Report Maccari Gianluca 0339210

Classe Cookie (Bookkeeper)

TestBookieHostIp

In questa classe viene testato il metodo isBookieHostCreatedFromIp(). Tale metodo non riceve parametri di input ma dalla documentazione emerge che il metodo analizza l'attributo "bookieId" dell'istanza della classe Cookie corrente. Tale attributo rappresenta il bookieHost, ovvero l'hostname con cui è stata creata l'istanza corrente della classe Cookie. Il metodo testato verifica se "bookieId" sia stato creato con un hostname o con un indirizzo IP. Inoltre, ritorna il valore true se il bookieHost è stato creato con un indirizzo IP mentre ritorna false se è stato creato con un hostname. Per questo i casi di test sono stati progettati in base all'attributo String bookieId.

Essendo bookieId una stringa, sono state individuate le seguenti classi di equivalenza: {stringa valida}, {stringa non valida}, {stringa vuota} e {null}.

Dalla documentazione si aspetta che solo la stringa valida ritorni il valore true, la stringa non valida e la stringa vuota ritornino false mentre per il valore null si aspetta che venga sollevata una istanza dell'eccezione NullPointerException.

I casi di test individuati sono riportati in tabella 1.

Dall'analisi di Jacoco emerge come i casi di test individuati, siano sufficienti per raggiungere instruction coverage e branches coverage pari a 1.

Anche dall'analisi di Ba-Dua emerge che la coverage è pari ad 1.

Dal report di Pit si nota come siano stati generati quattro mutazioni e i casi di test individuati riescano a uccidere tutte le mutazioni.

Si definisce un operational profile:

input: [string valida; stringa non valida; stringa vuota; null]

usage probability: [0.4; 0.4; 0.1; 0.1]

reliability: 0.9

(Nei profili operazionali riportati in tutto il report, il sollevamento dell'eccezione NullPointerException non gestito viene considerato come una failure. Questo poiché arresta l'esecuzione del programma e quindi si può osservare un crash. In questo report si assume il crash come una failure)

TestEncodeDirPaths

Test del metodo statico encodeDirPaths(String[] dirs), metodo privo di documentazione, quindi è stato utilizzato un approccio white box. Dall'analisi dell'implementazione, emerge che tale metodo riceve un array di stringhe che rappresentano il percorso di alcune directory, crea una stringa a cui accoda le stringhe dell'array e restituisce tale stringa.

Dato che il metodo riceve un array di stringhe, l'input è stato suddiviso nelle seguenti categorie: {array completo con stringhe valide}, {array tutte stringhe vuote}, {array con tutti null}, {null} e {array con metà delle stringhe valide e metà con stringa vuota e null}.

Dato che non ci sono vincoli sulla lunghezza dell'array non è stato possibile considerare una categoria con array che non rispettano vincoli di lunghezza.

Quindi i casi di test individuati sono quelli riportati nella tabella 2.

Dalle analisi di Jacoco e Ba-Dua,i casi di test individuati sono sufficienti per ottenere coverage pari a 1.

Dal report di Pit si nota come siano state generate due mutazioni che i casi di test individuati hanno ucciso.

Si definisce un operational profile:

input: [array con stringhe valide; array con stringhe vuote; array con null; array misto; null]

usage probability: [0.35; 0.1; 0.1; 0.35; 0.1]

reliability: 0.9

TestGenerateCookie

Il metodo in esame genera un cookie dalla configurazione data in input. Dalla documentazione si può notare che il metodo riceva in input una istanza della classa ServerConfiguration e restituisce in caso positovo una istanza della classe Builder, mentre in caso di errori viene sollevata l'eccezione UnknownHostException. tale eccezione viene sollevata in caso l'istanza di ServerConfiguration non sia correttamente impostata.

Poiché la classe ServerConfiguration possiede due costruttori, il primo di default e il secondo costruttore che riceve una istanza di AbstractConfiguration, una classe astratta estesa da ServerConfiguration e ClientConfiguration.

Per questo l'input è stato suddiviso in queste classi di equivalenza: {istanza valida creata con il costruttore di default}, {istanza valida con secondo costruttore}, {istanza non valida} e {null}.

Con istanza non valida si intende una istanza di ServerConfiguration che non sia stata impostata correttamente, in particolare non sia stato impostato il parametro "bookieId", e dunque si aspetta che il metodo in esame sollevi l'eccezione UnknownHostException.

Poiché il metodo sotto test ritorna una istanza di Builder, si aspetta che per le classi di istanze valide, il metodo ritorni una istanza di Builder, mentre per la classe di input null si aspetta venga sollevata una istanza dell'eccezione NullPointerException.

I casi di test sono dunque quelli riportati in tabella 3. (Nella tabella 3 così come in altre tabelle, come output atteso spesso sono inseriti dei booleani. Tali valori sono necessari ad assumere il comportamento di test corretto. Ovvero il valore di ouput delle tabelle differisce quello analizzato nel report ma l'implementazione segue gli output analizzati nel report)

Dal report di Jacoco e di Ba-Dua si nota che la coverage non è pari ad 1. Infatti, dal report di Jacoco (figura 1) si può osservare come la condizione dell'if a linea 338, non è mai soddisfatta, pertando non viene mai eseguito tale branch. Similmente, dal report di Ba-Dua (figura 2) possiamo notare come non siano stati coperti gli usi delle variabili conf a linea 338 e della variabile builder a linea 339. Quindi per coprire tali linee di codice e tali usi non è stato individuato il seguente caso di test: istanza valida ServerConfiguration di cui impostiamo anche le directory degli index, in tal modo la condizione dell'if viene rispettata. Per questo caso di test si attende e che l'esecuzione del metodo ritorni una istanza di Builder correttamente.

Dopo ulteriore analisi di Jacoco e Ba-Dua si può notare come la coverage sia pari ad 1. (P-use coverage iniziale era pari a 50% e c-use coverage era pari a 33%).

Dal report di Pit si può osservare che sono state generate due mutazioni e queste sono state uccise dai casi di test implementati.

Si definisce un operational profile:

input: [istanza valida con ServerConfiguration; instanza valida con ClientConfiguration; istanza non valida; null; istanza valida con directory index impostate]

usage probability: [0.3; 0.3; 0.1; 0.1; 0.2]

reliability: 0.8

(In questo metodo, le considerazioni fatta sull'eccezione NullPointerException nell'analisi della classe di test TestBookieHostIp, sono applicate all'eccezione UnknownHostException. Per questo anche questa il sollevamento di questa eccezione viene considerato una failure)

TestReadFromDirectory

Il metodo in esame ritorna un cookie data una directory. Il metodo readFromDirectory(File directory) riceve in input una istanza della classe File, che dalla documentazione, possiamo dedurre che sia il percorso di una directory in cui almeno un cookie sia stato scritto su file.

L’input è stato suddiviso quindi nelle seguenti classi di equivalenza: {directory valida}, {directory non valida}, {directory vuota} e {null}. Con directory vuota si intende che al metodo sotto test viene assegnato come input una istanza di File a cui viene passato il path “”, ovvero new File(“”).

Per una directory valida l’output atteso è la corretta creazione di una istanza di Cookie. Per il test è stato utilizzato un file temporaneo su cui viene prima salvato un Cookie e poi si verifica che la istanza di Cookie scritta sul file e letta dal metodo sotto test combacino. (Dati problemi tra Pit e operazioni di scrittura ed eliminazione file, non è stato possibile configurare un metodo di test con annotazione @After o @AfterClass che eliminasse il file temporaneo)

Per la directory non valida e la directory vuota si attende che il metodo non trovi una directory reale e quindi si attende che venga sollevata l’eccezione IOException.

Per il valore null si attende che venga sollevata l’eccezione NullPointerException.

I casi di test implementati sono riportati nella tabella 4.

Dall’analisi dei report di Jacoco e Ba-Dua, si nota che i casi di test sono sufficienti per raggiungere coverage pari a 1.

Dal report di Pit si nota che è stata generata una sola mutazione uccisa dai casi di test implementati.

Si definisce un operational profile:

input: [directory valida; directory non valida; directory vuota; null]

usage probability: [0.7; 0.1; 0.1; 0.1]

reliability: 0.7

(Anche per l’eccezione IOException che non viene gestita sono state applicate le stesse considerazioni delle eccezioni NullPointerException e UnknownHostException)

TestVerifyVerifySuperSet

In questa classe di test sono testati due metodi della classe Cookie: verify() e verifyIsSuperSet(). I due metodi sono stati testati nella stessa classe poiché hanno una simile implementazione, ovvero entrambi invocano l’esecuzione del metodo verifyInternal(), ma utilizzano un booleano diverso come parametro “checkIfSuperSet”.

Entrambi i metodi non possiedo documentazione, però inizialmente è stato utilizzato un approccio black box per poi passare ad un approccio white box in seguito. In particolare inizialmente è stato analizzato il metodo verifyIsSuperSet(). Per intuire la funzionalità di questo metodo è stata utilizzata anche la documentazione del metodo isSuperSet(), che verifica che uno dei due parametri di input che riceve, sia sottoinsieme del secondo parametro di input (i parametri di input sono array di stringhe). Come verrà detto più avanti, si è passati ad un approccio white box, e analizzando anche l’implementazione di verify(), si è intuito che i due metodi testati eseguono operazioni simili e necessitano della stessa configurazione. Dunque, a questo punto è stato deciso che entrambi i metodi potessero essere testati nella stessa classe di test.

Entrambi i metodi ricevono in input una istanza di Cookie. Le istanze di Cookie possono essere create solamente tramite la Classe Builder. Per tale motivo le classi di equivalenza sono state individuate anche sulla base degli attributi della classe Builder, i quali vengono acceduti durante la creazione di istanze di Cookie. Le classi di equivalenza individuate sono le seguenti: {istanza valida}, {istanza non valida} e {null}. Per istanza valida e non valida è stato effettuato category partitioning sugli attributi della classe Builder. Tali attributi sono: String bookieId, String journalDirs, String ledgerDirs, String instanceId, String indexDirs e int layoutVersion.

Per gli attributi di tipo stringa sono state individuate le seguenti classi di equivalenza: {stringa valida}, {stringa vuota} e {null}.

L’attributo layoutVersion di default è impostato al valore 5. Tale valore indica il numero della versione corrente, quindi invece di scegliere delle classi di equivalenza del tipo {<0} {>=0}, sono state scelte le seguenti classi di equivalenza: {<5}, {5}, {>5}. I valori di layoutVersion scelti per i casi di test sono: -1 e 0 per {<5}, 5 per {5} e 6 per {>5}. Per la classe {<5} sono stati scelti due valori per testare il comportamento del metodo sotto test in caso di layoutVersion (minore di 5) maggiore di zero e minore di zero.

Come combinazione dei valori scelti è stato scelto un approccio unidimensionale.

I valori di output attesi, ottenuti con un approccio black box, non erano corretti; perciò, si è passati ad un approccio white box per ottenere i corretti valori di output. I valori di input a questo punto sono stati modificati.

Dopo un'analisi dell'implementazione si è dedotto che l'istanza valida di Cookie comporta una corretta esecuzione del metodo, mentre le istanze non valide sollevano l'eccezione InvalidCookieException e NullPointerException. Mentre il valore null comporta l'eccezione NullPointerException.

Dunque, i casi di test implementati sono riportati nella tabella 5.

Dal report di Jacoco e Ba-Dua si nota che i metodi verify() e verifyIsSuperSet() hanno copertura pari a 1, mentre verifyInternal() non è totalmente coperto. Dal report di Pit si nota che per i metodi verify() e verifyIsSuperSet() è stata generata una mutazione a metodo, entrambe uccise dai casi di test implementati. Invece per il metodo verifyInternal() sono state generate tredici mutazioni di cui solo due sono state uccise.

Jacoco, segnala alcune linee di codice e alcune condizioni non coperte. La line coverage è del 60% e la branch coverage è del 50%. Per coprire le righe non coperte sono stati implementati i primi due casi di test della tabella 6.

Similmente il report di Ba-Dua segnala p-use coverage pari a 50%e c-use coverage pari a 33.3%. Quindi, alcuni usi delle varibili non sono stati coperti come: la variabile "c" definita a linea 154, non è stato coperto l'utilizzo a linea 158 (prima condizione della riga che usa "c") che porta all'esecuzione del ramo true dell'else if di riga 158, o l'utilizzo a linea 159 (seconda condizione della riga che utilizza "c") che porta all'esecuzione del ramo true dell'else if di riga 158 (vedere figura 3); oppure della variabile "checkIfSuperSet" definita a riga 154, non è stato coperto l'uso a riga 159 (prima condizione dell'else if che usa questa variabile) e l'uso a riga 160 (seconda condizione dell'else if che usa tale varibile) per finire nel branch positivo dell'else if di riga 158(vedere figura 4).

Per coprire i casi elencati sopra, è stato individuato l'ultimo caso di test della tabella 6. Anche questi casi di test sollevano l'eccezione InvalidCookieException, per questo l'output atteso è tale eccezione. Con questi ultimi casi di test la line coverage è del 97% e la branch coverage è del 59%. Dal report di Pit invece si può notare che con i casi di test aggiuntivi, le mutazioni uccise del metodo verifyInternal() sono cinque su tredici. Mentre dal nuovo report di Ba-Dua si nota che c-use coverage è pari a 1 e p-use è pari 62.5%.

Si definisce un operational profile (vedere tabella 5 e tabella 6):

input: [casoTest1; casoTest2; casoTest3; casoTest4; casoTest5; null; casoTest7; casoTest8; casoTest9]

usage probability: [0.6; 0.05; 0.05; 0.05; 0.05; 0.05; 0.05; 0.05; 0.05]

reliability: 0.6

(Sono state applicate le stesse considerazioni effettuate per le eccezioni delle classi di test precedente alle eccezioni sollevate dai casi di test non validi di questa classe)

ITDeleteFromRegistrationManager

Test di integrazione che è stato effettuato sul metodo deleteFromRegistrationManager(RegistrationManager rm, ServerConfiguration conf, Version version), il quale poi invoca anche il metodo deleteFromRegistrationManager(RegistrationManager rm, BookieId address, Version version).

In questi metodi la classe Cookie scambia messaggi con le classi BookieImpl e RegistrationManager. Della classe BookieImpl viene invocato il metodo statico getBookieId(ServerConfiguration conf), il quale ritorna il BookieId con cui è stato impostato il parametro di input conf di deleteFromRegistrationManager. Della classe RegistrationManager viene invocato il metodo removeCookie(BookieId bookieId, Version version).

Individuati gli scambi di messaggi, sono stati utilizzati dei mock che simulassero il comportamento atteso dagli scambi di messaggi.

Sono stati implementati tre metodi di test. allMockedTest() simula tutti i messaggi scambiati tramite mock; mockedRemoveCookieTest() simula solo l'invocazione del metodo removeCookie() di RegistrationManager; e almostCompleteTest() simula l'invocazione del metodo delete() di ZooKeper da parte di RegistrationManager.

(Per i test di integrazione non sono stati applicati i tool Ba-Dua e Pit. Poiché questi metodi utilizzano mock per simulare il comportamento di altre classi, per verificare la corretta esecuzione dei test sono stati usati i metodi verify() della libreria di Mockito. Inoltre è stato effettuato un controllo con Jacoco per verificare la copertura delle linee di codice in cui i metodi testati invocano metodi di altre classi. Controllo aggiuntivo ma superfluo)

ITWriteToRegistrationManager

Test di integrazione effettuato sul metodo writeToRegistrationManager(RegistrationManager rm, Server Configuration conf, Version version).

In questo metodo, la classe Cookie scambia messaggi con la classe BookieImpl e RegistrationManager. Della classe BookieImpl viene invocato il metodo statico getBookieId(ServerConfiguration conf), il quale ritorna il BookieId con cui è stato impostato il parametro di input conf di deleteFromRegistrationManager. Della classe RegistrationManager viene invocato il metodo writeCookie(BookieId bookieId, Versioned <byte[]> cookieData). Di conseguenza il comportamento di questi metodi è stato simulato con dei mock.

Sono stati implementati tre metodi di test: allMockedTest() simula tutti i messaggi scambiati tramite mock; mockedwriteCookieTest() simula solo l'invocazione del metodo writeCookie() di RegistrationManager; almostCompleteTest() simula l'invocazione del metodo setData() di ZooKeper da parte di RegistrationManager.

Classe ZKRegistrationManager (Bookkeeper)

La classe presa in considerazione scambia molti messaggi con altre classi; perciò, sono stati progettati e implementati dei test di integrazione.

ITInitNewCluster

Test di integrazione del metodo initNewCluster(). Tale metodo invoca le seguenti operazioni di altre classi: resolveZkServers() della classe ZKMetadataDriverBase; exists() e multi() della classe ZooKeeper; create() della classe Op; newLedgerManagerFactory() della classe AbstractZkLedgerManagerFactory.

Similmente, il metodo nukeExistingCluster() scambia messaggi con le classi elencate sopra. Quindi è stata implementata una classe astratta AbstractClusterTest che possiede i metodi per impostare i mock necessari e verificarli. Tale classe viene poi estesa dalla classe di test corrente e quella analizzata successivamente.

Tutti i metodi di classi esterne sono stati simulati tramite l'utilizzo di mock.

Sono stati implementati tre metodi di test: allMockedTest() che simula tutti gli scambi di messaggi tramite mock e verifica che i messaggi siano corretti tramite il metodo verify() della libreria Mockito; partialMockedTest() che non simula i metodi exists() di ZooKeeper e resolveZkServers() di ZKMetadataDriverBase; ed infine pureTest() che non simula alcun messaggio.

ITNukeExistingCluster

Test di integrazione del metodo nukeExistingCluster(). Tale metodo invoca le seguenti operazioni di altre classi: exists() della classe ZooKeeper; resolveZkServers() della classe ZKMetadataDriverBase. Anche questa classe di test estende la classe AbstractClusterTest.

Tutti i metodi di classi esterne elencati qui sopra sono stati simulati tramite l'utilizzo di mock.

Sono stati implementati due metodi di test: allMockedTest() che simula tutti gli scambi di messaggi; e pureTest() che non simula alcun messaggio.

ITRegisterBookie

Test di integrazione del metodo registerBookie(). Tale metodo invoca internamente i metodi doRegisterReadOnlyBookie() e doRegisterBookie(). Quindi questa classe di test copre i messaggi scambiati da questi tre metodi con altre classi, e tali messaggi invocano le seguenti operazioni: exists() e create() della classe ZooKeeper.

In questa classe di test sono stati individuati più di un caso di test, a differenza dei test di integrazione analizzati precedentemente. Il motivo è dovuto dal fatto che si è voluta testare la comunicazione dei metodi invocati internamente da registerBookie() (doRegisterReadOnlyBookie() e doRegisterBookie) con classi esterne, che però vengono invocati singolarmente in base ad un parametro di input di registerBookie(). In particolare,

il metodo registerBookie() riceve I seguenti parametri: una istanza bookieId della classe BookieId; un booleano readOnly; e una istanza bookieServiceInfo della classe BookieServiceInfo. “readOnly” è il parametro che decide quale dei due metodi invocare.

Poiché il focus di questa classe di test non è quella di individuare errori o problemi delle singole componenti software, sono stati implementati solo tre casi di test che non coprono tutte le classi di equivalenza dei tre parametri di input.

I casi di test individuati sono i seguenti: istanza di BookieId valida, false, istanza di BookieServiceInfo valida; istanza di BookieId valida, true, istanza di BookieServiceInfo valida; null,true,null.

Tutti i metodi di classi esterne elencati qui sopra sono stati simulati tramite l’utilizzo di mock.

Sono stati implementati due metodi di test: allMockedTest() che simula tutti gli scambi di messaggi; catchExceptionTest(), che a differenza del primo non simula l’esecuzione del metodo create() di ZooKeeper in caso il parametro readOnly sia impostato a false, quindi vengono generate delle eccezioni dalla classe ZooKeeper e il metodo di test verifica che siano state sollevate le eccezione aspettate.

Classe PreparedQueryCacheImpl (Openjpa)

TestCache

Test di unità del metodo cache(PreparedQuery q). Dalla documentazione si può comprendere che il metodo in questione salva nella lista di prepared query dell’istanza di PreparedQueryCacheImpl la PreparedQuery che riceve per input. Inoltre, nel farlo, verifica che non sia esclusa da un pattern Exclusion e che non sia marcata come non salvabile in cache. In caso sia trovato un pattern Exclusion che includa l’identificatore della query q, marca la query come non salvabile in cache. Il metodo ritorna il valore true se la query q è stata inserita nella cache e false altrimenti.

Sempre dalla documentazione si può comprendere che la classe Exclusion rappresenta una struttura che descrivere il livello di esclusione e il motivo per cui una query non possa essere salvata in cache.

Il metodo sotto test riceve solo un parametro in input che è una istanza della classe PreparedQuery; quindi, una prima suddivisione in classi di equivalenza è la seguente: {istanza valida}, {istanza non valida} e {null}. Con le informazioni ottenute dalla documentazione, la classe di equivalenza {istanza non valida} è stata estesa in questo modo: {istanza “uncachable”} e {istanza esclusa}. L’istanza “uncachable” è una istanza che è stata marcata come non salvabile in cache tramite operazione markUncachable() e l’istanza esclusa è una istanza per cui esiste un patter Exclusion creato e aggiunto alla configurazione della cache tramite operazione setExcludes().

Per i parametri di output si attendono: valore true per istanza valida; valore false per istanza “uncachable” e istanza esclusa; e NullPointerException per il valore di input null.

Inoltre, per verificare il corretto inserimento della istanza valida nella cache e non inserimento delle istanze non valide, è stato utilizzato il metodo get(String id), il quale ritorna l’istanza di PreparedQuery corrispondente all’identificatore id o null altrimenti. Quindi nei casi di test è incluso un ulteriore booleano utile a verificare l’esito del metodo get().

Dunque, i casi di test sono quelli riportati in tabella 7.

Dalle analisi di Jacoco si può osservare che line coverage è pari a 67% e branch coverage è pari a 50%.

Dalle analisi di Ba-Dua (figura 5) si può osservare come non siano stati coperti tutti gli usi delle varibili del metodo cache(), infatti la c-use coverage è pari a 95% e la p-use coverage è pari a 36.6%.

Dal report di Pit si nota che sono state generate quindici mutazioni e otto di esse sono state uccise dai casi di test implementati.

Si è scelto di coprire tutti gli usi delle varibili “id” e “_loc”. Gli utilizzi non coperti della varibile “id” sono a linea 130 e 140, dove viene utilizzato come parametro di input per il metodo get() ed a linea 124, dove viene utilizzato come parametro di input per il metodo containsKey(). Quest’ultimo utilizzo però è relativo al caso in cui la condizione dell’if viene rispettata e si prosegue con l’esecuzione della linea 125.

Per l’utilizzo della variabile “id” a linea 124 è stato individuato il caso di test riportato in tabella 7.1, dove tale istanza è stata precedentemente inserita in cache in fase di set up.

Per gli utilizzi della variabile “_loc” e i restanti usi della varibile “id”, è stato sufficiente impostare l’istanza di PreparedQueryCacheImpl con ulteriori metodi. In particolare, è stata impostata l’attributo “_log” usando un mock che ritornasse una istanza di LogImpl impostata in maniera tale da raggiungere le linee di codice non coperte 130 e 140.

Dopo aver implementato il caso di test aggiuntivo ed aver impostato ulteriormente l’istanza di PreparedQueryCacheImpl sotto test, il report di Jacoco indica line coverage pari a 1 e branch coverage pari al 78%.

Invece dal report di Ba-Dua (figura 6) si può notare che tutti gli usi delle variabili “id” e “_loc” sono stati coperti. La p-use coverage è pari a 60% e la c-use coverage è pari a 1.

Infine, dal report di Pit si nota che le mutazioni uccise dopo le modifiche apportate sono nove, una in più rispetto alla generazione del primo report.

Si definisce un operational profile (vedere tabella 7):

input: [casoTest1; casoTest2; casoTest3; casoTest4]

usage probability: [0.3; 0.3; 0.3; 0.1]

reliability: 0.9

TestIsCachable

Test di unità del metodo isCachable(String id). Dalla documentazione si comprende che tale metodo verifica che la PreparedQuery con identificatore id sia salvabile nella cache. Ritorna il valore true se tale query è salvabile in cache oppure false se la query è marcata come non salvabile o se esiste un pattern Exclusion che non permette il salvataggio della query in cache. Il metodo può ritornare anche null in caso non si riesca a stabilire se l’identificatore indica una query che si può salvare in cache.

Il metodo riceve un unico parametro di input di tipo String; quindi, la suddivisione in classi di equivalenza iniziale è la seguente: {stringa valida}, {stringa non valida} e {null}. Dove {stringa valida} comprende tutte le stringhe che rappresentano l’identificatore di una istanza di PreparedQuery salvata nella cache, oppure di una query segnalata come non salvabile nella cache. Con {stringa non valida} sono state identificate tutte quelle stringhe per cui non esiste istanza che sia presente nella cache o sia non salvabile nella cache.

Quindi la classe di stringhe valide è stata estesa in questo modo: {stringa di query in cache}, {stringa di query “uncachable”} e {stringa di query esclusa}.

I valori di output attesi inizialmente erano i seguenti: true per {stringa di query in cache}; false per {stringa di query “uncachable”}; false per {stringa di query esclusa}; false per {stringa non valida}; NullPointerException per {null}. In fase di test si è potuto notare che gli output attesi per le stringhe di query escluse e per le stringhe non valide non corrispondeva al valore restituito dal metodo, ovvero veniva sollevata una NullPointerException. Dopo un’analisi dell’implementazione e la consultazione della documentazione del metodo addExclusionPattern(), si è compreso che i valori attesi di queste due classi di equivalenza devono essere null. Il metodo però deve ritornare un booleano, quindi viene sollevata l’eccezione. Il problema è dovuto, in parte, anche dal comportamento di addExclusionPattern(), il quale cambia in base alla presenza in cache della query da escludere. Il caso di test individuato esclude l’identificatore di una query che non è in cache e quindi tale query non viene segnalata con “uncachable”. Di conseguenza non viene restituito false ma null.

Quindi i casi di test finali sono riportati in tabella 8.

Dalle analisi di Jacoco e Ba-dua si evince che i casi di test sono sufficienti per raggiungere coverage pari ad 1. Dal report di Pit si nota che sono state generate sette mutazioni di cui cinque sono state uccise, una è finita in time_out e l’ultima è sopravvissuta. Per uccidere la mutazione sopravvissuta è stato aggiunto un assertFalse() nel caso in cui venga sollevata l’eccezione NullPointerException. Il motivo di questo inserimento è dovuto dal fatto che la mutazione sopravvissuta differisce dal SUT originale nell’oggetto ritornato in caso di stringa valida; infatti, la mutazione invece di tornate true ritorna una istanza di NullPointerException. In questo modo le mutazioni uccise sono sei su sette.

Si definisce un operational profile (vedere tabella 8):

input: [“valid”; “uncachable”; “excluded”; “not valid”; null]

usage probability: [0.3; 0.3; 0.2; 0.1; 0.1]

reliability: 0.6

(Il sollevamento dell'eccezione NullPointerException è stato considerato come failure)

TestMarkUncachable

Test di unità del metodo markUncachable(String id, Exclusion exclusion). Dalla documentazione si comprende che questo metodo marca la stringa id, che rappresenta l'identificatore di una query, con non salvabile nella cache. Il parametro exclusion viene utilizzato per specificare se l'esclusione di tale identificatore è permanente oppure no. Inoltre, il metodo ritorna l'istanza di PreparedQuery marcata come non salvabile in cache se questa era presente in cache, null altrimenti.

Il metodo riceve due parametri in input: String id che corrisponde all'identificatore di una query ed Exclusion exclusion.

L'identificatore id può far riferimento ad una query già presente in cache oppure ad una query non in cache, per questo le classi di equivalenza individuate sono: {stringa valida query in cache}, {stringa valida query non in cache}, {stringa vuota} e {null}.

Poiché la classe Exclusion è una interfaccia che viene implementata dalla classe astratta ExclusionPattern estesa poi da StrongExclusion e WeakExclusion, le classi di equivalenza individuate sono: {istanza valida StrongExclusion}, {istanza valida WeakExclusion}, {istanza non valida} e {null}. Come istanza non valida è stata scelta una istanza di WeakExclusion con attributi impostati a null.

Come metodo di combinazione dei valori di input è stato utilizzato un approccio multidimensionale.

I valori di output attesi sono: il valore della istanza di PreparedQuery in cache per ogni combinazione con identificatore di query in cache; null per le combinazioni con identificatore di query non presente in cache e stringa vuota ma con exclusion valido; NullPointerException per le combinazioni con identificatore null o con istanza exclusion null o non valida. Tuttavia, in fase di test si è potuto notare che in nessun caso di test viene sollevata eccezione, in particolare per ogni caso non valido il metodo sotto esame ritorna null e marca come non salvabile in cache l'identificatore passato come input, anche il valore null. Quindi i valori di output finali sono stati modificati dopo analisi dell'implementazione.

I casi di test finali implementati sono riportati nella tabella 9.

Dalle analisi di Jacoco si nota che line coverage è pari a 86% e branch coverage è pari a 60%.

Dal report di Ba-Dua (figura 7) emerge che i casi di test non sono sufficienti per coprire tutti gli usi delle variabili definite nel metodo sotto test, in particolare la p-use coverage è pari a 63.6% e la c-use coverage è pari a 76.9%.

Si è dunque scelto di aumentare la coverage coprendo le linee di codice mancanti (linea 211) e di coprire tutti gli usi mancanti delle varibili: "_statsEnabled" a linea 210 (condizione con esito true); e "pq" a linea 210 sia per finire nel branch true sia per finire nel branch false dell'if di linea 210.

Per soddisfare questi obiettivi è stato sufficiente aggiungere nel set up dell'ambiente di una configurazione aggiuntiva dell'istanza di PreparedQueryCacheImpl, ovvero è stato impostato l'attributo "_statsEnabled" al valore true.

Con questa modifica si è raggiunta line coverage pari a 1 e branch coverage pari a 90%. Invece, dal report di Ba-Dua (figura 7.1) possiamo notare come tutti gli utilizzi delle variabili "_statsEnabled" e "pq" sono stati coperti ed in particolare la c-use coverage è pari a 1 e la p-use coverage è pari a 1.

Per raggiungere branch coverage pari ad 1, è stato implementato un caso di test che copra l'ultimo branch rimasto. Tale caso di test è il seguente: "notInCache" per l'input String id e una istanza di WeakExclusion valida come istanza di Exclusion. Inoltre, in fase di set up, l'identificatore "notInCache" è stato marcato come "uncachable" e come pattern di esclusione è stata associata un'istanza della classe StrongExclusion. In questo modo la condizione dell'if a linea 205 assume valore false e viene coperta anche tale condizione. Infatti, dal report di Jacoco si osserva che branch coverage assume valore 1.

Dal report di Pit si nota che sono state generate dieci mutazioni e tre sono state uccise.

Si definisce un operational profile (vedere tabella 9):

input: [c1; c2; c3; c4; c5; c6; c7; c8; c9; c10; c11; c12; c13; c14; c15; c16]

usage probability: [0.1; 0.05; 0.05; 0.05; 0.1; 0.05; 0.05; 0.05; 0.1; 0.05; 0.05; 0.05; 0.1; 0.05; 0.05; 0.05]

reliability: 1

TestRegister

Test di unità del metodo register(String id, Query query, FetchConfiguration hints). Dalla documentazione si comprende che il metodo sotto test registra la query passata come input nella cache con identificatore id. Se tale query non può essere inserita nella cache viene marcata come “uncachable”. Il metodo ritorna un booleano con valore true se la query è stata inserita nella cache, false se la query non può essere inserita in cache poiché marcata come “uncachable” o poiché il parametro hints segnala di ignorare la versione della query.

Per problemi dovuti alla configurazione dei parametri Query query e FetchConfiguration hints, si è passati da un approccio black box ad un approccio white box.

Il primo parametro di input è una stringa e come per i metodi analizzati sopra, tale stringa rappresenta l’identificatore di una query; quindi, le classi di equivalenza individuate sono: {stringa valida query in cache}, {stringa valida query non in cache}, {stringa vuota} e {null}.

Il secondo parametro di input è una istanza di Query query. Query è un’interfaccia implementata dalle classi: QueryImpl, DistributedQueryImpl e DelegatingQuery. Quindi come possibili classi di equivalenza sono state scelte le seguenti classi: {istanza valida QueryImpl}, {istanza valida DelegetingQuery} e {null}. Non sono state identificate possibili istanze non valide. La classe DistributedQueryImpl non è stata presa in considerazione per evitare dipendenze circolari.

Il terzo parametro di input è una istanza della classe FetchConfiguration hints. Questa, è un’interfaccia implementata dalle classi: FetchConfigurationImpl, DelegatingFetchConfiguration e JDBCFetchConfigurationImpl. Quindi le classi di equivalenza individuate sono le seguenti: {istanza valida FetchConfigurationImpl}, {istanza valida DelegatingFetchConfiguration}, {istanza valida JDBCFetchConfigurationImpl} e {null}. Anche per questo parametro non sono state individuate possibili istanze non valide.

Per la combinazione dei valori di input è stato scelto un approccio unidimensionale.

I valori di output attesi sono: true per le combinazioni con stringa valida di query non in cache e con istanza query valida; false per tutte le combinazioni con stringa id null e/o con istanza query null; null per tutte le combinazioni con stringa di query in cache e istanza query valida.

I casi di test implementati sono riportati in tabella 10.

Dal report di Ba-Dua (figura 8) si può notare come alcuni usi delle variabili non siano stati coperti dai casi di test, in particolare la p-use coverage è pari a 62.5% e la c-use coverage è pari a 85%.

Le analisi di Jacoco indicano che line coverage è pari a 96% e branch coverage è pari a 68%. In particolare, non sono stati coperte le condizioni (con esito true) delle linee 82-83-84-85. Identificando dei casi di test che coprono queste condizioni, si possono coprire anche gli usi delle variabili: “query” a linea 82 e 83 (esito true delle condizioni); “hints” linea 84 e 85 (esito true delle condizioni).

Dal report di Pit si nota che sono state generate dodici mutazioni e undici sono state uccise.

I casi di test individuati per aumentare la coverage sono riportati in tabella 10.1 .

Dalla generazione dei nuovi report di Jacoco si nota come line coverage non sia variata (come atteso) mentre branch coverage è pari a 93%. Mentre, dal nuovo report di Ba-Dua (figura 9) si può notare che gli usi riportati sopra sono stati coperti, in particolare la c-use coverage è ancora pari a 85% mentre la p-use coverage è diventata pari a 87.5%.

I casi di test aggiuntivi non uccidono l’unica mutazione sopravvissuta; quindi, per questo è stato implementato un ulteriore caso di test in cui l’identificatore corrisponde alla chiave di una query marcata come “uncachable”. In questo modo il metodo register() deve tornare false. La mutazione però altera il valore di ritorno in questo caso (query marcata come “uncachable”) e ritorna un’istanza dell’eccezione RunTimeException.

Generando il report di Pit con questo ultimo caso di test, si nota che le mutazioni sono state tutte uccise. Inoltre, il caso di test individuato per Pit, ha aumentato le line e branch coverage (attualmente pari ad 1). Da un’ulteriore analisi di Ba-Dua (figura 9.1) si nota che tutti gli usi sono stati coperti, quindi i valori finali di p-use coverage e c-use coverage sono pari ad 1.

Si definisce un operational profile (vedere tabella 10):

input: [c1; c2; c3; c4; c5; c6; c7; c8; c9; c10; c11; c12]

usage probability: [0.1; 0.1; 0.1; 0.1; 0.1; 0.1; 0.1; 0.1; 0.05; 0.05; 0.05; 0.05]

reliability: 1

TestRemoveExclusionPattern

Test di unità del metodo removeExclusionPattern(String pattern). Dalla documentazione si comprende che il metodo sotto test rimuove il pattern, passato come input, dalla lista delle esclusioni. Inoltre, se una query oggetto del pattern è stata marcata come “uncachable”, verrà eliminata dalla lista delle query “uncachable”.

Il metodo riceve un solo parametro di tipo stringa, che rappresenta l’attributo pattern di una istanza della classe Exclusion; quindi, come classi di equivalenza sono state individuate le seguenti: {stringa valida di pattern esistente}, {stringa valida di pattern non esistente}, {stringa vuota} e {null}. Come classe di stringa non valida è stata scelta la seconda classe dell’elenco.

Non ci sono valori di output poiché il metodo non ritorna parametri, però si attendono degli effetti dovuti dall’esecuzione del metodo. In particolare, per la classe {stringa valida pattern esistente}, si aspetta che tale pattern sia rimosso dalla lista di esclusione della cache, di conseguenza viene controllata tale lista.

Per la stringa null invece si attende che venga sollevata l’eccezione NullPointerException.

I casi di test sono dunque quelli riportati nella tabella 11.

Dalle analisi di Jacoco si nota come i casi di test non siano sufficienti a raggiungere coverage pari ad 1.

Dal report di Ba-Dua (figura 10) si nota come la p-use coverage sia pari a 45.4% e la c-use coverage sia pari a 37.5%.

Per aumentare line e branch coverage (inizialmente pari a 78% e 50% rispettiamente), è stata aggiunta un’ulteriore impostazione all’istanza di PreparedQueryCacheImpl di test (similmente alla classe di test TestCache) attivata solo per un nuovo caso di test. Il nuovo caso di test è identico al caso di stringa valida, con la differenza che in più imposta l’attributo “_log” manualmente. In questo modo viene coperta l’unica riga di codice e delle condizioni non coperte, ed inoltre, in questo modo si coprono anche gli utilizzi delle variabili: “_log”; “_loc”; “pattern”, “reborKey”.

Dall’analisi di Jacoco si nota che la line coverage è pari a 1 e la branch coverage è pari a 83%.

Dall’analisi di Ba-Dua (figura 11) si nota che molti degli usi delle variabili elencate sopra sono stati coperti, in particolare la p-use coverage è diventata pari a 81.8% e la c-use è diventata pari a 1.

Dal report di Pit si nota che sono state generate cinque mutazioni di cui due sono state uccise.

Si definisce un operational profile (vedere tabella 11):

input: [“testId”; “notInList”; “”; null]

usage probability: [0.4; 0.4; 0.1; 0.1]

reliability: 0.9

TestSetExcludes

Test di unità del metodo setExclusion(String excludes). Dalla documentazione si comprende che il metodo sotto test imposta una o più regola di esclusione (in caso di più di una regola, queste devono esser divise da “;”). Se una delle regole indica un identificatore salvato in cache, tale identificatore viene marcato come “uncachable”.

Il metodo riceve un solo parametro di input di tipo stringa. Inizialmente sono state identificate le seguenti classi di equivalenza: {stringa valida}, {stringa non valida}, {stringa vuota} e {null}. Successivamente sulla base della documentazione si estende la classe {stringa valida} in stringhe con più di una regola e stringhe con una regola. Le regole possono riferirsi a identificatori presenti e/o non in cache. Dalla documentazione non emergono possibili valori di stringa non valida; quindi, come stringa non valida si identifica una stringa

con regole ripetute. Quindi, le classi di equivalenza finali sono le seguenti: {stringa valida con più regole}, {stringa valida con una regola}, {stringa vuota}, {stringa regole ripetute} e {null}.

Il metodo non restituisce valori in output, tuttavia si attendono degli effetti dovuti dall'esecuzione del metodo, in particolare per le stringhe valide. In particolare, si aspetta che eventuali identificatori presenti in cache riferiti dalla stringa excludes, siano marcati come "uncachable" ed inoltre che la lista di esclusioni contenga le nuove esclusioni. Per la stringa null inizialmente si attendeva che venisse sollevata l'eccezione NullPointerException, ma in fase di test si è riscontrato che il metodo semplicemente ritorna senza fare nulla, per tanto l'output attesa è stato modificato. Per la stringa con regole ripetute inizialmente si è pensato che sollevasse una eccezione, invece dopo l'esecuzione dei test, si è potuto notare che anche tali stringhe sono valide.

Per i casi di test implementati si è scelto di includere sia una stringa della prima classe con regole che fanno tutte riferimento a identificatori presenti in cache sia ad una stringa con regole miste (alcune fanno riferimento a identificatori presenti in cache le restanti no). Similmente per la stringa con una sola regola.

I casi di test implementati sono riportati in tabella 12.

Dalle analisi di Jacoco e Ba-Dua si evince che i casi di test individuati sono sufficienti per avere coverage pari ad 1.

Dal report di Pit si nota che sono state generate quattro mutazioni di cui tre sono state uccise.

Si definisce un operational profile (vedere tabella 12):

input: ["testId2;testId3;testId4"; "testId;notInCache;test3"; "testId"; "notInCache"; "testId;testId"; ""; null]

usage probability: [0.2; 0.2; 0.2; 0.1; 0.1; 0.1; 0.1]

reliability: 1

Classe NativeJDBCSeq (Openjpa)

Come per la classe ZKRegistrationManager, la classe presa in considerazione scambia molti messaggi con altre classi; perciò, sono stati progettati e implementati dei test di integrazione.

Anche per i seguenti test di integrazione non sono stati applicati i tool Ba-Dua e Pit, ma sono stati utilizzati soltanto metodo verify() di Mockito e Jacoco.

ITAbstractNext

In questa classe di test si è voluto implementare un test di integrazione sui metodi privati nextInternal() e allocateInternal(). L'unico modo per eseguire tali metodi senza alterare la classe è invocare il metodo next() della classe astratta AbstractJDBCSeq, classe che viene estesa da NativeJDBCSeq. Infatti il metodo next(), implementato nella classe astratta, invoca internamente il metodo nextInternal() e quest'ultimo invoca il metodo allocateInternal().

Il metodo next() riceve come parametri di input due istanze: StoreContext ctx e ClassMetaData meta. Inoltre, invoca l'esecuzione del metodo getStoreManager() della classe DelegatingStoreManager e il metodo getInnermostDelegete() della classe StoreManager.

Il metodo nextInternal() riceve come parametri di input due istanze: JDBCStore store e ClassMapping mapping. Questo metodo non scambia messaggi con altre classi, invoca soltanto l'esecuzione del metodo allocateInternal().

Infine il metodo allocateInternal() riceve i seguenti parametri di input: int additional, JDBCStore store e ClassMapping mapping. Questo metodo invoca anche i seguenti metodi: getDBDictionaryInstance() e getLog() di JDBCConfiguration, isWarnEnabled() di Log. Inoltre, invoca anche altri metodi della classe NativeJDBCSeq getSequence(), getConnection(), updateSql() e closeConnection().

Tutti i metodi di classi esterne elencati qui sopra sono stati simulati tramite l'utilizzo di mock.

ITAddSchema

Test di integrazione del metodo addSchema(). Tale metodo riceve come input le istanze ClassMapping mapping e SchemaGroup group. Inoltre, invoca i seguenti metodi di altre classi: getPath() e getSchemaName()

di QualifiedDBIdentifier; isKnownSequence(), getSchema() e addSchema() di SchemaGroup; importSequence() di Schema.

Tutti i metodi di altre classi sono stati simulati tramite l'utilizzo di mock.

Sono stati implementati due metodi: mockedTest() che simula tutti gli scambi di messaggi tramite l'utilizzo di mock; testAddSchema() che non simula alcuno scambio di messaggi ed usa un solo caso di test con null come istanza di ClassMapping e con una istanza valida di SchemaGroup.

ITEndConfiguration

Test di integrazione del metodo endConfiguration(). Tale metodo invoca internamente i metodi: getNewTableSchemaIdentifier di Schemas; getFullName() di DBDictionary; getDBDictionaryInstance() di JDBCConfiguration.

Tutti i metodi di altre classi sono stati simulati tramite l'utilizzo di mock.

ITRefreshDropSequence

Test di integrazione dei metodi refreshSequence() e dropSequence(). Tali metodi sono stati testati insieme poiché la configurazione dell'ambiente di test è la stessa.

Entrambi i metodi invocano internamente i metodi: getLog(), getDataSource2() e getDBDictionaryInstance() di JDBCConfiguration; isInfoEnabled() di Log; getCreateSequenceSQL() e getDropSequenceSQL() di DBDictionary. In aggiunta, il metodo refreshSequence() invoca anche il metodo getCreateSequenceSQL della classe DBDictionary mentre il metodo dropSequence() invoca anche il metodo getDropSequenceSQL() di DBDictionary.

Tutti i metodi elencati sono stati simulati tramite l'utilizzo di mock.

Tabella 1

	Valore atteso	Input
Stringa valida	true	“160.160.160.160:800”
Stringa non valida	false	“test:test”
Stringa vuota	false	“”
Null	NullPointerException	null

Tabella 2

	Valore atteso	Input
Array con stringhe valide	“4\npath1\npath2\npath3\npath4\n“	{“path1”, “path2”, “path3”, “path4”}
Array con stringhe vuote	“4\n\n\n\n\n“	{“”, “”, “”, “”}
Array con null	“4\nnull\nnull\nnull\nnull\n“	{null, null, null, null}
Array misto	“4\npath1\npath2\n\nnull\n“	{“path1”, “path2”, “”, null}
Null	NullPointerException	null

Tabella 3

	Valore atteso	Input
Istanza valida	true	new ServerConfiguration() //viene invocato setBookieId()
Istanza valida	true	new ServerConfiguration(new ClientConfiguration) //viene invocato setBookieId()
Istanza non valida	UnknownHostException	new ServerConfiguration() //non viene invocato setBoo-kieId()
Null	NullPointerException	null

Tabella 4

	Valore atteso	Input
Directory valida	True	New File(**currDir**+ "/fortest/")
Directory non valida	false	new File("/bookkeeper-server/src/test/testFile")
Directory vuota	false	new File(“”)
Null	false	null

currDir corrisponde alla directory del modulo bookkeeper-server.

Tabella 5

	Valore atteso	bookieId	journalDirs	ledgerDirs	instanceId	indexDirs	layoutVersion
Istanza valida	“ok”	“bookieId”	“journal”	“ledger”	“instance”	“index”	6

Istanza non valida	“null”	null	“”	null	null	“index”	5
Istanza non valida	“too old”	null	null	“ledger”	“”	null	0
Istanza non valida	“too old”	null	null	“”	“instance”	“”	-1
Istanza non valida	“null”	null	null	null	“”	“index”	6
Null	“null”	null	/	/	/	/	/

Tabella 6

	Valore atteso	bookieId	journalDirs	ledgerDirs	instanceId	indexDirs	layoutVersion
Istanza Non valida	“not matching”	“bookieId”	“journal”	“ledger”	“instance”+”2”	“index”	6
Istanza non valida	“not matching”	“bookieId”+”2”	“journal”+”2”	“ledger”	“instance”+”2”	“index”	5
Istanza non valida	“not matching”	“bookieId”+”2”	“journal”	“ledger”	“instance”	“index”	6

Tabella 7

	Valore atteso	Istanza
Istanza valida	true	new PreparedQueryImpl("testId","testSql",null)
Istanza “uncachable”	false	new PreparedQueryImpl("unCache","testSql",null) //invocato metodo markUncachable(“unCache”)
Istanza esclusa	false	new PreparedQueryImpl("excluded","testSql",null) //invocato metodo setExcludes(“excluded”)
Null	NullPointerException	null

Tabella 7.1

	Valore atteso	Istanza
Istanza valida	false	new PreparedQueryImpl("cached","testSql",null) //invocato il metodo cache

Tabella 8

	Valore atteso	Input
Stringa di query in cache	true	"valid" //inserita la query in cache
Stringa di query “uncachable”	false	"uncachable" //marcato identificato con markUncachable()
Stringa di query esclusa	NullPointerException	"excluded" //escluso identificatore con setExcludes()

Stringa di query non in cache, non “uncachable”, non esclusa	NullPointerException	"not valid"
Null	NullPointerException	null

Tabella 9

Valore atteso	String id	Exclusion exclusion
PreparedQuery	“testId”	new StrongExclusion()
null	"notInCache"	new StrongExclusion()
null	“”	new StrongExclusion()
null	null	new StrongExclusion()
PreparedQuery	“testId”	new WeakExclusion()
null	"notInCache"	new WeakExclusion()
null	“”	new WeakExclusion()
null	null	new WeakExclusion()
PreparedQuery	“testId”	Invalid instance
null	"notInCache"	Invalid instance
null	“”	Invalid instance
null	null	Invalid instance
PreparedQuery	“testId”	null
null	"notInCache"	null
null	“”	null
null	null	null

Tabella 10

Valore atteso	String id	Query query	FetchConfiguration hints
null	“testId”	new QueryImpl()	null
true	"notInCache"	new QueryImpl()	new JDBCFetchConfigurationImpl()
true	“”	new QueryImpl ()	new FetchConfigurationImpl()
false	null	new QueryImpl ()	new DelegatingFetchConfiguration()
false	“testId”	null	new FetchConfigurationImpl()
false	"notInCache"	null	new JDBCFetchConfigurationImpl()
false	“”	null	new DelegatingFetchConfiguration()
false	null	null	null
null	“testId”	new DelegatingQuery()	new JDBCFetchConfigurationImpl()
true	"notInCache"	new DelegatingQuery()	new DelegatingFetchConfiguration()
true	“”	new DelegatingQuery()	new FetchConfigurationImpl()
false	null	new DelegatingQuery()	new FetchConfigurationImpl()

Tabella 10.1

Valore atteso	String id	Query query	FetchConfiguration hints
false	“testId”	new QueryImpl()	new FetchConfigurationImpl() //settato il primo hint
false	“testId”	new QueryImpl()	new FetchConfigurationImpl() //settato il secondo hint
false	“testId”	new QueryImpl () //settato il linguaggio openjpa.SQL	new FetchConfigurationImpl() //settato il secondo hint
false	“testId”	new QueryImpl () //settato il linguaggio openjpa.MethodQL	new FetchConfigurationImpl() //settato il secondo hint

Tabella 11

	Valore atteso	Istanza
Stringa valida pattern esistente	true	“testId”
Stringa valida pattern non esistente	false	“notInList”
Stringa vuota	false	“”
Null	NullPointerException	null

I valori true e false sono utili ad assumere il comportamento corretto del test. Tale comportamento corrisponde all’output atteso descritto nella descrizione del caso di test.

Tabella 12

	Valore atteso	Istanza
Stringa valida regole multiple	true	"testId2;testId3;testId4"
Stringa valida regole multiple miste	true	"testId;notInCache;test3"
Stringa valida regola singola	true	"testId"
Stringa valida regola singola non in cache	true	"notInCache"
Stringa regole ripetute	true	"testId;testId"
Stringa vuot	false	“”
null	false	null

I valori true e false sono utili ad assumere il comportamento corretto del test. Tale comportamento corrisponde all’output atteso descritto nella descrizione del caso di test.

Figura 1

```
331.    public static Builder generateCookie(ServerConfiguration conf)
332.            throws UnknownHostException {
333.        Builder builder = Cookie.newBuilder();
334.        builder.setLayoutVersion(CURRENT_COOKIE_LAYOUT_VERSION);
335.        builder.setBookieId(BookieImpl.getBookieId(conf).toString());
336.        builder.setJournalDirs(Joiner.on(',').join(conf.getJournalDirNames()));
337.        builder.setLedgerDirs(encodeDirPaths(conf.getLedgerDirNames()));
338.        if (null != conf.getIndexDirNames()) {
339.            builder.setIndexDirs(encodeDirPaths(conf.getIndexDirNames()));
340.        }
341.        return builder;
342.    }
```

Figura 2

```
-<method name="generateCookie" desc="(Lorg/apache/bookkeeper/conf/ServerConfiguration;)Lorg/apache/bookkeeper/bookie
 /Cookie$Builder;">
   <du var="conf" def="333" use="338" target="339" covered="0"/>
   <du var="conf" def="333" use="338" target="341" covered="1"/>
   <du var="conf" def="333" use="339" covered="0"/>
   <du var="builder" def="333" use="341" covered="1"/>
   <du var="builder" def="333" use="339" covered="0"/>
   <counter type="DU" missed="3" covered="2"/>
   <counter type="METHOD" missed="0" covered="1"/>
 </method>
```

Figura 3

```
−<method name="verifyInternal" desc="(Lorg/apache/bookkeeper/bookie/Cookie;Z)V">
    <du var="this" def="154" use="161" covered="0"/>
    <du var="this" def="154" use="158" target="158" covered="1"/>
    <du var="this" def="154" use="158" target="161" covered="0"/>
    <du var="this" def="154" use="159" target="159" covered="1"/>
    <du var="this" def="154" use="159" target="161" covered="0"/>
    <du var="this" def="154" use="159" target="159" covered="1"/>
    <du var="this" def="154" use="159" target="161" covered="0"/>
    <du var="this" def="154" use="160" target="161" covered="0"/>
    <du var="this" def="154" use="160" target="163" covered="1"/>
    <du var="this" def="154" use="163" target="163" covered="0"/>
    <du var="this" def="154" use="163" target="163" covered="1"/>
    <du var="this" def="154" use="163" target="163" covered="1"/>
    <du var="this" def="154" use="163" target="170" covered="0"/>
    <du var="this" def="154" use="164" target="166" covered="0"/>
    <du var="this" def="154" use="164" target="170" covered="1"/>
    <du var="this" def="154" use="166" covered="0"/>
    <du var="this" def="154" use="154" target="155" covered="1"/>
    <du var="this" def="154" use="154" target="158" covered="0"/>
    <du var="c" def="154" use="154" target="154" covered="1"/>
    <du var="c" def="154" use="154" target="158" covered="1"/>
    <du var="c" def="154" use="158" target="158" covered="1"/>
    <du var="c" def="154" use="158" target="161" covered="0"/>
    <du var="c" def="154" use="161" covered="0"/>
    <du var="c" def="154" use="158" target="158" covered="1"/>
    <du var="c" def="154" use="158" target="161" covered="0"/>
    <du var="c" def="154" use="159" target="159" covered="1"/>
    <du var="c" def="154" use="159" target="161" covered="0"/>
    <du var="c" def="154" use="159" target="159" covered="1"/>
    <du var="c" def="154" use="159" target="161" covered="0"/>
```

Figura 4

```
    <du var="this" def="154" use="164" target="170" covered="1"/>
    <du var="this" def="154" use="166" covered="0"/>
    <du var="this" def="154" use="154" target="155" covered="1"/>
    <du var="this" def="154" use="154" target="158" covered="0"/>
    <du var="c" def="154" use="154" target="154" covered="1"/>
    <du var="c" def="154" use="154" target="158" covered="1"/>
    <du var="c" def="154" use="158" target="158" covered="1"/>
    <du var="c" def="154" use="158" target="161" covered="0"/>
    <du var="c" def="154" use="161" covered="0"/>
    <du var="c" def="154" use="158" target="158" covered="1"/>
    <du var="c" def="154" use="158" target="161" covered="0"/>
    <du var="c" def="154" use="159" target="159" covered="1"/>
    <du var="c" def="154" use="159" target="161" covered="0"/>
    <du var="c" def="154" use="159" target="159" covered="1"/>
    <du var="c" def="154" use="159" target="161" covered="0"/>
    <du var="c" def="154" use="160" target="161" covered="0"/>
    <du var="c" def="154" use="160" target="163" covered="1"/>
    <du var="c" def="154" use="164" target="166" covered="0"/>
    <du var="c" def="154" use="164" target="170" covered="1"/>
    <du var="c" def="154" use="166" covered="0"/>
    <du var="c" def="154" use="163" target="163" covered="0"/>
    <du var="c" def="154" use="163" target="166" covered="0"/>
    <du var="c" def="154" use="154" target="155" covered="1"/>
    <du var="c" def="154" use="154" target="158" covered="0"/>
    <du var="c" def="154" use="155" covered="1"/>
    <du var="checkIfSuperSet" def="154" use="159" target="159" covered="1"/>
    <du var="checkIfSuperSet" def="154" use="159" target="161" covered="0"/>
    <du var="checkIfSuperSet" def="154" use="160" target="161" covered="0"/>
    <du var="checkIfSuperSet" def="154" use="160" target="163" covered="1"/>
    <du var="c.layoutVersion" def="154" use="154" target="154" covered="1"/>
    <du var="c.layoutVersion" def="154" use="154" target="158" covered="1"/>
    <du var="c.layoutVersion" def="154" use="158" target="158" covered="1"/>
```

Figura 5

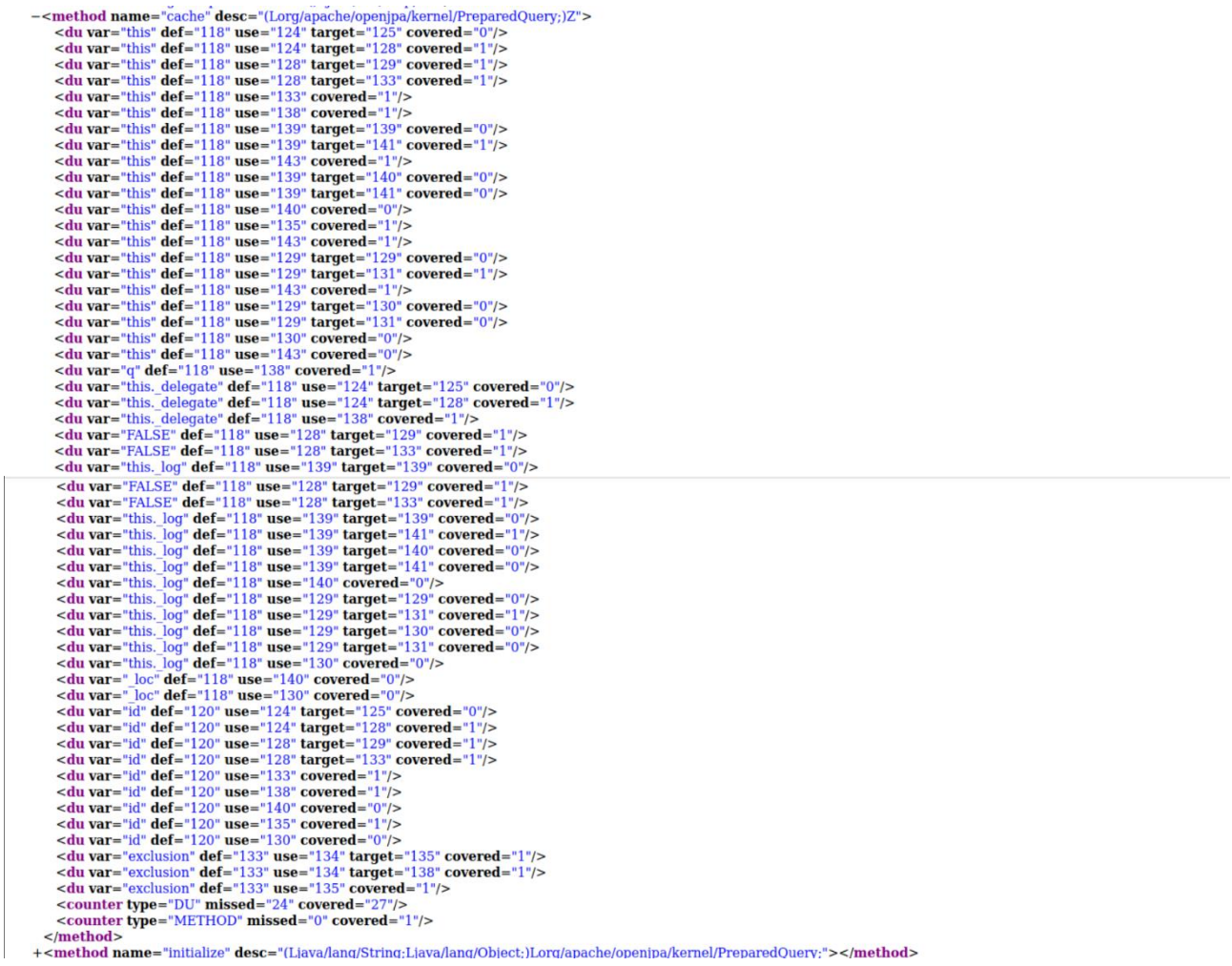
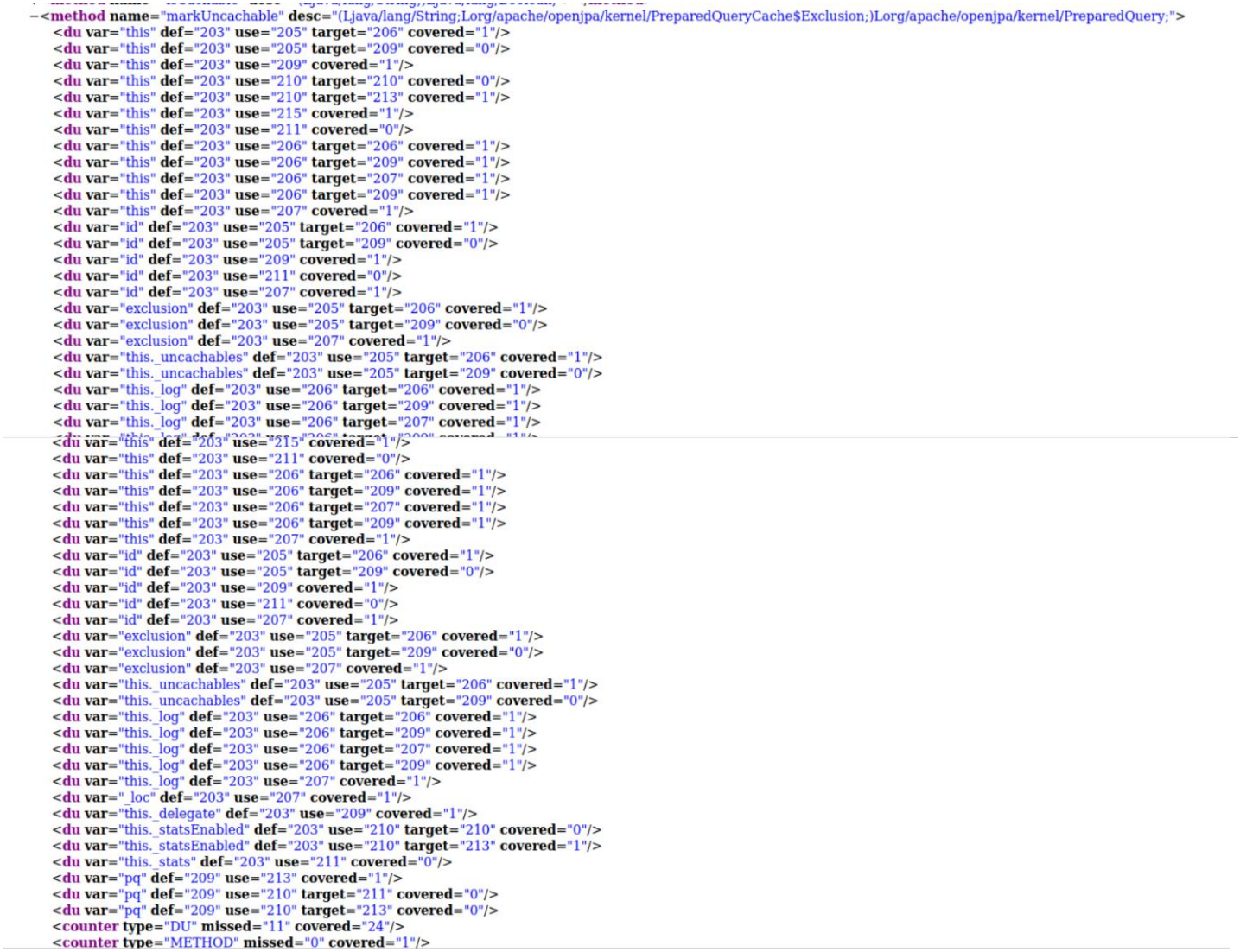


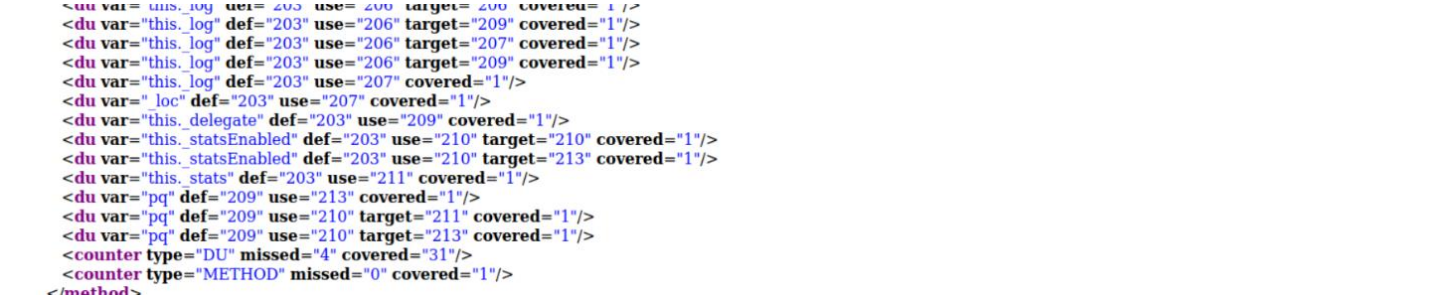
Figura 6

Figura 7



```
<method name="markUncachable" desc="(Ljava/lang/String;Lorg/apache/openjpa/kernel/PreparedQueryCache$Exclusion;)Lorg/apache/openjpa/kernel/PreparedQuery;">
<du var="this" def="203" use="205" target="206" covered="1"/>
<du var="this" def="203" use="205" target="209" covered="0"/>
<du var="this" def="203" use="209" covered="1"/>
<du var="this" def="203" use="210" target="210" covered="0"/>
<du var="this" def="203" use="210" target="213" covered="1"/>
<du var="this" def="203" use="215" covered="1"/>
<du var="this" def="203" use="211" covered="0"/>
<du var="this" def="203" use="206" target="206" covered="1"/>
<du var="this" def="203" use="206" target="209" covered="1"/>
<du var="this" def="203" use="206" target="207" covered="1"/>
<du var="this" def="203" use="206" target="209" covered="1"/>
<du var="this" def="203" use="207" covered="1"/>
<du var="id" def="203" use="205" target="206" covered="1"/>
<du var="id" def="203" use="205" target="209" covered="0"/>
<du var="id" def="203" use="209" covered="1"/>
<du var="id" def="203" use="211" covered="0"/>
<du var="id" def="203" use="207" covered="1"/>
<du var="exclusion" def="203" use="205" target="206" covered="1"/>
<du var="exclusion" def="203" use="205" target="209" covered="0"/>
<du var="exclusion" def="203" use="207" covered="1"/>
<du var="this._uncachables" def="203" use="205" target="206" covered="1"/>
<du var="this._uncachables" def="203" use="205" target="209" covered="0"/>
<du var="this._log" def="203" use="206" target="206" covered="1"/>
<du var="this._log" def="203" use="206" target="209" covered="1"/>
<du var="this._log" def="203" use="206" target="207" covered="1"/>
<du var="this" def="203" use="215" covered="1"/>
<du var="this" def="203" use="211" covered="0"/>
<du var="this" def="203" use="206" target="206" covered="1"/>
<du var="this" def="203" use="206" target="209" covered="1"/>
<du var="this" def="203" use="206" target="207" covered="1"/>
<du var="this" def="203" use="206" target="209" covered="1"/>
<du var="this" def="203" use="207" covered="1"/>
<du var="id" def="203" use="205" target="206" covered="1"/>
<du var="id" def="203" use="205" target="209" covered="0"/>
<du var="id" def="203" use="209" covered="1"/>
<du var="id" def="203" use="211" covered="0"/>
<du var="id" def="203" use="207" covered="1"/>
<du var="exclusion" def="203" use="205" target="206" covered="1"/>
<du var="exclusion" def="203" use="205" target="209" covered="0"/>
<du var="exclusion" def="203" use="207" covered="1"/>
<du var="this._uncachables" def="203" use="205" target="206" covered="1"/>
<du var="this._uncachables" def="203" use="205" target="209" covered="0"/>
<du var="this._log" def="203" use="206" target="206" covered="1"/>
<du var="this._log" def="203" use="206" target="209" covered="1"/>
<du var="this._log" def="203" use="206" target="207" covered="1"/>
<du var="this._log" def="203" use="206" target="209" covered="1"/>
<du var="this._log" def="203" use="207" covered="1"/>
<du var="_loc" def="203" use="207" covered="1"/>
<du var="this._delegate" def="203" use="209" covered="1"/>
<du var="this._statsEnabled" def="203" use="210" target="210" covered="0"/>
<du var="this._statsEnabled" def="203" use="210" target="213" covered="1"/>
<du var="this._stats" def="203" use="211" covered="0"/>
<du var="pq" def="209" use="213" covered="1"/>
<du var="pq" def="209" use="210" target="211" covered="0"/>
<du var="pq" def="209" use="210" target="213" covered="0"/>
<counter type="DU" missed="11" covered="24"/>
<counter type="METHOD" missed="0" covered="1"/>
```

Figura 7.1



```
<du var="this._log" def="203" use="206" target="209" covered="1"/>
<du var="this._log" def="203" use="206" target="207" covered="1"/>
<du var="this._log" def="203" use="206" target="209" covered="1"/>
<du var="this._log" def="203" use="207" covered="1"/>
<du var="_loc" def="203" use="207" covered="1"/>
<du var="this._delegate" def="203" use="209" covered="1"/>
<du var="this._statsEnabled" def="203" use="210" target="210" covered="1"/>
<du var="this._statsEnabled" def="203" use="210" target="213" covered="1"/>
<du var="this._stats" def="203" use="211" covered="1"/>
<du var="pq" def="209" use="213" covered="1"/>
<du var="pq" def="209" use="210" target="211" covered="1"/>
<du var="pq" def="209" use="210" target="213" covered="1"/>
<counter type="DU" missed="4" covered="31"/>
<counter type="METHOD" missed="0" covered="1"/>
</method>
```

Figura 8

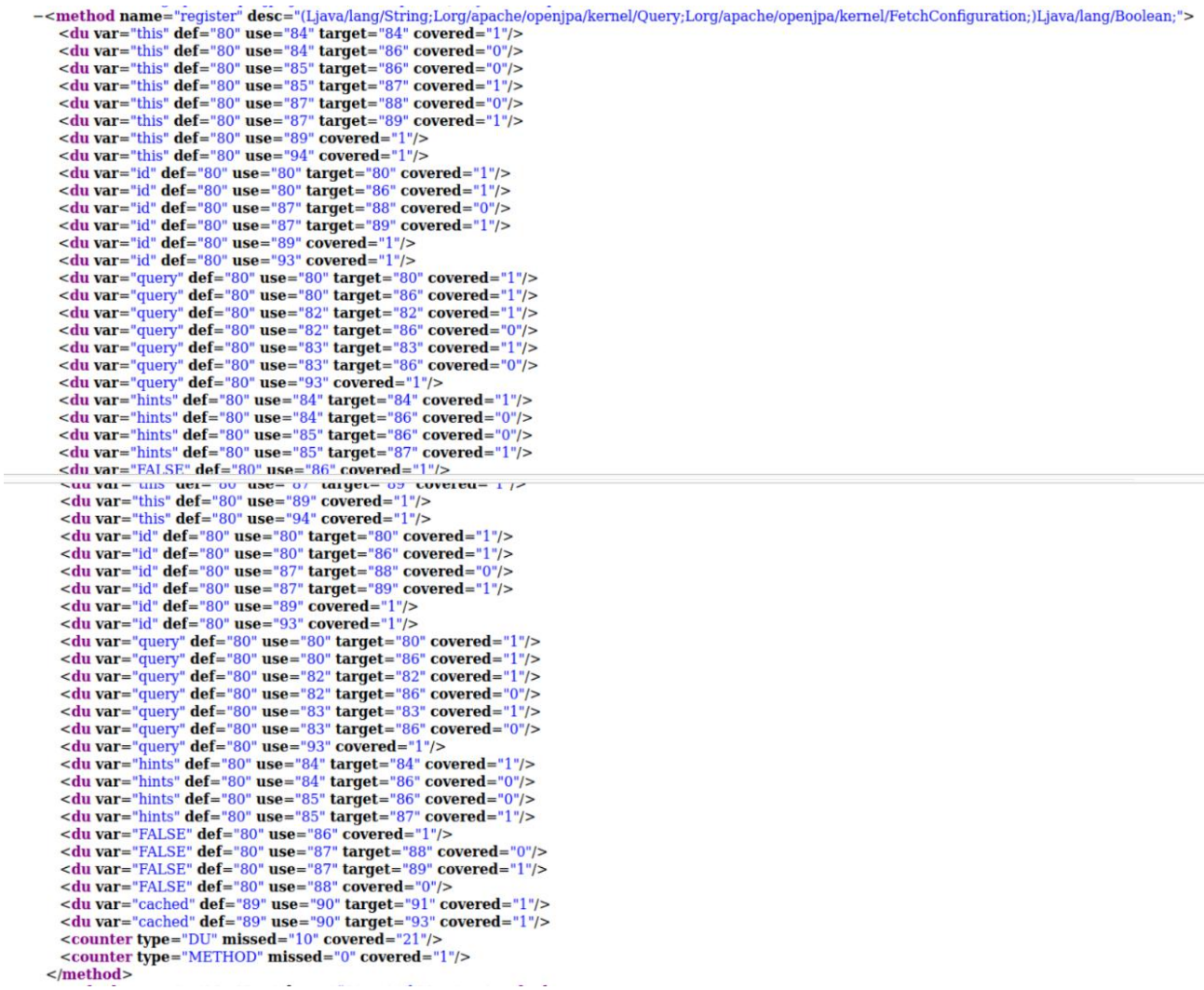


Figura 9

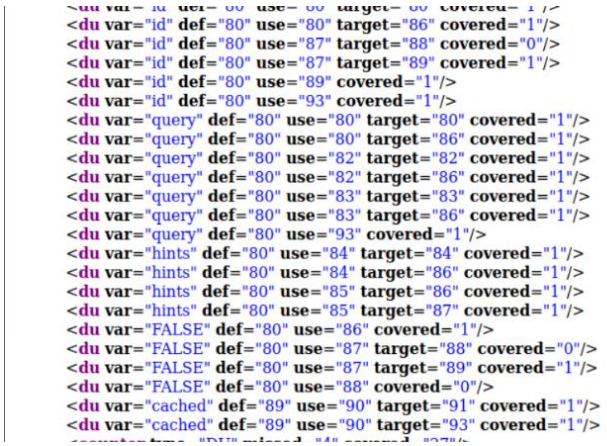
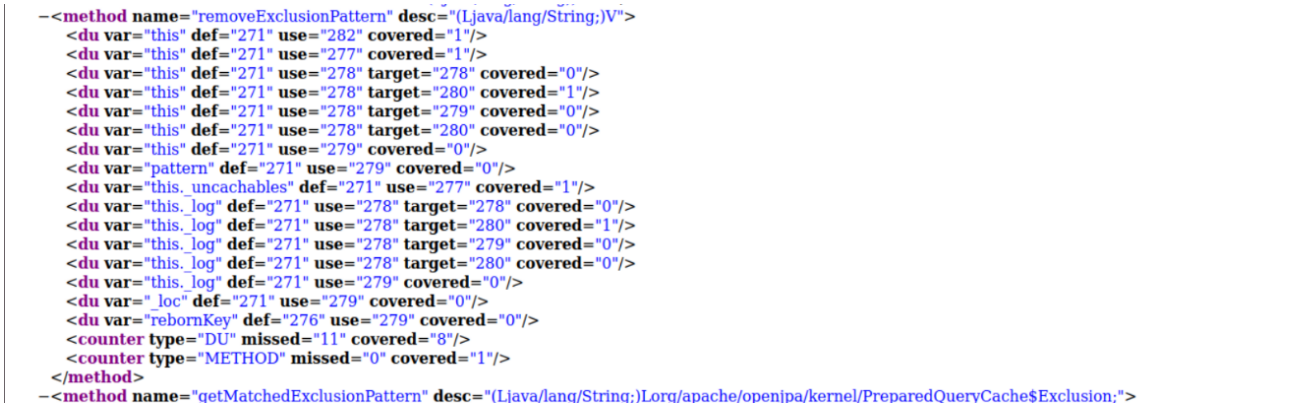


Figura 9.1

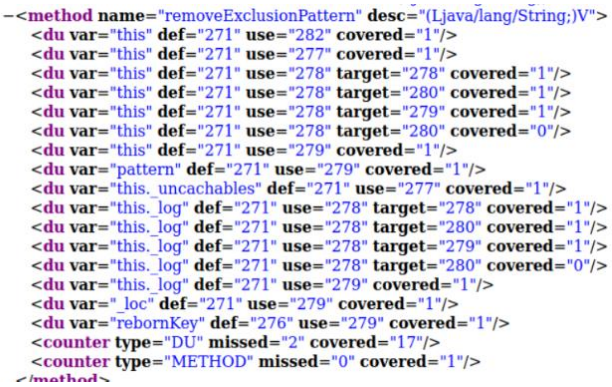
```
<method name="register" desc="(Ljava/lang/String;Lorg/apache/openjpa/kernel/Query;Lorg/apache/openjpa/kernel/FetchConfiguration;)Ljava/lang/Boolean;">
  <du var="this" def="80" use="84" target="84" covered="1"/>
  <du var="this" def="80" use="84" target="86" covered="1"/>
  <du var="this" def="80" use="85" target="86" covered="1"/>
  <du var="this" def="80" use="85" target="87" covered="1"/>
  <du var="this" def="80" use="87" target="88" covered="1"/>
  <du var="this" def="80" use="87" target="89" covered="1"/>
  <du var="this" def="80" use="89" covered="1"/>
  <du var="this" def="80" use="94" covered="1"/>
  <du var="id" def="80" use="80" target="80" covered="1"/>
  <du var="id" def="80" use="80" target="86" covered="1"/>
  <du var="id" def="80" use="87" target="88" covered="1"/>
  <du var="id" def="80" use="87" target="89" covered="1"/>
  <du var="id" def="80" use="89" covered="1"/>
  <du var="id" def="80" use="93" covered="1"/>
  <du var="query" def="80" use="80" target="80" covered="1"/>
  <du var="query" def="80" use="80" target="86" covered="1"/>
  <du var="query" def="80" use="82" target="82" covered="1"/>
  <du var="query" def="80" use="82" target="86" covered="1"/>
  <du var="query" def="80" use="83" target="83" covered="1"/>
  <du var="query" def="80" use="83" target="86" covered="1"/>
  <du var="query" def="80" use="93" covered="1"/>
  <du var="hints" def="80" use="84" target="84" covered="1"/>
  <du var="hints" def="80" use="84" target="86" covered="1"/>
  <du var="hints" def="80" use="85" target="86" covered="1"/>
  <du var="hints" def="80" use="85" target="87" covered="1"/>
  <du var="FALSE" def="80" use="86" covered="1"/>
  <du var="FALSE" def="80" use="87" target="88" covered="1"/>
  <du var="this" def="80" use="84" target="86" covered="1"/>
  <du var="this" def="80" use="85" target="86" covered="1"/>
  <du var="this" def="80" use="85" target="87" covered="1"/>
  <du var="this" def="80" use="87" target="88" covered="1"/>
  <du var="this" def="80" use="87" target="89" covered="1"/>
  <du var="this" def="80" use="89" covered="1"/>
  <du var="this" def="80" use="94" covered="1"/>
  <du var="id" def="80" use="80" target="80" covered="1"/>
  <du var="id" def="80" use="80" target="86" covered="1"/>
  <du var="id" def="80" use="87" target="88" covered="1"/>
  <du var="id" def="80" use="87" target="89" covered="1"/>
  <du var="id" def="80" use="89" covered="1"/>
  <du var="id" def="80" use="93" covered="1"/>
  <du var="query" def="80" use="80" target="80" covered="1"/>
  <du var="query" def="80" use="80" target="86" covered="1"/>
  <du var="query" def="80" use="82" target="82" covered="1"/>
  <du var="query" def="80" use="82" target="86" covered="1"/>
  <du var="query" def="80" use="83" target="83" covered="1"/>
  <du var="query" def="80" use="83" target="86" covered="1"/>
  <du var="query" def="80" use="93" covered="1"/>
  <du var="hints" def="80" use="84" target="84" covered="1"/>
  <du var="hints" def="80" use="84" target="86" covered="1"/>
  <du var="hints" def="80" use="85" target="86" covered="1"/>
  <du var="hints" def="80" use="85" target="87" covered="1"/>
  <du var="FALSE" def="80" use="86" covered="1"/>
  <du var="FALSE" def="80" use="87" target="88" covered="1"/>
  <du var="FALSE" def="80" use="87" target="89" covered="1"/>
  <du var="FALSE" def="80" use="88" covered="1"/>
  <du var="cached" def="89" use="90" target="91" covered="1"/>
  <du var="cached" def="89" use="90" target="93" covered="1"/>
  <counter type="DU" missed="0" covered="31"/>
```

Figura 10



```
<method name="removeExclusionPattern" desc="(Ljava/lang/String;)V">
  <du var="this" def="271" use="282" covered="1"/>
  <du var="this" def="271" use="277" covered="1"/>
  <du var="this" def="271" use="278" target="278" covered="0"/>
  <du var="this" def="271" use="278" target="280" covered="1"/>
  <du var="this" def="271" use="278" target="279" covered="0"/>
  <du var="this" def="271" use="278" target="280" covered="0"/>
  <du var="this" def="271" use="279" covered="0"/>
  <du var="pattern" def="271" use="279" covered="0"/>
  <du var="this._uncachables" def="271" use="277" covered="1"/>
  <du var="this._log" def="271" use="278" target="278" covered="0"/>
  <du var="this._log" def="271" use="278" target="280" covered="1"/>
  <du var="this._log" def="271" use="278" target="279" covered="0"/>
  <du var="this._log" def="271" use="278" target="280" covered="0"/>
  <du var="this._log" def="271" use="279" covered="0"/>
  <du var="_loc" def="271" use="279" covered="0"/>
  <du var="rebornKey" def="276" use="279" covered="0"/>
  <counter type="DU" missed="11" covered="8"/>
  <counter type="METHOD" missed="0" covered="1"/>
</method>
<method name="getMatchedExclusionPattern" desc="(Ljava/lang/String;)Lorg/apache/openjpa/kernel/PreparedQueryCache$Exclusion;">
```

Figura 11


−<method name="removeExclusionPattern" desc="(Ljava/lang/String;)V">
<du var="this" def="271" use="282" covered="1"/>
<du var="this" def="271" use="277" covered="1"/>
<du var="this" def="271" use="278" target="278" covered="1"/>
<du var="this" def="271" use="278" target="280" covered="1"/>
<du var="this" def="271" use="278" target="279" covered="1"/>
<du var="this" def="271" use="278" target="280" covered="0"/>
<du var="this" def="271" use="279" covered="1"/>
<du var="pattern" def="271" use="279" covered="1"/>
<du var="this._uncachables" def="271" use="277" covered="1"/>
<du var="this._log" def="271" use="278" target="278" covered="1"/>
<du var="this._log" def="271" use="278" target="280" covered="1"/>
<du var="this._log" def="271" use="278" target="279" covered="1"/>
<du var="this._log" def="271" use="278" target="280" covered="0"/>
<du var="this._log" def="271" use="279" covered="1"/>
<du var="_loc" def="271" use="279" covered="1"/>
<du var="rebornKey" def="276" use="279" covered="1"/>
<counter type="DU" missed="2" covered="17"/>
<counter type="METHOD" missed="0" covered="1"/>
</method>