Oracle Blog

Joseph D. Darcy's Oracle Weblog

Joseph D. Darcy's Oracle Weblog

« Coming in 2008: Tips... | Main | OpenJDK 6: Sources... »

How to cross-compile for older platform versions

By darcy on Feb 25, 2008

Besides compiling source code into class files suitable for the current JDK, `javac` can also be used as a cross-compiler to produce class files runnable on JDKs implementing an earlier version of the Java SE specification. However, just using

```
javac1.6 -source 1.4 Foo.java
```

is in general *not* sufficient to ensure that the resulting class file(s) will be usable in a 1.4 JDK. While this will work for many programs, benign evolution of the libraries and platform can cause failures if the program is run on the older JDK. As described in the cross-compilation example in the `javac` man page, the bootclasspath needs to be set to an appropriate library version too.

Targeting the right version of a library is important because libraries are not forward-compatible. That is, older versions of a library don't have facilities to handle calls to methods added in future versions of the library.

One fundamental job of `javac` is to translate symbolic name-based method and constructor calls in the source code to signature-based calls in the class file. Consider two versions of a library:

```
// Original
public class Library {
  public void foo(double d) {System.out.prinln("double foo");}
}

// Evolved
public class Library {
  public void foo(double d) {System.out.prinln("double foo");}
```

```
  public void foo(int i)    {System.out.prinln("int foo");}
}
```

and a client program

```
public class Client {
  public static void main(String... args) {
     (new Library()).foo(1);
  }
}
```

When processing a call site like "`.foo(1)`", `javac` goes through a nontrivial *method resolution* procedure. When `Client` is compiled against the original library, the method call resolves to `foo` taking a `double` parameter, `Library.foo(double)`, which is the only choice in this case. This in turn is represented in the class file as "invoke the method in class Library named foo with one double parameter that returns void." However, after the `foo` method is overloaded, the call "`.foo(1)`" will get resolved as `Library.foo(int)` since it is more specific than the other applicable choice and the class file will instead "invoke the method in class Library named foo with one **int** parameter that returns void" Therefore, regardless of the `-source` or `-target` options given to `javac`, the class file of `Client` compiled against the new library will reference a method specific to the new version of the library. When such a class file is then run against the old version of the library, a `NoSuchMethodError` will result since `Library.foo(int)` is not present.

Library writers should be cautious about overloading in general and especially wary of adding new overloadings. Besides this cross-compilation wrinkle, which has occurred in practice, new overloadings can alter the operational semantics of existing source code, as seen in this case where the printed message is changed.

The `@since` tags that appear in the platform javadoc are *not* used by `javac` to constrain the set of methods considered available when setting `-target`. These tags are only informative and are not represented in the compiled class files on the implicit or explicit bootclasspath. While it would be technically possible to store this information in the class files, for example as annotations or class file attributes, method overloading is already very complicated and this extra complexity would increase fragility of the compiler for little benefit.

Additionally, even when the bootclasspath and `-source`/`-target` are all set appropriately for cross-compilation, compiler-internal contracts, such as how anonymous inner classes are compiled, may differ between, say, `javac` in JDK 1.4.2 and `javac` in JDK 6 running with the `-target 1.4` option.

The most reliably way to produce class files that will work on a particular JDK and later is to compile the source files using the oldest JDK of interest. Barring that, the bootclasspath *must* be set for robust cross-compilation to an older JDK.

Category: Java

Tags: java

Permanent link to this entry

« Coming in 2008: Tips... | Main | OpenJDK 6: Sources... »
Comments:

I remember the same issue when StringBuffer.append(StringBuffer) was introduced in 1.4.

"Method overloading is already very complicated",
i agree, the introduction of varargs and autoboxing complexify the algorithm but
introduce @Since at runtime will only reduce the set of applicable methods (section 15.12.2.1 of the JLS3), it seems straightforward.

Rémi

Posted by **Rémi Forax** on February 28, 2008 at 06:45 AM PST #

Rémi,

Yes, I also closed bug 6578661 "StringBuffer.insert() compiles with CharSequence with 1.4 target." Many seemingly simple compilation changes can be surprising involved. For example, I'd anticipate complications from things like adding an interface to the platform in release N and declaring an existing class implements the interface in release (N+1) when the methods already exist; if compiling to release N, the class shouldn't implement the interface, etc. Another complicating factor is that the @since tags have been known to be incorrect from time to time (6367207 "Ensure correctness of all @since tags added since 1.2").

-Joe

Posted by **Joe Darcy** on February 28, 2008 at 04:28 PM PST #

Post a Comment:
Comments are closed for this entry.

About

darcy

Search

Enter search term:

Search only this blog

Recent Posts

- An apt ending draws nigh
- Project Coin at Devoxx 2011
- Project Coin at JavaOne 2011
- JDK 7 Changesets Over Time
- Coming soon: JavaOne 2011
- JDK 7: Small Library Treats
- A Pictorial View of a JSR Progressing through the JCP
- OSCON: The State of JDK OpenJDK
- OSCON: JDK 7 in a Nutshell
- Project Coin: JSR 334 Finale!

Top Tags

- annotationprocessing
- devoxx
- ee380
- fosdem
- fridayfun
- java
- javaone
- jck
- jdk
- jdk7
- jdk8
- jpr09
- jsr334
- jvmlang
- numerics
- openjdk
- openjdk6
- opensource
- oscon
- personal
- projectcoin
- projectlambda

Categories

- Annotation Processing
- General
- Java
- JavaOne
- Numerics
- OpenJDK

Archives

« January 2012

Sun	Mon	Tue	Wed	Thu	Fri	Sat
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

Today

News

No bookmarks in folder

Blogroll

- 

Menu

- Blogs Home
- Weblog
- Login

Feeds

RSS

- All
- /Annotation Processing
- /General
- /Java
- /JavaOne
- /Numerics
- /OpenJDK
- Comments

Atom

- All
- /Annotation Processing
- /General
- /Java
- /JavaOne
- /Numerics
- /OpenJDK
- Comments