RDF4J Planning Session

September 2022

Present:

Håvard Ottestad
Jeen Broekstra
Pavel Mihaylov
Tomas Kovachev
Andreas Schwarte
Bart Hanssens
Jerven Bolleman
Marco Neumann

Summary

RDF4J 5.0 will be our next major release and we aim to release it by Q1 2023. A major release allows us to remove or rework existing features. These changes should be marked with deprecations in the next (and hopefully last) minor release, 4.3.0.

We discussed some larger breaking changes, including the removal of the Iteration interface, removing support for serializable and defaulting to the extended evaluation strategy. We will also take the chance to clean up the code, fix unintended side effects and remove long deprecated classes.

New features would either be targeted at the next minor release (4.3.0), or the next major release (5.0.0). Two features are already in the works for the next minor release, transaction settings for changing the evaluation strategy and support for sails to provide optimised collections during query evaluation. Support for JSON-LD 1.1 and improved support for RDF-star are likely candidates for 5.0.0 as well as a new SHACL validation API and several potential performance and extendability improvements.

RDF4J 4.x

- Once “last” release to mark deprecations.
- Make it easier to switch between evaluation strategies.
    - Enum to set strategy mode
    - Make the extended -> standard. Strict is minimal compliance, Extended is useful within the standard. At least note that this is in the JavaDoc. See below.
- Collections factory to allow the sail to provide collection implementations to the query evaluation.
- CoreDatatype enum for XSD and Geo datatypes. Use those more often all over the query engine.
- Sparql order comparison (ValueComparator):
    - compare(value a, value b) current
    - compare(IRI a, IRI b) add type specific options

RDF4J 5.0

Removing or redesigning deprecated code

- Remove Iteration class and move methods to CloseableIteration. Messes up next() and hasNext() in one interface close in a second. This is due to i_stub hard to inline. (Note from jerven: also synchronized and isClosed check). Short term deprecation notices are difficult (for removal:true).
- Parser currently supports mutable Rio settings during parsing, which means that there are a lot of lookups. Making the Rio setting immutable would allow us to optimize the parser to read all the settings just once.
    - This might not require deprecation, if it is an internal and undocumented
- Move RIO settings to parser e.g. for JSON-LD. Change parser to Titanium, Rio depends on this JsonLdJava. Move most settings to JSON-LD so dependency not needed. Minimal dependencies for RIO/core.
- Remove the old RDFS reasoner?
- Remove limited size iterators.
- Change the default evaluation strategy from strict to extended.
- HashCode calculations.
- Remove serializable

New features:

- JSON-LD 1.1
    - Titanium
    - Should be possible to already select which JSON-LD processor to use. Using the service factory/mime-type. Must not depend on order of classpath

- Better support for custom Value implementations (like our NativeStore og LmdbStore which use a long id)
    - Many options. Main goal is to reduce the need to retrieve the actual value from disk. Two main issues at the moment are the equals and hashCode methods, which may require retrieving the actual value.
- Move more code to use CoreDatatype
- Improve extendability of Comparator implementations
    - One use case would be to allow comparing two custom IRI
- SHACL Validation API to support validation data that is already in a Sail/Repository without moving any data.
    - Currently a prototype PR that supports Sail, but needs to be extended to support Repository
    - Would be nice to support remote repositories, but that might be very slow. We could generate SPARQL queries, but the ShaclSail doesn't support generating queries for all constraints.
- RDF-Star is wanted but under specified. Do we implement CG publication or WG to be developed?
    - Also add RDF-Star to native store