JDiff — What Really Changed?

Comparing Java APIs

WRITTEN BY
MATTHEW B. DOAR

One of the most common questions Java developers ask after downloading a new version of a product is: "What really changed?" JDiff (www.jdiff.org) is an open source Java tool, based on Javadoc and developed by the author, that produces HTML documentation describing the precise API changes between two versions of a product.

This article uses JDiff to show what changed between J2SE 1.3 and J2SE 1.4, and describes how developers can use JDiff to document the changes between two versions of their own products as easily as running Javadoc.

What Changed Between J2SE 1.3 and J2SE 1.4?

Release notes are usually high-level descriptions of feature changes. Product reference manuals tend to be large. And it's hard to compare different product versions in Web browsers' windows. Sun's J2SE 1.4 product, with all its new features, is no exception. When you want to know exactly what changed between two versions of a file, "diff" is the familiar command-line tool for the job. When you want a precise comparison between two Java APIs, I suggest using JDiff.

Figures 1 and 2 show typical HTML documentation generated by JDiff. In this case, the J2SE 1.3.0 API and the J2SE 1.4.0 API are compared using JDiff 1.0.6. Figure 1 is a screenshot of some of the packages that were changed between versions, and Figure 2 shows the details for a particular class, java.lang.Throwable. Every change in the API is reported, from new methods and fields to changes in parameter types and which exceptions are thrown. Even the changes in the documentation for each class and method can be reported.

The best way to view a JDiff report is in a Web browser. (The report comparing J2SE 1.3 and J2SE 1.4 can be found at www.jdiff.org.)

What a JDiff Report Tells You

The HTML report generated by JDiff describes the differences between two Java APIs. The right-hand frame initially contains a summary of the packages that were removed, added, or otherwise changed in some way. There are links to other JDiff-generated pages that describe the changes for each package and class in more detail. To help recall what each package and class is used for, the first sentence of the Javadoc comment (a "documentation block") in the source code is shown to the right of each entry.

The layout of the report resembles Javadoc-generated HTML; to prevent confusion, JDiff uses a different colored background and all links from JDiff pages to Javadoc HTML pages are in a monospaced font.

Indexes

A good question whenever an API changes is: "What was removed?" This is because removed (and changed) constructors, methods, and fields will cause an application to fail, ideally at compile time. Constructors, methods, and fields that were newly added are less likely to cause an application to fail. JDiff provides indexes of which packages, classes, constructors, methods, and fields were removed, added, or changed. It also provides indexes of all the removals, additions, and changes. The indexes are all HTML links, but they're not underlined so they're easier to read quickly.

The indexes are a particularly useful feature when JDiff is used to track changes in an API as a product is being developed. Each part of the team can see precisely what has changed between the different versions.

Links

One feature that makes JDiff-generated reports useful is the large number of HTML links in a report. Every JDiff package and class page has links to the Javadoc-generated HTML pages for the specific package or class, making it easy to refer to an API's existing documentation. The JDiff navigation bar contains links to the page to each class's package, and also to the top-level summary page. Just like Javadoc, there are also links to the previous and next package or class, and to the sections within a page. All pages have links to nonframe versions of the page for browsers (and users) that can't deal with HTML frames.

Features for Developers

Two useful features for API developers are tracking changes in documentation and statistics about the changes between two APIs. This information is generally less useful to customers who use the API, so the features are optional in all JDiff reports.

Documentation Changes

JDiff can track changes in the Javadoc comments in the source code that's used to produce Javadoc HTML. While such changes are rarely of great interest to customers, it's very helpful for developers to know how the description of a method or field has changed during the development of an API. Each changed constructor, method, and field has links to the old documentation, the new documentation, and the highlighted differences between the two. Figure 3 shows some of the documentation changes between J2SE 1.3 and J2SE 1.4.

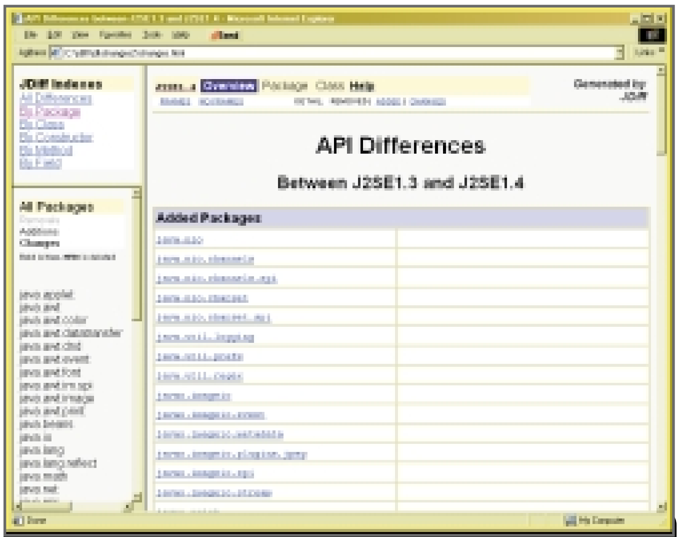
Statistics

Another good question whenever an API changes is: "How much has changed?" To answer this, JDiff can track the statistics for how many constructors, methods, and fields changed in a class, how many classes changed in a package, and so on. The formula is very simple:

```
Percentage Change = 100 x (Number of
Additions + Number of Removals + (2 x
Number of Changes))/ Total Number of
Packages or Classes in Both APIs
```

For example, suppose a Java API is made up of 15 Java packages, and in the

?

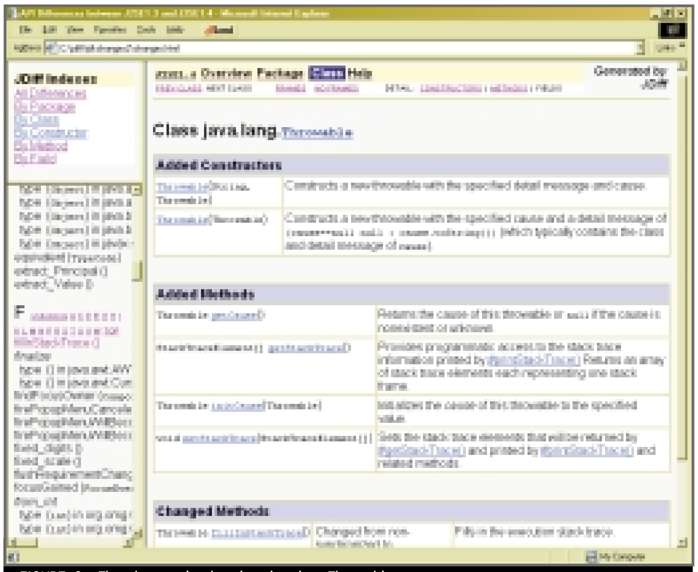


API Differences

Between J2SE1.3 and J2SE1.4

Added Packages

FIGURE 1 JDiff Report showing package changes



Class java.lang.Throwable

Added Constructors

Added Methods

Changed Methods

FIGURE 2 The changes in the class java.lang.Throwable

next release of the API two new packages are added and one existing package is removed so that there are now 16 packages total. Also suppose that 3 of the 16 existing packages have been changed. In this example, the percentage change between the two APIs is 100 x (2 + 1 + (2 x 3))/(15 + 16) = 29%. Using this formula, if two APIs are identical, the percentage change will be 0%, and if they're totally different, the percentage change between them will be 100%.

JDiff reports the percentage changes in each changed class, and also in each changed package, by applying the formula recursively. The percentage changes are also shown sorted in HTML tables in the report, and also in a format suitable for importing to popular spreadsheet applications. This makes it easy to identify when testing and documenting which parts of an API have changed most between different versions.

Table 1 shows the percentage changes for some popular APIs, not including documentation changes. Interestingly, the percentage change between J2SE 1.2 and J2SE 1.3.0 was about 11%, but was about 33% between J2SE 1.3.0 and J2SE 1.4.0, confirming opinions that the changes from J2SE 1.3 to J2SE 1.4 are larger than the changes in the previous major release. The breakdown of the statistics for the core Java classes show that Sun is very careful to add or change only packages and classes, and that minor releases really do contain only bug fixes, as opposed to API changes.
different APIs

How to Run JDiff on Your Own APIs

As shown in Figure 4, there are three fairly straightforward steps for using JDiff. Each step involves running Javadoc, and can be executed at the command line in a script or batch file, or added to a makefile or ANT build file as part of a build process.

Step 1: Use JDiff to generate an XML file that represents the old API's packages.

```
javadoc -doclet jdiff.JDiff
 -docletpath ..\..\lib\jdiff.jar
 -apiname "SuperProduct 1.0"
 -sourcepath ..\SuperProduct1.0 <old
packages>
```

This step scans the source code of the old API. The -doclet and -docletpath options are the standard options used by Javadoc to run the JDiff doclet. The -apiname option creates a unique identifier for the API, and the -sourcepath option indicates where to find the Java packages that make up the old API. <old packages> lists the precise packages that are scanned, just like Javadoc.

Step 2: Use JDiff to generate an XML file that represents the new API's packages.

```
javadoc -doclet jdiff.JDiff
 -docletpath ..\..\lib\jdiff.jar
 -apiname "SuperProduct 2.0"
 -sourcepath ..\SuperProduct2.0 <new
packages>
```

This step scans the source code of the new API, located in the "SuperProduct2.0" directory. The new API is given the unique identifier of "SuperProduct 2.0".

Step 3: Use JDiff to compare the contents of the two XML files and generate an HTML report of the differences.

```
javadoc -doclet jdiff.JDiff
 -docletpath ..\..\lib\jdiff.jar
 -d newdocs -stats
 -oldapi "SuperProduct 1.0"
 -newapi "SuperProduct 2.0"
 -javadocold "../../olddocs/"
 -javadocnew "../../newdocs/"
..\..\lib\Null.java
```

The final step compares the "SuperProduct 1.0" API and the "SuperProduct 2.0" API, with links to the Javadoc documentation in the olddocs and newdocs directories, respectively. The -d option makes the HTML report generated by JDiff appear in the directory newdocs\changes.html, and the -stats option reports statistics about the differences between the APIs. The file Null.java is present only because Javadoc has to read in at least one file, even if the doclet doesn't use it.

?

OLD API	NEW API	PERCENTAGE CHANGE
J2SE 1.2	J2SE 1.3.0	11%
J2SE 1.3.0	J2SE 1.4	33%
J2SE 1.3.0	J2SE 1.3.1	<1%
EJB 1.1	EJB 2.0	37%
Servlet 2.2	Servlet 2.3	49%

TABLE 1 Percentage changes between different APIs

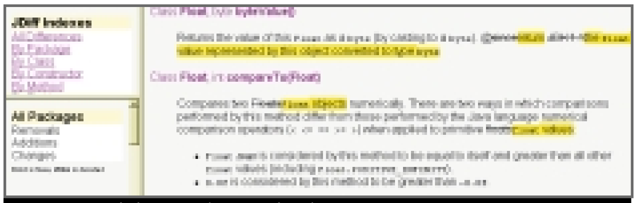


FIGURE 3 Typical J2SE1.4 documentation changes

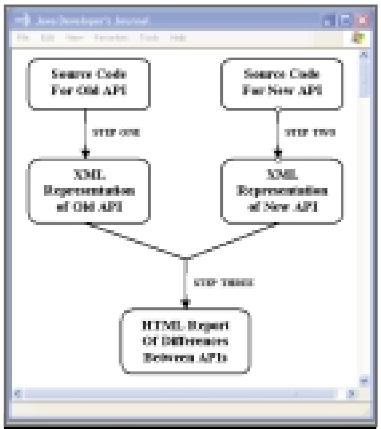

Java Developer's Journal
Source Code For Old API
Source Code For New API
STEP ONE
STEP TWO
XML Representation of Old API
XML Representation of New API
STEP THREE
HTML Report Of Differences Between APIs

FIGURE 4 Three steps for using JDiff

To make the JDiff report more useful, it helps if there's existing Javadoc HTML documentation for both APIs for HTML links from the report. It also helps if there are Javadoc comment blocks in the source code for the packages from each API, since they're used in the right-hand summary text in each entry in the report; however, you don't need them to compare APIs. You can even compare the APIs in two JAR files, where documentation blocks are not available.

JDiff also lets you write specific comments for each change between two APIs. You write a comment for each change into a "comments.xml" file with any text editor, and this file is read in when the report is generated. The comments file is regenerated after the report is finished so the comments are not lost after being incorporated into the report's HTML files. If no comments are provided, by default JDiff does its best to find appropriate comments for each change from the Javadoc comment blocks in the source code.

Scaling Issues

When scanning large APIs such as J2SE 1.4, which has 129 different packages in it, Steps 1 and 2 can take a few minutes to run on a 450MHz, 256MB Pentium III machine. Once the XML files have been generated in Steps 1 and 2 though, they don't need to be regenerated each time a JDiff report is created. Step 3 can be repeated with different options each time, using the same XML files from Steps 1 and 2. In Step 3 it took about three minutes to generate a JDiff report for the two J2SE APIs running on the same 450MHz, 256MB Pentium III machine.

With large APIs, the XML files generated in Steps 1 or 2 can be quite large – about 20MB in the case of J2SE 1.4, if all the documentation is included for comparison. These files can be archived with each release to avoid having to regenerate them later on. They can also be made smaller if changes in documentation are not tracked, since much of the content of each XML file is the entire documentation from each API's Javadoc comment block. The –firstsentence option can be used in Steps 1 and 2 to minimize the size of the XML files by storing only the first sentence of each Javadoc comment block in the XML file. The -docchanges option can be used in Step 3 to avoid tracking changes in documentation, which reduces both the size of the report and the report's index files.

How JDiff Works

JDiff uses the Javadoc doclet API (see the Javadoc homepage), which gives doclet developers a ready-made, easy-to-use tree structure of all the Java packages and classes in the files scanned by Javadoc. Doclets have been used to generate MIF and RTF documentation of APIs, to create customized Javadoc tags such as @todo, and even to generate source code for other applications using tags in the Javadoc comment blocks (see articles referenced at www.doclet.com).

The JDiff doclet has two modes of operation. In the first mode (Steps 1 and 2), it acts as an XML-generating doclet, which traverses all the known packages and classes and writes as much information about them as it can into an XML file. The XML file now represents everything the Javadoc knew about the scanned API. JDiff can also generate an XML Schema file (api.xsd) that describes the XML file and permits XML parsers to validate the XML file later on.

The second mode of operation (Step 3) takes two such XML files as input to an XML parser, such as the Xerces XML parser, and carefully compares them, populating an instance of the JDiff APIDiff.java class as it does so. The results of the comparison are then used by a number of JDiff classes to generate the HTML output. A summary page of all the packages that were removed, added, or changed is created, with links to pages for each package and class. Index pages are also generated for all the differences. Finally, statistics pages and other optional HTML pages are generated. All generated files except the top-level summary are in a single directory named "changes," which makes shipping the HTML files very easy.

The Benefits of Using JDiff

- JDiff is based on the standard Javadoc tool, and is just about as simple to use.
- Developers and documentation teams can produce release documentation describing precisely what has changed in each version of their product. Knowing what has changed between versions of a product leads to faster acceptance of new versions, and fewer frustrated customer calls when a product changes.
- Developers who work in different locations and time zones can use JDiff to summarize the changes in APIs and documentation blocks as the APIs change during development.
- QA and testing organizations can use JDiff to help identify which parts of an API have changed most between versions, indicating which areas need the most testing.

Resources

- *JDiff:* www.jdiff.org,
- *Project hosted by SourceForge:* http://javadiff.sourceforge.net
- *Javadoc Tool:* http://java.sun.com/j2se/javadoc/index.html
- *Writing your own doclet:* http://java.sun.com/j2se/1.4/docs/tooldocs/javadoc/overview.html
- *Third-party doclets:* http://java.sun.com/j2se/javadoc/faq.html#doclets
- *Doclet:* www.doclet.com

doar@pobox.com

AUTHOR BIO

Matthew Doar, a software developer at Vitria Technology, Inc., has worked with Java since the early days of JDK1.1. He wrote JDiff with the intention that it should be as easy to use as Javadoc, and to improve the overall level of documentation shipped with all Java products (www.pobox.com/~doar). Matthew holds a PhD in computer science from the University of Cambridge.