Search Developers Guide

Introduction

The eagle-i search application is packaged and deployed in two configurations: *institution* and *central*. The institution search application deployment supports the search of resource data in a single institution's eagle-i repository. The central search application deployment supports the search of data in a set of eagle-i institution repositories.

In the eagle-i source tree, the `webapps/institution` project is used to build the single institution search app. The `webapps/central` project is used to build the multi-institution search application.

The institution and central webapp projects contain very little code consisting of two java-based configuration files that determine in which mode the client and server search implementation will be executing.

`InstitutionApp.java` and `CentralApp.java` are the Google Web Toolkit (GWT) `EntryPoint` implementations that determine in which mode the GWT-based browser UI will execute. They instantiate a `SearchApplicationContext` with the parameter true or false that tells the UI whether it should display single or multi-institution search components. For example, in single institution mode, the Institution filter will not be displayed in the Location section of the sidebar, and the institution name is not displayed in the search result summary.

`InstitutionSearchConfig.java` and `CentralSearchConfig.java` are Spring configuration files that determine the back-end Spring beans that will be loaded for the two search app modes. In particular, it defines the `rootSearchProvider` singleton bean which is the core handler for search requests from the client.

The bulk of the search application implementation lives in the library project dependencies of institution and central WAR projects.

Search Back-End Components

Key components of the back-end and client implementation are described below:

Notes:

- **=>** Indicates a singleton (Spring Bean) dependency
- Italic comments in parenthesis indicates potential changes to the current implementation

`org.eaglei.ui.gwt.search.server.SearchServlet`

`SearchServlet` handles the core set of search operations requested by the the client. In the `SearchServlet init()` method, references to its singleton dependencies are obtained, and harvesting of eagle-i repository data is initiated.

=> **`ConnectionManager`**

- Called to validate user session ids (*Move into SearchProvider?*)

=> **`InstitutionRegistry`**

- API for getting Institution list when running in central mode *(Should be moved to some other servlet.)*

=> **`PollingDataHarvester`**

- Initiate search index population *(Make SearchProvider API?)*
- On search operation, first check if initial data indexing has completed *(Make SearchProvider API?)*

=>**`SearchProvider`**

- Process `SearchRequests`
- Get search result list: `SearchResultSet`, `SearchResult`
- Get supertype & subtype counts:`ClassCountResult`

- Get resource provider type counts:`ClassCountResult`
- Standard implementation is a `MultiDataSetSearchProvider`, described below

=>`*EIOntModel`

- Compute "root" type of search result (*have `SearchProvider` compute; part of `SearchResult`*)

=> **`AsynchronousLoggerSearch, AsynchronousLoggerCount`**

- DB logging of search request, counts, performance

`org.eaglei.services.InstitutionRegistry`

- Static list of institution information *(should be dynamically initialized, for example, determine which institutions are currently accessible)*
- Provides name, id, and `RepositoryHttpConfig` for each institution *(migrate from legacy URIs.)*
- Used to lookup institution EIEntity (label) from id
- `/common/services/src/main/resources/services-config.xml` is the primary configuration file for `InstitutionRegistry`. It is packaged in JAR *(Should be loaded from classpath location)*. It has a dependency on finding an institution-registry.properties file in the classpath which contain properties
    - `org.eaglei.tier`
        - used to compute URIs
        - (need to confirm that things work if prop is omitted, prod implicit)
    - `org.eaglei.subdomain`
        - defines what node app is running as: "search" for central, or institution subdomain/id
        - used to compute URIs
        - *(TODO: bad prop name, fairly hacky overloading...)*
    - `org.eaglei.repository.username`
    - `org.eaglei.repository.password` (*TODO: central: can't assume same for every institution*)

`org.eaglei.ui.gwt.suggest.server.DataSuggestServlet`

`DataSuggestServlet` is the server-side provider of the list of suggestions displayed as the user types in a search string. Note that client-server communication for this servlet is JSON-based rather than GWT RPC.

=>**DataHarvester**

- On suggest operation, first check if initial data indexing has completed *(make SuggestionProvider api?)*

=> **SuggestionProvider**

- Process EnityMatchRequest
- Get suggestions: `List<EntityMatch>`

`org.eaglei.search.harvest.PollingDataHarvester` extends `DataHarvester`

- `addChangeListener(ResourceChangeListener)`
- `(EIEntity institution)`
- `onChangeEvent(ResourceChangeEvent event)`
- `onChangeStreamEnd(EIEntity institution, Date lastModifiedDate)`
    - *(no rollback support. Needed?)*
- `optimize()`
- `harvest()`
- `hasInitalData()`; *(move to SearchProvider?)*
- `startPolling()`

`org.eaglei.search.harvest.RepositoryStreamHarvester` implements `PollingDataHarvester`

=>**`InstitutionRegistry`**

- Get `RepositoryHttpConfig` for a given institution, to call harvest

=> **EIOntModel**

- Lookup `EIEntity` from URIs (label)
- Lookup valid property list for type
- Get list of resource provider property URIs
- Harvests from a single eagle-i repository
- List of `ResourceChangeListeners`
- Parses harvest response body in to `ResourceChangeEvent`
- Two event types: Delete, Changed
    - *(Withdrawn, Return to Curation/Draft vs. Delete?)*
    - *(Create vs Changed?)*
- Drops all resource change notifications that don't have a label property, or type property
- Ignores all properties that don't conform to current `EIClass`
- Ignores all types that aren't in the current data model
- Exception in a notification is isolated to that listener *(no rollback support. Needed?)*

- *(add support for computing inverse properties, if not done in the repository; here or in indexer)*
- on call to `startPolling()`, creates a thread that calls `harvest()`
    - 10 sec sleep *(make configurable)*
- *(Timeout if `ResourceChangeListener` takes too long?)*
- *(Drop out on out of memory exception?)*
- *(Would be nice to get change notification from repo for referencing resources, on delete especially)*

`org.eaglei.search.harvest.MultiDataSourceStreamHarvester` implements `PollingDataHarvester`

- multiple institution repository DataHarvester
- holds a list of DataHarvester
- on call to startPolling(), creates thread that calls each DataHarvester.harvest() sequentially
- 2 sec sleep *(make configurable)*
- change events go each ResourceChangeListener from a single thread
  org.eaglei.solr.search.SolrSearchIndexer implements ResourceChangeListener
  *added to a DataHarvester as a ResourceChangeListener

=> **EIOntModel**

=> **Analyzer**

=> **Directory**

=> **ClassUsageCache**

- Used to avoid getting counts for unused classes

=> **ProviderUsageCache**

- Used to avoid getting counts for unused provider types

=> **SearchExcludeConfiguration**

- `/common/runtime/solr/src/main/resources/search-config.xml`

- Exclude all "Generic Lab" provided resources
- Exclude all non-resource provider Organization instances
- handle resource data received in any order
- unresolved object property values
- compute and store inferred supertypes
- store resource provider info
- assumption: embedded class data always received after containing class
- rolls up properties of "flatten" annotated instances (embedded & search_flatten) into containing instance
- exclude all flatten instances from search results *(add support for some flatten instances that can be search results)*
- *(Index-time boosting of doc types and fields)*

`org.eaglei.solr.suggest.SolrDataSuggestIndexer` implements `ResourceChangeListener`

- added to a `DataHarvester` as a `ResourceChangeListener`

=> **`EIOntModel`**

=> **`Analyzer`**

- This is a different analyzer than the search analyzer

=> **`Directory`**

- Separate index from search index (same index?)_
- Stores "terms" used in all resource documents
- Resource instance label
- Ontology class label
- Ontology class synonyms
- *(Resource instance Alternative Name)*
- *(Select free text property values:_ gene name)*
- *(Find term usage in free text)*
- Stores the "categories" in which the term is utilized
- The top-level search sidebar resource types
- UnknownCategory (People, Orgs)

`org.eaglei.search.provider.MultiDataSetSearchProvider` implements `SearchProvider`

- Maps a dataset id (eagle-i, PubMed, Entrez Gene, NIF, etc.) in SearchRequest to a SearchProvider implementation for that source of data.

`org.eaglei.solr.search.SolrSearchProvider` implements `SearchProvider`

SolrSearchProvider is class that executes a search query on the eagle-i data stored in a Solr indices.The data available to be searched may be single- or multi-institution based on application configuration. The SolrSearchProvider ensures that UI dependencies such as the match highlighting snippit are set in the search response.

=> **`InstitutionRegistry`**

- For lookup of institution EIEntity (label) to put in response

=> **`SolrServer`**

- The SolrServer to be queried -- implementation described below.

=> **ClassUsageCache**

- Get supertypes and subtypes to perform count queries on

=> **`ProviderUsageCache`**

- Get resource provider types to perform count queries on

=> **SolrDataSuggestProvider**

- Used to extract entity URI(s) from search string. For example, recognize that the query string "thermal cycler" is the label of a type of instrument in the eagle-i ontology. By determining a URI for a search string, matches on the URI can be prioritized.

=> **`SolrSearchQueryBuilder`**

- Generates a Lucene query from a `SearchRequest`

Solr

The eagle-i system supports three Apache.Solr index configurations: `ModelSuggest-core`, `Search-core`, and `DataSuggest-core`.

An instance of a `SolrServer` utilized by eagle-i web application (e.g. Search, SWEET, Ontology Browser) will contain one or more of these three indices. Currently, each web application instantiates its own embedded `SolrServer` with the cores that the particular web app need.It is anticipated that in the future, web applications will utilize a shared `SolrServer`.

- `ModelSuggest-core` is an index of all terms in the eagle-i ontology. This core is used to provide autosuggest of ontology class labels and class synonyms. The suggestion list returned can be filtered by type (class uri): for example, only return suggestions of type Instrument. Clients of `Mode lSuggest-core` include the SWEET form fields, and the Ontology Browser (Glossary) searchbox.

- Search-core is an index of eagle-i resource data. Querying this index produces the relevance ranked list of search results presented to the user in the search application. The central search application maintains an index of data from all institutions in the network. The search application on each institutional node maintains an index of the data of that particular institution.

- DataSuggest-core supports the suggestions offered by the search application search box. It is an index of all resource instance labels. It also indexes ontology terms and synonyms, however, unlike `ModelSuggest-core`, `DataSuggest-core` will only include ontology terms *currently used in the dataset*. For example, if the ontology term "Lepidosauria" is not currently used by any resource in the eagle-i dataset, it will not be offered in autosuggest.

Another feature of the `DataSuggest-core` is that it keeps track of the resource categories in which any given term (resource label or class label) is used. This supports the "in Instruments" and "in Reagents" feature of the search autosuggest.

`org.eaglei.solr.EagleISolrConfig`

This class is provides api for reading in solr configuration files, and determining the location that solr data files will be written.

By default, the configuration files for these solr cores are read from the solrhome directory in the eagle-`i-common-solr.ja`.

The location of data files created for the these core indices is the concatenation of the system property `org.eaglei.home`, a string passed to the constructor of `EagleISolrConfig` (normally the `application name` -- search, sweet, glossary), plus the name of the core.