CReport progetto Software Testing su Apache Bookkeeper e Apache Syncope

ISW2 0264636 Valentino Perrone

INTRODUZIONE

Il seguente report è stato tratto dallo studio di due progetti dell'Apache Software Foundation: Apache Bookkeeper comune a tutti gli studenti e Apache Syncope. Tale studio si pone l'obiettivo di migliorare la qualità del test effettuato su 4 classi Java, 2 classi per progetto. Inoltre sono stati presi in considerazione 4 metodi per le classi di Bookkeeper e 3 metodi per le classi di Syncope.

Per la scelta delle classi di Bookkeeper sono stati utilizzati in primo luogo i risultati ottenuti dal progetto di Machine Learning del professor Falessi e poi è stata effettuata una ricerca su quali classi avessero dei metodi pubblici con almeno un branch, almeno un parametro di input e un output, quindi che non fossero classi banali come setter e getter e che avessero un minimo di documentazione.

Per quanto riguarda la configurazione dell'ambiente di lavoro e per favorire il processo di sviluppo del continuous integration è stato utilizzato Github per ospitare i progetti Git Forkati, TravisCI per il build in Maven automatico ed infine SonarCloud collegato a TravisCI per misurare la qualità del software prodotto.

Lo studio presentato inizia con lo sviluppo di test di unità minimali con il framework JUnit e continua con l'obiettivo di migliorare l'adeguatezza dei test prodotti utilizzando il risultato dei criteri di copertura ottenuti da Jacoco e di data flow ottenuti da Ba-dua ed infine analizzando il risultato dei test di mutations ottenuti da Pit.

APACHE BOOKKEEPER

Il progetto Apache Bookkeeper è un progetto di logging distribuito. Bookkeeper lavora con Apache Zookkeeper il quale si occupa principalmente di mantenere i metadati del sistema e di eleggere il master dell'architettura di Bookkeeper nel caso andasse in down.

Per poter eseguire i test sulle classi di Bookkeeper si è proceduto eliminando le classi Java testate dai suoi sviluppatori così da non essere eseguite automaticamente da ciclo di build di Maven.

I metodi e le classi scelte per il testing sono le seguenti:

Per il progetto di Bookkeeper sono state scelte le seguenti classi e metodi:

- *org.apache.bookkeeper.client.BookKeeperAdmin.java*
  metodo1:
  public static boolean initBookie(ServerConfiguration conf) throws Exception

  metodo2:
  static String formatEnsemble(List<BookieSocketAddress> ensemble,Set<BookieSocketAddress> bookiesSrc, char marker)

- *org.apache.bookkeeper.bookie.storage.ldb.ReadCache.java*
  metodo1:
  public ByteBuf get(long ledgerId, long entryId)

  metodo2:
  public void put(long ledgerId, long entryId, ByteBuf entry)

BookkeeperAdmin:

BookkeeperAdmin è una classe che consente le varie inizializzazioni dei cluster di Zookkeeper e di Bookkeeper. Le misure estratte evidenziano che si tratta di una delle classi maggiormente modificate durante le release di sviluppo ed anche maggiormente revisionate. Verranno ora spiegati i metodi utilizzati più nel dettaglio.

Metodo1: *public static boolean initBookie(ServerConfiguration conf) throws Exception*
Tale metodo si occupa di inizializzare un BookkeeperServer secondo le configurazioni prese in ingresso. Al fine di simulare al meglio l'ambiente è stato possibile far partire il cluster di Zookkeeper nella fase di setup del test.
Al fine di scegliere dei valori accettabili come input al metodo si è scelto di adottare l'approccio category partition evidenziando le partizioni e le classi d'equivalenza per il parametro complesso "conf".
Le classi d'equivalenza banali sono [null] e [baseconf] che banalmente diventano i due input al metodo. Il risultato atteso rispettivamente è di esito negativo e positivo nel caso della baseConf.

Metodo2: *static String formatEnsemble(List<BookieSocketAddress> ensemble,Set<BookieSocketAddress> bookiesSrc, char marker)*

Questo metodo si occupa di formattare la lista di bookie mettendo un marker sul bookieSrc che ha fallito. Le partizioni minimali scelte per i 3 parametri sono le seguenti :

- marker: {null, '*'}
- ensemble: {null} {valori validi}
- bookieSrc: {valido} {null}

Adottando un approccio Boundary Value unidimensionale sono state provate le seguenti combinazioni:

- ensemble = array di 4 bookies, src = bookie 0 , marker = '*'
- ensemble = null, src = null, marker = null

In questo modo associo ad ogni valore almeno una volta la classe di appartenenza in maniera minimale. I risultati attesi sono:

- [bookie0*, bookie1, bookie2, bookie3]
- ""

ReadCache:

Questa classe è stata scelta perchè formata da 167 linee di codice e ha come age 132 settimane e churn 197. Dei parametri discreti che affermano la possibilità di avere dei bug da parte della classe considerata. ReadCache si occupa di salvare all'interno di segmenti delle porzioni momentanee di memoria.

Metodo1: *public void put(long ledgerId, long entryId, ByteBuf entry)*
Il metodo put inserisce all'interno della cache un'entry utilizzando un buffer di grandezza 64 byte o multipli indicizzato per entryId. Il ledgerId deve essere necessariamente >=0 mentre l'entryId può assumere qualsiasi valore. Le classi d'equivalenza considerate sono:

- ledgerId: <0, >=0
- entryId: <0, >=0
- entry: null, entryValida in un buffer di 64 bytes

Un primo test minimale unidimensionale riguarda i seguenti input:

- ledgerId = -1 , entryId = -1, buffer di 64 byte, esito sperato = negativo
- ledgerId = 0, entryId = 1, buffer 64 byte, esito sperato = positive

- ledgerId = 0, entryId = 0, buffer di 64 bytes , esito positive

Il caso con buffer null non è stato inserito poiché non si arrivava a testare la Put dato che un buffer Null non crea nessuna entry.

Metodo2: *public ByteBuf get(long ledgerId, long entryId)*
Per testare la funzione get si è scelto di utilizzare la funzione Put per inserire ciò che si vuole verificare che sia presente in cache. Valori inseriti come input in una fase preliminare:

- ledgerId = 0,entryId = -1, buffer 64, esito positivo
- ledgerId = 0, entryId= 0, buffer 64, esito positive

Non è stato inserito un valore negativo nel ledgerId poiché non effettuerebbe la fase di Put e sarebbe inutile al fine del testing della funzione get.

MIGLIORAMENTO ADEGUATEZZA (statement coverage e branch coverage)
Una volta ottenuto un primo caso di test si è voluto indagare sulla bontà di tale test ossia l'adeguatezza, cioè su quanto il System Under Testing fosse testato bene in maniera meticolosa e su quanto i test fossero sufficientemente buoni e conformi agli obiettivi.
Al fine di ottenere risultati migliori riguardo la suite di test strutturali, sono stati utilizzati criteri di copertura per ottenere degli indici sulla porzione di codice del SUT che i casi di test riescono a coprire.
A questo proposito è stato utilizzato il plugin Jacoco incluso all'interno del Pom di Maven che restituisce come indici la Branch Coverage e la Statement Coverage.
Come si evince dalla [Figura 1] i metodi della classe BookkeeperAdmin, initBookie(ServerConfiguration conf) e formatEnsemble(list, set, char) sono stati coperti per lo statement coverage rispettivamente per il 72% e 100% mentre per il branch coverage 37% e 100%. Ovviamente sono stati inclusi nella coverage anche metodi che sono richiamati dai metodi target sotto testing. Per migliorare la coverage sul metodo initBookie è stato doveroso studiare l'implementazione in maniera più dettagliata per poter aumentare il valore della coverage e quindi ottenere una maggiore adeguatezza del test.
Dalla documentazione è possibile capire che initBookie restituisce false in case in cui le directory : journalDir, ledgerDir e indexDir non fossero vuote. Per aumentare la statement coverage fino all'83% e una branch coverage fino al 50% [Figura 2] sono stati creati dei file nella directory journalDir mentre non è stato possibile creare dei file nelle altre directory per problemi annessi ai permessi Unix.
Per quanto concerne la classe ReadCache[Figura3] in una fase preliminare di test si è riuscito a coprire il 42% per lo statemente coverage e 25% branch coverage per il metodo put e il 94% per statemente e 50% per branch coverage per il metodo get. Per migliorare l'operazione di get è bastato inserire il caso in cui venisse effettuata la get di un'entry non disponibile arrivando ad una coverage del 100%[Figura4] sia per statement che per branch.

MIGLIORAMENTO ADEGUATEZZA (mutation testing)
Per migliorare ulteriormente l'adeguatezza viene utilizzato PIT. Pit esegue un mutation testing ed utilizza dei flag in grado di cambiare il tipo e il numero di mutazioni. In entrambi i progetti, sono stati modificati opportunamente i file pom.xml, così da aggiungere un profilo di attivazione per evitare di problemi nel build di travis a causa del tempo elevato richiesto a PIT per generare le mutazioni. A tal proposito si è deciso di impostare un numero massimo di mutazioni per classe a 50.
La [Figura 5] mostra il miglioramento dovuto ai test sopra citati rispetto alla [Figura 6]. E' possibile notare una percentuale dell'1% in più sulle coverage.

Rimane comunque molto bassa la coverage della classe BookkeeperAdmin, in particolare nel metodo InitBookie viene impedito l'accesso dei mutanti alle varie condition così da ridurre la coverage della classe. Nel metodo formatEnsemble della medesima classe sono presenti solamente mutanti killati. Migliore in termini di coverage è risultata la classe ReadCache. E' possibile notare dalla [Figura 7] e [Figura 8] i mutanti sopravvissuti in rosso e quelli killati in verde per quanto riguarda i metodi put e get.

APACHE SYNCOPE

Per il progetto apache Syncope la scelta delle classi da testare è stata complicata dalla poca documentazione del progetto e dalle numerose classi presenti con metodi privati o con annotazioni spring. Le classi infine scelte fanno parte del package core e sono :

- org.apache.syncope.core.provisioning.api.jexl.JexlUtils
- org.apache.syncope.core.provisioning.api.utils.FormatUtils

In particolare la classe Format esegue delle formattazioni sulla data e sui numeri mentre la classe JexUtils esegue controlli sull espressioni.
I metodi presi in considerazioni rispettivamente per le classi precedenti sono spiegati di seguito.

Metodo 1: *public static boolean isExpressionValid(final String expression)*
Tale metodo controlla se l'espressione in ingresso è valida. Essendo una string si è scelto di utilizzare come classi di equivalenza la stringa nulla e un'espressione banale valida ottenendo in output un esito negativo nel primo caso e positivo nel secondo.

Per quanto riguarda la seconda classe i metodi utilizzati sono:
Metodo 1: public static String format(final Date date, final boolean lenient, final String conversionPattern)
Questo metodo esegue una formattazione della data inserendo come parametro il lenient che consente di essere più flessibili quando il pattern non matcha in maniera esatta.
Le partizioni individuate sono:

- date: {null}, {base date}
- lenient: {true}, {false}
- pattern: {pattern valido}, {pattern invalido} , {null}

I parametri testati in una prima analisi sono:

- date= valid date, lenient = false, pattern = valid
- date = null, lenient = true, pattern nullo
- date = valid date, lenient = true, pattern invalido

Metodo 2: public static Number parseNumber(final String source, final String conversionPattern) throws ParseException

Questo metodo esegue il parsing di una stringa secondo un pattern verso un tipo di dato complesso Number che viene utilizzato per le serializzazioni.
Classi di equivalenza individuate:

- source = {stringa1 ricavata da getTime() di Calendar()}, {null}, {invalid source}
- pattern = {null}, {###,###}, {invalid pattern}

la stringa invalida è una stringa non convertibile in numero. Per la scelta di un pattern non valido si è scelta una combinazione di caratteri random.

Avendo successivamente il seguente input:

- source = valore valido, pattern = pattern valido
- source = null, pattern = pattern invalido
- source = stringa invalida, pattern = null

MIGLIORAMENTO ADEGUATEZZA (statement coverage e branch coverage)
E' possibile verificare attraverso la [Figura 7] e [Figura 8] come i test effettuati inizialmente su entrambe le classi riescono a coprire tutta la statement coverage e branch coverage. Tale risultato permette di capire che dei buoni test iniziali permettono di coprire gran parte del sistema under testing.

MIGLIORAMENTO ADEGUATEZZA (mutations)
La [Figura 11] mostra la coverage ottenuta sulle classi JexlUtils e FormatUtils. Appare evidente come la coverage ottenuta in FormatUtils risulta essere maggiore. Tale risultato potrebbe essere dovuto al numero minore di condition presenti nella classe FormatUtils.
Entrando nel dettaglio nei metodi trattati è possibile notare dalla [Figura 12] come nel metodo isExpressionValid della classe JexlUtils vengano killati 2 mutanti in confronto a 0 sopravvissuti, indice del fatto che il test si sia comportato in maniera adeguata.
Nell'altra classe invece, analizzando il metodo parseNumber [Figura 13] si nota che un mutante sopravvive all'applicazione del pattern ma viene killato durante il parsing. Analogamente nel metodo format [Figura 14] trattato durante lo studio sono presenti 2 mutanti sopravvissuti in confronto a 3 killati.

CONFIGURAZIONE E PROBLEMI RISCONTRATI

Per la configurazione del Pom di Apache Bookkeeper è stata seguita la guida consigliata per maven Multimodulo per quanto riguarda l'inserimento del plugin di Jacoco e Surefire per generare i report ed avviare in automatico i test nella fase di verify di maven. E' stato poi aggiunto il plugin per pitest per visualizzare il report per le mutazioni.
Riguardo Syncope la configurazione del pom è stata più ostile perché la struttura è organizzata in maniera gerarchica. Pitest sono stati inseriti in un profilo per essere attivati solamente in locale. E' servito inoltre scaricare il Jar di Badua ed utilizzare dei comandi di badua dal terminale per poter deserializzare il report generato. Badua è stato inserito nel pom per uno studio tecnico di avviamento, ma per problemi di tempo non è stato possibile includerlo nel report per il miglioramento dei test.

Figura 1

getUnderreplicationManager()		0%		0%	2	2	3	3	1	1
initBookie(ServerConfiguration)		72%		37%	4	5	4	11	0	1
validateDirectoriesAreEmpty(File[], String)		69%		50%	2	4	2	6	0	1
lambda$waitForLedgersToBeReplicated$9(BookieSocketAddress, LedgerManager, Long)		0%		0%	2	2	1	1	1	1
asyncRecoverLedgerFragment(LedgerHandle, LedgerFragment, AsyncCallback.VoidCallback, Set, BiConsumer)		0%		n/a	1	1	2	2	1	1
format(ServerConfiguration, boolean, boolean)		0%		n/a	1	1	1	1	1	1
asyncRecoverBookieData(Set, AsyncCallback.RecoverCallback, Object)		0%		n/a	1	1	2	2	1	1
asyncRecoverBookieData(Set, boolean, boolean, AsyncCallback.RecoverCallback, Object)		0%		n/a	1	1	2	2	1	1
isEnsembleAdheringToPlacementPolicy(List, int, int)		0%		n/a	1	1	1	1	1	1
lambda$asyncRecoverBookieData$2(AsyncCallback.RecoverCallback, Object, int, String, Object)		0%		n/a	1	1	1	1	1	1
setLostBookieRecoveryDelay(int)		0%		n/a	1	1	3	3	1	1
asyncGetListOfEntriesOfLedger(BookieSocketAddress, long)		0%		n/a	1	1	1	1	1	1
lambda$decommissionBookie$8(List, UnderreplicatedLedger)		0%		n/a	1	1	2	2	1	1
recoverBookieData(Set)		0%		n/a	1	1	2	2	1	1
initNewCluster(ServerConfiguration)		0%		n/a	1	1	1	1	1	1
getLostBookieRecoveryDelay()		0%		n/a	1	1	2	2	1	1
BookKeeperAdmin(BookKeeper)		0%		n/a	1	1	2	2	1	1
getAllBookies()		0%		n/a	1	1	1	1	1	1
lambda$decommissionBookie$7(BookieSocketAddress, List)		0%		n/a	1	1	1	1	1	1
close()		57%		50%	1	2	1	3	0	1
getLedgerMetadata(LedgerHandle)		0%		n/a	1	1	1	1	1	1
lambda$static$0(Long, Long)		0%		n/a	1	1	1	1	1	1
formatEnsemble(List, Set, char)		100%		100%	0	4	0	11	0	1
BookKeeperAdmin(BookKeeper, StatsLogger)		100%		n/a	0	1	0	7	0	1

Figura 2

initBookie(ServerConfiguration)	83%		50%	4	5	3	11	0	1
recoverBookieData(Set)	0%		n/a	1	1	2	2	1	1
initNewCluster(ServerConfiguration)	0%		n/a	1	1	1	1	1	1
getLostBookieRecoveryDelay()	0%		n/a	1	1	2	2	1	1
BookKeeperAdmin(BookKeeper)	0%		n/a	1	1	2	2	1	1
getAllBookies()	0%		n/a	1	1	1	1	1	1
lambda$decommissionBookie$7(BookieSocketAddress, List)	0%		n/a	1	1	1	1	1	1
close()	57%		50%	1	2	1	3	0	1
getLedgerMetadata(LedgerHandle)	0%		n/a	1	1	1	1	1	1
lambda$static$0(Long, Long)	0%		n/a	1	1	1	1	1	1
formatEnsemble(List, Set, char)	100%		100%	0	4	0	11	0	1
BookKeeperAdmin(BookKeeper, StatsLogger)	100%		n/a	0	1	0	7	0	1

Figura 3

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
put(long, long, ByteBuf)		42%		25%	2	3	11	21	0	1
size()		0%		0%	4	4	9	9	1	1
get(long, long)		94%		50%	2	3	1	13	0	1
ReadCache(ByteBufAllocator, long, int)		100%		100%	0	2	0	12	0	1

Figura 4

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
put(long, long, ByteBuf)		42%		25%	2	3	11	21	0	1
size()		0%		0%	4	4	9	9	1	1
count()		0%		0%	2	2	6	6	1	1
ReadCache(ByteBufAllocator, long, int)		100%		100%	0	2	0	12	0	1
get(long, long)		100%		100%	0	3	0	13	0	1
ReadCache(ByteBufAllocator, long)		100%		n/a	0	1	0	2	0	1

Figura 5

Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage		Mutation Coverage	
2	18%	81/452	9%	20/229

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage	
org.apache.bookkeeper.bookie.storage.ldb	1	64%	45/70	26%	12/46
org.apache.bookkeeper.client	1	9%	36/382	4%	8/183

Report generated by PIT 1.5.1

Figura 6

Pit Test Coverage Report

Project Summary

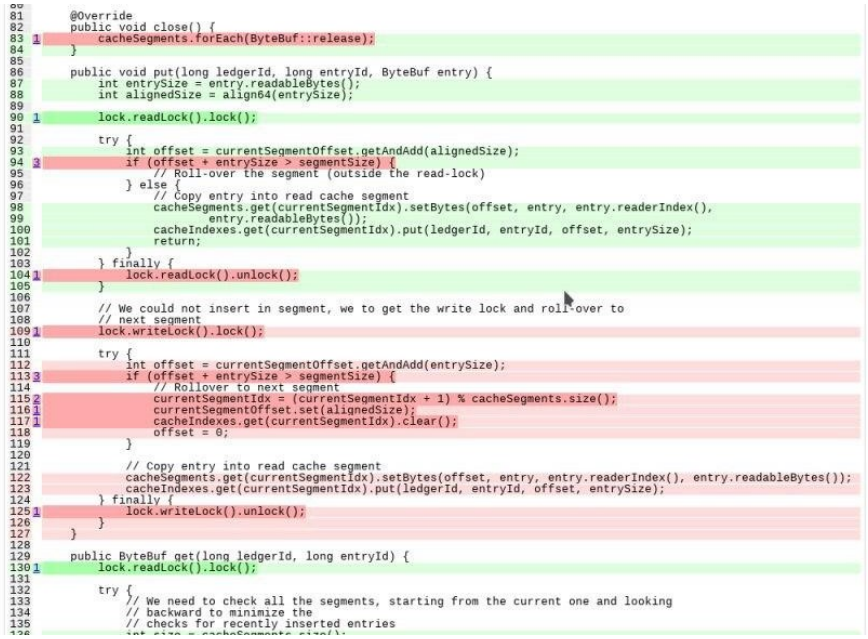
Number of Classes	Line Coverage		Mutation Coverage	
2	17%	77/452	8%	18/229

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage	
org.apache.bookkeeper.bookie.storage.ldb	1	59%	41/70	22%	10/46
org.apache.bookkeeper.client	1	9%	36/382	4%	8/183

Report generated by PIT 1.5.1

Figura 7



```
    @Override
    public void close() {
        cacheSegments.forEach(ByteBuf::release);
    }

    public void put(long ledgerId, long entryId, ByteBuf entry) {
        int entrySize = entry.readableBytes();
        int alignedSize = align64(entrySize);

        lock.readLock().lock();

        try {
            int offset = currentSegmentOffset.getAndAdd(alignedSize);
            if (offset + entrySize > segmentSize) {
                // Roll-over the segment (outside the read-lock)
            } else {
                // Copy entry into read cache segment
                cacheSegments.get(currentSegmentIdx).setBytes(offset, entry, entry.readerIndex(),
                        entry.readableBytes());
                cacheIndexes.get(currentSegmentIdx).put(ledgerId, entryId, offset, entrySize);
                return;
            }
        } finally {
            lock.readLock().unlock();
        }

        // We could not insert in segment, we to get the write lock and roll-over to
        // next segment
        lock.writeLock().lock();

        try {
            int offset = currentSegmentOffset.getAndAdd(entrySize);
            if (offset + entrySize > segmentSize) {
                // Rollover to next segment
                currentSegmentIdx = (currentSegmentIdx + 1) % cacheSegments.size();
                currentSegmentOffset.set(alignedSize);
                cacheIndexes.get(currentSegmentIdx).clear();
                offset = 0;
            }

            // Copy entry into read cache segment
            cacheSegments.get(currentSegmentIdx).setBytes(offset, entry, entry.readerIndex(), entry.readableBytes());
            cacheIndexes.get(currentSegmentIdx).put(ledgerId, entryId, offset, entrySize);
        } finally {
            lock.writeLock().unlock();
        }
    }

    public ByteBuf get(long ledgerId, long entryId) {
        lock.readLock().lock();

        try {
            // We need to check all the segments, starting from the current one and looking
            // backward to minimize the
            // checks for recently inserted entries
            int size = cacheSegments.size();
            for (int i = 0; i < size; i++) {
                int segmentIdx = (currentSegmentIdx + (size - i)) % size;

                LongPair res = cacheIndexes.get(segmentIdx).get(ledgerId, entryId);
                if (res != null) {
                    int entryOffset = (int) res.first;
                    int entryLen = (int) res.second;

                    ByteBuf entry = allocator.directBuffer(entryLen, entryLen);
                    entry.writeBytes(cacheSegments.get(segmentIdx), entryOffset, entryLen);
                    return entry;
                }
            }
        } finally {
            lock.readLock().unlock();
        }

        // Entry not found in any segment
        return null;
    }

    /**
     * @return the total size of cached entries
     */
    public long size() {
        lock.readLock().lock();

        try {
            long size = 0;
            for (int i = 0; i < cacheIndexes.size(); i++) {
                if (i == currentSegmentIdx) {
                    size += currentSegmentOffset.get();
                } else if (!cacheIndexes.get(i).isEmpty()) {
                    size += segmentSize;
                } else {
                    // the segment is empty
                }
            }

            return size;
        } finally {
            lock.readLock().unlock();
        }
    }

    /**
```

Figura 8

```
    public ByteBuf get(long ledgerId, long entryId) {
        lock.readLock().lock();

        try {
            // We need to check all the segments, starting from the current one and looking
            // backward to minimize the
            // checks for recently inserted entries
            int size = cacheSegments.size();
            for (int i = 0; i < size; i++) {
                int segmentIdx = (currentSegmentIdx + (size - i)) % size;

                LongPair res = cacheIndexes.get(segmentIdx).get(ledgerId, entryId);
                if (res != null) {
                    int entryOffset = (int) res.first;
                    int entryLen = (int) res.second;

                    ByteBuf entry = allocator.directBuffer(entryLen, entryLen);
                    entry.writeBytes(cacheSegments.get(segmentIdx), entryOffset, entryLen);
                    return entry;
                }
            }
        } finally {
            lock.readLock().unlock();
        }

        // Entry not found in any segment
        return null;
    }

    /**
     * @return the total size of cached entries
     */
    public long size() {
        lock.readLock().lock();

        try {
            long size = 0;
            for (int i = 0; i < cacheIndexes.size(); i++) {
                if (i == currentSegmentIdx) {
                    size += currentSegmentOffset.get();
                } else if (!cacheIndexes.get(i).isEmpty()) {
                    size += segmentSize;
                } else {
                    // the segment is empty
                }
            }

            return size;
        } finally {
            lock.readLock().unlock();
        }
    }

    /**
```

Figura 9

FormatUtils

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
format(long, String)		0%		0%	2	2	7	7	1	1
format(double, String)		0%		0%	2	2	7	7	1	1
parseDate(String, String)		0%		n/a	1	1	4	4	1	1
format(Date, boolean)		0%		n/a	1	1	1	1	1	1
clear()		0%		n/a	1	1	3	3	1	1
format(Date)		0%		n/a	1	1	1	1	1	1
format(long)		0%		n/a	1	1	1	1	1	1
format(double)		0%		n/a	1	1	1	1	1	1
parseDate(String)		0%		n/a	1	1	1	1	1	1
format(Date, boolean, String)		100%		100%	0	2	0	6	0	1
parseNumber(String, String)		100%		n/a	0	1	0	3	0	1
lambda$static$0()		100%		n/a	0	1	0	3	0	1
static {...}		100%		n/a	0	1	0	2	0	1
Total	82 of 130	36%	4 of 6	33%	11	16	26	40	9	13

Figura 10

JexlUtils

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
addFieldsToContext(Object, JexlContext)		0%		0%	8	8	17	17	1	1
lambda$addFieldsToContext$0(Object, Class)		0%		0%	7	7	16	16	1	1
lambda$addFieldsToContext$1(Object, JexlContext, Pair)		0%		0%	5	5	19	19	1	1
evaluate(String, JexlContext)		0%		0%	4	4	11	11	1	1
lambda$addPlainAttrsToContext$5(JexlContext, PlainAttr)		0%		0%	3	3	9	9	1	1
lambda$addAttrsToContext$3(JexlContext, Attr)		0%		0%	3	3	8	8	1	1
evaluateMandatoryCondition(String, Any, DerAttrHandler)		0%		n/a	1	1	4	4	1	1
addDerAttrsToContext(Any, DerAttrHandler, JexlContext)		0%		n/a	1	1	3	3	1	1
addAttrsToContext(Collection, JexlContext)		0%		n/a	1	1	2	2	1	1
addPlainAttrsToContext(Collection, JexlContext)		0%		n/a	1	1	2	2	1	1
lambda$addPlainAttrsToContext$4(PlainAttr)		0%		0%	2	2	1	1	1	1
lambda$addAttrsToContext$2(Attr)		0%		0%	2	2	1	1	1	1
lambda$addDerAttrsToContext$6(JexlContext, DerSchema, String)		0%		n/a	1	1	1	1	1	1
newJxltEngine()		0%		n/a	1	1	1	1	1	1
getEngine()		100%		100%	0	2	0	12	0	1
static {...}		100%		n/a	0	1	0	4	0	1
isExpressionValid(String)		100%		n/a	0	1	0	7	0	1
Total	416 of 497	16%	52 of 54	3%	40	44	92	115	14	17

Figura 11

Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage		Mutation Coverage	
2	23%	37/159	9%	7/78

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage	
org.apache.syncope.core.provisioning.api.jexl	1	20%	23/117	6%	3/51
org.apache.syncope.core.provisioning.api.utils	1	33%	14/42	15%	4/27

Report generated by PIT 1.5.1

Figura 12

```
92
93      public static boolean isExpressionValid(final String expression) {
94          boolean result;
95          try {
96              getEngine().createExpression(expression);
97              result = true;
98          } catch (JexlException e) {
99              LOG.error("Invalid jexl expression: " + expression, e);
100             result = false;
101         }
102
103 2       return result;
104     }
105
```

Figura 13

```
public static Number parseNumber(final String source, final String conversionPattern) throws ParseException {
    DecimalFormat df = DECIMAL_FORMAT.get();
    df.applyPattern(conversionPattern);
    return df.parse(source);
}
```

Figura 14

```
public static String format(final Date date, final boolean lenient, final String conversionPattern) {
    SimpleDateFormat sdf = DATE_FORMAT.get();

    if (conversionPattern == null) {
        sdf.applyPattern(SyncopeConstants.DEFAULT_DATE_PATTERN);
    } else {
        sdf.applyPattern(conversionPattern);
    }

    sdf.setLenient(lenient);

    return sdf.format(date);
}
```