Esercizio preliminare

Luca Mastrobattista

Indice

1 Traccia

Considerare il progetto Apache Bookkeeper. In particolare la documentazione, con riferimento alla classe `org.apache.bookkeeper.client.Bookkeeper`, identificare partizioni di dominio per i parametri dei metodi: `createLedger`.

Suggerimento: iniziare a considerare la sua implementazione.

Scrivere un breve report descrivendo:

1. come sono state identificate le partizioni, e come è stata condotta la *boundary analysis*
2. suggerimenti preliminari di test da includere, con risultato atteso

2 Svolgimento

La segnatura del metodo è:

```
public LedgerHandle createLedger(int ensSize,
      int writeQuorumSize,
      int ackQuorumSize,
      DigestType digestType,
      byte[] passwd)
```

Prende in inpunt 5 parametri:

1. `ensSize`: il numero di nodi in cui il *ledger* è memorizzato. La segnatura prevede un tipo intero.
2. `writeQuorumSize`: il numero di nodi in cui ogni *entry* è memorizzata e rappresenta di fatto il massimo grado di replicazione di ogni *entry*. La segnatura prevede un tipo intero.
3. `ackQuorumSize`: il numero di nodi da cui ogni *entry* deve essere conosciuta e rappresenta di fatto il minimo grado di replicazione di ogni *entry*. Anche qui è previsto un intero.
4. `digestType`: dovrebbe rappresentare una *signature* del *ledger* calcolata sulle varie *entry* per verificarle. È una *enum* e sono possibili solo i seguenti valori:

(a) `CRC32`

(b) `MAC`

(c) `CRC32C`

(d) `DUMMY`

5. `passwd`: nella documentazione non è riportata l'utilità di questo campo, ma è possibile leggere che sono i bytes ottenuti da una password inserita. È quindi un array di bytes.

2.1 Definizione delle classi di equivalenza

2.1.1 `ensSize`

È un tipo di dato intero che rappresenta il numero di nodi su cui il *ledger* è definito. Si definiscono 3 classi di equivalenza, una invalida, una che potrebbe essere invalida, e una valida:
$\{< 0\}$, $\{= 0\}$, $\{> 0\}$

2.1.2 `writeQuorumSize`

È un tipo di dato intero che rappresenta il numero di nodi su cui ogni *entry* è memorizzata, e quindi il suo grado massimo di replicazione. Si definiscono 3 classi di equivalenza, una invalida, una che potrebbe essere invalida, e una valida:
$\{< 0\}$, $\{= 0\}$, $\{> 0\}$

2.1.3 `ackQuorumSize`

È un tipo di dato intero che rappresenta il numero di nodi su cui ogni *entry* deve essere conosciuta, e quindi il suo grado minimo di replicazione. Si definiscono 3 classi di equivalenza, una invalida, una che potrebbe essere invalida, e una valida:
$\{<$ `writeQuorumSize`$\}$, $\{=$ `writeQuorumSize`$\}$, $\{>$ `writeQuorumSize`$\}$

2.1.4 `digestType`

Si definisce una classe di equivalenza per ogni *enum*:
{`CRC32`}, {`MAC`}, {`CRC32C`}, {`DUMMY`}

2.1.5 `passwd`

È un array di bytes. Si definiscono le seguenti classi:
`NULL`, {}, `"password".getBytes()`.

2.2 Boundary analisys

Le linee guida prevedono che si scelgano valori ai bordi delle classi di equivalenza dove, secondo uno studio empirico, si riscontrano più spesso problemi.

2.2.1 `ensSize`

$\{< 0\}$, $\{= 0\}$, $\{> 0\} \rightarrow$ {-1}, {0}, {1}

2.2.2 `writeQuorumSize`

$\{< 0\}$, $\{= 0\}$, $\{> 0\} \rightarrow$ {-1}, {0}, {1}

2.2.3 `ackQuorumSize`

{$<$ `writeQuorumSize`}, {$=$ `writeQuorumSize`}, {$>$ `writeQuorumSize`} $\rightarrow$ {`writeQuorumSize` -1}, {`writeQuorumSize`}, {`writeQuorumSize` +1}

2.2.4 `digestType`

{`CRC32`}, {`MAC`}, {`CRC32C`}, {`DUMMY`}

2.2.5 `passwd`

`NULL`, {}, `"password".getBytes()`

2.3 Test da includere

2.3.1 Analisi unidimensionale

`createLedger(-1, -1, -2, CRC32, NULL)` $\rightarrow$ Ci si aspetta il sollevamento di una eccezione: si suppone che i gradi di replicazione di un nodo debbano essere maggiori o al massimo uguali a 0.

`createLedger(0, 0, 0, MAC, {})` $\rightarrow$ Ci si aspetta il sollevamento di una eccezione: nella documentazione è specificato che è necessario almeno una password e un `digestType` per creare un ledger, ma qui la password è vuota.

Inoltre non avrebbe molto senso creare un *ledger* che non viene memorizzato su nessun nodo.

`createLedger(1, 1, 2, CRC32C, "password".getBytes())` → Ci si aspetta il sollevamento di una eccezione: il grado di replicazione minimo è superiore al grado di replicazione massimo.

`createLedger(1, 1, 2, DUMMY, "password".getBytes())` → Ci si aspetta che venga restituito un `LedgerHandle` correttamente creato.

L'ultimo test è risultato necessario per coprire interamente le classi di equivalenza del parametro `digestType`, ma è stato utile per testare che il metodo restituisca un oggetto correttamente inizializzato.

2.3.2 Analisi multidimensionale

Le possibili combinazioni sono:

$$\prod_{C_i} |C_i| = 3 \cdot 3 \cdot 3 \cdot 4 \cdot 3 = 3^4 \cdot 4 = 324$$

dove C_i è la classe di equivalenza per il parametro i.

2.3.3 Conclusioni

Il secondo metodo, quello multidimensionale, è decisamente più costoso ma sicuramente più esaustivo: tutte le possibili combinazioni delle classi di equivalenza vengono infatti testate. Non è detto, però, che eventuali malfunzionamenti dipendano dalla combinazione di tutti e 5 i parametri: personalmente, osservando che i test definiti con un approccio unidimensionale sono un sottoinsieme dei casi definiti con un approccio multidimensione, proverei prima ad eseguire i test del sottoinsieme e poi, eventualmente, eseguirei i test mancanti per ottenere l'insieme dei test definiti nel caso multidimensionale.