mjdecker / **survey_tools**

Sample 10 modified code.

master

Browse files

mjdecker committed 8 hours ago   1 parent 4378bad   commit 6e80e8be3271e1a827aaba305d8015c0802aeb28

Showing **1 changed file** with **44 additions** and **36 deletions**.   Unified Split

80 🟩🟩🟥🟥 tool1/sample10.java   View

```
...  @@ -1,25 +1,27 @@
```

```
 package com.google.common.cache;

 import com.google.common.base.Function;
 import com.google.common.base.Objects;
 import com.google.common.base.Optional;
 import com.google.common.base.Preconditions;
 import com.google.common.cache.LocalCache.Strength;
 import com.google.common.collect.Iterables;
 import com.google.common.collect.Lists;
 import com.google.common.collect.Sets;

 import java.util.List;
 import java.util.Set;
 import java.util.concurrent.TimeUnit;

 import javax.annotation.Nullable;

 class CacheBuilderFactory {
    private Set<Integer> concurrencyLevels =
Sets.newHashSet((Integer) null);
    private Set<Integer> initialCapacities =
Sets.newHashSet((Integer) null);
    private Set<Integer> maximumSizes =
Sets.newHashSet((Integer) null);
-   private Set<ExpirationSpec> expirations =
Sets.newHashSet((ExpirationSpec) null);
    private Set<Strength> keyStrengths =
Sets.newHashSet((Strength) null);
    private Set<Strength> valueStrengths =
Sets.newHashSet((Strength) null);

    CacheBuilderFactory withConcurrencyLevels(Set<Integer>
concurrencyLevels) {
      this.concurrencyLevels =
Sets.newLinkedHashSet(concurrencyLevels);
      return this;
    }

    CacheBuilderFactory withInitialCapacities(Set<Integer>
initialCapacities) {
      this.initialCapacities =
Sets.newLinkedHashSet(initialCapacities);
      return this;
    }

    CacheBuilderFactory withMaximumSizes(Set<Integer>
```

```
 package com.google.common.cache;

 import com.google.common.base.Function;
 import com.google.common.base.Objects;
 import com.google.common.base.Optional;
 import com.google.common.base.Preconditions;
 import com.google.common.cache.LocalCache.Strength;
 import com.google.common.collect.Iterables;
 import com.google.common.collect.Lists;
 import com.google.common.collect.Sets;

 import java.util.List;
 import java.util.Set;
 import java.util.concurrent.TimeUnit;

 import javax.annotation.Nullable;

 class CacheBuilderFactory {
    private Set<Integer> concurrencyLevels =
Sets.newHashSet((Integer) null);
    private Set<Integer> initialCapacities =
Sets.newHashSet((Integer) null);
    private Set<Integer> maximumSizes =
Sets.newHashSet((Integer) null);
+   private Set<DurationSpec> expireAfterWrites =
Sets.newHashSet((DurationSpec) null);
+   private Set<DurationSpec> expireAfterAccesses =
Sets.newHashSet((DurationSpec) null);
+   private Set<DurationSpec> refreshes =
Sets.newHashSet((DurationSpec) null);
    private Set<Strength> keyStrengths =
Sets.newHashSet((Strength) null);
    private Set<Strength> valueStrengths =
Sets.newHashSet((Strength) null);

    CacheBuilderFactory withConcurrencyLevels(Set<Integer>
concurrencyLevels) {
      this.concurrencyLevels =
Sets.newLinkedHashSet(concurrencyLevels);
      return this;
    }

    CacheBuilderFactory withInitialCapacities(Set<Integer>
initialCapacities) {
      this.initialCapacities =
Sets.newLinkedHashSet(initialCapacities);
      return this;
    }

    CacheBuilderFactory withMaximumSizes(Set<Integer>
maximumSizes) {
    this.maximumSizes = Sets.newLinkedHashSet(maximumSizes);
    return this;
  }

- CacheBuilderFactory withExpirations(Set<ExpirationSpec> expirations) {
-   this.expirations = Sets.newLinkedHashSet(expirations);
    return this;
  }

  CacheBuilderFactory withKeyStrengths(Set<Strength> keyStrengths) {
    this.keyStrengths = Sets.newLinkedHashSet(keyStrengths);
    Preconditions.checkArgument(!this.keyStrengths.contains(Strength.SOFT));
    return this;
  }

  CacheBuilderFactory withValueStrengths(Set<Strength> valueStrengths) {
    this.valueStrengths = Sets.newLinkedHashSet(valueStrengths);
    return this;
  }

  Iterable<CacheBuilder<Object, Object>> buildAllPermutations() {
    @SuppressWarnings("unchecked")
    Iterable<List<Object>> combinations = buildCartesianProduct(concurrencyLevels,
-       initialCapacities, maximumSizes, expirations, keyStrengths, valueStrengths);

    return Iterables.transform(combinations,
        new Function<List<Object>, CacheBuilder<Object, Object>>() {
          @Override public CacheBuilder<Object, Object> apply(List<Object> combination) {
            return createCacheBuilder(
                (Integer) combination.get(0),
                (Integer) combination.get(1),
                (Integer) combination.get(2),
-               (ExpirationSpec) combination.get(3),
-               (Strength) combination.get(4),
-               (Strength) combination.get(5));
          }
        });
  }

  private static final Function<Object, Optional<?>> NULLABLE_TO_OPTIONAL =
      new Function<Object, Optional<?>>() {
        @Override public Optional<?> apply(@Nullable Object obj) {
          return Optional.fromNullable(obj);
```

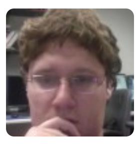
```
maximumSizes) {
    this.maximumSizes = Sets.newLinkedHashSet(maximumSizes);
    return this;
  }

+ CacheBuilderFactory withExpireAfterWrites(Set<DurationSpec> durations) {
+   this.expireAfterWrites = Sets.newLinkedHashSet(durations);
+   return this;
+ }
+
+ CacheBuilderFactory withExpireAfterAccesses(Set<DurationSpec> durations) {
+   this.expireAfterAccesses = Sets.newLinkedHashSet(durations);
+   return this;
+ }
+
+ CacheBuilderFactory withRefreshes(Set<DurationSpec> durations) {
+   this.refreshes = Sets.newLinkedHashSet(durations);
    return this;
  }

  CacheBuilderFactory withKeyStrengths(Set<Strength> keyStrengths) {
    this.keyStrengths = Sets.newLinkedHashSet(keyStrengths);
    Preconditions.checkArgument(!this.keyStrengths.contains(Strength.SOFT));
    return this;
  }

  CacheBuilderFactory withValueStrengths(Set<Strength> valueStrengths) {
    this.valueStrengths = Sets.newLinkedHashSet(valueStrengths);
    return this;
  }

  Iterable<CacheBuilder<Object, Object>> buildAllPermutations() {
    @SuppressWarnings("unchecked")
    Iterable<List<Object>> combinations = buildCartesianProduct(concurrencyLevels,
+       initialCapacities, maximumSizes, expireAfterWrites, expireAfterAccesses, refreshes,
+       keyStrengths, valueStrengths);
    return Iterables.transform(combinations,
        new Function<List<Object>, CacheBuilder<Object, Object>>() {
          @Override public CacheBuilder<Object, Object> apply(List<Object> combination) {
            return createCacheBuilder(
                (Integer) combination.get(0),
                (Integer) combination.get(1),
                (Integer) combination.get(2),
+               (DurationSpec) combination.get(3),
+               (DurationSpec) combination.get(4),
+               (DurationSpec) combination.get(5),
+               (Strength) combination.get(6),
+               (Strength) combination.get(7));
          }
        });
  }

  private static final Function<Object, Optional<?>> NULLABLE_TO_OPTIONAL =
      new Function<Object, Optional<?>>() {
        @Override public Optional<?> apply(@Nullable Object obj) {
          return Optional.fromNullable(obj);
        }
      };

  private static final Function<Optional<?>, Object> OPTIONAL_TO_NULLABLE =
      new Function<Optional<?>, Object>() {
        @Override public Object apply(Optional<?> optional) {
          return optional.orNull();
        }
      };

  private Iterable<List<Object>> buildCartesianProduct(Set<?>... sets) {
    List<Set<Optional<?>>> optionalSets = Lists.newArrayListWithExpectedSize(sets.length);
    for (Set<?> set : sets) {
      Set<Optional<?>> optionalSet =
          Sets.newLinkedHashSet(Iterables.transform(set, NULLABLE_TO_OPTIONAL));
      optionalSets.add(optionalSet);
    }
    Set<List<Optional<?>>> cartesianProduct = Sets.cartesianProduct(optionalSets);
    return Iterables.transform(cartesianProduct,
        new Function<List<Optional<?>>, List<Object>>() {
          @Override public List<Object> apply(List<Optional<?>> objs) {
            return Lists.transform(objs, OPTIONAL_TO_NULLABLE);
          }
        });
  }

  private CacheBuilder<Object, Object> createCacheBuilder(
      Integer concurrencyLevel, Integer initialCapacity, Integer maximumSize,
-      ExpirationSpec expiration, Strength keyStrength, Strength valueStrength) {

    CacheBuilder<Object, Object> builder = CacheBuilder.newBuilder();
    if (concurrencyLevel != null) {
      builder.concurrencyLevel(concurrencyLevel);
    }
    if (initialCapacity != null) {
      builder.initialCapacity(initialCapacity);
    }
    if (maximumSize != null) {
      builder.maximumSize(maximumSize);
    }
-    if (expiration != null) {
-      if (expiration.expireAfterAccessMillis != null) {
-        builder.expireAfterAccess(expiration.expireAfterAccessMillis, TimeUnit.MILLISECONDS);
    }
-      if (expiration.expireAfterWriteMillis != null) {
-        builder.expireAfterWrite(expiration.expireAfterWriteMillis, TimeUnit.MILLISECONDS);
    }
    }
    if (keyStrength != null) {
      builder.setKeyStrength(keyStrength);
    }
    if (valueStrength != null) {
      builder.setValueStrength(valueStrength);
    }
```

```
        }
      };

  private static final Function<Optional<?>, Object> OPTIONAL_TO_NULLABLE =
      new Function<Optional<?>, Object>() {
        @Override public Object apply(Optional<?> optional) {
          return optional.orNull();
        }
      };

  private Iterable<List<Object>> buildCartesianProduct(Set<?>... sets) {
    List<Set<Optional<?>>> optionalSets = Lists.newArrayListWithExpectedSize(sets.length);
    for (Set<?> set : sets) {
      Set<Optional<?>> optionalSet =
          Sets.newLinkedHashSet(Iterables.transform(set, NULLABLE_TO_OPTIONAL));
      optionalSets.add(optionalSet);
    }
    Set<List<Optional<?>>> cartesianProduct = Sets.cartesianProduct(optionalSets);
    return Iterables.transform(cartesianProduct,
        new Function<List<Optional<?>>, List<Object>>() {
          @Override public List<Object> apply(List<Optional<?>> objs) {
            return Lists.transform(objs, OPTIONAL_TO_NULLABLE);
          }
        });
  }

  private CacheBuilder<Object, Object> createCacheBuilder(
      Integer concurrencyLevel, Integer initialCapacity, Integer maximumSize,
+      DurationSpec expireAfterWrite, DurationSpec expireAfterAccess, DurationSpec refresh,
+      Strength keyStrength, Strength valueStrength) {

    CacheBuilder<Object, Object> builder = CacheBuilder.newBuilder();
    if (concurrencyLevel != null) {
      builder.concurrencyLevel(concurrencyLevel);
    }
    if (initialCapacity != null) {
      builder.initialCapacity(initialCapacity);
    }
    if (maximumSize != null) {
      builder.maximumSize(maximumSize);
    }
+    if (expireAfterWrite != null) {
+      builder.expireAfterWrite(expireAfterWrite.duration, expireAfterWrite.unit);
    }
+    if (expireAfterAccess != null) {
+      builder.expireAfterAccess(expireAfterAccess.duration, expireAfterAccess.unit);
    }
+    if (refresh != null) {
+      builder.refreshInterval(refresh.duration, refresh.unit);
    }
    if (keyStrength != null) {
      builder.setKeyStrength(keyStrength);
    }
    if (valueStrength != null) {
      builder.setValueStrength(valueStrength);
    }
```

```diff
     return builder;
   }

-  static class ExpirationSpec {
-    @Nullable
-    private final Long expireAfterAccessMillis;
-    @Nullable
-    private final Long expireAfterWriteMillis;
-
-    private ExpirationSpec(Long expireAfterAccessMillis, Long expireAfterWriteMillis) {
-      Preconditions.checkArgument(
-          expireAfterAccessMillis == null || expireAfterWriteMillis == null);
-      this.expireAfterAccessMillis = expireAfterAccessMillis;
-      this.expireAfterWriteMillis = expireAfterWriteMillis;
-    }
+  static class DurationSpec {
+    private final long duration;
+    private final TimeUnit unit;

-    public static ExpirationSpec afterAccess(long afterAccess, TimeUnit unit) {
-      return new ExpirationSpec(unit.toMillis(afterAccess), null);
+    private DurationSpec(long duration, TimeUnit unit) {
+      this.duration = duration;
+      this.unit = unit;
     }

-    public static ExpirationSpec afterWrite(long afterWrite, TimeUnit unit) {
-      return new ExpirationSpec(null, unit.toMillis(afterWrite));
+    public static DurationSpec of(long duration, TimeUnit unit) {
+      return new DurationSpec(duration, unit);
     }

     @Override
     public int hashCode() {
-      return Objects.hashCode(expireAfterAccessMillis, expireAfterWriteMillis);
+      return Objects.hashCode(duration, unit);
     }

     @Override
     public boolean equals(Object o) {
-      if (o instanceof ExpirationSpec) {
-        ExpirationSpec that = (ExpirationSpec) o;
-        return Objects.equal(this.expireAfterAccessMillis, that.expireAfterAccessMillis)
-            && Objects.equal(this.expireAfterWriteMillis, that.expireAfterWriteMillis);
+      if (o instanceof DurationSpec) {
+        DurationSpec that = (DurationSpec) o;
+        return unit.toNanos(duration) == that.unit.toNanos(that.duration);
       }
       return false;
     }

     @Override
     public String toString() {
       return Objects.toStringHelper(this)
-          .add("expireAfterAccessMillis", expireAfterAccessMillis)
-          .add("expireAfterWriteMillis", expireAfterWriteMillis)
+          .add("duration", duration)
+          .add("unit", unit)
           .toString();
     }
   }
 }
```

0 comments on commit 6e80e8b

Lock conversation



Write   Preview