Advanced JSON

Persistence Mapping, RPCs, Cross-Domain and More.

Kris Zyp
kriszyp@xucia.com
www.xucia.com
www.json.com

Overview

- Specifications - JSON
  - JSONT
  - JSONPath
  - JSON Referencing
  - JSON Schema
  - JSON-RPC
  - JSPON
  - JSON-P
- Tools
  - CrossSafe
  - JSPON Browser
  - Persevere

JSON Overview/History

- JSON born from JavaScript object and array initializer syntax
- {“property”:”value”, “number”:2, “boolean”:true, “oddNumbers”:[1,3,5]}
- XML vs JSON
  - Strengths of each

About JSON

- JSON = {
  “size” : “compact”,
  “readable” : true,
  “purposes” :
  [“simple data structures”,
  ”object serialization”],
  “simple” : true,
  “easily parsed” : true
  }

XML…

```
<?xml version="1.0" encoding="utf-8"?>
<xml xmlns:ajson="http://www.xucia.com/
   page/Advanced_JSON" >
   <size>bigger</size>
   <readable type=“boolean” expectation=“consumer
   defines booleans the same way I do”>true</readable>
   <purpose ambiguity=“Is this an array?”>Give semantic
   meaning to documents</purpose>
   <derived_from>SGML</derived_from>
   <legacy_uses>many</legacy_uses>
</xml>
```

Interoperability

- Minimizing cost of communication.
- JSON has a lot of flexibility.
- JSON Definitions help us to interoperate without having to create or understand custom JSON usage.
- Analogous to RSS, SOAP, and XHTML in the XML world.

Cross Domain JSON

- XHR Same Origin Policy
- Script tags can access cross domain scripts
- Dynamic Script Tag Insertion
  - How to know when it loads?
  - How to know what was returned?
  - Is JSON even a valid script?

JSONP

http://bob.pythonmac.org/archives/2005/12/05/remote-json-jsonp/

- Defines a parameter for defining a prefix for JSON returned in a script
- www.jsonsource.com/?callback=done
- done({“name”:”my json object”})
- Yahoo, Flickr provide this for their webservices
- Unsecure used alone!

CrossSafe

- How to access cross-domain data securely?
  - Proxy – Secure, but slower and more work.
  - Dynamic Script Tags – Faster, more direct, but insecure, cross domain has full access to your JS environment.
  - Alternate technologies – Flash, signed applets
  - CrossSafe – Fast, direct access that is secure.
    - Implements JSONRequest Specification
    - Implements Subspace Approach
      - Uses Dynamic Script Tag insertion in nested iframes with domain containment for sandboxing

CrossSafe

Must use hostname.domain.com and make webservice.domain.com accessible

Servers must support JSONP or other callback parameter

JSONRequest.get(“http://www.yahoo.com/..”, function(id, value) { … }

Show Demo:

http://www.xucia.com/page/CrossSafe

JSONT

http://goessner.net/articles/jsont/

- Data: { *"link"*: {*"uri"*:*"http://company.com"*, *"title"*:*"company homepage"* }}
- Transformation: { *"link"*: *"{link.title}"* }
- Result: company homepage

Referencing

- Circular References

  Me = {“name”:”Kris Zyp”,
  “spouse” : {“name”:”Nikki Zyp”}}

  MyWife = Me.spouse;

  MyWife.spouse = Me;

- Multiples References to single objects

  list = [{“name”:”first”,”child”:{“name”:”the child”}},
  {“name”:”second”}]

  list[1].child = list[0].child;

JSON Referencing

www.json.com

- Scope
  - Intramessage
    - Circular References
    - Multiple References
  - Intermessage
  - Cross-Domain references
- Reference Methods
  - Path - ref: “$.prop[4]”
  - ID - ref: “111”

JSON Referencing (Intramessage)

www.json.com

- Reference Methods
  - Conventions – in string, fixups, in object
  - Path – use $ for JSON root (per JSONPath)

    [{“child”:{“name”:”the child”}},
    {“child”:{“$ref”:”$[0].child”}}]
  - ID

    [{“child”:{“id”:”1”,“name”:”the child”}},
    {“child”:{“$ref”:”1”}}]
  - Combine – {“$ref”:”1.name”}

JSON Referencing

www.json.com

- Intermessage – must use ID referencing
  - Intermessage
    - {“id”:”1”,”name”:”first”,
      “child”:{“id”:”3”,“name”:”the child”}}
    - {“id”:”2”,”name”:”second”,
      “child”:{“$ref”:”3”}}
  - Remote References
    - {“id”:”2”,”name”:”second”,
      “child”:{“$ref”:”http://www.json.com/jsonObject”}}
  - URL rules
    - Use the standard HTML rules for relative URLs in context
      GET /users/2
      {“id”:”2”,”name”:”john”,”group”:{“$ref”:”../groups/1”}}

Identification

- Circular and Multiple References

```
Me = {“id”:”kris”,
    “name”:”Kris Zyp”,
    “spouse”:{“id”:”nikki”,
                “name”:”Nikki Zyp”:
                “spouse”:{“$ref”:”kris”},
                “child”:{“$ref”:”jennika”}},
    “child”:{“id”:”jennika”,
      ”name”:”Jennika Zyp”,
      “age”:0.1}}
Me.spouse.spouse == Me
Me.child.age = 2;
Me.spouse.child.age -> 2
```

JSPON

www.jspon.org

- RESTful approach for interaction and interoperability with persistent data
- Most JSON is used to describe persisted data
- Deals with complex data structures, modifications, and more

Identification

- Partial Graph Transfer/Lazy Loading

  {“id”:”myObj”

  “name”:”my object”,

  “moreInfo”:{“$ref”:”largeObjectNotNeededNow”}
  }

- Object Modification
- Cross Domain Object Trees

JSPON

- Array String keys

  Me.children = []; MyWife.children = Me.children;

  {”children”:
  {“name”:”zypchildren”,
  “array”:[{“id”:”jennika”}]}}

- Prototypes

  function Zyp() {};

  Zyp.prototype={“lastName”:”Zyp”}

  Me = new Zyp();

  Me.firstName = “Kris”;

  JSPON:

  Me = {“id”:”kris”,”firstName”:”Kris”,
  “basis”:{“id”:”zyp”, “lastName”:”Zyp”}}

JSPON
Over
HTTP

- Every id has an implicit (or explicit) URL using the standard relative URL approach
- References to ids can be used for unloaded values (lazy loading)
- REST service
  - POST to an id/url to append an object
  - PUT to an id/url to change an object
  - GET to an id/url to get an object
  - DELETE to an id/url to delete an object

Persevere Server

- JSON Server
  - Object database server
  - Interaction through REST commands with JSON
  - JSPON id approach
  - Persistent Object Mapping
    - Rhino Environment
  - Supports JSONP
  - JSON-RPC method calls
    - Demo distributed calls later
  - JSON Schema support

Persevere Server

- Exposes multiple data sources as JSON (JSPON):
  - Internal OO Database
  - SQL Databases (MySQL, HSQL, etc)
  - XML files
- Data Centric Security
  - Avoids application logic centric security
- Modular server – can be used with other servers

JSONPath

- XPath for JSON
- Influenced by XPath and E4X syntax, and traditional C/Java/JS property access syntax
- Examples:
- $.store.book[0].title
- $..author
- $.book[:4] → first four books
- $.book[?(@.price<10)] -> books under 10

JSON-RPC

http://json-rpc.org/

- Request
  - **method** - A string containing the name of the method to be invoked.
  - **params** - An array of objects to pass as arguments to the method.
  - **id** - The request id. This can be of any type. It is used to match the response with the request that it is replying to.

JSON-RPC

- Response
  - **result** - The object that was returned by the invoked method. This must be null in case there was an error invoking the method.
  - **error** - An error object if there was an error invoking the method. It must be null if there was no error.
  - **id** - This must be the same id as the request it is responding to.

JSON-RPC

- Example

{ "method": "echo", "params": ["Hello JSON-RPC"], "id": 1}

{ "result": "Hello JSON-RPC", "error": null, "id": 1}

JSON Schema

http://www.json.com/json-schema-proposal/

- Contract about valid data
- Analogous to Classes in OO Languages or DTDs/Schemas in XML
- Defines requirements for JSON properties and other property attributes
- Uses
  - Validation - data integrity
  - Documentation
  - Interaction
    - UI generation (forms and code)
- Can be used on the client and server
- Compact Implementation

JSON Schema Example

```
{“name”:{
    “type”:”string”,
    “required”:true,
    “nullable”:false,
    “length”:25,
    “description”:”Name of the person”},
“age”:{
    “type”:”number”,
    “minimum”:0,
    “maximum”:125}
}
```

JSON Schema Example

```
{“name”:”Kris Zyp”
“age”:30,
“spouse”:{“id”:”nikki”},
“schema”:{“id”:”person”,
   “name”:{“type”:”string”},
   “age”:{“type”:”number”},
   “spouse”:{“type”:{“$ref”:”person”}} // recursive def
}}
```

JSPON Object Browser

- Capable of browsing, modifying, and observing structural and access rules of data sources.
- Demo at: http://www.xucia.com/browser.html?id=dyna%2F100788

Persevere

- Implements JSPON for browser persistent object mapping with REST web services
- Orthogonal Persistence and Lazy Loading Capabilities
- Implements JSON-RPC Method Calls
- Implements Persistent JavaScript API:
  http://www.persistentjavascript.org

Example Usage of Persevere

```
var result = pjs.load(“data or query id”);
var object = result[1];
var value = object.prop;
object.prop = “new value”;
result.push({name:”new object/row”})
Array.remove(result,object); // or splice
```

What Did We Not Have to Write?

- doAjax… (break up into multiple slides)
- Request controller
- Do sql…
- API to learn… Almost zero, majority of operations are standard JavaScript

Persevere Capabilities

- Compared to Jester – Jester is perfect for RoR
- Robust and scalable
  - Lazy loading
  - Automatic and Explicit Transactions
  - Designed for optimistic locking/merging
- Auto save/Orthogonality
- Integration with Strands for threading/continuations
- Bi-directional dynamic object and structure creation and modification

Demo

- Demonstration of CRUD application with customer database.
- Fits well with other JavaScript libraries

Persevere
Persisted Applications

- Functions/Methods can be stored in the persisted object stores
  - Applications can exist as persistent object graphs.
- Persevere server implements Persistent JavaScript on the server.
- Distributed Computing capable with other JSPON/RPC implementators.
  - Transparent remote calls

Persisted Application & Distributed Computing Demonstration

- Customers Demo
- Find Primes Demo
- Reverse Ajax
- Inheritance

Thank you for your time

See the presentation and links to projects at:

www.xucia.com/page/AdvancedJSON

Xucia Incorporation

www.xucia.com

www.json.com

email: kriszyp@xucia.com

Kris Zyp

503-806-1841