CS 416
Web Programming

Structured Content Part 2: JSON

Dr. Williams
Central Connecticut State University

Structured content part 2

- As seen with XML, structured content allows the recipient to determine what is important
- Huge benefit in providing single interface to multiple consumers
- XML is one option, another is JSON
  - **J**ava**S**cript **O**bject **N**otation

What is JSON

- Text syntax for transmitting JavaScript objects
- Result is rather than parsing and navigating structure, JavaScript can load objects automatically

```
var obj = JSON.parse(text);
```

- Thus, perfect for most AJAX applications!
- So why did you learn XML then???

JSON pros/cons

- Pros
  - Simple – less markup syntax overhead compared to XML
  - JSON Schema – data type / structure validation
  - Support for generation, parsing in most common web back-end languages (Java, C#, PHP, Ruby)
  - Created for AJAX front-end, extremely simple on front-end
- Cons
  - Limited to JavaScript data types
  - Limited in ability to represent complex relationships and deep structures

XML pros/cons

- Pros
  - Generalized markup – ultimate in flexibility for representing relationships and any data type
  - XML schema – data type / structure validation, ability to define new data types
  - XPath/XQuery – query language for extracting complex data – equivalent to SQL for DBs
  - XSLT – language independent specification for output transformation
- Cons
  - Lots of syntax overhead
  - Front-end (JavaScript) processing not as simple

Which one to use in practice?

- XML
  - Complex relationships
  - Deep data structures
  - More specific datatype validation needed
  - Providing generic interface to multiple consumers with multiple purposes i.e. not just front end
- JSON
  - Simple relationships
  - Simple to medium data structures (arrays, object with 1-2 level of nested objects)
  - Interface used exclusively by front-end and/or back-ends that support JSON

JSON syntax – human readable as well

```
{
  "firstName":"John",
  "lastName":"Doe",
  "age":25,
  "address":{
    "street":"1 Main St.",
    "city":"New Britain",
    "state":"CT",
    "zipCode":"06050"},
  "phoneNumber":[
    {"type":"home","number":"860 867-5309"},
    {"type":"fax","number":"860 123-4567"}
  ]
}
```

JSON basics

- Objects encapsulated in curly brackets { }
- Name value pairs

```
"lastName":"Doe",
"age":25,
```

- Arrays in square brackets [ ]

```
"phoneNumber":[
    {"type":"home","number":"860 867-5309"},
    {"type":"fax","number":"860 123-4567"}
 ]
```

Comparison

- Create example of XML and JSON of:
  - Request return of all CDs with matching name with: artist, title, year, list of musicians with their first and last names
  - Sample of syntax as reminder

```
{
  "City":"New Britain",
  "age":25,
  "phoneNumber":[
    {"type":"home","number":"860 867-5309"},
    {"type":"fax","number":"860 123-4567"}
  ]
}
```

JSON solution

```
[
  {"artist":"Green Day",
   "title":”Dookie”,
   "year":1994,
   "musician":[{"first":"Billy Joe","last":"Armstrong"},
               {"first":"Tre","last":"Cool"}]
  },
  {"artist":"Pear Jam",
   "title":”Ten”,
   "year":1991,
   "musician":[{"first":"Eddie","last":"Vedder"},
               {"first":"Mike","last":"McCready"}]
  }
]
```

XML solution

```xml
<CDs>
    <CD>
        <artist>Green Day</artist>
        <musicians>
            <musician>
                <first>Billy Joe</first>
                <last>Armstrong</last>
            </musician>
            <musician>
                <first>Tre</first>
                <last>Cool</last>
            </musician>
        </musicians>
        <title>Dookie</title>
        <year>1994</year>
    </CD>
    <CD>
        <artist>Pearl Jam</artist>
        <musicians>
            <musician>
                <first>Eddie</first>
                <last>Vedder</last>
            </musician>
            <musician>
                <first>Mike</first>
                <last>McCready</last>
            </musician>
        </musicians>
        <title>Ten</title>
        <year>1991</year>
    </CD>
</CDs>
```

JSON browser side

Incoming JSON response:

```
{"name": "brett", "country": "Australia"}
```

AJAX processing of response:

```
var jsonObj = JSON.parse(http_request.responseText);

// jsonObj variable now contains the data structure
document.getElementById("Name").innerHTML = jsonObj.name;
document.getElementById("Country").innerHTML = jsonObj.country;
```

JSON server side

- javax.json package contains API for generation/parsing
- Uses builder pattern:

```
JsonBuilderFactory factory =
Json.createBuilderFactory(null);
JsonArray jsonArray = factory.createArrayBuilder()
                .add(factory.createObjectBuilder()
                        .add("name", "Brett")
                        .add("country", "Australia"))
                .add(factory.createObjectBuilder()
                        .add("name", "Sam")
                        .add("country", "France"))
                .build();
System.out.println(jsonArray);

[{"name": "Brett", "country": "Australia"}
 {"name": "Sam", "country": "France"}]
```

XML server side generation

JAXB (Java Architecture for XML Binding) makes this easy

- Add `@XmlRootElement` to any class
  - All public attributes or public getters have automatic XML generation
  - Must have default constructor
- One annoyance, need for wrapper classes for having parent element for multiples, see Musicians class (inside of Musician file)

JSON vs XML server side generation

Problem from before…

- Request return of all CDs with matching name with: artist, title, year, list of musicians with their first and last names
  - See AjaxJsonCDDemoServlet
  - See AjaxXmlCDDemoServlet (and CD and Musician)

- AJAX use of output:
  - AjaxJsonCDDemo.html
  - AjaxXmlCDDemo.html

AJAX FindByNames DB lookup revisited

- JSON
  - JsonResultSetConverter
  - AjaxJsonFindByNameServlet
  - AjaxJsonPersonLookup.html
- XML
  - Note using JPA for data retrieval as natural fit for JAXB
  - AjaxJpaXmlFindByNameServlet
  - AjaxJpaXmlPersonLookup.html

Which to choose?

- For most situations either one is fine
  - JSON
    - Flat structure like record sets easy
    - **Simple browser side processing**
  - XML
    - Browser side can be somewhat more complex
    - Very easy to add to existing class model
    - **Clear winner for deep structured data particularly with server side processing**