ISL@b
Databases Framework 2

QUICK START GUIDE - VERSION 1.2

Indice

1 Configurazione

I passi necessari per il corretto funzionamento delle librerie IDBS2 consistono nell'importazione delle librerie, nella configurazione dei file per la connessione ai database e nell'aggiunta al progetto di un file per la configurazione del sistema di logging.

1.1 Importazione librerie

Per far si che le librerie IDBS2 possano essere utilizzate all'interno del codice è necessario includerle nel classpath del progetto.

Inclusione in Eclipse

Per includere le librerie lavorando sulla piattaforma Eclipse è necessario accedere alle proprietà del progetto mediante click destro sulla cartella contenente il progetto stesso.
Nel menu a sinistra della finestra che verrà mostrata è necessario spostarsi sulla scheda *Java Build Path* e, successivamente, selezionare il tab *Libraries* dal blocco centrale. A questo punto è sufficiente aggiungere il file *IDBS2 [version].jar* mediante il bottone *Add JARs...*, se tale file è all'interno della cartella di progetto, oppure mediante il bottone *Add External JARs...* se le librerie sono state estratte in una cartella esterna[1]. Per il corretto funzionamento delle librerie IDBS2, occorrerà importare all'interno del progetto anche la libreria JDBC per il DBMS utilizzato (all'interno delle librerie di supporto disponibili sul sito del laboratorio sono contenute le librerie per PostgreSQL e MySQL).

1.2 Configurazione dell'accesso al database

I file per l'accesso al database sono tre (*cloudDatabaseConfiguration.xml*, *kcDatabaseConfiguration.xml* e *mapDatabaseConfiguration.xml*), possono essere scaricati dalla pagina http://island.ricerca.di.unimi.it nella sezione relativa alle librerie in questione e devono essere posizionati in una cartella *config* situata nella root del progetto. Tutti e tre i file si basano sulla stessa struttura di rappresentazione. Per ottenere una corretta configurazione della libreria per l'accesso ad uno dei tre database su PostgreSQL o MySQL è sufficiente modificare il file corrispondente andando ad impostare i valori corretti nei campi *connection.url*, *connection.username* e *connection.password*. Nel caso si decida di utilizzare un DBMS differente sarà necessario modificare i parametri *connection.driver_class* e *dialect* oltre che cambiare il protocollo del parametro *connection.url* [2].

[1] Si consiglia di creare una cartella *lib* all'interno del progetto e di copiarci tutte le librerie richieste. In questo modo chi dovrà riutilizzare il software in seguito si troverà con un pacchetto funzionante.

[2] I file di configurazione pronti sono disponibili online per PostgreSQL e MySQL, per altri DBMS è necessario provvedere a modificarli cercando online i parametri corretti.

1.3 Configurazione del sistema di logging (*log4j*)

Il file per la configurazione del sistema di logging può essere scaricato dalla pagina http://island.ricerca.di.unimi.it nella sezione relativa alle librerie in questione. Una volta scaricato il file dovrà essere inserito nella cartella *src* del progetto (o nella cartella principale di esecuzione). All'interno è possibile vedere tre parametri che controllano il livello di logging delle 3 librerie:

- log4j.logger.idbs2.kc=warn
- log4j.logger.idbs2.map=debug
- log4j.logger.idbs2.cloud=info

Per ogni libreria è possibile impostare il livello di logging desiderato tra *error, warning, info, debug, trace*, in ordine di verbosità da sinistra verso destra.

2 Funzionamento delle librerie

Prima di procedere con la spiegazione del funzionamento delle librerie IDBS2 è necessario soffermarsi sul significato degli oggetti resi disponibili da questa libreria (che eredita i concetti definiti dal framework Hibernate). Un oggetto mappato tramite Hibernate (e.g., KnowledgeChunk, CloudNode, MappingSet) può trovarsi in tre stati:

- **Transient**. Un oggetto si trova in questo stato quando viene istanziato mediante un comando *new*. In questa situazione un oggetto è in uno stato NON persistente e totalmente svincolato dalla base di dati fino alla chiamate di un metodo che lo colleghi ad una sessione (come ad esempio *saveOrUpdate*).

- **Persistent**. Quando un oggetto viene caricato tramite la sessione Hibernate o viene collegato ad essa mediante la chiamata ad un metodo, esso risulta collegato ad una sessione e, conseguentemente, alla base di dati. Quando un oggetto si trova in questo stato qualunque modifica effettuata su di esso verrà riportata sulle tuple della base di dati delle quali l'oggetto è rappresentazione. Un oggetto resta in questo stato fino alla chiamata di una operazione di commit esplicita.

- **Detached**. Si trovano in questo stato oggetti che sono stati collegati ad una sessione che venga chiusa ad un certo punto dell'esecuzione. Questo stato è praticamente identico allo stato *Transient* con la differenza che gli oggetti possono essere ricollegati successivamente ad una sessione in modo da materializzare automaticamente le modifiche avvenute durante il periodo di non-persistenza.

In seguito verrà illustrato il funzionamento delle librerie IDBS2 mediante una serie di esempi basati sul package *kc* (KnowledgeChunk). Tali esempi possono essere presi come spunto per il funzionamento dei package *ic* e *map* in quanto tutti e 3 i package rispondono alle stesse logiche di funzionamento.

2.1 Libreria KC

Per introdurre l'utente all'utilizzo della libreria, vediamo un esempio di creazione e salvataggio di KnowledgeChunk su database remoto, un esempio di lettura e modifica ed infine un esempio di gestione manuale della sessione (utile per effettuare operazioni avanzate non disponibili nei gestori di sessione di queste librerie).

2.1.1 Inizializzazione del database

A partire dalla versione 1.2 delle librerie, è resa disponibile la funzione *createDb()*, pubblicata dalla classe statica *KCSessionManager*. Una volta invocata, questa funzione genera una versione essenziale dello schema della base di dati KnowledgeChunk sul database configurato per la connessione. Questa funzione

è molto utile per la generazione dello schema a runtime ma deve essere usata con cautela in quanto lo schema generato sarà poco ottimizzato (mancanza di indici, operazioni di default per i vincoli di integrità referenziale, funzioni di utilità).

2.1.2 Creazione e salvataggio

Di seguito un breve estratto di codice che provvede a generare un nuovo KnowledgeChunk e a salvarlo sulla base di dati mediante le librerie IDBS2.

```
//Creo un nuovo KC con id "/en/woody_allen"
KnowledgeChunk woody_allen = new KnowledgeChunk("/en/woody_allen");
//definisco questo kc come istanza
woody_allen.setType('I');

//Aggiungo dei termini associati a woody_allen specificando relevance del termine
//e frequency (numero di volte in cui quel termine compare all'interno del KC)
woody_allen.addTerm("Termine 1", 0.4, 1);
woody_allen.addTerm("Termine 2", 0.8, 3);

//Creo un nuovo KC che rappresenta il tipo "persona"
KnowledgeChunk persona = new KnowledgeChunk("/people/person");

//Dichiaro questo KC come classe (tipo).
persona.setType('C');

//Definisco woody_allen di tipo persona.
woody_allen.addType(persona);

//DA QUI INIZIO AD INTERAGIRE CON IL DATABASE MEDIANTE I GESTORI DI SESSIONE

//Avvio una nuova transazione che rimarrà attiva fino al commit esplicito.
KCSessionManager.beginTransaction();

//Memorizzo in due variabili temporanee i singoletti KCManager e DocumentManager.
KCManager kcm = KCSessionManager.kcm;
DocumentManager dm = KCSessionManager.dm;

//

//Carico un documento
Document doc = dm.getDocumentByUrl("people_from_freebase");

//aggiungo woody_allen a questo documento
woody_allen.addDocument(doc);
```

```
//Utilizzo il KCManager per salvare il kc appena creato.
//N.B.: Il kc "persona" verrà salvato
//automaticamente in cascata poiché fa parte del contesto di woody_allen.
kcm.saveOrUpdate(woody_allen);

//Termino la sessione. Questa operazione garantisce la materializzazione dei dati
//sul DB collegato.
KCSessionManager.commitTransaction();
```

Nel codice appena visto, l'operazione *saveOrUpdate* provvede al salvataggio dell'oggetto passato come argomento o al suo aggiornamento nel caso tale oggetto fosse già collegato ad una sessione Hibernate.

2.1.3 Caricamento ed aggiornamento

Vediamo ora come caricare un oggetto dalla base di dati sfruttando i metodi base forniti dalla classe KCManager[3].

```
//Avvio una transazione (indispensabile per la comunicazione con il DB)
KCSessionManager.beginTransaction();

//Richiamo il KCManager tramite il campo statico kcm di KCSessionManager e
//eseguo un'operazione di caricamento di un KC mediante il suo id.
KnowledgeChunk woody_allen =
KCSessionManager.kcm.getKnowledgeChunkById("/en/woody_allen");

//Rimuovo un termine dal suo terminological equipment
woody_allen.removeTerm("Termine 2");

//Memorizzo un nome human readable
woody_allen.setName("Woody Allen");

//Genero un kc che rappresenti il tipo "attore"
KnowledgeChunk actor = new KnowledgeChunk("/film/actor");

//Definisco woody_allen di tipo attore
woody_allen.addType(actor);

//N.B.: Tutte le operazioni effettuate sono state fatte su di un oggetto in stato
//persistent e verranno quindi riportate sulla base di dati al termine della sessione.
```

[3]Le classi per la gestione della sessione vengono aggiornate e potenziate costantemente. Fare riferimento alle API ufficiali per la versione più recente (https://island.ricerca.di.unimi.it/javadoc/idbs1.0/

```
//Termino la sessione e rendo definitive le modifiche
KCSessionManager.commitTransaction();
```

2.1.4 Gestione manuale della sessione

Fino a questo punto tutte le chiamate al database sono state gestite automaticamente attraverso una sessione interna alla classe KCSessionManager. Per ottenere un generatore di sessioni utilizzabile in maniera diretta occorre chiamare il metodo KCSessionManager.getSessionFactory()[4]: attraverso la SessionFactory restituita sarà possibile gestire manualmente la sessione Hibernate. Vediamo ora un esempio di gestione manuale della sessione.

```
//Carico una sessione per la gestione manuale
Session s = KCSessionManager.getSessionFactory().getCurrentSession();

//Avvio una transazione
s.beginTransaction();

//Eseguo un'interrogazione per caricare tutti i kc aventi "allen" nel nome
List<KnowledgeChunk> kcs =
s.createQuery("FROM KnowledgeChunk kc WHERE kc.name LIKE '%allen%'").list();

//Scorro l'elenco di kc appena caricato e li elimino dalla base di dati.
for(KnowledgeChunk kc : kcs){
s.delete(kc);
}

//Faccio il commit della sessione attuale (questa
//operazione chiude la transazione corrente e salva
//le modifiche)
s.getTransaction.commit();
```

Attraverso la gestione manuale della sessione è possibile eseguire qualunque genere di query HQL[5] per il reperimento di dati, per la modifica o la cancellazione.

[4] Attraverso il metodo getSessionFactory() viene restituita una SessionFactory configurata mediante il file .xml contenuto nella cartella *config* relativo alla base di dati corrente (e.g. nel caso del database dei KnowledgeChunk sarà il file kcDatabaseConfiguration.xml). Per ottenere una SessionFactory basata su di un file di configurazione differente (nel caso si volesse gestire più di un database) è possibile chiamare la versione sovraccaricata dello stesso metodo KCSessionManager.getSessionFactory(File conf) che richiede in input un file di configurazione xml Hibernate.

[5] Hibernate Query Language. Fare riferimento alla guida ufficiale di Hibernate alla versione **3.6** per avere informazioni dettagliate sul funzionamento e sulla sintassi di questo linguaggio (http://www.hibernate.org/docs)