

REPORT SOFTWARE TESTING

Projects: BookKeeper, Syncope

De Angelis Alessandro
0317176

Github repositories:

BookKeeper: <https://github.com/AlessDea/bookkeeper>

Syncope: <https://github.com/AlessDea/syncope>

BookKeeper

BookKeeperAdmin.java

La scelta di questa classe è dovuta al fatto che dall'analisi delle classi condotta risulta essere una delle più anziane e più 'buggy'. In particolare, dalla release 1 (4.0.0) alla 7 (4.2.3), è stata coinvolta in fix di bug per ben 5 volte.

```
nukeExistingCluster(ServerConfiguration conf, String  
ledgersRootPath, String instanceId, boolean force)
```

Questo metodo 'distrugge' i metadati di un cluster esistente. I paramteri indicano rispettivamente: la configurazione del server Zookeeper, il path della root del ledger, l'id dell'istanza del cluster e un booleano che serve ad indicare se l' `instanceId` debba essere validata o meno.

In particolare, la distruzione dei metadati viene effettuata solamente se `ledgersRootPath` corrisponde con `zkLedgersRootPath` che è mantenuto nella configurazione del server di Zookeeper (`conf`). Inoltre `instanceId` deve corrispondere a quello contenuto nei metadati del cluster ma se il parametro `force` è pari a `true` allora la 'distruzione' dei metadati avviene senza controllare se `instanceId` corrisponde a quello nei metadati del cluster.

Category Partition

```
ServerConfiguration conf: {Invalid, Valid, null}  
String ledgersRootPath: {Invalid, valid, null, empty}  
String instanceId: {Invalid, valid, null, empty}  
boolean force: {true, false}
```

N.B.

Dalla documentazione, un instance id deve essere un oggetto di tipo `immutable universally unique identifier (UUID)` che rappresenta un valore di 128 bit. In particolare è una stringa di 36 caratteri divisa in 5 gruppi separati da trattini, per esempio:

```
"7940c93b-5da8-4fa7-941e-d254d678fb1c"
```

Boundary values

Definisco:

```
valid_ild = "7940c93b-5da8-4fa7-941e-d254d678fb1c"
```

```
inv_ild = "12345"
```

Per creare un oggetto `ServerConfiguration` invalido si può creare uno valido per poi settare un attributo in modo che la configurazione non si valida. Questo può essere fatto per esempio settando a `-1` la porta del bookie: `conf.setBookiePort(-1)`

```
ServerConfiguration conf: {invalid_c, valid_c, null}
String ledgersRootPath: {".../", "/ledgers", null, ""}
String instanceId: {valid_ild, inv_ild, null, ""}
boolean force: {true, false}
```

I parametri che sono legati fra loro sono:

- `conf`, `ledgersRootPath`: poichè viene fatto il controllo se il path della root del ledger corrisponde con quello nella configurazione
- `instanceId`, `force`: poichè in caso di `force` valido allora `instanceId` non viene considerato

Per i primi due si può optare per un approccio unidimensionale ed identificare i seguenti casi limite:

- `{invalid_c, "/ledgers"}` -> permette di verificare che venga effettivamente fatto il controllo con il path specificato nella configurazione e che non ci si basi solo da quello fornito con `ledgerRootPath`
- `{valid_c, ".../"}` -> permette di verificare che un path errato non venga preso per buono o non causi errori non gestiti
- `{null, "/ledgers"}` -> come il primo caso ed inoltre garantisce la buona gestione in caso di configurazione nulla
- `{valid_c, ""}`

Quindi si hanno per `conf` e `ledgersRootPath`: `{invalid_c, "/ledgers"}`, `{valid_c, ".../"}`, `{null, "/ledgers"}`, `{valid_c, ""}`

Per `instanceId` e `force` invece, seguendo sempre un approccio unidimensionale:

- `{valid_ild, true}`
- `{inv_ild, false}` -> verificare la corretta gestione del controllo di un id non valido
- `{null, false}` -> verificare che un id nullo non causi problemi nel controllo
- `{ "", false}` -> verificare che un id vuoto non causi problemi nel controllo

Ricapitolando si hanno: `{valid_ild, true}`, `{inv_ild, false}`, `{null, false}`, `{ "", false}`

A questo punto si può effettuare un approccio multidimensionale fra i due insiemi creati.

```
{false, INVALID, "/ledgers", valid_ild, true},
{false, INVALID, "/ledgers", inv_ild, false},
{false, INVALID, "/ledgers", null, false},
{false, INVALID, "/ledgers", "", false},
{false, VALID, ".../", valid_ild, true},
{false, VALID, ".../", inv_ild, false},
{false, VALID, ".../", null, false},
{false, VALID, ".../", "", false},
```

```

{false, NULL, "/ledgers", valid_ild, true},
{false, NULL, "/ledgers", inv_ild, false},
{false, NULL, "/ledgers", null, false},
{false, NULL, "/ledgers", "", false},
{false, VALID, "", valid_ild, true},
{false, VALID, "", inv_ild, false},
{false, VALID, "", null, false},
{false, VALID, "", "", false},

```

Tutti i test hanno esito corretto.

Jacoco & PIT

```

1308. public static boolean nukeExistingCluster(ServerConfiguration conf, String ledgersRootPath, String instanceId,
1309. boolean force) throws Exception {
1310.     String confLedgersRootPath = ZKMetadataDriverBase.resolveZkLedgersRootPath(conf);
1311.     if (!confLedgersRootPath.equals(ledgersRootPath)) {
1312.         LOG.error("Provided ledgerRootPath : {} is not matching with config's ledgerRootPath: {}, "
1313.             + "so exiting nuke operation", ledgersRootPath, confLedgersRootPath);
1314.         return false;
1315.     }
1316.
1317.     return runFunctionWithRegistrationManager(conf, rm -> {
1318.         try {
1319.             if (!force) {
1320.                 String readInstanceId = rm.getClusterInstanceId();
1321.                 if ((instanceId == null) || !instanceId.equals(readInstanceId)) {
1322.                     LOG.error("Provided InstanceId : {} is not matching with cluster InstanceId in ZK: {}",
1323.                         instanceId, readInstanceId);
1324.                     return false;
1325.                 }
1326.             }
1327.             return rm.nukeExistingCluster();
1328.         } catch (Exception e) {
1329.             throw new UncheckedExecutionException(e.getMessage(), e);
1330.         }
1331.     });
1332. }

```

La coverage di tale metodo è del 63%:

● <code>nukeExistingCluster(ServerConfiguration,String,String,boolean)</code>		63%
---	---	-----

```

1308 public static boolean nukeExistingCluster(ServerConfiguration conf, String ledgersRootPath, String instanceId,
1309 boolean force) throws Exception {
1310     String confLedgersRootPath = ZKMetadataDriverBase.resolveZkLedgersRootPath(conf);
1311 1 if (!confLedgersRootPath.equals(ledgersRootPath)) {
1312     LOG.error("Provided ledgerRootPath : {} is not matching with config's ledgerRootPath: {}, "
1313         + "so exiting nuke operation", ledgersRootPath, confLedgersRootPath);
1314 1 return false;
1315 }
1316
1317 2 return runFunctionWithRegistrationManager(conf, rm -> {
1318     try {
1319 1 if (!force) {
1320         String readInstanceId = rm.getClusterInstanceId();
1321 2 if ((instanceId == null) || !instanceId.equals(readInstanceId)) {
1322             LOG.error("Provided InstanceId : {} is not matching with cluster InstanceId in ZK: {}",
1323                 instanceId, readInstanceId);
1324 1 return false;
1325         }
1326     }
1327 2 return rm.nukeExistingCluster();
1328 } catch (Exception e) {
1329     throw new UncheckedExecutionException(e.getMessage(), e);
1330 }
1331 });
1332 }
1333

```

Ba-dua

```
-<method name="lambda$hukeExistingCluster$4" desc="(ZLjava/lang/String;Lorg/apache/bookkeeper/discover/RegistrationManager;)Ljava/lang/Boolean;">
  <du var="force" def="1319" use="1319" target="1320" covered="1"/>
  <du var="force" def="1319" use="1319" target="1327" covered="1"/>
  <du var="instanceId" def="1319" use="1321" target="1321" covered="0"/>
  <du var="instanceId" def="1319" use="1321" target="1322" covered="0"/>
  <du var="instanceId" def="1319" use="1322" covered="0"/>
  <du var="instanceId" def="1319" use="1321" target="1322" covered="0"/>
  <du var="instanceId" def="1319" use="1321" target="1327" covered="0"/>
  <du var="rm" def="1319" use="1327" covered="1"/>
  <du var="rm" def="1319" use="1320" covered="1"/>
  <du var="LOG" def="1319" use="1322" covered="0"/>
  <du var="readInstanceId" def="1320" use="1322" covered="0"/>
  <du var="readInstanceId" def="1320" use="1321" target="1322" covered="0"/>
  <du var="readInstanceId" def="1320" use="1321" target="1327" covered="0"/>
  <counter type="DU" missed="9" covered="4"/>
  <counter type="METHOD" missed="0" covered="1"/>
</method>
```

format(ServerConfiguration conf, boolean isInteractive, boolean force)

Tale metodo è utilizzato per formattare o inizializzare un insieme di journal e ledger per l'istanza di BookKeeper specificata. Può utilizzato per avviare un'istanza di BookKeeper da zero, rimuovendo tutti i dati precedenti e creando una nuova configurazione iniziale. Prende come parametri una configurazione iniziale del server (`conf`), un booleano che indica se la formattazione deve avvenire in modo interattivo (`isInteractive`) e un booleano che indica se bisogna chiedere conferma o meno prima di rimuovere i vecchi dati (`force`). In particolare se `force` è `true` , e lo è anche `isInteractive` , allora non viene chiesto all'utente di confermare, altrimenti è richiesta la conferma.

Category Partition

```
ServerConfiguration conf: {Invalid, Valid, null}
boolean isInteractive: {true, false}
boolean force: {true, false}
```

Boundary values

Per un `ServerConfiguration` invalido: come prima.

```
ServerConfiguration conf: {invalid_c, valid_c, null}
boolean isInteractive: {true, false}
boolean force: {true, false}
```

I parametri `isInteractive` e `force` sono strettamente legati in quanto se `isInteractive` è pari a `true` allora è richiesta l'interazione con l'utente ma se anche `force` fosse pari a `true` allora questo implicherebbe che bisognerebbe saltare l'interazione con l'utente.

Di conseguenza è stato utilizzato un approccio unidimensionale:

- `{true, true}` -> in questo modo si può verificare se anche se abilitata l'interattività, non venga comunque utilizzata in caso di `force` pari a `true`
- `{true, false}`
- `{false, false}`

I casi in cui `force` è uguale a `true` non è stato testato in realtà poichè non si è riusciti a simulare l'interazione con l'utente.

Viene applicato poi un approccio multidimensionale tra i due insiemi per ottenere i casi di test.

```
{true, true, false, true},
{false, true, false, false},
```

```
{false, false, false, true},  
{false, false, false, false},
```

A causa della mancata possibilità di imitare l'interazione con l'utente non è stato possibile testare i casi in cui `isInteractive` fosse pari a `true`.

Jacoco & PIT

```
1238.     public static boolean format(ServerConfiguration conf,  
1239.         boolean isInteractive, boolean force) throws Exception {  
1240.         return runFunctionWithMetadataBookieDriver(conf, new Function<MetadataBookieDriver, Boolean>() {  
1241.             @Override  
1242.             @SuppressWarnings("RCN_REDUNDANT_NULLCHECK_WOULD_HAVE_BEEN_A_NPE")  
1243.             public Boolean apply(MetadataBookieDriver driver) {  
1244.                 try {  
1245.                     try (RegistrationManager regManager = driver.createRegistrationManager()) {  
1246.                         boolean ledgerRootExists = regManager.prepareFormat();  
1247.                         // If old data was there then confirm with admin.  
1248.                         boolean doFormat = true;  
1249.                         if (ledgerRootExists) {  
1250.                             if (!isInteractive) {  
1251.                                 // If non interactive and force is set, then delete old data.  
1252.                                 doFormat = force;  
1253.                             } else {  
1254.                                 // Confirm with the admin.  
1255.                                 doFormat = IOUtils  
1256.                                     .confirmPrompt("Ledger root already exists. "  
1257.                                         + "Are you sure to format bookkeeper metadata? "  
1258.                                         + "This may cause data loss.");  
1259.                             }  
1260.                         }  
1261.                         if (!doFormat) {  
1262.                             return false;  
1263.                         }  
1264.                         driver.getLedgerManagerFactory().format(  
1265.                             conf,  
1266.                             driver.getLayoutManager());  
1267.                         return regManager.format();  
1268.                     }  
1269.                 } catch (Exception e) {  
1270.                     System.out.println("EXC:" + Arrays.toString(e.getStackTrace()));  
1271.                     throw new UncheckedExecutionException(e.getMessage(), e);  
1272.                 }  
1273.             }  
1274.         });  
1275.     }
```

La coverage di tale metodo è del 100%:

format(ServerConfiguration, boolean, boolean)	100%
---	------

```
1238     public static boolean format(ServerConfiguration conf,  
1239         boolean isInteractive, boolean force) throws Exception {  
1240.         return runFunctionWithMetadataBookieDriver(conf, new Function<MetadataBookieDriver, Boolean>() {  
1241.             @Override  
1242.             @SuppressWarnings("RCN_REDUNDANT_NULLCHECK_WOULD_HAVE_BEEN_A_NPE")  
1243.             public Boolean apply(MetadataBookieDriver driver) {  
1244.                 try {  
1245.                     try (RegistrationManager regManager = driver.createRegistrationManager()) {  
1246.                         boolean ledgerRootExists = regManager.prepareFormat();  
1247.                         // If old data was there then confirm with admin.  
1248.                         boolean doFormat = true;  
1249.                         if (ledgerRootExists) {  
1250.                             if (!isInteractive) {  
1251.                                 // If non interactive and force is set, then delete old data.  
1252.                                 doFormat = force;  
1253.                             } else {  
1254.                                 // Confirm with the admin.  
1255.                                 doFormat = IOUtils  
1256.                                     .confirmPrompt("Ledger root already exists. "  
1257.                                         + "Are you sure to format bookkeeper metadata? "  
1258.                                         + "This may cause data loss.");  
1259.                             }  
1260.                         }  
1261.                         if (!doFormat) {  
1262.                             return false;  
1263.                         }  
1264.                         driver.getLedgerManagerFactory().format(  
1265.                             conf,  
1266.                             driver.getLayoutManager());  
1267.                         return regManager.format();  
1268.                     }  
1269.                 } catch (Exception e) {  
1270.                     System.out.println("EXC:" + Arrays.toString(e.getStackTrace()));  
1271.                     throw new UncheckedExecutionException(e.getMessage(), e);  
1272.                 }  
1273.             }  
1274.         });  
1275.     }
```

A quanto pare PIT non segue le funzioni nidificate.

Ba-dua

```
-<class name="org/apache/bookkeeper/client/BookKeeperAdmin$7">
-<method name="apply" desc="(Lorg/apache/bookkeeper/meta/MetadataBookieDriver;)Ljava/lang/Boolean;">
  <du var="this" def="1245" use="1264" covered="1"/>
  <du var="this" def="1245" use="1250" target="1252" covered="1"/>
  <du var="this" def="1245" use="1250" target="1255" covered="0"/>
  <du var="this" def="1245" use="1252" covered="1"/>
  <du var="driver" def="1245" use="1264" covered="1"/>
  <du var="this.val$isActive" def="1245" use="1250" target="1252" covered="1"/>
  <du var="this.val$isActive" def="1245" use="1250" target="1255" covered="0"/>
  <du var="this.val$force" def="1245" use="1252" covered="1"/>
  <du var="this.val$conf" def="1245" use="1264" covered="1"/>
  <du var="regManager" def="1245" use="1267" covered="1"/>
  <du var="regManager" def="1245" use="1268" target="1268" covered="1"/>
  <du var="regManager" def="1245" use="1268" target="1267" covered="0"/>
  <du var="regManager" def="1245" use="1268" covered="1"/>
  <du var="regManager" def="1245" use="1268" covered="0"/>
  <du var="regManager" def="1245" use="1268" target="1268" covered="1"/>
  <du var="regManager" def="1245" use="1268" target="1262" covered="0"/>
  <du var="regManager" def="1245" use="1268" covered="1"/>
  <du var="regManager" def="1245" use="1268" covered="0"/>
  <du var="ledgerRootExists" def="1246" use="1249" target="1250" covered="1"/>
  <du var="ledgerRootExists" def="1246" use="1249" target="1261" covered="0"/>
  <du var="doFormat" def="1248" use="1261" target="1262" covered="0"/>
  <du var="doFormat" def="1248" use="1261" target="1264" covered="0"/>
  <du var="doFormat" def="1252" use="1261" target="1262" covered="1"/>
  <du var="doFormat" def="1252" use="1261" target="1264" covered="1"/>
  <du var="doFormat" def="1256" use="1261" target="1262" covered="0"/>
  <du var="doFormat" def="1256" use="1261" target="1264" covered="0"/>
  <counter type="DU" missed="13" covered="19"/>
  <counter type="METHOD" missed="0" covered="1"/>
</method>
```


ReadCache.java

Questa classe è stata scelta perchè molto più recente rispetto a quelle analizzate e quindi potrebbe contenere dei bug.

`get(long ledgerId, long entryId)`

Il metodo `get` di `ReadCache` prende come parametri due `long` che identificano l'id del ledger e l'id dell'entry all'interno del **ledger**. Tale metodo restituisce un oggetto di tipo `ByteBuf` nel caso l'entry identificata da `entryId` esista all'interno del ledger specificato da `ledgerId`. In caso negativo ci si aspetta una certa eccezione o il valore `null`.

Category Partition

```
long ledgerId : {≤ 0, > 0}
```

```
long entryId : {≤ 0, > 0}
```

Boundary values

```
long ledgerId : {-1, 0, 1}
```

```
long entryId : {-1, 0, 1}
```

I due parametri non sono strettamente legati fra di loro nel senso che il valriare di uno non influisce sull'altro; di conseguenza non è naturale derivare dei casi di test in modo unidimensionale perchè non si riesce a identificarne alcun caso che sia effettivamente più valido degli altri. Si preferisce quindi usare l'approccio multidimensionale ottenendo così seguenti casi:

```
{false, -1, -1},
```

```
{false, -1, 0},
```

```
{false, -1, 1},
```

```
{false, 0, -1},
```

```
{false, 0, 0},
```

```
{false, 0, 1},
```

```
{false, 1, -1},
```

```
{false, 1, 0},
```

```
{false, 1, 1},
```

```
{true, 123, 0}
```

Jacoco & PIT

```

139.     public ByteBuf get(long ledgerId, long entryId) {
140.         lock.readLock().lock();
141.
142.         try {
143.             // We need to check all the segments, starting from the current one and looking
144.             // backward to minimize the
145.             // checks for recently inserted entries
146.             int size = cacheSegments.size();
147.             for (int i = 0; i < size; i++) {
148.                 int segmentIdx = (currentSegmentIdx + (size - i)) % size;
149.
150.                 LongPair res = cacheIndexes.get(segmentIdx).get(ledgerId, entryId);
151.                 if (res != null) {
152.                     int entryOffset = (int) res.first;
153.                     int entryLen = (int) res.second;
154.
155.                     ByteBuf entry = allocator.buffer(entryLen, entryLen);
156.                     entry.writeBytes(cacheSegments.get(segmentIdx), entryOffset, entryLen);
157.                     return entry;
158.                 }
159.             }
160.         } finally {
161.             lock.readLock().unlock();
162.         }
163.
164.         // Entry not found in any segment
165.         return null;
166.     }

```

La coverage di tale metodo è del 100%:

get(long, long)	100%
-----------------	------

```

139     public ByteBuf get(long ledgerId, long entryId) {
140         lock.readLock().lock();
141
142         try {
143             // We need to check all the segments, starting from the current one and looking
144             // backward to minimize the
145             // checks for recently inserted entries
146             int size = cacheSegments.size();
147             for (int i = 0; i < size; i++) {
148                 int segmentIdx = (currentSegmentIdx + (size - i)) % size;
149
150                 LongPair res = cacheIndexes.get(segmentIdx).get(ledgerId, entryId);
151                 if (res != null) {
152                     int entryOffset = (int) res.first;
153                     int entryLen = (int) res.second;
154
155                     ByteBuf entry = allocator.buffer(entryLen, entryLen);
156                     entry.writeBytes(cacheSegments.get(segmentIdx), entryOffset, entryLen);
157                     return entry;
158                 }
159             }
160         } finally {
161             lock.readLock().unlock();
162         }
163
164         // Entry not found in any segment
165         return null;
166     }

```

Per questo metodo solamente un mutante non è stato 'intercettato'.

Ba-dua

```
-<method name="get" desc="(J)Lio/netty/buffer/ByteBuf;">
  <du var="this" def="140" use="161" covered="1"/>
  <du var="this" def="140" use="148" covered="1"/>
  <du var="this" def="140" use="150" covered="1"/>
  <du var="this" def="140" use="155" covered="1"/>
  <du var="this" def="140" use="156" covered="1"/>
  <du var="this" def="140" use="161" covered="1"/>
  <du var="ledgerId" def="140" use="150" covered="1"/>
  <du var="entryId" def="140" use="150" covered="1"/>
  <du var="this.lock" def="140" use="161" covered="1"/>
  <du var="this.lock" def="140" use="161" covered="1"/>
  <du var="this.cacheSegments" def="140" use="156" covered="1"/>
  <du var="this.currentSegmentIdx" def="140" use="148" covered="1"/>
  <du var="this.cacheIndexes" def="140" use="150" covered="1"/>
  <du var="this allocator" def="140" use="155" covered="1"/>
  <du var="size" def="146" use="147" target="148" covered="1"/>
  <du var="size" def="146" use="147" target="161" covered="1"/>
  <du var="size" def="146" use="148" covered="1"/>
  <du var="i" def="147" use="147" target="148" covered="1"/>
  <du var="i" def="147" use="147" target="161" covered="0"/>
  <du var="i" def="147" use="148" covered="1"/>
  <du var="i" def="147" use="147" covered="1"/>
  <du var="segmentIdx" def="148" use="156" covered="1"/>
  <du var="res" def="150" use="151" target="152" covered="1"/>
  <du var="res" def="150" use="151" target="147" covered="1"/>
  <du var="res" def="150" use="152" covered="1"/>
  <du var="res" def="150" use="153" covered="1"/>
  <du var="res.first" def="150" use="152" covered="1"/>
  <du var="res.second" def="150" use="153" covered="1"/>
  <du var="i" def="147" use="147" target="148" covered="1"/>
  <du var="i" def="147" use="147" target="161" covered="1"/>
  <du var="i" def="147" use="148" covered="1"/>
  <du var="i" def="147" use="147" covered="1"/>
  <counter type="DU" missed="1" covered="31"/>
  <counter type="METHOD" missed="0" covered="1"/>
</method>
```

`hasEntry(long ledgerId, long entryId)`

Il metodo `hasEntry` di `ReadCache` prende come parametri due `long` che identificano l'id del ledger e l'id dell'entry all'interno del **ledger**. Tale metodo restituisce `true` nel caso l'entry identificata da `entryId` esista all'interno del ledger specificato da `ledgerId`. In caso negativo ritorna `false`.

Category Partition

`long ledgerId: { ≤ 0 , > 0 }`

`long entryId: { ≤ 0 , > 0 }`

Boundary values

`long ledgerId: {-1, 0, 1}`

`long entryId: {-1, 0, 1}`

Per lo stesso motivo di `get` si è utilizzato un approccio multidimensionale:

`{false, -1, -1},`

`{false, -1, 0},`

`{false, -1, 1},`

`{false, 0, -1},`

`{false, 0, 0},`

`{false, 0, 1},`

`{false, 1, -1},`

`{false, 1, 0},`

`{false, 1, 1},`

`{true, 123, 0}`

Jacoco & PIT

```

168.     public boolean hasEntry(long ledgerId, long entryId) {
169.         lock.readLock().lock();
170.
171.         try {
172.             int size = cacheSegments.size();
173.             for (int i = 0; i < size; i++) {
174.                 int segmentIdx = (currentSegmentIdx + (size - i)) % size;
175.
176.                 LongPair res = cacheIndexes.get(segmentIdx).get(ledgerId, entryId);
177.                 if (res != null) {
178.                     return true;
179.                 }
180.             }
181.         } finally {
182.             lock.readLock().unlock();
183.         }
184.
185.         // Entry not found in any segment
186.         return false;
187.     }

```

La coverage è del 100%:  **hasEntry(long, long)** 100%

```

168     public boolean hasEntry(long ledgerId, long entryId) {
169 1     lock.readLock().lock();
170
171     try {
172         int size = cacheSegments.size();
173 2     for (int i = 0; i < size; i++) {
174 3         int segmentIdx = (currentSegmentIdx + (size - i)) % size;
175
176         LongPair res = cacheIndexes.get(segmentIdx).get(ledgerId, entryId);
177 1     if (res != null) {
178 2         return true;
179     }
180     } finally {
181 1     lock.readLock().unlock();
182     }
183
184     // Entry not found in any segment
186 1     return false;
187 }

```

solamente due mutanti non sono stati 'intercettati'.

Ba-dua

```

<method name="hasEntry" desc="(JJ)Z">
  <du var="this" def="169" use="182" covered="1"/>
  <du var="this" def="169" use="174" covered="1"/>
  <du var="this" def="169" use="176" covered="1"/>
  <du var="this" def="169" use="182" covered="1"/>
  <du var="ledgerId" def="169" use="176" covered="1"/>
  <du var="entryId" def="169" use="176" covered="1"/>
  <du var="this.lock" def="169" use="182" covered="1"/>
  <du var="this.lock" def="169" use="182" covered="1"/>
  <du var="this.currentSegmentIdx" def="169" use="174" covered="1"/>
  <du var="this.cacheIndexes" def="169" use="176" covered="1"/>
  <du var="size" def="172" use="173" target="174" covered="1"/>
  <du var="size" def="172" use="173" target="182" covered="1"/>
  <du var="size" def="172" use="174" covered="1"/>
  <du var="i" def="173" use="173" target="174" covered="1"/>
  <du var="i" def="173" use="173" target="182" covered="0"/>
  <du var="i" def="173" use="174" covered="1"/>
  <du var="i" def="173" use="173" covered="1"/>
  <du var="res" def="176" use="177" target="178" covered="1"/>
  <du var="res" def="176" use="177" target="173" covered="1"/>
  <du var="i" def="173" use="173" target="174" covered="1"/>
  <du var="i" def="173" use="173" target="182" covered="1"/>
  <du var="i" def="173" use="174" covered="1"/>
  <du var="i" def="173" use="173" covered="1"/>
  <counter type="DU" missed="1" covered="22"/>
  <counter type="METHOD" missed="0" covered="1"/>
</method>

```

Syncope

RealmUtils.java

Questa classe è relativamente nuova (fino alla venticinquesima release, ultima tra quelle su cui è stata calcolata la buggyness, non è presente), quindi potrebbe contenere dei bug.

```
normalizingAddTo(final Set<String> realms, final String newRealm)
```

Tale metodo si occupa dell'aggiunta di un nuovo Realm. Prende come parametro il set dei Realm già esistenti ed il nome del nuovo.

Dalla documentazione si può evincere che un real è una sorta di albero che definisce un dominio di sicurezza ed è rappresentato come un path:

```
Realms define a hierarchical security domain tree, primarily meant for containing Users, Groups and Any Objects.
```

```
Each realm:
```

1. has a unique name and a parent realm except for the pre-defined root realm, which is named `/'`;
2. is either a leaf or root of a sub-tree of realms;
3. is uniquely identified by the path from the root realm, e.g. `/a/b/c` identifies the sub-realm `c` in the sub-tree rooted at `b`, having in turn `a` as parent realm, directly under the root realm;

ref. <https://syncope.apache.org/docs/reference-guide.html#realms>

Category Partition

```
final Set<String> realms: {valid, invalid, null, empty}  
final String newRealm: {valid, invalid, null, ""}
```

Partendo dal presupposto che il nome di un realm è sostanzialmente un path, un nome di Real non valido potrebbe essere una stringa che non rappresenta un path o che contiene caratteri che non possono essere utilizzati in un path.

Il fatto però è che non si sta parlando di un path relativo a dei file in un certo FS quindi la precedente affermazioni è probabilmente errata e i casi di test prodotti seguendo questa logica potrebbero non produrre il risultato atteso.

Boundary values

```
final Set<String> realms: {valid_set, invalid_set, null, empty_set}
final String newRealm: {valid_path, invalid_path, null, ""}
```

Con un approccio multidimensionale si ottengono i seguenti casi di test:

{valid_set, valid_path} -> questo caso di test è stato replicato per far sì di testare il metodo in caso di path già presenti nella lista e anche per nuovi path.

```
{valid_set, invalid_path}
{valid_set, null}
{valid_set, empty_set}
{invalid_set, valid_path}
{invalid_set, invalid_path}
{invalid_set, null}
{invalid_set, empty_set}
{null, valid_path}
{null, invalid_path}
{null, null}
{null, empty_set}
{empty_set, valid_path}
{empty_set, invalid_path}
{empty_set, null}
{empty_set, empty_set}
```

Eseguendo i test ho potuto verificare che il metodo non controlla il nome del realm ma semplicemente si accerta che nel set ci sia o meno il nuovo realm. Da qui quindi ho dovuto procedere con un approccio white box e verificare l'effettiva implementazione, dalla quale si evince che ogni volta che il nuovo realm è presente nel set allora non viene aggiunto. Se nel set c'è un realm il cui path inizia con lo stesso nome del nuovo realm allora questo viene rimosso dal set. Se nessuno dei due casi è vero allora il nuovo realm viene aggiunto nel set. Esempi:

1. immaginiamo che il set di realm esistenti sia [/a/b, /c, /d/e] e che il nuovo realm sia /a
 - allora in questo caso il realm /a/b viene eliminato e /a non viene aggiunto
2. in questo caso sia [/c, /d/e, /f] il set di realm esistenti e /a il nuovo realm
 - in tal caso il nuovo realm viene aggiunto
3. nel caso in cui il set fosse [/a, /c, /d/e] e si volesse ancora aggiungere /a/b allora:
 - in questo caso il nuovo realm non viene aggiunto poichè ne esiste già uno con path che inizia in modo uguale
4. Se il set fosse [/a/b, /c, /d/e] e si volesse aggiungere /a/c:
 - in questo caso /a/c viene aggiunto

Da questi esempi si evince quindi che vengono aggiunti nuovi realm solo se non esistono, nel set, dei realm di livello superiore a quello da inserire; inoltre due realm identici non faranno parte del set.

`/a` è un realm di livello superiore rispetto a `/a/b`.

Non è possibile quindi pensare a casi di test basandosi sulla validità dei nomi dei realm, ma bisogna solamente tenere conto se un realm possa essere aggiunto o meno a seconda delle regole esplicitate sopra. Da qui sono stati elaborati dei nuovi casi di test:

```
exp set newRealm
{true, VALID, newR1},
{true, VALID, newR2},
{false, VALID, newR3},
{false, VALID, null},
{false, VALID, ""},

{true, EMPTY, newR1},
{false, EMPTY, null},
{false, EMPTY, ""},
```

Dove newR1, newR2, newR3 sono semplicemente dei valori validi differenti (vedi implementazione).

POSSIBILE BUG 1 (P.Bug1)

Tutti i test, tranne quelli evidenziati, producono un risultato corretto. I test che falliscono sono quelli in cui il `newRealm` è una stringa vuota, in particolare tali test provocano l'eliminazione di tutto il set di realm e l'inserimento proprio di tale realm 'vuoto'. Questo è dovuto al fatto che nell'implementazione del metodo viene utilizzato il metodo `startsWith(String prefix)` che restituisce `true` in caso l'argomento sia una stringa vuota. Di conseguenza, restituendo `true` per ogni realm presente nel test, questi vengono tutti eliminati, provocando quindi lo svuotamento del set.

POSSIBILE BUG 2 (P.Bug2)

Un altro problema è stato riscontrato nel caso di test in cui il nuovo realm da inserire è `null`. In particolare, se il set di realm è vuoto e si prova ad inserire un nuovo realm `null` allora l'inserimento viene effettuato (non ci si aspetta questo) mentre se si prova ad inserire un oggetto `null` in un set non vuoto allora l'operazione non viene effettuata poichè viene correttamente lanciata un'eccezione da `startsWith(String prefix)` poichè l'oggetto su cui viene chiamata è `null`. Nell'altro caso invece non viene lanciata eccezione poichè, essendo la lista vuota, non si entra nel ciclo e quindi la chiamata a `startsWith(String prefix)` non viene effettuata ma si procede direttamente con l'inserimento.

Sono stati effettuati altri test per verificare la correttezza del metodo ed andare a fondo ai casi precedenti. In particolare è stato implementato un test (non parametrizzato) che invece di verificare il valore di ritorno del metodo verifica i cambiamenti effettuati sul set (in caso ci

fossero stati), in questo modo si è verificato in maniera più puntuale la correttezza del metodo testato e i problemi riscontrati.

Inoltre è stato implementato un ulteriore test che verifica che due chiamate consecutive a `normalizeAddTo` in cui la prima prova a fare un inserimento valido in un set vuoto e la seconda prova a fare un inserimento nullo non provochino risultati inattesi. Questo è stato fatto perchè il P.Bub2 è stato scoperto in quanto si verificava che con i casi di test iniziali tale errore veniva prodotto solo quando, nell'inserimento in un set vuoto, il caso null veniva eseguito prima del caso valido.

Le due classi di test ulteriori sono: `RealmUtilsSecondTest` e `RealmUtilsEmptySetTest`.

Jacoco & PIT

```
43.     public static boolean normalizingAddTo(final Set<String> realms, final String newRealm)
44.     {
45.         boolean dontAdd = false;
46.         Set<String> toRemove = new HashSet<>();
47.         for (String realm : realms) {
48.             if (newRealm.startsWith(realm)) {
49.                 dontAdd = true;
50.             } else if (realm.startsWith(newRealm)) {
51.                 toRemove.add(realm);
52.             }
53.         }
54.         realms.removeAll(toRemove);
55.         if (!dontAdd) {
56.             realms.add(newRealm);
57.         }
58.         return !dontAdd;
59.     }
```

La coverage per questo metodo è del 100%:

● [normalizingAddTo\(Set, String\)](#)  100%

```
43     public static boolean normalizingAddTo(final Set<String> realms, final String newRealm) {
44         boolean dontAdd = false;
45         Set<String> toRemove = new HashSet<>();
46         for (String realm : realms) {
47     1         if (newRealm.startsWith(realm)) {
48             dontAdd = true;
49     1         } else if (realm.startsWith(newRealm)) {
50             toRemove.add(realm);
51         }
52     }
53
54     realms.removeAll(toRemove);
55     1     if (!dontAdd) {
56         realms.add(newRealm);
57     }
58     2     return !dontAdd;
59 }
```

Tutti i mutanti sono stati 'intercettati'.

Encryptor.java

Su tale classe non è stato mai effettuato un fix di un bug e non è stata etichettata come buggy dall'analisi della buggyness. Tale classe però ha subito delle modifiche in ognuna delle 25 release analizzate quindi potrebbe ospitare dei bug ancora dormienti.

```
verify(final String value, final CipherAlgorithm  
cipherAlgorithm, final String encoded)
```

Questo metodo prende una stringa, un algoritmo di cifratura ed una stringa che rappresenta la password cifrata. La stringa `value` viene cifrata tramite `cipherAlgorithm` e confrontata poi con la stringa `encoded`. Restituisce un booleano.

In Syncope, con `CipherAlgorithm`, si identificano indistintamente sia algoritmi di hashing che ciphers. Per semplicità e facilità di comprensione del report verrà utilizzata la parola cipher tutti.

I cipher che possono essere utilizzati sono:

- producono digest di lunghezza fissa:
 - SHA1: 20 byte
 - SHA256: 32 byte
 - SHA512 : 64 byte
 - MD5: 16 byte
 - BCRYPT: 23 byte
 - salted: SSHA, SSHA1, SSHA256, SSHA512 (uguali ai base)
- Cifrario a blocchi quindi lavora con blocchi da 16 byte, può cifrare qualsiasi lunghezza:
 - AES

Category Partition

```
String value: {invalid, valid, null, ""}
```

```
CipherAlgorithm cipherAlgorithm: {invalid, valid, null}
```

```
String encoded: {invalid, valid, null, ""}
```

Boundary values

Per `encoded` definisco:

invalid: un digest che supera la lunghezza massima prodotta da tutte le funzioni hash utilizzabili che è 64 (eccetto AES che è un cipher a blocchi e non produce digest) oppure uno più corto della lunghezza minima. Per esempio:

```
"a0c299b71a9e59d5ebb07917e70601a3570aa103e99a7bb65a58e780ec9077b1902d1d  
edb31b1457beda595fe4d71d779b6ca9cad476266cc07590e31d84b206"
```

Considero prima gli algoritmi che producono un digest di lunghezza fissa:
SHA, SHA1, SHA256, SHA512, SMD5, BCRYPT

```
String value: {valid_str, invalid_str, "", null}  
CipherAlgorithm cipherAlgorithm: {valid, invalid, null}  
String encoded: {valid, invalid, "", null}
```

Utilizzando un approccio multidimensionale tra `cipherAlgorithm` e `encoded`, che sono due parametri relazionati, ottengo:

{valid, invalid},{valid, null},{valid, ""},{invalid, valid},{null, valid}

Utilizzando poi un approccio multidimensionale tra i casi prodotti sopra e il parametro `value` si ottengono:

{valid_str, valid, invalid}
{valid_str, valid, null}
{valid_str, valid, ""}
{valid_str, invalid, valid}
{valid_str, null, valid}}

{invalid_str, valid, invalid}
{invalid_str, valid, null}
{invalid_str, valid, ""}
{invalid_str, invalid, valid}
{invalid_str, null, valid}

I casi evidenziati non hanno senso di esistere in quanto non si riesce a produrre una stringa non valida e di conseguenza ci si basa solamente a considerarli nulli, quindi come i casi successivi.



{ "", valid, invalid}
{ "", valid, null}
{ "", valid, ""}
{ "", invalid, valid}
{ "", null, valid}
{ null, valid, invalid}
{ null, valid, null}
{ null, valid, ""}
{ null, invalid, valid}
{ null, null, valid}

Ovviamente nel caso di cipher valido vengono considerati tutti i cipher disponibili
Quando il cipher specificato è null viene utilizzato AES

Tutti i test restituiscono risultati coerenti con quelli attesi, tranne quelli per cui il cipher è salted.

Jacoco & PIT

```
116.     public boolean verify(final String value, final CipherAlgorithm cipherAlgorithm, final String encoded) {
117.         boolean verified = false;
118.
119.         try {
120.             if (value != null) {
121.                 if (cipherAlgorithm == null || cipherAlgorithm == CipherAlgorithm.AES) {
122.                     verified = encode(value, cipherAlgorithm).equals(encoded);
123.                 } else if (cipherAlgorithm == CipherAlgorithm.BCRYPT) {
124.                     verified = BCrypt.checkpw(value, encoded);
125.                 } else {
126.                     verified = getDigester(cipherAlgorithm).matches(value, encoded);
127.                 }
128.             }
129.         } catch (Exception e) {
130.             LOG.error("Could not verify encoded value", e);
131.         }
132.
133.         return verified;
134.     }
```

La coverage è del 100%:  [verify\(String, CipherAlgorithm, String\)](#)  100%

```
116     public boolean verify(final String value, final CipherAlgorithm cipherAlgorithm, final String encoded) {
117         boolean verified = false;
118
119         try {
120             if (value != null) {
121                 if (cipherAlgorithm == null || cipherAlgorithm == CipherAlgorithm.AES) {
122                     verified = encode(value, cipherAlgorithm).equals(encoded);
123                 } else if (cipherAlgorithm == CipherAlgorithm.BCRYPT) {
124                     verified = BCrypt.checkpw(value, encoded);
125                 } else {
126                     verified = getDigester(cipherAlgorithm).matches(value, encoded);
127                 }
128             }
129         } catch (Exception e) {
130             LOG.error("Could not verify encoded value", e);
131         }
132
133         return verified;
134     }
```

Tutti i mutanti sono stati 'intercettati'.

`encode(final String value, final CipherAlgorithm cipherAlgorithm)`

Tale metodo prende come parametro una stringa ed un algoritmo di cifratura. Tramite l'algoritmo la stringa (password) viene cifrata e restituita.

Category Partition

`String value`: {invalid, valid, null, ""}

`CipherAlgorithm cipherAlgorithm`: {invalid, valid, null}

Boundary values

`String value`: {valid_str, invalid_str, "", null}

`CipherAlgorithm cipherAlgorithm`: {valid, invalid, null}

{valid_str, valid}
{valid_str, invalid}
{valid_str, null}
{invalid_str, valid,}
{invalid_str, invalid}
{invalid_str, null}
{ "", valid}
{ "", invalid}
{ "", null}
{null, valid}
{null, invalid}
{null, null}

Nell'impossibilità di creare una Stringa non valida si considera null.

Il metodo Encode ha un Cipher di default quindi quando viene passato un cipher non valido o nullo viene effettuata una cifratura usando il cifrario a blocchi AES, utilizzando una chiave di default.

A questo punto però è utile verificare se tutti i Cipher producano un risultato corretto, di conseguenza devo fare un test per ogni cipher quando questo è valido.

Durante la fase di testing ci sono state difficoltà nell'utilizzo dei cipher salted. Infatti l'utilizzo di essi richiede che venga creato un certo Context che però non è stato possibile creare in alcun modo in quanto veniva lanciata un'eccezione indicando che il contesto non fosse valido. Per scrupolo sono stati controllati anche i test nativi di Syncope nei quali si è riscontrato lo stesso problema.

Jacoco & PIT

```

94.     public String encode(final String value, final CipherAlgorithm cipherAlgorithm)
95.         throws UnsupportedOperationException, NoSuchAlgorithmException, NoSuchPaddingException, InvalidKeyException,
96.             IllegalBlockSizeException, BadPaddingException {
97.
98.         String encoded = null;
99.
100.        if (value != null) {
101.            if (cipherAlgorithm == null || cipherAlgorithm == CipherAlgorithm.AES) {
102.                Cipher cipher = Cipher.getInstance(CipherAlgorithm.AES.getAlgorithm());
103.                cipher.init(Cipher.ENCRYPT_MODE, keySpec);
104.
105.                encoded = Base64.getEncoder().encodeToString(cipher.doFinal(value.getBytes(StandardCharsets.UTF_8)));
106.            } else if (cipherAlgorithm == CipherAlgorithm.BCRYPT) {
107.                encoded = BCrypt.hashpw(value, BCrypt.gensalt());
108.            } else {
109.                encoded = getDigester(cipherAlgorithm).digest(value);
110.            }
111.        }
112.
113.        return encoded;
114.    }

```

La coverage è del 100%:  [encode\(String, CipherAlgorithm\)](#)  100%

```

94     public String encode(final String value, final CipherAlgorithm cipherAlgorithm)
95         throws UnsupportedOperationException, NoSuchAlgorithmException, NoSuchPaddingException, InvalidKeyException,
96             IllegalBlockSizeException, BadPaddingException {
97.
98         String encoded = null;
99.
100.        if (value != null) {
101.            if (cipherAlgorithm == null || cipherAlgorithm == CipherAlgorithm.AES) {
102.                Cipher cipher = Cipher.getInstance(CipherAlgorithm.AES.getAlgorithm());
103.                cipher.init(Cipher.ENCRYPT_MODE, keySpec);
104.
105.                encoded = Base64.getEncoder().encodeToString(cipher.doFinal(value.getBytes(StandardCharsets.UTF_8)));
106.            } else if (cipherAlgorithm == CipherAlgorithm.BCRYPT) {
107.                encoded = BCrypt.hashpw(value, BCrypt.gensalt());
108.            } else {
109.                encoded = getDigester(cipherAlgorithm).digest(value);
110.            }
111.        }
112.
113.        return encoded;
114.    }

```

Tutti i mutanti sono stati 'intercettati'.

Note

L'applicazione di ba-dua per il progetto Syncope non è stata possibile. Questo è dovuto al fatto che Syncope ha bisogno della jdk-20 per essere compilato ma ba-dua necessita della jdk-8.