Linked Data Gazetteer PR

This page includes information about the gazetteer PR and the two LRs used by the gazetteer - TRIE Cache and Metadata.

Purpose of the Linked Data Gazetteer Processing Resource

A gazetteer component is generally used to recognize given pieces of text as objects, meaningful for the user. The Linked Data Gazetteer processes texts and creates Lookup annotations over words or groups of words (so called tokens), and assigns different features to these annotations. It uses a pre-filled cache structured as `(label -> (instance identifier, instance type))`. The contents of this cache is defined by the TRIE Cache for the Linked Data Gazetteer Language Resource, which is attached as a runtime parameter to the Gazetteer Processing Resource. There is a possibility to add other annotation features to the Lookups generated by the gazetteer, the Metadata Language Resource responsible for these. An instance of the Metadata language resource can be attached as an optional runtime parameter to the Linked Data Gazetteer PR. Its cache structure is usually represented as `(identifier(instance identifier, instance type) -> list (feature name, feature value)|(feature name, feature value))`.

Resource workflow

1. The Linked Data Gazetteer PR retrieves all annotations of type Token.
2. The set of Token annotations is passed to the TRIE Cache Language Resource for matching against the known entries.
3. For every match found by the matching routine, a Lookup annotation is added. At this point, the generated annotation has the following features available:
    a. inst - contains the instance URI for the entity;
    b. class - contains the URI of the class with which the entity is stored within the gazetteer cache;
    c. string - contains the label that was matched;
    d. id - a unique integer value, identifying the (instance identifier, instance class) pair within the resource cache.

Specifics

1. Generally, the processing resource works over the "string" feature of the Token annotations. However, there is an option to match against additional features of the Token annotation, such as "stem" or "lemma". These features can be generated by using specific processing resources, usually distributed with the GATE platform.
2. The Linked Data Gazetteer should be considered as a semantic resource. Before running an instance of the processing resource over a document, the document must be annotated with Token annotations, using the default GATE tokenizer - the ANNIE English Tokeniser Processing Resource. Any other processing resources that create Token annotations can be used as well. Considering the above mentioned option for matching against more Token features, the pipeline must contain processing resources able to generate these features for the Token annotations.
3. Developers: one can implement their own cache and matching routine by implementing the com.ontotext.gate.gazetteer.MatchLR interface.

TRIE Cache for the Linked Data Gazetteer Language Resource

1. Using the Resource: This resource represents a cache structure containing `(label -> (instance identifier, instance type))` pairs. TRIE matches labels from the cache against the provided tokenized text and returns entries in the (instance identifier, instance type) form that match the text.
2. Initialization: The TRIE Cache Language Resource has various initialization-time parameters and most of them are mandatory. The most important parameters describe the remote repository endpoint, case sensitivity of the gazetteer, matching strategy, paths to serialized cache on disk and queries used for loading. For more information, refer to the Parameters section.
3. Tuning: If the process of adding new entries to the cache needs to be fine-tuned, there are parameters for setting paths for stop words file, label-processing rules, list of Token annotation feature names for the matcher to run over, and a helper pipeline for adding new Token annotation features. For more information, refer to the Parameters section.
4. Parameters:
    a. `connectionString` - SPARQL endpoint for the remote repository where semantic data is stored;
    b. `isCaseSensitive` - if the matching should be case sensitive or not;
    c. `lemmetizerHelper` - a helper pipeline for adding Token annotations with important features;
    d. `matchStrategy` - how matching should be performed. There are two possible options:

        i. LEFTMOST_LONGEST - matches only the leftmost longest possible match. Example: if you have "Anton Atanassov" and "Atanassov" as labels in the cache (they are going to be matched) and you have a sentence "Anton Atanassov works at Ontotext" the algorithm will match only "Anton Atanassov" and there will be no "Atanassov" Lookup annotation. This way no nested annotations will be created.
        ii. ALL - matches all that can be matched. The above example will match both "Anton Atanassov" and "Atanassov".
    e. `path2dic` - the path where cache will be serialized and from which the LR tries to deserialize it on subsequent initialization;
    f. (optional) `path2ignorewords` - the path to a plain text file containing words that should be ignored when filling the cache;
    g. (optional) `path2rules` - the path to a groovy source code file containing rules for re-writing a label. The source must contain a workFlow method that returns a set of Strings. The groovy code is used for creating derivatives of a label. When a label is added to the cache, all its derivatives are added as well. For example, let's say you want to lookup the label "Obama, Barak" but in most articles the mention of the name is "Barak Obama". You can use a rule in the groovy code so when you pass "Obama, Barak" you get "Obama, Barak", "Barak Obama" and both labels are added to the cache for matching.
    h. `queryFile` - URL of a file containing SPARQL queries that return a list of triplets - `<URI instance, Literal label, URI type>`. The SPARQL queries in the file are separated by `@@@`. They are executed one after another and the result of each is fed to the cache. If there is no file for deserialization in `path2dic`, the queries are executed on the SPARQL server defined in `connectionString`. SPARQL variables bound to elements of the triplets:

```
?concept <-> URI instance;
?literal <-> Literal label;
?type <-> URI type.
```

    i. `shouldCreateDump` - if the cache should be serialized (default is true);
    j. `tokenFeatures` - the cache matches according to the values of the features listed in this list. It is tightly coupled with the parameter `lemmetizerHelper`;
    k. (optional) `updateable` - in some cases, after initial loading, the cache should not be available for updates. Set this to 'false', if needed.

Note: The SPARQL endpoint has to be opened or the security has to be off, because there are no parameters for user name and password.

Metadata Language Resource

1. Using the Resource: This resource represents a cache structure containing `(identifier(instance URI, class URI)` -> list of semantic metadata features in the `(identifier(feature value, feature name)) form)`. It is used by the Linked Data Gazetteer Processing Resource to assign additional features to the generated Lookup annotations. The Metadata Language Resource and the TRIE Cache Language Resource are very tightly coupled as they need to use common structures to identify the entity instances represented by the Lookup annotations.
2. Initialization: This Language Resource has several initialization parameters, two of which are mandatory. The parameters include location of the SPARQL endpoint for cache loading, paths to the shared structures with the TRIE Cache Language Resource, and path to a file containing queries used to load the cache.
3. Parameters:
    a. connectionString - SPARQL endpoint for the remote repository where semantic data is stored;
    b. indexFilePath - path to the filesystem location where cache will be serialized;
    c. pathToEntityPoolFolder - the location of the Entity Pool structure, shared with the TRIE Cache Language Resource. The value of this parameter must be exactly the same as the value of the 'path2dic' parameter of the TRIE Cache.
    d. queryFilePath - the path to the file with queries to load the resource data.

Linked Data Gazetteer Processing Resource

1. Using the Resource: This Processing Resource runs over the document text and produces Lookup annotations with (optionally) semantic data features. It uses the TRIE Cache Language Resource and (optionally) the Metadata Language Resource.
2. Runtime parameters:
    a. cacheLR - this is the TRIE Cache Language Resource that will be used for matching;
    b. (optional) inputAsName - the name of the annotation set where the Token annotations are, the default is <null>, i.e. the default annotation setting;
    c. (optional) metadataLR - the Metadata Language resource bound to the corresponding TRIE Cache Language resource.

Step-by-step Guide for Creating and Adding a Linked Data Gazetteer Processing Resource into a Pipeline

1. Open a GATE Developer instance.
2. (Optional) Load the GATE application where the Gazetteer PR is to be added.
3. Load the gazetteer CREOLE plugin:
    a. File -> Manage CREOLE plugins.
    b. Click on + button.
    c. Type in the directory location (prefixing it with 'file://') or use the 'Select a Directory' button.
    d. Click 'OK' and select the 'Load Now' option for the newly loaded plugin.

    e. Click 'Apply All'
4. Prepare a helper application:
    a. Right-click 'Applications' from the left-hand side menu, select 'Create New Application -> Conditional Corpus Pipeline'.
    b. Double click the created pipeline and add an existing tokenization PR into it; if no tokenization Processing Resources are present, use the following procedure:
        i. Load the ANNIE CREOLE plugin from the 'Manage CREOLE plugins' screen.
        ii. Right-click 'Processing Resources -> ANNIE English Tokeniser'.
        iii. Add the newly created resource to the helper pipeline.
5. Right-click 'Language Resources -> TRIE Cache for Linked Data Gazetteer' to create a TRIE Cache for the Linked Data Gazetteer Language Resource.
6. Set parameter values, refer to the TRIE Cache Parameters section for more information.
7. Click 'OK' and the TRIE Cache Language resource will begin preparing its cache by evaluating the queries against the SPARQL endpoint or deserializing existing data.
8. (Optional) Right-click 'Language Resources -> Metadata LR' to create a Metadata Language Resource.
9. (Optional) Set parameter values, refer to the Metadata Language Resource Parameters section for more information.
10. (Optional) Click 'OK' and the Metadata Language resource will begin preparing its cache by evaluating the queries against the SPARQL endpoint or deserializing existing data.
11. Right-click 'Processing Resources -> Linked Data Gazetteer' to create a Linked Data Gazetteer Processing Resource.
12. Open the pipeline where the Linked Data Gazetteer must be added and add the newly created instance at the desired position within the pipeline.
13. Select the Linked Data Gazetteer Processing Resource and set its runtime parameters, refer to the Linked Data Gazetteer Processing Resource Runtime parameters section.