Item 87
git_comments:

git_commits:

1. **summary:** Merged revision(s) 1618148 from lucene/dev/trunk:
   message: Merged revision(s) 1618148 from lucene/dev/trunk: LUCENE-5887: Remove WeakIdentityMap caching in AttributeFactory, AttributeSource, and VirtualMethod in favour of Java 7's ClassValue. Always use MethodHandles to create AttributeImpl classes. git-svn-id: https://svn.apache.org/repos/asf/lucene/dev/branches/branch_4x@1618149 13f79535-47bb-0310-9956-ffa450edef68

github_issues:

github_issues_comments:

github_pulls:

github_pulls_comments:

github_pulls_reviews:

jira_issues:

1. **summary:** Remove horrible WeakIdentityMap caching in AttributeFactory, AttributeSource and VirtualMethod
   description: Especially the use case in AttributeFactory is horrible: Because of ClassLoader issues we cannot hold strong references (see LUCENE-5640 for explanation), we need WeakIdentityMap<Class, WeakReference<someVal>>. You could say: let's use a strong value for stuff like MethodHandles (used in AttributeFactory), but because those have a strong reference to the class, our reference to key would be strong, so garbage collector can no longer unload the class. This is why we use the WeakReference also on the value. The problem is if the value is something like a MethodHandle, which itsself has hard reference to (so it gets garbage collected). Then the cache is useless. In DefaultAttributeFactory I decided, to make methodhandles strong references, but then I needed to restrict it to our own classloader, otherwise we would have strong references to foreign classloaders. Since Java 7 there is java.lang.ClassValue, that fixes the following JVM bug: http://bugs.java.com/bugdatabase/view_bug.do?bug_id=6389107 See also: http://stackoverflow.com/questions/7444420/classvalue-in-java-7 In fact internally, there is a also a WeakReference/WeakHashMap used, but only as fallback - and its only one globally, used by many other JVM internals, too. By default it has a very fast path and the call to ClassValue.get() is incredibly fast. This should therefore also improve AttributeFactory alltogether. Next to AttributeFactory, I also improved the Interfaces cache of AttributeSource (this one assigns an array of Attribute interfaces to an AttributeImpl). The other one is VirtualMethod (assigns its own implementationDistance for every seen subclass). This also removes almost all uses of WeakIdentityMap, the remaining one is the ByteBuffer stuff in MMapDirectory. Unfortunately I have still no idea how to remove that one... :(
   label: code-design
2. **summary:** Remove horrible WeakIdentityMap caching in AttributeFactory, AttributeSource and VirtualMethod
   description: Especially the use case in AttributeFactory is horrible: Because of ClassLoader issues we cannot hold strong references (see LUCENE-5640 for explanation), we need WeakIdentityMap<Class, WeakReference<someVal>>. You could say: let's use a strong value for stuff like MethodHandles (used in AttributeFactory), but because those have a strong reference to the class, our reference to key would be strong, so garbage collector can no longer unload the class. This is why we use the WeakReference also on the value. The problem is if the value is something like a MethodHandle, which itsself has hard reference to (so it gets garbage collected). Then the cache is useless. In DefaultAttributeFactory I decided, to make methodhandles strong references, but then I needed to restrict it to our own classloader, otherwise we would have strong references to foreign classloaders. Since Java 7 there is java.lang.ClassValue, that fixes the following JVM bug: http://bugs.java.com/bugdatabase/view_bug.do?bug_id=6389107 See also: http://stackoverflow.com/questions/7444420/classvalue-in-java-7 In fact

internally, there is a also a WeakReference/WeakHashMap used, but only as fallback - and its only one globally, used by many other JVM internals, too. By default it has a very fast path and the call to ClassValue.get() is incredibly fast. This should therefore also improve AttributeFactory alltogether. Next to AttributeFactory, I also improved the Interfaces cache of AttributeSource (this one assigns an array of Attribute interfaces to an AttributeImpl). The other one is VirtualMethod (assigns its own implementationDistance for every seen subclass). This also removes almost all uses of WeakIdentityMap, the remaining one is the ByteBuffer stuff in MMapDirectory. Unfortunately I have still no idea how to remove that one... :(
label: code-design

3. **summary:** Remove horrible WeakIdentityMap caching in AttributeFactory, AttributeSource and VirtualMethod
   description: Especially the use case in AttributeFactory is horrible: Because of ClassLoader issues we cannot hold strong references (see LUCENE-5640 for explanation), we need WeakIdentityMap<Class, WeakReference<someVal>>. You could say: let's use a strong value for stuff like MethodHandles (used in AttributeFactory), but because those have a strong reference to the class, our reference to key would be strong, so garbage collector can no longer unload the class. This is why we use the WeakReference also on the value. The problem is if the value is something like a MethodHandle, which itsself has hard reference to (so it gets garbage collected). Then the cache is useless. In DefaultAttributeFactory I decided, to make methodhandles strong references, but then I needed to restrict it to our own classloader, otherwise we would have strong references to foreign classloaders. Since Java 7 there is java.lang.ClassValue, that fixes the following JVM bug: http://bugs.java.com/bugdatabase/view_bug.do?bug_id=6389107 See also: http://stackoverflow.com/questions/7444420/classvalue-in-java-7 In fact internally, there is a also a WeakReference/WeakHashMap used, but only as fallback - and its only one globally, used by many other JVM internals, too. By default it has a very fast path and the call to ClassValue.get() is incredibly fast. This should therefore also improve AttributeFactory alltogether. Next to AttributeFactory, I also improved the Interfaces cache of AttributeSource (this one assigns an array of Attribute interfaces to an AttributeImpl). The other one is VirtualMethod (assigns its own implementationDistance for every seen subclass). This also removes almost all uses of WeakIdentityMap, the remaining one is the ByteBuffer stuff in MMapDirectory. Unfortunately I have still no idea how to remove that one... :(
4. **summary:** Remove horrible WeakIdentityMap caching in AttributeFactory, AttributeSource and VirtualMethod
   description: Especially the use case in AttributeFactory is horrible: Because of ClassLoader issues we cannot hold strong references (see LUCENE-5640 for explanation), we need WeakIdentityMap<Class, WeakReference<someVal>>. You could say: let's use a strong value for stuff like MethodHandles (used in AttributeFactory), but because those have a strong reference to the class, our reference to key would be strong, so garbage collector can no longer unload the class. This is why we use the WeakReference also on the value. The problem is if the value is something like a MethodHandle, which itsself has hard reference to (so it gets garbage collected). Then the cache is useless. In DefaultAttributeFactory I decided, to make methodhandles strong references, but then I needed to restrict it to our own classloader, otherwise we would have strong references to foreign classloaders. Since Java 7 there is java.lang.ClassValue, that fixes the following JVM bug: http://bugs.java.com/bugdatabase/view_bug.do?bug_id=6389107 See also: http://stackoverflow.com/questions/7444420/classvalue-in-java-7 In fact internally, there is a also a WeakReference/WeakHashMap used, but only as fallback - and its only one globally, used by many other JVM internals, too. By default it has a very fast path and the call to ClassValue.get() is incredibly fast. This should therefore also improve AttributeFactory alltogether. Next to AttributeFactory, I also improved the Interfaces cache of AttributeSource (this one assigns an array of Attribute interfaces to an AttributeImpl). The other one is VirtualMethod (assigns its own implementationDistance for every seen subclass). This also removes almost all uses of WeakIdentityMap, the remaining one is the ByteBuffer stuff in MMapDirectory. Unfortunately I have still no idea how to remove that one... :(
5. **summary:** Remove horrible WeakIdentityMap caching in AttributeFactory, AttributeSource and VirtualMethod
   description: Especially the use case in AttributeFactory is horrible: Because of ClassLoader issues we cannot hold strong references (see LUCENE-5640 for explanation), we need WeakIdentityMap<Class, WeakReference<someVal>>. You could say: let's use a strong value for stuff like MethodHandles (used in AttributeFactory), but because those have a strong reference to the class, our reference to key would be strong, so garbage collector can no longer unload the class. This is why we use the WeakReference also on the value. The problem is if the value is something like a MethodHandle, which itsself has hard

reference to (so it gets garbage collected). Then the cache is useless. In DefaultAttributeFactory I decided, to make methodhandles strong references, but then I needed to restrict it to our own classloader, otherwise we would have strong references to foreign classloaders. Since Java 7 there is java.lang.ClassValue, that fixes the following JVM bug: http://bugs.java.com/bugdatabase/view_bug.do?bug_id=6389107 See also: http://stackoverflow.com/questions/7444420/classvalue-in-java-7 In fact internally, there is a also a WeakReference/WeakHashMap used, but only as fallback - and its only one globally, used by many other JVM internals, too. By default it has a very fast path and the call to ClassValue.get() is incredibly fast. This should therefore also improve AttributeFactory alltogether. Next to AttributeFactory, I also improved the Interfaces cache of AttributeSource (this one assigns an array of Attribute interfaces to an AttributeImpl). The other one is VirtualMethod (assigns its own implementationDistance for every seen subclass). This also removes almost all uses of WeakIdentityMap, the remaining one is the ByteBuffer stuff in MMapDirectory. Unfortunately I have still no idea how to remove that one... :(
label: code-design

6. **summary:** Remove horrible WeakIdentityMap caching in AttributeFactory, AttributeSource and VirtualMethod
   description: Especially the use case in AttributeFactory is horrible: Because of ClassLoader issues we cannot hold strong references (see LUCENE-5640 for explanation), we need WeakIdentityMap<Class, WeakReference<someVal>>. You could say: let's use a strong value for stuff like MethodHandles (used in AttributeFactory), but because those have a strong reference to the class, our reference to key would be strong, so garbage collector can no longer unload the class. This is why we use the WeakReference also on the value. The problem is if the value is something like a MethodHandle, which itsself has hard reference to (so it gets garbage collected). Then the cache is useless. In DefaultAttributeFactory I decided, to make methodhandles strong references, but then I needed to restrict it to our own classloader, otherwise we would have strong references to foreign classloaders. Since Java 7 there is java.lang.ClassValue, that fixes the following JVM bug: http://bugs.java.com/bugdatabase/view_bug.do?bug_id=6389107 See also: http://stackoverflow.com/questions/7444420/classvalue-in-java-7 In fact internally, there is a also a WeakReference/WeakHashMap used, but only as fallback - and its only one globally, used by many other JVM internals, too. By default it has a very fast path and the call to ClassValue.get() is incredibly fast. This should therefore also improve AttributeFactory alltogether. Next to AttributeFactory, I also improved the Interfaces cache of AttributeSource (this one assigns an array of Attribute interfaces to an AttributeImpl). The other one is VirtualMethod (assigns its own implementationDistance for every seen subclass). This also removes almost all uses of WeakIdentityMap, the remaining one is the ByteBuffer stuff in MMapDirectory. Unfortunately I have still no idea how to remove that one... :(

jira_issues_comments:

1. **body:** Patch removing most of the code... Very nice cleanup!
   label: code-design
2. **body:** New patch with some cleanups (renames of fields,...). Also made sure that the collected interfaces for a given AttributeImpl dont have duplicae interfaces. This was not guaranteed before, but was possible, if superclass defined same interfaces as class. This happened with the Token class... Will commit this now.
   label: code-design
3. Commit 1618148 from [~thetaphi] in branch 'dev/trunk' [https://svn.apache.org/r1618148] LUCENE-5887: Remove WeakIdentityMap caching in AttributeFactory, AttributeSource, and VirtualMethod in favour of Java 7's ClassValue. Always use MethodHandles to create AttributeImpl classes.
4. Commit 1618149 from [~thetaphi] in branch 'dev/branches/branch_4x' [https://svn.apache.org/r1618149] Merged revision(s) 1618148 from lucene/dev/trunk: LUCENE-5887: Remove WeakIdentityMap caching in AttributeFactory, AttributeSource, and VirtualMethod in favour of Java 7's ClassValue. Always use MethodHandles to create AttributeImpl classes.