javasphinx

Release 0.9.15

Oct 01, 2018

Contents

Welcome to the javasphinx user's guide.

CHAPTER 1

Introduction

javasphinx is a Sphinx extension that provides a Sphinx domain for documenting Java projects and a `javasphinx-apidoc` command line tool for automatically generating API documentation from existing Java source code and Javadoc documentation.

CHAPTER 2

Installing

javasphinx is available in the Python Package Index (PyPi) and can be installed using tools such as `pip` or `easy_install`,

```
$ pip install javasphinx
```

or,

```
$ easy_install -U javasphinx
```

CHAPTER 3

Configuration

To enable javasphinx for your existing Sphinx configuration add `'javasphinx'` to the list of extensions in your conf.py file. javasphinx can be configured to cross link to external sources of documentation using the `javadoc_url_map` option,

```
javadoc_url_map = {
    'com.netflix.curator' : ('http://netflix.github.com/curator/doc', 'javadoc'),
    'org.springframework' : ('http://static.springsource.org/spring/docs/3.1.x/
↪javadoc-api/', 'javadoc'),
    'org.springframework.data.redis' : ('http://static.springsource.org/spring-data/
↪data-redis/docs/current/api/', 'javadoc')
}
```

Each key in the map should be a Java package. Each value is a tuple of the form `(base_url, doc_type)` where `base_url` is the base URL of the documentation source, and `doc_type` is one of,

`javadoc` For documentation generated by the Javadoc tool *before* version 8.

`javadoc8` For documentation generated by the Javadoc tool after version 8. This is required due to changes in how method anchors are generated (see JDK-8144118).

`sphinx` For external documentation generated by javasphinx.

When comparing referenced types to the list of available packages the longest match will be used. Entries for `java`, `javax`, `org.xml`, and `org.w3c` packages pointing to http://docs.oracle.com/javase/8/docs/api are included automatically and do not need to be defined explicitly.

CHAPTER 4

Java domain

4.1 Directives

The Java domain uses the name **java** and provides the following directives,

`.. java:type:: type-signature`

Describe a Java type. The signature can represent either a class, interface, enum or annotation declaration.

Use the `param` field to document type parameters.

Example,

```
.. java:type:: public interface List<E> extends Collection<E>, Iterable<E>

   An ordered collection (also known as a *sequence*)

   :param E: type of item stored by the list
```

produces,

public interface **`List`**<E> extends Collection<E>, Iterable<E>

An ordered collection (also known as a *sequence*)

Parameters

- **`E`** – type of item stored by the list

`.. java:field:: field-signature`

Describe a Java field.

`.. java:method:: method-signature`

Describe a Java method.

Use the `param` field to document parameters.

Use the `throws` field to document exceptions thrown by the method.

Use the `return` field to document the return type

.. java:constructor:: `constructor-signature`

Describe a Java constructor.

Use the `param` field to document parameters.

Use the `throws` field to document exceptions thrown by the constructor.

.. java:package:: `package`

Provide package-level documentation and also sets the active package for the type, method, field, constructors, and references that follow.

Use the `:noindex:` option if the directive is only being used to specify the active package. Only one directive for a given package should exclude `:noindex:`.

.. java:import:: `package type`

Declare the given type as being provided by the given package. This information helps javasphinx create cross references for types in type, method, and field declarations. It also allows explicit cross references (using the `java:ref` role) to exclude the package qualification.

The method, construct, field, and type directives all accept the following standard options,

package

Specify the package the declaration is within. Can be used instead of, or to override, a `java:package` directive.

outertype

Specify the class/interface the documented object is contained within. This option should be provided for any constructor, method, or field directive that isn't nested within a corresponding type directive.

4.2 Roles

The following roles are provided,

:java:ref:

This role can be used to create a cross reference to any object type within the Java domain. Aliases for this role include `java:meth`, `java:type`, `java:field`, `java:package`, and `java:construct`.

An explicit title can be provided by using the standard `title <reference>` syntax.

:java:extdoc:

This role can be used to explicitly link to an externally documented type. The reference must be fully qualified and supports an explicit title using the `title <reference>` syntax.

The `java:ref` role will also create external references as a fall-back if it can't find a matching local declaration so using this role is not strictly necessary.

CHAPTER 5

javasphinx-apidoc

The `javasphinx-apidoc` tool is the counterpoint to the `sphinx-apidoc` tool within the Java domain. It can be used to generate reST source from existing Java source code which has been marked up with Javadoc-style comments. The generated reST is then processed alongside hand-written documentation by Sphinx.

At minimum a source and destination directory must be provided. The input directory will be scanned for .java files and documentation will be generated for all non-private types and members. A separate output file will be generated for each type (including inner classes). Each file is put within a directory corresponding to its package (with periods replaced by directory separators) and with the basename of the file deriving from the type name. Inner types are placed in files with a basename using a hyphen to separate inner and outer types, e.g. `OuterType-InnerType.rst`.

By default `javasphinx-apidoc` will not override existing files. Two options can change this behavior,

`-f, --force`
: All existing output files will be rewritten. If a cache directory is specified it will be rebuilt.

`-u, --update`
: Updated source files will have their corresponding output files updated. Unchanged files will be left alone. Most projects will want to use this option.

For larger projects it is recommended to use a cache directory. This can speed up subsequent runs by an order of magnitude or more. Specify a directory to store cached output using the *`-c`* option,

`-c, --cache-dir`
: Specify a directory to cache intermediate documentation representations. This directory will be created if it does not already exist.

Index