Sling Type System: motivation and requirements

STATUS as of December 2020: see SLING-9950

I've been working on an off this year (2018) on creating stronger models at the core of Sling applications, to represent Sling Resources, their content models, relationships (like parent/child) to other resource types, etc.

This requires lots of thinking and I also had other things to do, so far I only have a very rough prototype at https://github.com/apache/sling-whiteboard/tree/master/resource-schemas but discussions at adaptTo() and with colleagues show a definite need for something like that, provided we can integrate it with the core of Sling. Or maybe create new default servlets that use such a model, "details" to be defined...

The goal of this page is to collect ideas and requirements, opening up this exploration to others.

Feel free to expand this page with additional ideas, requirements and comments as needed!

Motivation

Sling applications provide a rich HTTP API but that's generally underspecified and out of the box it is hard to create a fully self-describing and hypermedia driven HTTP API.

Several modules help with this: Sling Models, Sling Validation, HApi can all improve the situation (and I'm probably forgetting other modules) but they feel more like isolated islands of functionality rather than a consistent toolset.

The Sling resource type and servlet/script resolution concepts provide a powerful and mostly declarative way to implement HTTP request processing, but we do not expose the corresponding links in the default HTTP API, which would make that API self-describing and hypertext-driven. And the resource type and the things that it points to does not explicitly constrain the resources that are managed. JCR node types could help with that but they might not be rich enough and we generally want to limit the dependencies on the JCR API to allow Sling to run on other content storage systems.

All this shows a need for a unifying concept where Sling Resource Types are described with much more details, which can include content models (based on https://github.com/adobe/xdm maybe, or at least on JSON Schema), relationships between resource types to indicate allowed parent/child relationships, default content and access control for newly created resources, as well as any other relevant attributes of a Resource Type that can help create self-describing APIs, be they based on HTTP, front-end user interfaces or maybe things like GraphQL or command-line query/content manipulation languages.

Having such a unified model, and deriving APIs, validation and default content from it, should help bring Sling to the next level by making it natural to create self-describing APIs, as well as deriving documentation and validation rules from the same model.

I suggest calling this the Sling Type System, I think that's consistent with the Wikipedia definition of that term.

Notes from other discussions

- On our dev list, Jason mentions that overriding resource types in RequestDispatcherOptions for sling:include operations is useful and commonly used. This sounds similar to casting objects, and we probably should apply similar rules than Java when doing that - is the object being cast compatible with the target type.

See Also

- https://introspected.rest/ - exposes interesting and somewhat similar concepts with MicroTypes, as well as delivering hypermedia information separately from content, in parallel
- https://blog.heroku.com/heroku-http-api-toolchain - Heroku's toolchain to generate HTTP API docs and (Ruby and Go) client code based on JSON Schema
- http://json-schema.org/ which now includes *JSON Hyper-Schema* allowing hypertext links to be included in JSON Schema in a standard way, including templated links.
- https://brandur.org/elegant-apis discusses these techniques with examples.

FAQ

What's wrong with Swagger / OpenAPI ?

I'd be happy to be proven wrong but so far it feels like Swagger is strongly oriented towards a path-driven view of the world, like "all books are found under /books".

That doesn't fly for the rich content-driven view of the world that Sling promotes, where users are free to arrange their content objects as they see fit, letting real-world concerns drive the content hierarchy.

Sling applications are driven by resource types, which ultimately (but in an incomplete way so far, as explained in the "motivation" section) define the behavior of resources, without taking into account their position in the content hierarchy.

That being said, it should be possible to derive OpenAPI information from a Sling Type System model, as that model will describe everything that we know about the underlying content.

Why do we need to do this differently from everybody else?

See the above question about OpenAPI - it seems like many people are happy to let object types drive the content hierarchy, while we have strong reasons to allow users to drive that hierarchy based on business requirements.

Isn't GraphQL much more modern than REST anyway?

That's not really a good question as those are very different animals: GraphQL is a query language while REST is an architectural style.

https://philsturgeon.uk/api/2017/01/24/graphql-vs-rest-overview/ has a more elaborate view on this.

From a Sling point of view, exposing a strictly controlled set GraphQL queries over a Sling Type System model might be useful, as a way of allowing rich clients to retrieve "ideal" documents for building their user interfaces. But I think we should see this only as an alternate view on the same thing, which is the content and objects model that the Sling Type System defines.

Requirements

TDB - let's discuss the concept and motivations first, maybe.