Title

Jim Higosn

2013

Contents

Abstract

100-200 words

Introduction

introduction should be 2-5 pages

Background

background should be 2-10 pages

SOA

REST/WebServices (WSDL etc)

What is a rest client in this context (a client library)

Marshalling/ de-marshalling. Benefits and the problems that it causes. Allows one model to be written out to XML or JSON

Big/small message problem and granularity. With small: http overhead. With big: not all may be needed.

Parsing: SAX and Dom

Why sax is difficult

DOM parser can be built on a SAX parser

State of http as a streaming technology

Dichotamy between streaming and downloading in the browser for downloading data. But not for html (progressive rendering) or images (progressive PNGs and progressive JPEGs)

Lack of support in browser Long poll - for infrequent push messages. Must be read Writing script tags

All require server to have a special mode. Encoding is specific to get arround restrictions.

XmlHttpRequest

XmlHttpRequest (XHR)

Xhr2 and the .onprogress callback

Applicaiton and Reflection

40 to 60 pages

Focus on replacing ajax, rather than streaming. In older browsers, getting the whole message at once is no worse than it is now.

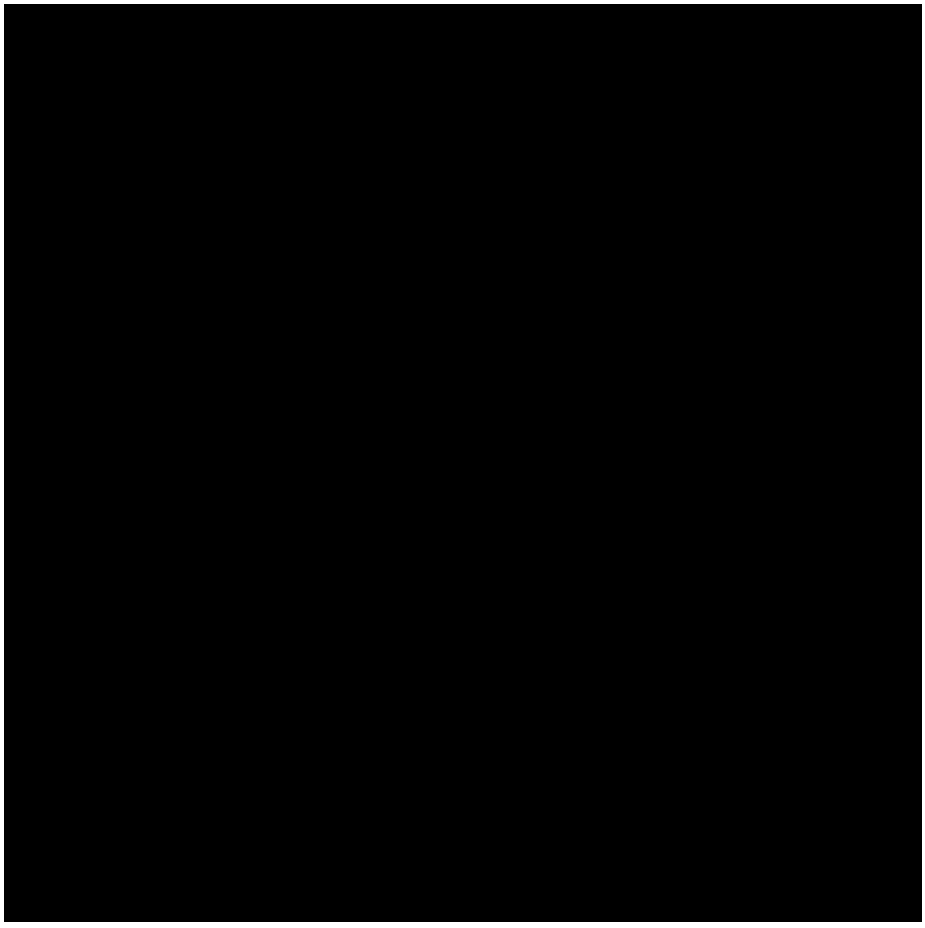

Figure 1: Over seveeral hops of aggregation,

choice of technologies

can justify why js as:

Most widely deployable.

asynchronous model built into language already, no 'concurrent' library needed compare to Erlang.

funcitonal, pure functional possible [FPR] but not as nicely as in a pure functional language, ie function caches although can be implemented, not universal on all functions.

easy to distribute softare (npm etc)

summary of json

Why json?

A bridge between languages that isn't too different from the common types in the languages themselves a common bridge between languages

Also very simple. Easy to parse.

identifying interesting objects in the stream

The failure of sax: requires programmer to do a lot of work to identify interesting things. Eg, to find tag address inside tag person with a given name, have to recognise three things while reieving a callback for every single element and attribute in the document. As a principle, the programmer should only have to handle the cases which are interesting to them, not wade manually through a haystack in search of a needle, which means the library should provide an expressive way of associating the nodes of interest with their targetted callbacks.

First way to identify an interesting thing is by its location in the document. In the absense of node typing beyond the categorisation as objects, arrays and various primative types, the key immediately mapping to the object is often taken as a lose concept of the type of the object. Quite fortunately, rather than because of a well considered object design, this tends to play well with automaticly marshalling of domain objects expressed in a Java-style OO language because there is a stong tendency for field names – and by extension, 'get' methods – to be named after the *type* of the field, the name of the type also serving as a rough summary of the relationship between two objects. See figure 2 below.

By sensible convention, even in a serialisation format with only a loose definition of lists, lists contain only items of the same type. This gives way to a sister convention, that for lists of items, the key immediately linking to the

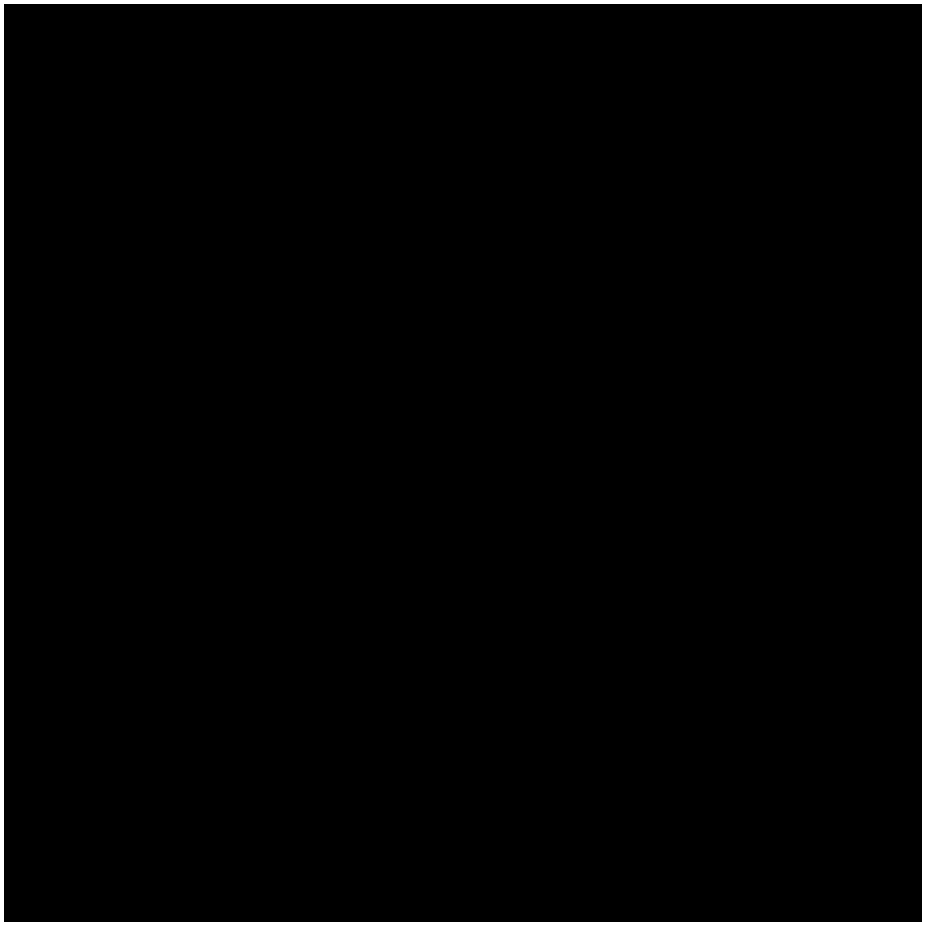

Figure 2: UML class diagram showing a person class in relationship with an address class. In implementation as Java the 'hasAdress' relationship would typically be reified as a getAddress method. This co-incidence of object type and the name of the field refering to the type lends itself well to the tendency for the immediate key before an object to be taken as the type when Java models are marshalled into json

Essentially two ways to identify an interesting node - by location (covered by existing jsonpath)

Why duck typing is desirable in absense of genuine types in the json standard (ala tag names in XML). or by a loose concept of type which is not well supported by existing jsonpath spec. Object syntax for listing fields.

`Examples of jsonpath syntax`

breaking out of big/small tradeoff

Best of both modes

Granularity: only need read as far as necessary. Services could be designed to write the big picture first. Alternatively, where resources link to one another, can stop reading at the link. Eg, looking for a person's publications, start with an index of people but no need to read whole list of people.

Aborting http request may not stop processing on the server. Why this is perhaps desirable - transactions, leaving resources in a half-complete state.

styles of programming

some of it is pure functional (jsonPath, controller) ie, only semantically different from a Haskell programme others, syntaxticaly functional but stateful to fit in with expected APIs etc

jsonpath implementation allows the compilation of complex expressions into an executable form, but each part implementing the executable form is locally simple. By using recursion, assembling the simple functions into a more function expressing a more complex rule also follows as being locally simple but gaining a usefully sophisticated behaviour through composition of simple parts. Each recursive call of the parser identifies one token for non-empty input and then recursively digests the rest.

automated testing

testing via node - slowserver. Proxy. why jstd's built in proxy isn't sufficient

testing pyramid

Could implement a resume function for if transmission stops halfway

```
.onError( error ) {
   this.resume();
}
```

stability over upgrades

why jsonpath-like syntax allows upgrading message semantics without causing problems [SOA] how to guarantee non-breakages? could publish 'supported queries' that are guaranteed to work

support for older browsers

Still works as well as non-progressive json Could be used for content that is inherently streaming (wouldn't make sense without streaming)

suitability for databases

Databases offer data one row at a time, not as a big lump.

weaknesses

implementation keeps 'unreachable' listeners difficult decidability/proof type problem to get completely right but could cover most of the easy cases

Parse time for large files spread out over a long time. Reaction to parsed content spread out over a long time, for example de-marshalling to domain objects. For UX may be preferable to have many small delays rather than one large one.

Doesn't support all of jsonpath. Not a strict subset of the language.

Rest client as a library is passing mutable objects to the caller. too inefficient to re-create a new map/array every time an item is not as efficient in immutability as list head-tail type storage

An imutability wrapper might be possible with defineProperty. Can't casually overwrite via assignment but still possible to do defineProperty again.

Would benefit from a stateless language where everything is stateless at all times to avoid having to program defensively.

Conclusion

1 to 5 pages

Bibliography

Appendix