org.springframework.orm.hibernate3

Class HibernateTemplate

```
java.lang.Object
  └ org.springframework.orm.hibernate3.HibernateAccessor
      └ org.springframework.orm.hibernate3.HibernateTemplate
```

All Implemented Interfaces:
BeanFactoryAware, InitializingBean, HibernateOperations

```
public class HibernateTemplate
extends HibernateAccessor
implements HibernateOperations
```

Helper class that simplifies Hibernate data access code. Automatically converts HibernateExceptions into DataAccessExceptions, following the `org.springframework.dao` exception hierarchy.

The central method is `execute`, supporting Hibernate access code implementing the `HibernateCallback` interface. It provides Hibernate Session handling such that neither the HibernateCallback implementation nor the calling code needs to explicitly care about retrieving/closing Hibernate Sessions, or handling Session lifecycle exceptions. For typical single step actions, there are various convenience methods (find, load, saveOrUpdate, delete).

Can be used within a service implementation via direct instantiation with a SessionFactory reference, or get prepared in an application context and given to services as bean reference. Note: The SessionFactory should always be configured as bean in the application context, in the first case given to the service directly, in the second case to the prepared template.

NOTE: As of Hibernate 3.0.1, transactional Hibernate access code can also be coded in plain Hibernate style. Hence, for newly started projects, consider adopting the standard Hibernate3 style of coding data access objects instead, based on `SessionFactory.getCurrentSession()`.

This class can be considered as direct alternative to working with the raw Hibernate3 Session API (through `SessionFactory.getCurrentSession()`). The major advantage is its automatic conversion to DataAccessExceptions as well as its capability to fall back to 'auto-commit' style behavior when used outside of transactions. **Note that HibernateTemplate will perform its own Session management, not participating in a custom Hibernate CurrentSessionContext unless you explicitly switch `"allowCreate"` to "false".**

`LocalSessionFactoryBean` is the preferred way of obtaining a reference to a specific Hibernate SessionFactory, at least in a non-EJB environment. The Spring application context will manage its lifecycle, initializing and shutting down the factory as part of the application.

Note that operations that return an Iterator (i.e. *iterate*) are supposed to be used within Spring-driven or JTA-driven transactions (with HibernateTransactionManager, JtaTransactionManager, or EJB CMT). Else, the Iterator won't be able to read results from its ResultSet anymore, as the underlying Hibernate Session will already have been closed.

Lazy loading will also just work with an open Hibernate Session, either within a transaction or within OpenSessionInViewFilter/Interceptor. Furthermore, some operations just make sense within transactions, for example: *contains*, *evict*, *lock*, *flush*, *clear*.

Since:
1.2

Author:
Juergen Hoeller

See Also:
HibernateAccessor.setSessionFactory(org.hibernate.SessionFactory), *HibernateCallback*, *Session*, *LocalSessionFactoryBean*, *HibernateTransactionManager*, *JtaTransactionManager*, *OpenSessionInViewFilter*, *OpenSessionInViewInterceptor*

Field Summary

Fields inherited from class org.springframework.orm.hibernate3.HibernateAccessor
FLUSH_ALWAYS, *FLUSH_AUTO*, *FLUSH_COMMIT*, *FLUSH_EAGER*, *FLUSH_NEVER*, *logger*

Constructor Summary

HibernateTemplate() Create a new HibernateTemplate instance.
HibernateTemplate(SessionFactory sessionFactory) Create a new HibernateTemplate instance.
HibernateTemplate(SessionFactory sessionFactory, boolean allowCreate) Create a new HibernateTemplate instance.

Method Summary

protected void	*applyNamedParameterToQuery(Query queryObject, String paramName, Object value)* Apply the given name parameter to the given Query object.
int	*bulkUpdate(String queryString)* Update/delete all objects according to the given query.
int	*bulkUpdate(String queryString, Object value)* Update/delete all objects according to the given query, binding one value to a "?"
int	*bulkUpdate(String queryString, Object[] values)* Update/delete all objects according to the given query, binding a number of values to "?"
protected	*checkWriteOperationAllowed(Session session)* Check whether write operations are allowed on the given Session.

void	
void	*clear()* Remove all objects from the *Session* cache, and cancel all pending saves, updates and deletes.
void	*closeIterator(Iterator it)* Immediately close an *Iterator* created by any of the various *iterate(..)*
boolean	*contains(Object entity)* Check whether the given object is in the Session cache.
protected Session	*createSessionProxy(Session session)* Create a close-suppressing proxy for the given Hibernate Session.
void	*delete(Object entity)* Delete the given persistent instance.
void	*delete(Object entity, LockMode lockMode)* Delete the given persistent instance.
void	*delete(String entityName, Object entity)* Delete the given persistent instance.
void	*delete(String entityName, Object entity, LockMode lockMode)* Delete the given persistent instance.
void	*deleteAll(Collection entities)* Delete all given persistent instances.
Filter	*enableFilter(String filterName)* Return an enabled Hibernate *Filter* for the given filter name.
void	*evict(Object entity)* Remove the given object from the *Session* cache.
Object	*execute(HibernateCallback action)* Execute the action specified by the given action object within a *Session*.
Object	*execute(HibernateCallback action, boolean exposeNativeSession)* Execute the action specified by the given action object within a Session.
List	*executeFind(HibernateCallback action)* Execute the specified action assuming that the result object is a *List*.
List	*find(String queryString)* Execute an HQL query.
List	*find(String queryString, Object value)* Execute an HQL query, binding one value to a "?"
List	*find(String queryString, Object[] values)* Execute an HQL query, binding a number of values to "?"
List	*findByCriteria(DetachedCriteria criteria)* Execute a query based on a given Hibernate criteria object.

List	findByCriteria(DetachedCriteria criteria, int firstResult, int maxResults) Execute a query based on the given Hibernate criteria object.
List	findByExample(Object exampleEntity) Execute a query based on the given example entity object.
List	findByExample(Object exampleEntity, int firstResult, int maxResults) Execute a query based on a given example entity object.
List	findByExample(String entityName, Object exampleEntity) Execute a query based on the given example entity object.
List	findByExample(String entityName, Object exampleEntity, int firstResult, int maxResults) Execute a query based on a given example entity object.
List	findByNamedParam(String queryString, String[] paramNames, Object[] values) Execute an HQL query, binding a number of values to ":" named parameters in the query string.
List	findByNamedParam(String queryString, String paramName, Object value) Execute an HQL query, binding one value to a ":" named parameter in the query string.
List	findByNamedQuery(String queryName) Execute a named query.
List	findByNamedQuery(String queryName, Object value) Execute a named query, binding one value to a "?"
List	findByNamedQuery(String queryName, Object[] values) Execute a named query binding a number of values to "?"
List	findByNamedQueryAndNamedParam(String queryName, String[] paramNames, Object[] values) Execute a named query, binding a number of values to ":" named parameters in the query string.
List	findByNamedQueryAndNamedParam(String queryName, String paramName, Object value) Execute a named query, binding one value to a ":" named parameter in the query string.
List	findByNamedQueryAndValueBean(String queryName, Object valueBean) Execute a named query, binding the properties of the given bean to ":" named parameters in the query string.
List	findByValueBean(String queryString, Object valueBean) Execute an HQL query, binding the properties of the given bean to *named* parameters in the query string.
void	flush() Flush all pending saves, updates and deletes to the database.
Object	get(Class entityClass, Serializable id) Return the persistent instance of the given entity class with the given identifier, or *null* if not found.

Object	*get(Class entityClass, Serializable id, LockMode lockMode)* Return the persistent instance of the given entity class with the given identifier, or *null* if not found.
Object	*get(String entityName, Serializable id)* Return the persistent instance of the given entity class with the given identifier, or *null* if not found.
Object	*get(String entityName, Serializable id, LockMode lockMode)* Return the persistent instance of the given entity class with the given identifier, or *null* if not found.
int	*getFetchSize()* Return the fetch size specified for this HibernateTemplate.
int	*getMaxResults()* Return the maximum number of rows specified for this HibernateTemplate.
String	*getQueryCacheRegion()* Return the name of the cache region for queries executed by this template.
protected Session	*getSession()* Return a Session for use by this template.
void	*initialize(Object proxy)* Force initialization of a Hibernate proxy or persistent collection.
boolean	*isAllowCreate()* Return if a new Session should be created if no thread-bound found.
boolean	*isAlwaysUseNewSession()* Return whether to always use a new Hibernate Session for this template.
boolean	*isCacheQueries()* Return whether to cache all queries executed by this template.
boolean	*isCheckWriteOperations()* Return whether to check that the Hibernate Session is not in read-only mode in case of write operations (save/update/delete).
boolean	*isExposeNativeSession()* Return whether to expose the native Hibernate Session to HibernateCallback code, or rather a Session proxy.
Iterator	*iterate(String queryString)* Execute a query for persistent instances.
Iterator	*iterate(String queryString, Object value)* Execute a query for persistent instances, binding one value to a "?"
Iterator	*iterate(String queryString, Object[] values)* Execute a query for persistent instances, binding a number of values to "?"
Object	*load(Class entityClass, Serializable id)* Return the persistent instance of the given entity class with the given identifier, throwing an exception if not found.

Object	load(Class entityClass, Serializable id, LockMode lockMode) Return the persistent instance of the given entity class with the given identifier, throwing an exception if not found.
void	load(Object entity, Serializable id) Load the persistent instance with the given identifier into the given object, throwing an exception if not found.
Object	load(String entityName, Serializable id) Return the persistent instance of the given entity class with the given identifier, throwing an exception if not found.
Object	load(String entityName, Serializable id, LockMode lockMode) Return the persistent instance of the given entity class with the given identifier, throwing an exception if not found.
List	loadAll(Class entityClass) Return all persistent instances of the given entity class.
void	lock(Object entity, LockMode lockMode) Obtain the specified lock level upon the given object, implicitly checking whether the corresponding database entry still exists.
void	lock(String entityName, Object entity, LockMode lockMode) Obtain the specified lock level upon the given object, implicitly checking whether the corresponding database entry still exists.
Object	merge(Object entity) Copy the state of the given object onto the persistent object with the same identifier.
Object	merge(String entityName, Object entity) Copy the state of the given object onto the persistent object with the same identifier.
void	persist(Object entity) Persist the given transient instance.
void	persist(String entityName, Object entity) Persist the given transient instance.
protected void	prepareCriteria(Criteria criteria) Prepare the given Criteria object, applying cache settings and/or a transaction timeout.
protected void	prepareQuery(Query queryObject) Prepare the given Query object, applying cache settings and/or a transaction timeout.
void	refresh(Object entity) Re-read the state of the given persistent instance.
void	refresh(Object entity, LockMode lockMode) Re-read the state of the given persistent instance.
void	replicate(Object entity, ReplicationMode replicationMode) Persist the state of the given detached instance according to the given replication mode, reusing the current identifier value.
void	replicate(String entityName, Object entity, ReplicationMode replicationMode) Persist the state of the given detached instance according to the given replication mode, reusing the current identifier value.

Serializable	save(Object entity) Persist the given transient instance.
Serializable	save(String entityName, Object entity) Persist the given transient instance.
void	saveOrUpdate(Object entity) Save or update the given persistent instance, according to its id (matching the configured "unsaved-value"?).
void	saveOrUpdate(String entityName, Object entity) Save or update the given persistent instance, according to its id (matching the configured "unsaved-value"?).
void	saveOrUpdateAll(Collection entities) Save or update all given persistent instances, according to its id (matching the configured "unsaved-value"?).
void	setAllowCreate(boolean allowCreate) Set if a new Session should be created when no transactional Session can be found for the current thread.
void	setAlwaysUseNewSession(boolean alwaysUseNewSession) Set whether to always use a new Hibernate Session for this template.
void	setCacheQueries(boolean cacheQueries) Set whether to cache all queries executed by this template.
void	setCheckWriteOperations(boolean checkWriteOperations) Set whether to check that the Hibernate Session is not in read-only mode in case of write operations (save/update/delete).
void	setExposeNativeSession(boolean exposeNativeSession) Set whether to expose the native Hibernate Session to HibernateCallback code.
void	setFetchSize(int fetchSize) Set the fetch size for this HibernateTemplate.
void	setMaxResults(int maxResults) Set the maximum number of rows for this HibernateTemplate.
void	setQueryCacheRegion(String queryCacheRegion) Set the name of the cache region for queries executed by this template.
void	update(Object entity) Update the given persistent instance, associating it with the current Hibernate Session.
void	update(Object entity, LockMode lockMode) Update the given persistent instance, associating it with the current Hibernate Session.
void	update(String entityName, Object entity) Update the given persistent instance, associating it with the current Hibernate Session.
void	update(String entityName, Object entity, LockMode lockMode) Update the given persistent instance, associating it with the current Hibernate Session.

Methods inherited from class org.springframework.orm.hibernate3.HibernateAccessor
afterPropertiesSet, applyFlushMode, convertHibernateAccessException, convertJdbcAccessException, convertJdbcAccessException, disableFilters, enableFilters, flushIfNecessary, getDefaultJdbcExceptionTranslator, getEntityInterceptor, getFilterNames, getFlushMode, getJdbcExceptionTranslator, getSessionFactory, setBeanFactory, setEntityInterceptor, setEntityInterceptorBeanName, setFilterName, setFilterNames, setFlushMode, setFlushModeName, setJdbcExceptionTranslator, setSessionFactory

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

HibernateTemplate

*public **HibernateTemplate**()*

Create a new HibernateTemplate instance.

HibernateTemplate

*public **HibernateTemplate**(SessionFactory sessionFactory)*

Create a new HibernateTemplate instance.

Parameters:
sessionFactory - SessionFactory to create Sessions

HibernateTemplate

```
public HibernateTemplate(SessionFactory sessionFactory,
                         boolean allowCreate)
```

Create a new HibernateTemplate instance.

Parameters:
sessionFactory - SessionFactory to create Sessions
allowCreate - if a non-transactional Session should be created when no transactional Session can be found for the current thread

Method Detail

setAllowCreate

*public void **setAllowCreate**(boolean allowCreate)*

Set if a new *Session* should be created when no transactional *Session* can be found for the current thread. The default value is *true*.

HibernateTemplate is aware of a corresponding *Session* bound to the current thread, for example when using *HibernateTransactionManager*. If *allowCreate* is *true*, a new non-transactional *Session* will be created if none is found, which needs to be closed at the end of the operation. If *false*, an *IllegalStateException* will get thrown in this case.

NOTE: As of Spring 2.5, switching *allowCreate* to *false* will delegate to Hibernate's *SessionFactory.getCurrentSession()* method, which - with Spring-based setup - will by default delegate to Spring's *SessionFactoryUtils.getSession(sessionFactory, false)*. This mode also allows for custom Hibernate CurrentSessionContext strategies to be plugged in, whereas *allowCreate* set to *true* will always use a Spring-managed Hibernate Session.

See Also:

SessionFactoryUtils.getSession(SessionFactory, boolean)

---

isAllowCreate

public boolean ***isAllowCreate()***

Return if a new Session should be created if no thread-bound found.

---

setAlwaysUseNewSession

public void ***setAlwaysUseNewSession****(boolean alwaysUseNewSession)*

Set whether to always use a new Hibernate Session for this template. Default is "false"; if activated, all operations on this template will work on a new Hibernate Session even in case of a pre-bound Session (for example, within a transaction or OpenSessionInViewFilter).

Within a transaction, a new Hibernate Session used by this template will participate in the transaction through using the same JDBC Connection. In such a scenario, multiple Sessions will participate in the same database transaction.

Turn this on for operations that are supposed to always execute independently, without side effects caused by a shared Hibernate Session.

---

isAlwaysUseNewSession

public boolean ***isAlwaysUseNewSession()***

Return whether to always use a new Hibernate Session for this template.

---

setExposeNativeSession

public void ***setExposeNativeSession****(boolean exposeNativeSession)*

Set whether to expose the native Hibernate Session to HibernateCallback code.

Default is "false": a Session proxy will be returned, suppressing *close* calls and automatically applying query cache settings and transaction timeouts.

See Also:

HibernateCallback, Session, setCacheQueries(boolean), setQueryCacheRegion(java.lang.String), prepareQuery(org.hibernate.Query), prepareCriteria(org.hibernate.Criteria)

---

isExposeNativeSession

*public boolean **isExposeNativeSession()***

Return whether to expose the native Hibernate Session to HibernateCallback code, or rather a Session proxy.

---

setCheckWriteOperations

*public void **setCheckWriteOperations**(boolean checkWriteOperations)*

Set whether to check that the Hibernate Session is not in read-only mode in case of write operations (save/update/delete).

Default is "true", for fail-fast behavior when attempting write operations within a read-only transaction. Turn this off to allow save/update/delete on a Session with flush mode NEVER.

See Also:

HibernateAccessor.setFlushMode(int), checkWriteOperationAllowed(org.hibernate.Session), TransactionDefinition.isReadOnly()

---

isCheckWriteOperations

*public boolean **isCheckWriteOperations()***

Return whether to check that the Hibernate Session is not in read-only mode in case of write operations (save/update/delete).

---

setCacheQueries

*public void **setCacheQueries**(boolean cacheQueries)*

Set whether to cache all queries executed by this template.

If this is "true", all Query and Criteria objects created by this template will be marked as cacheable (including all queries through find methods).

To specify the query region to be used for queries cached by this template, set the "queryCacheRegion" property.

See Also:

setQueryCacheRegion(java.lang.String), Query.setCacheable(boolean), Criteria.setCacheable(boolean)

---

isCacheQueries

public boolean ***isCacheQueries()***

Return whether to cache all queries executed by this template.

---

setQueryCacheRegion

public void ***setQueryCacheRegion****(String queryCacheRegion)*

Set the name of the cache region for queries executed by this template.

If this is specified, it will be applied to all Query and Criteria objects created by this template (including all queries through find methods).

The cache region will not take effect unless queries created by this template are configured to be cached via the "cacheQueries" property.

See Also:

setCacheQueries(boolean), Query.setCacheRegion(java.lang.String), Criteria.setCacheRegion(java.lang.String)

---

getQueryCacheRegion

public String ***getQueryCacheRegion()***

Return the name of the cache region for queries executed by this template.

---

setFetchSize

public void ***setFetchSize****(int fetchSize)*

Set the fetch size for this HibernateTemplate. This is important for processing large result sets: Setting this higher than the default value will increase processing speed at the cost of memory consumption; setting this lower can avoid transferring row data that will never be read by the application.

Default is 0, indicating to use the JDBC driver's default.

---

getFetchSize

public int ***getFetchSize()***

Return the fetch size specified for this HibernateTemplate.

---

setMaxResults

*public void **setMaxResults**(int maxResults)*

Set the maximum number of rows for this HibernateTemplate. This is important for processing subsets of large result sets, avoiding to read and hold the entire result set in the database or in the JDBC driver if we're never interested in the entire result in the first place (for example, when performing searches that might return a large number of matches).

Default is 0, indicating to use the JDBC driver's default.

---

getMaxResults

*public int **getMaxResults**()*

Return the maximum number of rows specified for this HibernateTemplate.

---

execute

*public Object **execute**(HibernateCallback action)*
throws DataAccessException

Description copied from interface: *HibernateOperations*
Execute the action specified by the given action object within a *Session*.

Application exceptions thrown by the action object get propagated to the caller (can only be unchecked). Hibernate exceptions are transformed into appropriate DAO ones. Allows for returning a result object, that is a domain object or a collection of domain objects.

Note: Callback code is not supposed to handle transactions itself! Use an appropriate transaction manager like *HibernateTransactionManager*. Generally, callback code must not touch any *Session* lifecycle methods, like close, disconnect, or reconnect, to let the template do its work.

Specified by:
execute in interface *HibernateOperations*

Parameters:
action - callback object that specifies the Hibernate action
Returns:
a result object returned by the action, or *null*
Throws:
DataAccessException - in case of Hibernate errors
See Also:
HibernateTransactionManager, *org.springframework.dao*, *org.springframework.transaction*, *Session*

---

executeFind

*public List **executeFind**(HibernateCallback action)*
throws DataAccessException

Description copied from interface: *HibernateOperations*

Execute the specified action assuming that the result object is a *List*.

This is a convenience method for executing Hibernate find calls or queries within an action.

Specified by:

executeFind in interface *HibernateOperations*

Parameters:

action - calback object that specifies the Hibernate action

Returns:

a List result returned by the action, or *null*

Throws:

DataAccessException - in case of Hibernate errors

---

execute

```
public Object execute(HibernateCallback action,
                      boolean exposeNativeSession)
               throws DataAccessException
```

Execute the action specified by the given action object within a Session.

Parameters:

action - callback object that specifies the Hibernate action

exposeNativeSession - whether to expose the native Hibernate Session to callback code

Returns:

a result object returned by the action, or *null*

Throws:

DataAccessException - in case of Hibernate errors

---

getSession

```
protected Session getSession()
```

Return a Session for use by this template.

Returns a new Session in case of "alwaysUseNewSession" (using the same JDBC Connection as a transactional Session, if applicable), a pre-bound Session in case of "allowCreate" turned off, and a pre-bound or new Session else (new only if no transactional or otherwise pre-bound Session exists).

Returns:

the Session to use (never *null*)

See Also:

SessionFactoryUtils.getSession(org.hibernate.SessionFactory, boolean), *SessionFactoryUtils.getNewSession(org.hibernate.SessionFactory)*, *setAlwaysUseNewSession(boolean)*, *setAllowCreate(boolean)*

---

createSessionProxy

```
protected Session createSessionProxy(Session session)
```

Create a close-suppressing proxy for the given Hibernate Session. The proxy also prepares returned Query and Criteria objects.

Parameters:
session - the Hibernate Session to create a proxy for

Returns:
the Session proxy

See Also:
`Session.close()`, `prepareQuery(org.hibernate.Query)`, `prepareCriteria(org.hibernate.Criteria)`

---

get

```
public Object get(Class entityClass,
                  Serializable id)
           throws DataAccessException
```

Description copied from interface: `HibernateOperations`
Return the persistent instance of the given entity class with the given identifier, or *null* if not found.

This method is a thin wrapper around `Session.get(Class, java.io.Serializable)` for convenience. For an explanation of the exact semantics of this method, please do refer to the Hibernate API documentation in the first instance.

Specified by:
`get` in interface `HibernateOperations`

Parameters:
entityClass - a persistent class
id - the identifier of the persistent instance

Returns:
the persistent instance, or *null* if not found

Throws:
`DataAccessException` - in case of Hibernate errors

See Also:
`Session.get(Class, java.io.Serializable)`

---

get

```
public Object get(Class entityClass,
                  Serializable id,
                  LockMode lockMode)
           throws DataAccessException
```

Description copied from interface: `HibernateOperations`

Return the persistent instance of the given entity class with the given identifier, or *null* if not found.

Obtains the specified lock mode if the instance exists.

This method is a thin wrapper around *Session.get(Class, java.io.Serializable, LockMode)* for convenience. For an explanation of the exact semantics of this method, please do refer to the Hibernate API documentation in the first instance.

Specified by:

get in interface *HibernateOperations*

Parameters:

entityClass - a persistent class

id - the identifier of the persistent instance

lockMode - the lock mode to obtain

Returns:

the persistent instance, or *null* if not found

Throws:

DataAccessException - in case of Hibernate errors

See Also:

Session.get(Class, java.io.Serializable, org.hibernate.LockMode)

---

get

```
public Object get(String entityName,
                  Serializable id)
           throws DataAccessException
```

Description copied from interface: *HibernateOperations*

Return the persistent instance of the given entity class with the given identifier, or *null* if not found.

This method is a thin wrapper around *Session.get(String, java.io.Serializable)* for convenience. For an explanation of the exact semantics of this method, please do refer to the Hibernate API documentation in the first instance.

Specified by:

get in interface *HibernateOperations*

Parameters:

entityName - the name of the persistent entity

id - the identifier of the persistent instance

Returns:

the persistent instance, or *null* if not found

Throws:

DataAccessException - in case of Hibernate errors

See Also:

Session.get(Class, java.io.Serializable)

---

get

```
public Object get(String entityName,
                  Serializable id,
                  LockMode lockMode)
           throws DataAccessException
```

Description copied from interface: *HibernateOperations*

Return the persistent instance of the given entity class with the given identifier, or *null* if not found. Obtains the specified lock mode if the instance exists.

This method is a thin wrapper around *Session.get(String, java.io.Serializable, LockMode)* for convenience. For an explanation of the exact semantics of this method, please do refer to the Hibernate API documentation in the first instance.

Specified by:

get in interface *HibernateOperations*

Parameters:

entityName - the name of the persistent entity

id - the identifier of the persistent instance

lockMode - the lock mode to obtain

Returns:

the persistent instance, or *null* if not found

Throws:

DataAccessException - in case of Hibernate errors

See Also:

Session.get(Class, java.io.Serializable, org.hibernate.LockMode)

---

load

```
public Object load(Class entityClass,
                   Serializable id)
            throws DataAccessException
```

Description copied from interface: *HibernateOperations*

Return the persistent instance of the given entity class with the given identifier, throwing an exception if not found.

This method is a thin wrapper around *Session.load(Class, java.io.Serializable)* for convenience. For an explanation of the exact semantics of this method, please do refer to the Hibernate API documentation in the first instance.

Specified by:

load in interface *HibernateOperations*

Parameters:

entityClass - a persistent class

id - the identifier of the persistent instance

Returns:

the persistent instance

Throws:

ObjectRetrievalFailureException - if not found

DataAccessException - in case of Hibernate errors

See Also:

Session.load(Class, java.io.Serializable)

---

load

```
public Object load(Class entityClass,
                   Serializable id,
                   LockMode lockMode)
            throws DataAccessException
```

Description copied from interface: *HibernateOperations*

Return the persistent instance of the given entity class with the given identifier, throwing an exception if not found. Obtains the specified lock mode if the instance exists.

This method is a thin wrapper around *Session.load(Class, java.io.Serializable, LockMode)* for convenience. For an explanation of the exact semantics of this method, please do refer to the Hibernate API documentation in the first instance.

Specified by:

load in interface *HibernateOperations*

Parameters:

entityClass - a persistent class

id - the identifier of the persistent instance

lockMode - the lock mode to obtain

Returns:

the persistent instance

Throws:

ObjectRetrievalFailureException - if not found

DataAccessException - in case of Hibernate errors

See Also:

Session.load(Class, java.io.Serializable)

---

load

```
public Object load(String entityName,
                   Serializable id)
            throws DataAccessException
```

Description copied from interface: *HibernateOperations*

Return the persistent instance of the given entity class with the given identifier, throwing an exception if not found.

This method is a thin wrapper around *Session.load(String, java.io.Serializable)* for convenience. For an explanation of the exact semantics of this method, please do refer to the Hibernate API documentation in the first instance.

Specified by:

load in interface *HibernateOperations*

Parameters:

entityName - the name of the persistent entity

id - the identifier of the persistent instance

Returns:

the persistent instance

Throws:

ObjectRetrievalFailureException - if not found

DataAccessException - in case of Hibernate errors

See Also:

Session.load(Class, java.io.Serializable)

---

load

```
public Object load(String entityName,
                   Serializable id,
                   LockMode lockMode)
            throws DataAccessException
```

Description copied from interface: *HibernateOperations*

Return the persistent instance of the given entity class with the given identifier, throwing an exception if not found.

Obtains the specified lock mode if the instance exists.

This method is a thin wrapper around *Session.load(String, java.io.Serializable, LockMode)* for convenience. For an explanation of the exact semantics of this method, please do refer to the Hibernate API documentation in the first instance.

Specified by:

load in interface *HibernateOperations*

Parameters:

entityName - the name of the persistent entity

id - the identifier of the persistent instance

lockMode - the lock mode to obtain

Returns:

the persistent instance

Throws:

ObjectRetrievalFailureException - if not found

`DataAccessException` - in case of Hibernate errors

See Also:

`Session.load(Class, java.io.Serializable)`

---

loadAll

```
public List loadAll(Class entityClass)
             throws DataAccessException
```

Description copied from interface: `HibernateOperations`

Return all persistent instances of the given entity class. Note: Use queries or criteria for retrieving a specific subset.

Specified by:

`loadAll` in interface `HibernateOperations`

Parameters:

entityClass - a persistent class

Returns:

a `List` containing 0 or more persistent instances

Throws:

`DataAccessException` - if there is a Hibernate error

See Also:

`Session.createCriteria(java.lang.Class)`

---

load

```
public void load(Object entity,
                 Serializable id)
          throws DataAccessException
```

Description copied from interface: `HibernateOperations`

Load the persistent instance with the given identifier into the given object, throwing an exception if not found.

This method is a thin wrapper around `Session.load(Object, java.io.Serializable)` for convenience. For an explanation of the exact semantics of this method, please do refer to the Hibernate API documentation in the first instance.

Specified by:

`load` in interface `HibernateOperations`

Parameters:

entity - the object (of the target class) to load into

id - the identifier of the persistent instance

Throws:

`ObjectRetrievalFailureException` - if not found

`DataAccessException` - in case of Hibernate errors

See Also:

Session.load(Object, java.io.Serializable)

---

refresh

```
public void refresh(Object entity)
             throws DataAccessException
```

Description copied from interface: *HibernateOperations*
Re-read the state of the given persistent instance.

Specified by:

refresh in interface *HibernateOperations*

Parameters:

entity - the persistent instance to re-read

Throws:

DataAccessException - in case of Hibernate errors

See Also:

Session.refresh(Object)

---

refresh

```
public void refresh(Object entity,
                    LockMode lockMode)
             throws DataAccessException
```

Description copied from interface: *HibernateOperations*
Re-read the state of the given persistent instance. Obtains the specified lock mode for the instance.

Specified by:

refresh in interface *HibernateOperations*

Parameters:

entity - the persistent instance to re-read

lockMode - the lock mode to obtain

Throws:

DataAccessException - in case of Hibernate errors

See Also:

Session.refresh(Object, org.hibernate.LockMode)

---

contains

```
public boolean contains(Object entity)
                 throws DataAccessException
```

Description copied from interface: *HibernateOperations*

Check whether the given object is in the Session cache.

Specified by:
contains in interface *HibernateOperations*

Parameters:
entity - the persistence instance to check

Returns:
whether the given object is in the Session cache

Throws:
DataAccessException - if there is a Hibernate error

See Also:
Session.contains(java.lang.Object)

---

evict

```
public void evict(Object entity)
           throws DataAccessException
```

Description copied from interface: *HibernateOperations*

Remove the given object from the *Session* cache.

Specified by:
evict in interface *HibernateOperations*

Parameters:
entity - the persistent instance to evict

Throws:
DataAccessException - in case of Hibernate errors

See Also:
Session.evict(java.lang.Object)

---

initialize

```
public void initialize(Object proxy)
                throws DataAccessException
```

Description copied from interface: *HibernateOperations*
Force initialization of a Hibernate proxy or persistent collection.

Specified by:
initialize in interface *HibernateOperations*

Parameters:
proxy - a proxy for a persistent object or a persistent collection

Throws:
DataAccessException - if we can't initialize the proxy, for example because it is not associated with an active Session

See Also:

Hibernate.initialize(java.lang.Object)

---

enableFilter

```
public Filter enableFilter(String filterName)
                    throws IllegalStateException
```

Description copied from interface: *HibernateOperations*

Return an enabled Hibernate *Filter* for the given filter name. The returned *Filter* instance can be used to set filter parameters.

Specified by:

enableFilter in interface *HibernateOperations*

Parameters:

filterName - the name of the filter

Returns:

the enabled Hibernate *Filter* (either already enabled or enabled on the fly by this operation)

Throws:

IllegalStateException - if we are not running within a transactional Session (in which case this operation does not make sense)

---

lock

```
public void lock(Object entity,
                 LockMode lockMode)
          throws DataAccessException
```

Description copied from interface: *HibernateOperations*

Obtain the specified lock level upon the given object, implicitly checking whether the corresponding database entry still exists.

Specified by:

lock in interface *HibernateOperations*

Parameters:

entity - the persistent instance to lock

lockMode - the lock mode to obtain

Throws:

ObjectOptimisticLockingFailureException - if not found

DataAccessException - in case of Hibernate errors

See Also:

Session.lock(Object, org.hibernate.LockMode)

---

lock

```
public void lock(String entityName,
                 Object entity,
                 LockMode lockMode)
          throws DataAccessException
```

Description copied from interface: *HibernateOperations*
Obtain the specified lock level upon the given object, implicitly checking whether the corresponding database entry still exists.

Specified by:
lock in interface *HibernateOperations*

Parameters:
entityName - the name of the persistent entity
entity - the persistent instance to lock
lockMode - the lock mode to obtain

Throws:
ObjectOptimisticLockingFailureException - if not found
DataAccessException - in case of Hibernate errors

See Also:
Session.lock(String, Object, org.hibernate.LockMode)

---

save

```
public Serializable save(Object entity)
                  throws DataAccessException
```

Description copied from interface: *HibernateOperations*
Persist the given transient instance.

Specified by:
save in interface *HibernateOperations*

Parameters:
entity - the transient instance to persist

Returns:
the generated identifier

Throws:
DataAccessException - in case of Hibernate errors

See Also:
Session.save(Object)

---

save

```
public Serializable save(String entityName,
                         Object entity)
                  throws DataAccessException
```

Description copied from interface: *HibernateOperations*
Persist the given transient instance.

Specified by:
save in interface *HibernateOperations*

Parameters:
entityName - the name of the persistent entity
entity - the transient instance to persist

Returns:
the generated identifier

Throws:
DataAccessException - in case of Hibernate errors

See Also:
Session.save(String, Object)

---

update

```
public void update(Object entity)
            throws DataAccessException
```

Description copied from interface: *HibernateOperations*
Update the given persistent instance, associating it with the current Hibernate *Session*.

Specified by:
update in interface *HibernateOperations*

Parameters:
entity - the persistent instance to update

Throws:
DataAccessException - in case of Hibernate errors

See Also:
Session.update(Object)

---

update

```
public void update(Object entity,
                   LockMode lockMode)
            throws DataAccessException
```

Description copied from interface: *HibernateOperations*
Update the given persistent instance, associating it with the current Hibernate *Session*.

Obtains the specified lock mode if the instance exists, implicitly checking whether the corresponding database entry still exists.

Specified by:
update in interface *HibernateOperations*

Parameters:

entity - the persistent instance to update

lockMode - the lock mode to obtain

Throws:

ObjectOptimisticLockingFailureException - if not found

DataAccessException - in case of Hibernate errors

See Also:

Session.update(Object)

---

update

```
public void update(String entityName,
                   Object entity)
            throws DataAccessException
```

Description copied from interface: *HibernateOperations*

Update the given persistent instance, associating it with the current Hibernate *Session*.

Specified by:

update in interface *HibernateOperations*

Parameters:

entityName - the name of the persistent entity

entity - the persistent instance to update

Throws:

DataAccessException - in case of Hibernate errors

See Also:

Session.update(String, Object)

---

update

```
public void update(String entityName,
                   Object entity,
                   LockMode lockMode)
            throws DataAccessException
```

Description copied from interface: *HibernateOperations*

Update the given persistent instance, associating it with the current Hibernate *Session*.

Obtains the specified lock mode if the instance exists, implicitly checking whether the corresponding database entry still exists.

Specified by:

update in interface *HibernateOperations*

Parameters:

entityName - the name of the persistent entity

entity - the persistent instance to update

lockMode - the lock mode to obtain

Throws:

ObjectOptimisticLockingFailureException - if not found

DataAccessException - in case of Hibernate errors

See Also:

Session.update(String, Object)

---

saveOrUpdate

```
public void saveOrUpdate(Object entity)
                  throws DataAccessException
```

Description copied from interface: *HibernateOperations*

Save or update the given persistent instance, according to its id (matching the configured "unsaved-value"?). Associates the instance with the current Hibernate *Session*.

Specified by:

saveOrUpdate in interface *HibernateOperations*

Parameters:

entity - the persistent instance to save or update (to be associated with the Hibernate *Session*)

Throws:

DataAccessException - in case of Hibernate errors

See Also:

Session.saveOrUpdate(Object)

---

saveOrUpdate

```
public void saveOrUpdate(String entityName,
                         Object entity)
                  throws DataAccessException
```

Description copied from interface: *HibernateOperations*

Save or update the given persistent instance, according to its id (matching the configured "unsaved-value"?). Associates the instance with the current Hibernate *Session*.

Specified by:

saveOrUpdate in interface *HibernateOperations*

Parameters:

entityName - the name of the persistent entity

entity - the persistent instance to save or update (to be associated with the Hibernate *Session*)

Throws:

DataAccessException - in case of Hibernate errors

See Also:

Session.saveOrUpdate(String, Object)

saveOrUpdateAll

```
public void saveOrUpdateAll(Collection entities)
                     throws DataAccessException
```

Description copied from interface: *HibernateOperations*
Save or update all given persistent instances, according to its id (matching the configured "unsaved-value"?). Associates the instances with the current Hibernate *Session*.

Specified by:
saveOrUpdateAll in interface *HibernateOperations*

Parameters:
entities - the persistent instances to save or update (to be associated with the Hibernate *Session*)

Throws:
DataAccessException - in case of Hibernate errors

---

replicate

```
public void replicate(Object entity,
                      ReplicationMode replicationMode)
               throws DataAccessException
```

Description copied from interface: *HibernateOperations*
Persist the state of the given detached instance according to the given replication mode, reusing the current identifier value.

Specified by:
replicate in interface *HibernateOperations*

Parameters:
entity - the persistent object to replicate
replicationMode - the Hibernate ReplicationMode

Throws:
DataAccessException - in case of Hibernate errors

See Also:
Session.replicate(Object, org.hibernate.ReplicationMode)

---

replicate

```
public void replicate(String entityName,
                      Object entity,
                      ReplicationMode replicationMode)
               throws DataAccessException
```

Description copied from interface: *HibernateOperations*
Persist the state of the given detached instance according to the given replication mode, reusing the current identifier value.

Specified by:
replicate in interface *HibernateOperations*

Parameters:
entityName - the name of the persistent entity
entity - the persistent object to replicate
replicationMode - the Hibernate ReplicationMode

Throws:
DataAccessException - in case of Hibernate errors

See Also:
Session.replicate(String, Object, org.hibernate.ReplicationMode)

---

persist

```
public void persist(Object entity)
             throws DataAccessException
```

Description copied from interface: *HibernateOperations*
Persist the given transient instance. Follows JSR-220 semantics.

Similar to *save*, associating the given object with the current Hibernate *Session*.

Specified by:
persist in interface *HibernateOperations*

Parameters:
entity - the persistent instance to persist

Throws:
DataAccessException - in case of Hibernate errors

See Also:
Session.persist(Object), *HibernateOperations.save(java.lang.Object)*

---

persist

```
public void persist(String entityName,
                    Object entity)
             throws DataAccessException
```

Description copied from interface: *HibernateOperations*
Persist the given transient instance. Follows JSR-220 semantics.

Similar to *save*, associating the given object with the current Hibernate *Session*.

Specified by:

persist in interface *HibernateOperations*

Parameters:

entityName - the name of the persistent entity

entity - the persistent instance to persist

Throws:

DataAccessException - in case of Hibernate errors

See Also:

Session.persist(String, Object), *HibernateOperations.save(java.lang.Object)*

---

merge

```
public Object merge(Object entity)
             throws DataAccessException
```

Description copied from interface: *HibernateOperations*

Copy the state of the given object onto the persistent object with the same identifier. Follows JSR-220 semantics.

Similar to *saveOrUpdate*, but never associates the given object with the current Hibernate Session. In case of a new entity, the state will be copied over as well.

Note that *merge* will *not* update the identifiers in the passed-in object graph (in contrast to TopLink)! Consider registering Spring's *IdTransferringMergeEventListener* if you would like to have newly assigned ids transferred to the original object graph too.

Specified by:

merge in interface *HibernateOperations*

Parameters:

entity - the object to merge with the corresponding persistence instance

Returns:

the updated, registered persistent instance

Throws:

DataAccessException - in case of Hibernate errors

See Also:

Session.merge(Object), *HibernateOperations.saveOrUpdate(java.lang.Object)*, *IdTransferringMergeEventListener*

---

merge

```
public Object merge(String entityName,
                    Object entity)
             throws DataAccessException
```

Description copied from interface: *HibernateOperations*

Copy the state of the given object onto the persistent object with the same identifier. Follows JSR-220 semantics.

Similar to *saveOrUpdate*, but never associates the given object with the current Hibernate *Session*. In the case of a new entity, the state will be copied over as well.

Note that *merge* will *not* update the identifiers in the passed-in object graph (in contrast to TopLink)! Consider registering Spring's *IdTransferringMergeEventListener* if you would like to have newly assigned ids transferred to the original object graph too.

Specified by:

merge in interface *HibernateOperations*

Parameters:

entityName - the name of the persistent entity

entity - the object to merge with the corresponding persistence instance

Returns:

the updated, registered persistent instance

Throws:

DataAccessException - in case of Hibernate errors

See Also:

Session.merge(String, Object), *HibernateOperations.saveOrUpdate(java.lang.Object)*

---

delete

```
public void delete(Object entity)
            throws DataAccessException
```

Description copied from interface: *HibernateOperations*

Delete the given persistent instance.

Specified by:

delete in interface *HibernateOperations*

Parameters:

entity - the persistent instance to delete

Throws:

DataAccessException - in case of Hibernate errors

See Also:

Session.delete(Object)

---

delete

```
public void delete(Object entity,
                   LockMode lockMode)
            throws DataAccessException
```

Description copied from interface: *HibernateOperations*

Delete the given persistent instance.

Obtains the specified lock mode if the instance exists, implicitly checking whether the corresponding database entry still exists.

Specified by:

delete in interface *HibernateOperations*

Parameters:

entity - the persistent instance to delete

lockMode - the lock mode to obtain

Throws:

ObjectOptimisticLockingFailureException - if not found

DataAccessException - in case of Hibernate errors

See Also:

Session.delete(Object)

---

delete

```
public void delete(String entityName,
                   Object entity)
            throws DataAccessException
```

Description copied from interface: *HibernateOperations*

Delete the given persistent instance.

Specified by:

delete in interface *HibernateOperations*

Parameters:

entityName - the name of the persistent entity

entity - the persistent instance to delete

Throws:

DataAccessException - in case of Hibernate errors

See Also:

Session.delete(Object)

---

delete

```
public void delete(String entityName,
                   Object entity,
                   LockMode lockMode)
            throws DataAccessException
```

Description copied from interface: *HibernateOperations*

Delete the given persistent instance.

Obtains the specified lock mode if the instance exists, implicitly checking whether the corresponding database entry still exists.

Specified by:

delete in interface *HibernateOperations*

Parameters:

entityName - the name of the persistent entity

entity - the persistent instance to delete

lockMode - the lock mode to obtain

Throws:

ObjectOptimisticLockingFailureException - if not found

DataAccessException - in case of Hibernate errors

See Also:

Session.delete(Object)

---

deleteAll

```
public void deleteAll(Collection entities)
               throws DataAccessException
```

Description copied from interface: *HibernateOperations*

Delete all given persistent instances.

This can be combined with any of the find methods to delete by query in two lines of code.

Specified by:

deleteAll in interface *HibernateOperations*

Parameters:

entities - the persistent instances to delete

Throws:

DataAccessException - in case of Hibernate errors

See Also:

Session.delete(Object)

---

flush

```
public void flush()
           throws DataAccessException
```

Description copied from interface: *HibernateOperations*

Flush all pending saves, updates and deletes to the database.

Only invoke this for selective eager flushing, for example when JDBC code needs to see certain changes within the same transaction. Else, it is preferable to rely on auto-flushing at transaction completion.

Specified by:

flush in interface *HibernateOperations*

Throws:

DataAccessException - in case of Hibernate errors

See Also:

Session.flush()

---

clear

public void clear()
throws DataAccessException

Description copied from interface: HibernateOperations

Remove all objects from the Session cache, and cancel all pending saves, updates and deletes.

Specified by:

clear in interface HibernateOperations

Throws:

DataAccessException - in case of Hibernate errors

See Also:

Session.clear()

---

find

public List find(String queryString)
throws DataAccessException

Description copied from interface: HibernateOperations

Execute an HQL query.

Specified by:

find in interface HibernateOperations

Parameters:

queryString - a query expressed in Hibernate's query language

Returns:

a List containing the results of the query execution

Throws:

DataAccessException - in case of Hibernate errors

See Also:

Session.createQuery(java.lang.String)

---

find

public List find(String queryString,
Object value)
throws DataAccessException

Description copied from interface: HibernateOperations

Execute an HQL query, binding one value to a "?" parameter in the query string.

Specified by:

find in interface *HibernateOperations*

Parameters:

queryString - a query expressed in Hibernate's query language

value - the value of the parameter

Returns:

a *List* containing the results of the query execution

Throws:

DataAccessException - in case of Hibernate errors

See Also:

Session.createQuery(java.lang.String)

---

find

```
public List find(String queryString,
                 Object[] values)
          throws DataAccessException
```

Description copied from interface: *HibernateOperations*

Execute an HQL query, binding a number of values to "?" parameters in the query string.

Specified by:

find in interface *HibernateOperations*

Parameters:

queryString - a query expressed in Hibernate's query language

values - the values of the parameters

Returns:

a *List* containing the results of the query execution

Throws:

DataAccessException - in case of Hibernate errors

See Also:

Session.createQuery(java.lang.String)

---

findByNamedParam

```
public List findByNamedParam(String queryString,
                             String paramName,
                             Object value)
                      throws DataAccessException
```

Description copied from interface: *HibernateOperations*

Execute an HQL query, binding one value to a ":" named parameter in the query string.

Specified by:

findByNamedParam in interface *HibernateOperations*

Parameters:

queryString - a query expressed in Hibernate's query language

paramName - the name of the parameter

value - the value of the parameter

Returns:

a *List* containing the results of the query execution

Throws:

DataAccessException - in case of Hibernate errors

See Also:

Session.getNamedQuery(String)

---

findByNamedParam

```
public List findByNamedParam(String queryString,
                             String[] paramNames,
                             Object[] values)
                      throws DataAccessException
```

Description copied from interface: *HibernateOperations*

Execute an HQL query, binding a number of values to ":" named parameters in the query string.

Specified by:

findByNamedParam in interface *HibernateOperations*

Parameters:

queryString - a query expressed in Hibernate's query language

paramNames - the names of the parameters

values - the values of the parameters

Returns:

a *List* containing the results of the query execution

Throws:

DataAccessException - in case of Hibernate errors

See Also:

Session.getNamedQuery(String)

---

findByValueBean

```
public List findByValueBean(String queryString,
                            Object valueBean)
                     throws DataAccessException
```

Description copied from interface: *HibernateOperations*

Execute an HQL query, binding the properties of the given bean to *named* parameters in the query string.

Specified by:

findByValueBean in interface *HibernateOperations*

Parameters:

queryString - a query expressed in Hibernate's query language

valueBean - the values of the parameters

Returns:

a *List* containing the results of the query execution

Throws:

DataAccessException - in case of Hibernate errors

See Also:

Query.setProperties(java.lang.Object), *Session.createQuery(java.lang.String)*

---

findByNamedQuery

```
public List findByNamedQuery(String queryName)
                      throws DataAccessException
```

Description copied from interface: *HibernateOperations*

Execute a named query.

A named query is defined in a Hibernate mapping file.

Specified by:

findByNamedQuery in interface *HibernateOperations*

Parameters:

queryName - the name of a Hibernate query in a mapping file

Returns:

a *List* containing the results of the query execution

Throws:

DataAccessException - in case of Hibernate errors

See Also:

Session.getNamedQuery(String)

---

findByNamedQuery

```
public List findByNamedQuery(String queryName,
                             Object value)
                      throws DataAccessException
```

Description copied from interface: *HibernateOperations*

Execute a named query, binding one value to a "?" parameter in the query string.

A named query is defined in a Hibernate mapping file.

Specified by:

findByNamedQuery in interface *HibernateOperations*

Parameters:

queryName - the name of a Hibernate query in a mapping file

value - the value of the parameter

Returns:

a *List* containing the results of the query execution

Throws:

DataAccessException - in case of Hibernate errors

See Also:

Session.getNamedQuery(String)

---

findByNamedQuery

```
public List findByNamedQuery(String queryName,
                             Object[] values)
                      throws DataAccessException
```

Description copied from interface: *HibernateOperations*

Execute a named query binding a number of values to "?" parameters in the query string.

A named query is defined in a Hibernate mapping file.

Specified by:

findByNamedQuery in interface *HibernateOperations*

Parameters:

queryName - the name of a Hibernate query in a mapping file

values - the values of the parameters

Returns:

a *List* containing the results of the query execution

Throws:

DataAccessException - in case of Hibernate errors

See Also:

Session.getNamedQuery(String)

---

findByNamedQueryAndNamedParam

```
public List findByNamedQueryAndNamedParam(String queryName,
                                          String paramName,
                                          Object value)
                                   throws DataAccessException
```

Description copied from interface: *HibernateOperations*

Execute a named query, binding one value to a ":" named parameter in the query string.

A named query is defined in a Hibernate mapping file.

Specified by:

findByNamedQueryAndNamedParam in interface *HibernateOperations*

Parameters:

queryName - the name of a Hibernate query in a mapping file

paramName - the name of parameter

value - the value of the parameter

Returns:

a *List* containing the results of the query execution

Throws:

DataAccessException - in case of Hibernate errors

See Also:

Session.getNamedQuery(String)

---

findByNamedQueryAndNamedParam

```
public List findByNamedQueryAndNamedParam(String queryName,
                                          String[] paramNames,
                                          Object[] values)
                                   throws DataAccessException
```

Description copied from interface: *HibernateOperations*

Execute a named query, binding a number of values to ":" named parameters in the query string.

A named query is defined in a Hibernate mapping file.

Specified by:

findByNamedQueryAndNamedParam in interface *HibernateOperations*

Parameters:

queryName - the name of a Hibernate query in a mapping file

paramNames - the names of the parameters

values - the values of the parameters

Returns:

a *List* containing the results of the query execution

Throws:

DataAccessException - in case of Hibernate errors

See Also:

Session.getNamedQuery(String)

---

findByNamedQueryAndValueBean

```
public List findByNamedQueryAndValueBean(String queryName,
                                         Object valueBean)
                                  throws DataAccessException
```

Description copied from interface: *HibernateOperations*

Execute a named query, binding the properties of the given bean to ":" named parameters in the query string.

A named query is defined in a Hibernate mapping file.

Specified by:
findByNamedQueryAndValueBean in interface *HibernateOperations*

Parameters:
queryName - the name of a Hibernate query in a mapping file
valueBean - the values of the parameters

Returns:
a *List* containing the results of the query execution

Throws:
DataAccessException - in case of Hibernate errors

See Also:
Query.setProperties(java.lang.Object), *Session.getNamedQuery(String)*

---

findByCriteria

```
public List findByCriteria(DetachedCriteria criteria)
                    throws DataAccessException
```

Description copied from interface: *HibernateOperations*
Execute a query based on a given Hibernate criteria object.

Specified by:
findByCriteria in interface *HibernateOperations*

Parameters:
criteria - the detached Hibernate criteria object, which can for example be held in an instance variable of a DAO

Returns:
a *List* containing 0 or more persistent instances

Throws:
DataAccessException - in case of Hibernate errors

See Also:
DetachedCriteria.getExecutableCriteria(org.hibernate.Session)

---

findByCriteria

```
public List findByCriteria(DetachedCriteria criteria,
                           int firstResult,
                           int maxResults)
                    throws DataAccessException
```

Description copied from interface: *HibernateOperations*
Execute a query based on the given Hibernate criteria object.

Specified by:
findByCriteria in interface *HibernateOperations*

Parameters:

criteria - the detached Hibernate criteria object, which can for example be held in an instance variable of a DAO
firstResult - the index of the first result object to be retrieved (numbered from 0)
maxResults - the maximum number of result objects to retrieve (or <=0 for no limit)

Returns:

a *List* containing 0 or more persistent instances

Throws:

DataAccessException - in case of Hibernate errors

See Also:

DetachedCriteria.getExecutableCriteria(org.hibernate.Session), *Criteria.setFirstResult(int)*, *Criteria.setMaxResults(int)*

---

findByExample

```
public List findByExample(Object exampleEntity)
                   throws DataAccessException
```

Description copied from interface: *HibernateOperations*
Execute a query based on the given example entity object.

Specified by:

findByExample in interface *HibernateOperations*

Parameters:

exampleEntity - an instance of the desired entity, serving as example for "query-by-example"

Returns:

a *List* containing 0 or more persistent instances

Throws:

DataAccessException - in case of Hibernate errors

See Also:

Example.create(Object)

---

findByExample

```
public List findByExample(String entityName,
                          Object exampleEntity)
                   throws DataAccessException
```

Description copied from interface: *HibernateOperations*
Execute a query based on the given example entity object.

Specified by:

findByExample in interface *HibernateOperations*

Parameters:

entityName - the name of the persistent entity

exampleEntity - an instance of the desired entity, serving as example for "query-by-example"

Returns:

a *List* containing 0 or more persistent instances

Throws:

DataAccessException - in case of Hibernate errors

See Also:

Example.create(Object)

---

findByExample

```
public List findByExample(Object exampleEntity,
                          int firstResult,
                          int maxResults)
                   throws DataAccessException
```

Description copied from interface: *HibernateOperations*

Execute a query based on a given example entity object.

Specified by:

findByExample in interface *HibernateOperations*

Parameters:

exampleEntity - an instance of the desired entity, serving as example for "query-by-example"

firstResult - the index of the first result object to be retrieved (numbered from 0)

maxResults - the maximum number of result objects to retrieve (or <=0 for no limit)

Returns:

a *List* containing 0 or more persistent instances

Throws:

DataAccessException - in case of Hibernate errors

See Also:

Example.create(Object), *Criteria.setFirstResult(int)*, *Criteria.setMaxResults(int)*

---

findByExample

```
public List findByExample(String entityName,
                          Object exampleEntity,
                          int firstResult,
                          int maxResults)
                   throws DataAccessException
```

Description copied from interface: *HibernateOperations*

Execute a query based on a given example entity object.

Specified by:

findByExample in interface *HibernateOperations*

Parameters:

entityName - the name of the persistent entity

exampleEntity - an instance of the desired entity, serving as example for "query-by-example"

firstResult - the index of the first result object to be retrieved (numbered from 0)

maxResults - the maximum number of result objects to retrieve (or <=0 for no limit)

Returns:

a *List* containing 0 or more persistent instances

Throws:

DataAccessException - in case of Hibernate errors

See Also:

Example.create(Object), *Criteria.setFirstResult(int)*, *Criteria.setMaxResults(int)*

---

iterate

```
public Iterator iterate(String queryString)
                 throws DataAccessException
```

Description copied from interface: *HibernateOperations*
Execute a query for persistent instances.

Returns the results as an *Iterator*. Entities returned are initialized on demand. See the Hibernate API documentation for details.

Specified by:

iterate in interface *HibernateOperations*

Parameters:

queryString - a query expressed in Hibernate's query language

Returns:

an *Iterator* containing 0 or more persistent instances

Throws:

DataAccessException - in case of Hibernate errors

See Also:

Session.createQuery(java.lang.String), *Query.iterate()*

---

iterate

```
public Iterator iterate(String queryString,
                        Object value)
                 throws DataAccessException
```

Description copied from interface: *HibernateOperations*
Execute a query for persistent instances, binding one value to a "?" parameter in the query string.

Returns the results as an *Iterator*. Entities returned are initialized on demand. See the Hibernate API documentation for details.

Specified by:

iterate in interface *HibernateOperations*

Parameters:

queryString - a query expressed in Hibernate's query language

value - the value of the parameter

Returns:

an *Iterator* containing 0 or more persistent instances

Throws:

DataAccessException - in case of Hibernate errors

See Also:

Session.createQuery(java.lang.String), *Query.iterate()*

---

iterate

```
public Iterator iterate(String queryString,
                        Object[] values)
                 throws DataAccessException
```

Description copied from interface: *HibernateOperations*

Execute a query for persistent instances, binding a number of values to "?" parameters in the query string.

Returns the results as an *Iterator*. Entities returned are initialized on demand. See the Hibernate API documentation for details.

Specified by:

iterate in interface *HibernateOperations*

Parameters:

queryString - a query expressed in Hibernate's query language

values - the values of the parameters

Returns:

an *Iterator* containing 0 or more persistent instances

Throws:

DataAccessException - in case of Hibernate errors

See Also:

Session.createQuery(java.lang.String), *Query.iterate()*

---

closeIterator

```
public void closeIterator(Iterator it)
                   throws DataAccessException
```

Description copied from interface: *HibernateOperations*

Immediately close an *Iterator* created by any of the various *iterate(..)* operations, instead of waiting until the session is closed or disconnected.

Specified by:

closeIterator in interface *HibernateOperations*

Parameters:
it - the *Iterator* to close

Throws:
DataAccessException - if the *Iterator* could not be closed

See Also:
Hibernate.close(java.util.Iterator)

---

bulkUpdate

```
public int bulkUpdate(String queryString)
               throws DataAccessException
```

Description copied from interface: *HibernateOperations*
Update/delete all objects according to the given query.

Specified by:
bulkUpdate in interface *HibernateOperations*

Parameters:
queryString - an update/delete query expressed in Hibernate's query language

Returns:
the number of instances updated/deleted

Throws:
DataAccessException - in case of Hibernate errors

See Also:
Session.createQuery(java.lang.String), *Query.executeUpdate()*

---

bulkUpdate

```
public int bulkUpdate(String queryString,
                      Object value)
               throws DataAccessException
```

Description copied from interface: *HibernateOperations*
Update/delete all objects according to the given query, binding one value to a "?" parameter in the query string.

Specified by:
bulkUpdate in interface *HibernateOperations*

Parameters:
queryString - an update/delete query expressed in Hibernate's query language
value - the value of the parameter

Returns:
the number of instances updated/deleted

Throws:

DataAccessException - in case of Hibernate errors

See Also:

Session.createQuery(java.lang.String), *Query.executeUpdate()*

---

bulkUpdate

```
public int bulkUpdate(String queryString,
                      Object[] values)
               throws DataAccessException
```

Description copied from interface: *HibernateOperations*

Update/delete all objects according to the given query, binding a number of values to "?" parameters in the query string.

Specified by:

bulkUpdate in interface *HibernateOperations*

Parameters:

queryString - an update/delete query expressed in Hibernate's query language

values - the values of the parameters

Returns:

the number of instances updated/deleted

Throws:

DataAccessException - in case of Hibernate errors

See Also:

Session.createQuery(java.lang.String), *Query.executeUpdate()*

---

checkWriteOperationAllowed

```
protected void checkWriteOperationAllowed(Session session)
                                   throws InvalidDataAccessApiUsageException
```

Check whether write operations are allowed on the given Session.

Default implementation throws an InvalidDataAccessApiUsageException in case of *FlushMode.NEVER/MANUAL*. Can be overridden in subclasses.

Parameters:

session - current Hibernate Session

Throws:

InvalidDataAccessApiUsageException - if write operations are not allowed

See Also:

setCheckWriteOperations(boolean), *HibernateAccessor.getFlushMode()*, *HibernateAccessor.FLUSH_EAGER*, *Session.getFlushMode()*, *FlushMode.NEVER*, *FlushMode.MANUAL*

---

prepareQuery

protected void ***prepareQuery***(*Query queryObject*)

Prepare the given Query object, applying cache settings and/or a transaction timeout.

Parameters:

queryObject - the Query object to prepare

See Also:

setCacheQueries(boolean), *setQueryCacheRegion(java.lang.String)*, *SessionFactoryUtils.applyTransactionTimeout(org.hibernate.Query, org.hibernate.SessionFactory)*

---

prepareCriteria

protected void ***prepareCriteria***(*Criteria criteria*)

Prepare the given Criteria object, applying cache settings and/or a transaction timeout.

Parameters:

criteria - the Criteria object to prepare

See Also:

setCacheQueries(boolean), *setQueryCacheRegion(java.lang.String)*, *SessionFactoryUtils.applyTransactionTimeout(org.hibernate.Query, org.hibernate.SessionFactory)*

---

applyNamedParameterToQuery

```
protected void applyNamedParameterToQuery(Query queryObject,
                                          String paramName,
                                          Object value)
                                   throws HibernateException
```

Apply the given name parameter to the given Query object.

Parameters:

queryObject - the Query object

paramName - the name of the parameter

value - the value of the parameter

Throws:

HibernateException - if thrown by the Query object
