Introduzione

In questa pagina viene descritto il modulo di gestione delle code in itaEngine. Possiamo definire una coda come un'insieme di messaggi, che dovranno essere processati in ordine di inserimento (FIFO), al fine di rendere asincroni alcuni processi (Es. chiamate a web services).

Configurazione

Al file **config.ini** dovrà essere gestita la sesione [queue], che contiene le seguenti chiavi:

- **queueType**: Indica il tipo di implementazione utilizzata. Attualmente l'unico valore possibile è “cache”.
- **cacheRoot**: L'implementazione cache utilizzata è forzatamente “file”, in quanto “apc” vive solamente all'interno di un processo php che l'ha lanciata, quindi utilizzando la CLI per elaborare la coda, apc non sarà collegata ai dati di cache dell'applicativo, in quanto i processi sono differenti.

Descrizione delle singole componenti

Messaggio

Il messaggio rappresenta il singolo elemento della coda. E' un'istanza della classe **itaQueueMessage** (presente in lib/itaPHPQueue/itaQueueMessage.class.php) ed ha le seguenti proprietà:

- **uuid**: UUID del messaggio (chiave assegnata dal sistema)
- **alias**: Alias che identifica univocamente il messaggio (assegnata dalla singola procedura)
- **data**: Dati del messaggio
- **retries**: Numero massimo di tentativi (in caso di errore durante esecuzione)
- **executionMode**: Modalità di esecuzione (immediata/differita)
- **dateTimeDeferredExecution**: Data e ora di esecuzione differita
- **disabled**: Messaggio disabilitato
- **username**: Nome utente utile per sapere chi ha inserito il messaggio in coda

Coda

La coda è composta dalle seguenti informazioni:

- **id**: Identificativo coda (fisso in base alla tipologia, es. ANPR)
- **status**: Contiene lo stato della coda
- **messages**: Contiene la lista dei messaggi attualmente presenti nella coda

Lo stato, a sua volta, contiene le seguenti informazioni:

- **lastMessageInserted**: informazioni riguardanti l'ultimo messaggio inserito (uuid, alias e timestamp)
- **lastMessageProcessed**: informazioni riguardanti l'ultimo messaggio elaborato (uuid, alias e timestamp)
- **lastQueueModifyTime**: Data/ora ultima modifica
- **messagesToProcess**: numero di messaggi in coda da elaborare
- **customAttributes**: attributi custom della coda (array associativo, gestione totalmente libera)

Gestore della Coda

Il gestore si occupa delle operazioni che andranno ad interagire con la coda. L'implementazione specifica è determinata dal parametro “queueType” presente nel file di configurazione (Utilizzando apposita factory). Attualmente l'unica implementazione possibile è quella fatta con la cache, la classe corrispondente è 'itaQueueManagerCache' (presente il lib/itaPHPQueue/ityaQueueManagerCache.class.php). Il gestore della coda si occupa delle seguenti operazioni:

- **createQueue**: Crea una nuova coda
- **destroyQueue**: Distrugge coda
- **addMessage**: Aggiunge un nuovo messaggio in coda
- **getMessage**: Preleva il primo messaggio (non disabilitato) dalla coda (con rimozione)
- **queueExists**: Verifica esistenza della coda
- **queueStatus**: Restituisce lo stato della coda
- **getLastError**: Restituisce l'ultimo errore avvenuto
- **updateLastMessageProcessed**: Effettua l'aggiornamento della coda con l'esecuzione del messaggio appena eseguito.
- **findMessage**: Ricerca i messaggi sulla coda
- **updateMessage**: Aggiorna messaggio sulla cosa

Per reperire l'oggetto gestore, occorre fare in questo modo:

```
$qm = itaQueueFactory::getQueueManager();
```

Esempio di creazione di una coda:

```
$created = $qm->createQueue($itaQueueManagerBase::MESSAGE_TYPE_ANPR);
```

Worker

Ogni tipologia di coda ha una classe worker specifica, che si occupa dell'elaborazione del singolo messaggio. Ad esempio, per ANPR, la classe specifica è 'itaQueueWorkerANPR' presente in 'lib/itaPHPQueue/workers/itaQueueWorkerANPR.class.php'. Il worker è chiamato dal CLI, anch'esso specifico. Sempre nel caso di ANPR, lo script CLI da lanciare è 'queueWorkerANPR.php'.

Daemon

Per fare in modo che la coda giri come servizio, è opportuno scrivere un apposito daemon e registrarlo nel sistema come servizio. Per creare un nuovo daemon, basta semplicemente creare una nuova classe in /daemon/daemons/ con la seguente nomenclatura:

```
<nome-daemon>Daemon.class.php.
```

I metodi da implementare sono:

- **executeStart**: Metodo che viene invocato ad ogni iterazione.
- **executeStop**: Metodo che viene invocato allo stop del servizio.
- **getSleepTime**: Restituisce il numero di secondi di inattività tra un'iterazione e l'altra.

Configurazione

Oltre alla creazione del demone sotto forma di classe PHP risulta necessario inserire il demone appena creato nel file di configurazione /daemon/config/autostart.ini Ogni sezione del file di configurazione rappresenta un demone. Per ogni demone sono presenti due chiavi:

- **autostart** può essere impostata a true o false e indica se il demone va lanciato automaticamente dallo script */daemon/itaDaemonAutostart.php*
- **parameters** sono i parametri addizionali da inviare al demone, possono essere definiti più parametri usando il carattere di pipe | come divisore.

Servizio

Centos7: creazione di un nuovo script (/etc/systemd/system/testphp.service):

```
After=network.target

[Service]
Type=forking
User=root
ExecStart=/bin/bash /vagrant/system/start-daemon.sh par1 par2
ExecStop=/bin/bash /vagrant/system/stop-daemon.sh

[Install]
WantedBy=multi-user.target
```

start-daemon.sh:

```
php /vagrant/itaEngine/daemon/itaDaemonExecutor.php dummy start $1 $2 &
```

stop-daemon.sh:

```
php /vagrant/itaEngine/daemon/itaDaemonExecutor.php dummy stop
```