go into great detail about what a collection's representation should look like.

- The media type of the representation tells you what you can do with the resource. If you get an application/vnd.collection+json representation, you know that the rules of the Collection+JSON standard apply. If the representation is application/atom+xml, you know that AtomPub rules apply.

- If the representation is application/json, you're out of luck, because the JSON standard doesn't say anything about collection resources. You're using an API that went off on its own and defined a fiat standard. You'll need to look up the details for the specific API you're using.

POST-to-Append

- The defining characteristic of a collection is its behaviour under HTTP POST.
- Unless a collection is read-only (like a collection of search results), a client can create a new item inside it by sending it a POST request.
- When you POST a representation to a collection, the server creates a new resource based on your representation.
- That resource becomes the latest member of the collection.

PUT and PATCH

- None of the main collection standards define a collection's response to PUT or PATCH.
- Some applications implement these methods as a way of modifying several elements at once, or of removing individual elements from a collection.
- Collection+JSON, AtomPub, and OData all define an *item's* response to PUT: they say that PUT is how clients should change the state of an item. But these standards are just repeating what the HTTP standard says. They're not putting new restrictions on item resources. PUT is how clients change the state of *any* HTTP resource.

DELETE

- None of the three big standards define how a collection should respond to DELETE.
- Some applications implement DELETE by deleting the collection; others delete the collection and every resource listed as an item in the collection.
- The main collection standards all define an item's response to DELETE, but again, they're just restating what the HTTP standard says. The DELETE method is for deleting things.

Pagination

- A collection may contain millions of items, but the server is under no obligation to serve millions of links in a single document. The most common alternative is pagination.
- A server can choose to serve the first 10 items in the collection, and give the client a link to the rest:

```
<link rel="next" href="/collection/4iz6"/>
```

- The "next" link relation is registered with the IANA to mean “the next in the series.” and by following that link we get the second page of the collection.
- We can keep following rel="next" links until we reach the end of the collection.
- There are a number of generic link relations for navigating paginated lists. These include "next", "previous", "first", and "last".
- These link relations were originally defined for HTML, but now they’re registered with the IANA, so you can use them with any media type.

The Atom Publishing Protocol (AtomPub)

- The Atom file format was developed for syndicating news articles and blog posts. It's defined in RFC 4287, which was finalized in 2005.

- The Atom Publishing Protocol is a standardized workflow for editing and publishing news articles, using the Atom file format as the representation format. It's defined in RFC 5023, which was finalized in 2007.

- Those are pretty early dates in the world of REST APIs. In fact, AtomPub was the first standard to describe the collection pattern.

- AtomPub has the same concepts as Collection+JSON, but uses different terminology. Instead of a "collection" that contains "items," this is a "feed" that contains "entries."

An Atom representation of a microblog

```
<feed xmlns="http://www.w3.org/2005/Atom">
    <title>You Type It, We Post It</title>
    <link href="http://www.youtypeitwepostit.com/api" rel="self" />
    <id>http://www.youtypeitwepostit.com/api</id>
    <updated>2013-04-22T05:33:58.930Z</updated>
    <entry>
        <title>Test.</title>
        <link href="http://www.youtypeitwepostit.com/api/messages/2181852539069950 6" />
        <link rel="edit"  href="http://www.youtypeitwepostit.com/api/messages/2181852539069950 6" />
        <id>http://www.youtypeitwepostit.com/api/messages/2181852539069950 6</id>
        <updated>2013-04-22T05:33:58.930Z</updated>
        <author><name/></author>
    </entry>
    ….
```

Some Atom Features

- The previous document is served with the media type application/atom+xml
- We can POST a new Atom entry to the href of the collection. An entry's rel="edit" link is the URL you send a PUT to if you want to edit the entry, or send a DELETE to if you want to delete the entry.
- There's one big conceptual difference between Collection+JSON and AtomPub. Collection+ JSON defines no particular application semantics for "item." An "item" can look like anything. But since Atom was designed to syndicate news articles, every AtomPub entry looks a bit like a news article.
- Every entry in an AtomPub feed must have a unique ID (the URL of the post in the previous example), a title (the text of the post in the previous example), and the date and time it was published or last updated.
- The Atom file format defines little bits of application semantics for news stories: fields like "subtitle" and "author."

Atom Applications

- Despite this focus on news and blog posts, AtomPub is a fully general implementation of the collection pattern.
- Google, the biggest corporate adopter of AtomPub, uses Atom documents to represent videos, calendar events, cells in a spreadsheet, places on a map, and more.
- The secret is extensibility. You're allowed to extend Atom's vocabulary with whatever application semantics you care to define.
- Google defined a common Atom extension called GData for all of its Atom-based APIs, then defined additional extensions for videos, calendars, spreadsheets, and so on.

A few interesting facts about AtomPub

- Since news articles are often classified under one or more categories, the Atom file format defines a simple category system, and AtomPub defines a separate media type for a list of categories (application/atomcat+xml).
- AtomPub also defines a media type for a Service Document—effectively a collection of collections.
- Atom is strictly an XML-based file format. AtomPub installations do not serve JSON representations. This makes it difficult to consume an AtomPub API from an Ajax (Asynchronous JavaScript And XML) client.
- Although Atom is an XML file format, clients may POST binary files to an AtomPub API. An uploaded file is represented on the server as two distinct resources: a Media Resource whose representation is the binary data, and an Entry Resource whose representation is metadata in Atom format.
- This feature lets you use AtomPub to store a collection of photos or audio files, along with Atom documents containing descriptions and related links.

Why Doesn't Everyone Use AtomPub?

- Many years after finalizing the AtomPub standard, it never got much traction outside of Google, and even Google seems to be phasing it out.
- The problem stems from the decision to use XML documents for AtomPub representations
- It was obviously a correct decision in 2003, but over the next 10 years, as in-browser API clients became more and more popular, JSON gained an overwhelming popularity as a representation format.
- It's a lot easier to process JSON from in-browser JavaScript code than it is to process XML.
- Today, the vast majority of APIs either serve JSON representations exclusively, or offer a choice between XML and JSON representations.
- The AtomPub story shows that "nothing wrong with the standard" isn't good enough.
- People won't go through the trouble of learning a standard unless it's directly relevant to their needs.
- It's easier to reinvent the "collection" pattern using a fiat standard based on JSON, so that's what thousands of developers did—and continue to do.
- The question is whether we'll collectively allow ourselves to reinvent the same basic ideas over and over again.

That's it Folks

Further Reading

Chapter 6: RESTful Web APIs by Leonard Richardson and Mike Amundsen