**StatelessSession**

StatelessSession is a command-oriented API provided by Hibernate. Use it to stream data to and from the database in the form of detached objects. A StatelessSession has no persistence context associated with it and does not provide many of the higher-level life cycle semantics. Some of the things not provided by a StatelessSession include:

**Features and behaviors not provided by StatelessSession**

* a first-level cache
* interaction with any second-level or query cache
* transactional write-behind or automatic dirty checking

**Limitations of StatelessSession**

* Operations performed using a stateless session never cascade to associated instances.
* Collections are ignored by a stateless session.
* Operations performed via a stateless session bypass Hibernate's event model and interceptors.
* Due to the lack of a first-level cache, Stateless sessions are vulnerable to data aliasing effects.
* A stateless session is a lower-level abstraction that is much closer to the underlying JDBC.

**Example 4.4. Using a StatelessSession**

StatelessSession session = sessionFactory.openStatelessSession();

Transaction tx = session.beginTransaction();

ScrollableResults customers = session.getNamedQuery("GetCustomers")

.scroll(ScrollMode.FORWARD\_ONLY);

**while** ( customers.next() ) {

Customer customer = (Customer) customers.get(0);

customer.updateStuff(...);

session.update(customer);

}

tx.commit();

session.close();

The Customer instances returned by the query are immediately detached. They are never associated with any persistence context.

The insert(), update(), and delete() operations defined by the StatelessSession interface operate directly on database rows. They cause the corresponding SQL operations to be executed immediately. They have different semantics from the save(), saveOrUpdate(), and delete() operations defined by the Session interface.