What do you need to do in Spring if you would like to work with JPA?

Since there are two ways in order to create a JPA EntityManagerFactory, Spring also offers functionality to work with those two types:

LocalEntityManagerFactoryBean
LocalContainerEntityManagerFactoryBean

It is preferably to use the container-managed type because in this case you will not need the persistence.xml file (located in META-INF) and also it allows for scanning packages for @Entity-annotated classes (using setPackagesToScan() of LocalContainerEntityManagerFactoryBean).

org.springframework.orm.jpa

Class AbstractEntityManagerFactoryBean

java.lang.Object
  org.springframework.orm.jpa.AbstractEntityManagerFactoryBean

All Implemented Interfaces:

Serializable, Aware, BeanClassLoaderAware, BeanFactoryAware, BeanNameAware, DisposableBean, FactoryBean<EntityManagerFactory>, InitializingBean, PersistenceExceptionTranslator, EntityManagerFactoryInfo

Direct Known Subclasses:

LocalContainerEntityManagerFactoryBean, LocalEntityManagerFactoryBean

---

```
public abstract class AbstractEntityManagerFactoryBean
extends Object
implements FactoryBean<EntityManagerFactory>, BeanClassLoaderAware, BeanFactoryAware, BeanNameAware, InitializingBean, DisposableE
```

Abstract FactoryBean that creates a local JPA EntityManagerFactory instance within a Spring application context.

Encapsulates the common functionality between the different JPA bootstrap contracts (standalone as well as container).

Implements support for standard JPA configuration conventions as well as Spring's customizable JpaVendorAdapter mechanism, and controls the EntityManagerFactory's lifecycle.

This class also implements the PersistenceExceptionTranslator interface, as autodetected by Spring's PersistenceExceptionTranslationPostProcessor, for AOP-based translation of native exceptions to Spring DataAccessExceptions. Hence, the presence of e.g. LocalEntityManagerFactoryBean automatically enables a PersistenceExceptionTranslationPostProcessor to translate JPA exceptions.

Since:

2.0

Author:

Juergen Hoeller, Rod Johnson

See Also:

LocalEntityManagerFactoryBean, LocalContainerEntityManagerFactoryBean, Serialized Form

Field Summary

Fields

Modifier and Type	Field and Description
protected Log	**logger** Logger available to subclasses

Constructor Summary

Constructors

Constructor and Description
AbstractEntityManagerFactoryBean()

Method Summary

All Methods Instance Methods Abstract Methods Concrete Methods

Modifier and Type	Method and Description
void	**afterPropertiesSet**() Invoked by a BeanFactory after it has set all bean properties supplied (and satisfied BeanFactoryAware and ApplicationContextAware).
protected EntityManagerFactory	**createEntityManagerFactoryProxy**(EntityManagerFactory emf) Create a proxy of the given EntityManagerFactory.
protected abstract EntityManagerFactory	**createNativeEntityManagerFactory**()

	Subclasses must implement this method to create the EntityManagerFactory that will be returned by the `getObject()` method.
void	**destroy**() Close the EntityManagerFactory on bean factory shutdown.
ClassLoader	**getBeanClassLoader**() Return the ClassLoader that the application's beans are loaded with.
AsyncTaskExecutor	**getBootstrapExecutor**() Return the asynchronous executor for background bootstrapping, if any.
DataSource	**getDataSource**() Return the JDBC DataSource that this EntityManagerFactory obtains its JDBC Connections from.
Class<? extends **EntityManager**>	**getEntityManagerInterface**() Return the (potentially vendor-specific) EntityManager interface that this factory's EntityManagers will implement.
JpaDialect	**getJpaDialect**() Return the vendor-specific JpaDialect implementation for this EntityManagerFactory, or `null` if not known.
Map<**String**,**Object**>	**getJpaPropertyMap**() Allow Map access to the JPA properties to be passed to the persistence provider, with the option to add or override specific entries.
JpaVendorAdapter	**getJpaVendorAdapter**() Return the JpaVendorAdapter implementation for this EntityManagerFactory, or `null` if not known.
EntityManagerFactory	**getNativeEntityManagerFactory**() Return the raw underlying EntityManagerFactory.
EntityManagerFactory	**getObject**() Return the singleton EntityManagerFactory.
Class<? extends **EntityManagerFactory**>	**getObjectType**() Return the type of object that this FactoryBean creates, or `null` if not known in advance.
PersistenceProvider	**getPersistenceProvider**() Return the underlying PersistenceProvider that the underlying EntityManagerFactory was created with.
PersistenceUnitInfo	**getPersistenceUnitInfo**() Return the PersistenceUnitInfo used to create this EntityManagerFactory, if the in-container API was used.
String	**getPersistenceUnitName**() Return the name of the persistence unit used to create this EntityManagerFactory, or `null` if it is an unnamed default.
boolean	**isSingleton**() Is the object managed by this factory a singleton? That is, will **FactoryBean.getObject()** always return the same object (a reference that can be cached)?
void	**setBeanClassLoader**(**ClassLoader** classLoader) Callback that supplies the bean **class loader** to a bean instance.
void	**setBeanFactory**(**BeanFactory** beanFactory) Callback that supplies the owning factory to a bean instance.
void	**setBeanName**(**String** name) Set the name of the bean in the bean factory that created this bean.
void	**setBootstrapExecutor**(**AsyncTaskExecutor** bootstrapExecutor) Specify an asynchronous executor for background bootstrapping, e.g.
void	**setEntityManagerFactoryInterface**(**Class**<? extends **EntityManagerFactory**> emfInterface) Specify the (potentially vendor-specific) EntityManagerFactory interface that this EntityManagerFactory proxy is supposed to implement.
void	**setEntityManagerInterface**(**Class**<? extends **EntityManager**> emInterface) Specify the (potentially vendor-specific) EntityManager interface that this factory's EntityManagers are supposed to implement.
void	**setJpaDialect**(**JpaDialect** jpaDialect)

	Specify the vendor-specific JpaDialect implementation to associate with this EntityManagerFactory.
void	**setJpaProperties**(**Properties** jpaProperties) Specify JPA properties, to be passed into `Persistence.createEntityManagerFactory` (if any).
void	**setJpaPropertyMap**(**Map**<**String**,?> jpaProperties) Specify JPA properties as a Map, to be passed into `Persistence.createEntityManagerFactory` (if any).
void	**setJpaVendorAdapter**(**JpaVendorAdapter** jpaVendorAdapter) Specify the JpaVendorAdapter implementation for the desired JPA provider, if any.
void	**setPersistenceProvider**(**PersistenceProvider** persistenceProvider) Set the PersistenceProvider instance to use for creating the EntityManagerFactory.
void	**setPersistenceProviderClass**(**Class**<? extends **PersistenceProvider**> persistenceProviderClass) Set the PersistenceProvider implementation class to use for creating the EntityManagerFactory.
void	**setPersistenceUnitName**(**String** persistenceUnitName) Specify the name of the EntityManagerFactory configuration.
DataAccessException	**translateExceptionIfPossible**(**RuntimeException** ex) Implementation of the PersistenceExceptionTranslator interface, as autodetected by Spring's PersistenceExceptionTranslationPostProcessor.
protected **Object**	**writeReplace**()

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

logger

```
protected final Log logger
```

Logger available to subclasses

Constructor Detail

AbstractEntityManagerFactoryBean

```
public AbstractEntityManagerFactoryBean()
```

Method Detail

setPersistenceProviderClass

```
public void setPersistenceProviderClass(Class<? extends PersistenceProvider> persistenceProviderClass)
```

Set the PersistenceProvider implementation class to use for creating the EntityManagerFactory. If not specified, the persistence provider will be taken from the JpaVendorAdapter (if any) or retrieved through scanning (as far as possible).

See Also:

JpaVendorAdapter.getPersistenceProvider(), PersistenceProvider, Persistence

setPersistenceProvider

```
public void setPersistenceProvider(PersistenceProvider persistenceProvider)
```

Set the PersistenceProvider instance to use for creating the EntityManagerFactory. If not specified, the persistence provider will be taken from the JpaVendorAdapter (if any) or determined by the persistence unit deployment descriptor (as far as possible).

See Also:

JpaVendorAdapter.getPersistenceProvider(), PersistenceProvider, Persistence

getPersistenceProvider

```
public PersistenceProvider getPersistenceProvider()
```

Description copied from interface: EntityManagerFactoryInfo
Return the underlying PersistenceProvider that the underlying EntityManagerFactory was created with.

Specified by:
`getPersistenceProvider` in interface `EntityManagerFactoryInfo`

Returns:
the PersistenceProvider used to create this EntityManagerFactory, or null if the standard JPA provider autodetection process was used to configure the EntityManagerFactory

setPersistenceUnitName

```
public void setPersistenceUnitName(String persistenceUnitName)
```

Specify the name of the EntityManagerFactory configuration.

Default is none, indicating the default EntityManagerFactory configuration. The persistence provider will throw an exception if ambiguous EntityManager configurations are found.

See Also:
`Persistence.createEntityManagerFactory(String)`

getPersistenceUnitName

```
public String getPersistenceUnitName()
```

Description copied from interface: EntityManagerFactoryInfo
Return the name of the persistence unit used to create this EntityManagerFactory, or `null` if it is an unnamed default.

If `getPersistenceUnitInfo()` returns non-null, the result of `getPersistenceUnitName()` must be equal to the value returned by `PersistenceUnitInfo.getPersistenceUnitName()`.

Specified by:
`getPersistenceUnitName` in interface `EntityManagerFactoryInfo`

See Also:
`EntityManagerFactoryInfo.getPersistenceUnitInfo()`, `PersistenceUnitInfo.getPersistenceUnitName()`

setJpaProperties

```
public void setJpaProperties(Properties jpaProperties)
```

Specify JPA properties, to be passed into `Persistence.createEntityManagerFactory` (if any).

Can be populated with a String "value" (parsed via PropertiesEditor) or a "props" element in XML bean definitions.

See Also:
`Persistence.createEntityManagerFactory(String, java.util.Map)`,
`PersistenceProvider.createContainerEntityManagerFactory(javax.persistence.spi.PersistenceUnitInfo, java.util.Map)`

setJpaPropertyMap

```
public void setJpaPropertyMap(Map<String,?> jpaProperties)
```

Specify JPA properties as a Map, to be passed into `Persistence.createEntityManagerFactory` (if any).

Can be populated with a "map" or "props" element in XML bean definitions.

See Also:
`Persistence.createEntityManagerFactory(String, java.util.Map)`,
`PersistenceProvider.createContainerEntityManagerFactory(javax.persistence.spi.PersistenceUnitInfo, java.util.Map)`

getJpaPropertyMap

```
public Map<String,Object> getJpaPropertyMap()
```

Allow Map access to the JPA properties to be passed to the persistence provider, with the option to add or override specific entries.

Useful for specifying entries directly, for example via "jpaPropertyMap[myKey]".

setEntityManagerFactoryInterface

```
public void setEntityManagerFactoryInterface(Class<? extends EntityManagerFactory> emfInterface)
```

Specify the (potentially vendor-specific) EntityManagerFactory interface that this EntityManagerFactory proxy is supposed to implement.

The default will be taken from the specific JpaVendorAdapter, if any, or set to the standard `javax.persistence.EntityManagerFactory` interface else.

See Also:

`JpaVendorAdapter.getEntityManagerFactoryInterface()`

setEntityManagerInterface

```
public void setEntityManagerInterface(Class<? extends EntityManager> emInterface)
```

Specify the (potentially vendor-specific) EntityManager interface that this factory's EntityManagers are supposed to implement.

The default will be taken from the specific JpaVendorAdapter, if any, or set to the standard `javax.persistence.EntityManager` interface else.

See Also:

`JpaVendorAdapter.getEntityManagerInterface()`, `EntityManagerFactoryInfo.getEntityManagerInterface()`

getEntityManagerInterface

```
public Class<? extends EntityManager> getEntityManagerInterface()
```

Description copied from interface: `EntityManagerFactoryInfo`

Return the (potentially vendor-specific) EntityManager interface that this factory's EntityManagers will implement.

A `null` return value suggests that autodetection is supposed to happen: either based on a target `EntityManager` instance or simply defaulting to `javax.persistence.EntityManager`.

Specified by:

`getEntityManagerInterface` in interface `EntityManagerFactoryInfo`

setJpaDialect

```
public void setJpaDialect(JpaDialect jpaDialect)
```

Specify the vendor-specific JpaDialect implementation to associate with this EntityManagerFactory. This will be exposed through the EntityManagerFactoryInfo interface, to be picked up as default dialect by accessors that intend to use JpaDialect functionality.

See Also:

`EntityManagerFactoryInfo.getJpaDialect()`

getJpaDialect

```
public JpaDialect getJpaDialect()
```

Description copied from interface: `EntityManagerFactoryInfo`

Return the vendor-specific JpaDialect implementation for this EntityManagerFactory, or `null` if not known.

Specified by:

`getJpaDialect` in interface `EntityManagerFactoryInfo`

setJpaVendorAdapter

```
public void setJpaVendorAdapter(JpaVendorAdapter jpaVendorAdapter)
```

Specify the JpaVendorAdapter implementation for the desired JPA provider, if any. This will initialize appropriate defaults for the given provider, such as persistence provider class and JpaDialect, unless locally overridden in this FactoryBean.

getJpaVendorAdapter

```
public JpaVendorAdapter getJpaVendorAdapter()
```

Return the JpaVendorAdapter implementation for this EntityManagerFactory, or `null` if not known.

setBootstrapExecutor

```
public void setBootstrapExecutor(AsyncTaskExecutor bootstrapExecutor)
```

Specify an asynchronous executor for background bootstrapping, e.g. a `SimpleAsyncTaskExecutor`.

`EntityManagerFactory` initialization will then switch into background bootstrap mode, with a `EntityManagerFactory` proxy immediately returned for injection purposes instead of waiting for the JPA provider's bootstrapping to complete. However, note that the first actual call to a `EntityManagerFactory` method will then block until the JPA provider's bootstrapping completed, if not ready by then. For maximum benefit, make sure to avoid early `EntityManagerFactory` calls in init methods of related beans, even for metadata introspection purposes.

Since:

4.3

getBootstrapExecutor

```
public AsyncTaskExecutor getBootstrapExecutor()
```

Return the asynchronous executor for background bootstrapping, if any.

Since:

4.3

setBeanClassLoader

```
public void setBeanClassLoader(ClassLoader classLoader)
```

Description copied from interface: `BeanClassLoaderAware`

Callback that supplies the bean `class loader` to a bean instance.

Invoked *after* the population of normal bean properties but *before* an initialization callback such as `InitializingBean's InitializingBean.afterPropertiesSet()` method or a custom init-method.

Specified by:

`setBeanClassLoader` in interface `BeanClassLoaderAware`

Parameters:

`classLoader - the owning class loader; may be null in which case a default ClassLoader must be used, for example the ClassLoader obtained via ClassUtils.getDefaultClassLoader()`

getBeanClassLoader

```
public ClassLoader getBeanClassLoader()
```

Description copied from interface: `EntityManagerFactoryInfo`

Return the ClassLoader that the application's beans are loaded with.

Proxies will be generated in this ClassLoader.

Specified by:

`getBeanClassLoader` in interface `EntityManagerFactoryInfo`

setBeanFactory

```
public void setBeanFactory(BeanFactory beanFactory)
```

Description copied from interface: `BeanFactoryAware`

Callback that supplies the owning factory to a bean instance.

Invoked after the population of normal bean properties but before an initialization callback such as `InitializingBean.afterPropertiesSet()` or a custom init-method.

Specified by:

`setBeanFactory` in interface `BeanFactoryAware`

Parameters:

`beanFactory - owning BeanFactory (never null). The bean can immediately call methods on the factory.`

See Also:

`BeanInitializationException`

setBeanName

```
public void setBeanName(String name)
```

Description copied from interface: `BeanNameAware`

Set the name of the bean in the bean factory that created this bean.

Invoked after population of normal bean properties but before an init callback such as `InitializingBean.afterPropertiesSet()` or a custom init-method.

Specified by:

`setBeanName` in interface `BeanNameAware`

Parameters:

`name` - the name of the bean in the factory. Note that this name is the actual bean name used in the factory, which may differ from the originally specified name: in particular for inner bean names, the actual bean name might have been made unique through appending "#..." suffixes. Use the `BeanFactoryUtils.originalBeanName(String)` method to extract the original bean name (without suffix), if desired.

afterPropertiesSet

```
public final void afterPropertiesSet()
                            throws PersistenceException
```

Description copied from interface: `InitializingBean`

Invoked by a BeanFactory after it has set all bean properties supplied (and satisfied BeanFactoryAware and ApplicationContextAware).

This method allows the bean instance to perform initialization only possible when all bean properties have been set and to throw an exception in the event of misconfiguration.

Specified by:

`afterPropertiesSet` in interface `InitializingBean`

Throws:

`PersistenceException`

createEntityManagerFactoryProxy

```
protected EntityManagerFactory createEntityManagerFactoryProxy(EntityManagerFactory emf)
```

Create a proxy of the given EntityManagerFactory. We do this to be able to return transaction-aware proxies for application-managed EntityManagers, and to introduce the NamedEntityManagerFactory interface

Parameters:

`emf` - EntityManagerFactory as returned by the persistence provider

Returns:

proxy entity manager

createNativeEntityManagerFactory

```
protected abstract EntityManagerFactory createNativeEntityManagerFactory()
                                                                  throws PersistenceException
```

Subclasses must implement this method to create the EntityManagerFactory that will be returned by the `getObject()` method.

Returns:

EntityManagerFactory instance returned by this FactoryBean

Throws:

`PersistenceException` - if the EntityManager cannot be created

translateExceptionIfPossible

```
public DataAccessException translateExceptionIfPossible(RuntimeException ex)
```

Implementation of the PersistenceExceptionTranslator interface, as autodetected by Spring's PersistenceExceptionTranslationPostProcessor.

Uses the dialect's conversion if possible; otherwise falls back to standard JPA exception conversion.

Specified by:

`translateExceptionIfPossible` in interface `PersistenceExceptionTranslator`

Parameters:

`ex` - a RuntimeException to translate

Returns:

the corresponding DataAccessException (or null if the exception could not be translated, as in this case it may result from user code rather than from an actual persistence problem)

See Also:

`PersistenceExceptionTranslationPostProcessor`, `PersistenceExceptionTranslator.translateExceptionIfPossible(java.lang.RuntimeException)`,

EntityManagerFactoryUtils.convertJpaAccessExceptionIfPossible(java.lang.RuntimeException)

getNativeEntityManagerFactory

public EntityManagerFactory getNativeEntityManagerFactory()

Description copied from interface: EntityManagerFactoryInfo
Return the raw underlying EntityManagerFactory.

Specified by:
getNativeEntityManagerFactory in interface EntityManagerFactoryInfo

Returns:
the unadorned EntityManagerFactory (never null)

getPersistenceUnitInfo

public PersistenceUnitInfo getPersistenceUnitInfo()

Description copied from interface: EntityManagerFactoryInfo
Return the PersistenceUnitInfo used to create this EntityManagerFactory, if the in-container API was used.

Specified by:
getPersistenceUnitInfo in interface EntityManagerFactoryInfo

Returns:
the PersistenceUnitInfo used to create this EntityManagerFactory, or null if the in-container contract was not used to configure the EntityManagerFactory

getDataSource

public DataSource getDataSource()

Description copied from interface: EntityManagerFactoryInfo
Return the JDBC DataSource that this EntityManagerFactory obtains its JDBC Connections from.

Specified by:
getDataSource in interface EntityManagerFactoryInfo

Returns:
the JDBC DataSource, or null if not known

getObject

public EntityManagerFactory getObject()

Return the singleton EntityManagerFactory.

Specified by:
getObject in interface FactoryBean<EntityManagerFactory>

Returns:
an instance of the bean (can be null)

See Also:
FactoryBeanNotInitializedException

getObjectType

public Class<? extends EntityManagerFactory> getObjectType()

Description copied from interface: FactoryBean
Return the type of object that this FactoryBean creates, or `null` if not known in advance.

This allows one to check for specific types of beans without instantiating objects, for example on autowiring.

In the case of implementations that are creating a singleton object, this method should try to avoid singleton creation as far as possible; it should rather estimate the type in advance. For prototypes, returning a meaningful type here is advisable too.

This method can be called *before* this FactoryBean has been fully initialized. It must not rely on state created during initialization; of course, it can still use such state if available.

NOTE: Autowiring will simply ignore FactoryBeans that return `null` here. Therefore it is highly recommended to implement this method properly, using the current state of the FactoryBean.

Specified by:
`getObjectType` in interface `FactoryBean<EntityManagerFactory>`

Returns:
the type of object that this FactoryBean creates, or null if not known at the time of the call

See Also:
`ListableBeanFactory.getBeansOfType(java.lang.Class<T>)`

isSingleton

```
public boolean isSingleton()
```

Description copied from interface: `FactoryBean`
Is the object managed by this factory a singleton? That is, will `FactoryBean.getObject()` always return the same object (a reference that can be cached)?

NOTE: If a FactoryBean indicates to hold a singleton object, the object returned from `getObject()` might get cached by the owning BeanFactory. Hence, do not return `true` unless the FactoryBean always exposes the same reference.

The singleton status of the FactoryBean itself will generally be provided by the owning BeanFactory; usually, it has to be defined as singleton there.

NOTE: This method returning `false` does not necessarily indicate that returned objects are independent instances. An implementation of the extended `SmartFactoryBean` interface may explicitly indicate independent instances through its `SmartFactoryBean.isPrototype()` method. Plain `FactoryBean` implementations which do not implement this extended interface are simply assumed to always return independent instances if the `isSingleton()` implementation returns `false`.

Specified by:
`isSingleton` in interface `FactoryBean<EntityManagerFactory>`

Returns:
whether the exposed object is a singleton

See Also:
`FactoryBean.getObject()`, `SmartFactoryBean.isPrototype()`

destroy

```
public void destroy()
```

Close the EntityManagerFactory on bean factory shutdown.

Specified by:
`destroy` in interface `DisposableBean`

writeReplace

```
protected Object writeReplace()
                     throws ObjectStreamException
```

Throws:
`ObjectStreamException`

org.springframework.orm.jpa

Class LocalContainerEntityManagerFactoryBean

java.lang.Object
  org.springframework.orm.jpa.AbstractEntityManagerFactoryBean
    org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean

All Implemented Interfaces:

Serializable, Aware, BeanClassLoaderAware, BeanFactoryAware, BeanNameAware, DisposableBean, FactoryBean<EntityManagerFactory>, InitializingBean, ResourceLoaderAware, LoadTimeWeaverAware, PersistenceExceptionTranslator, EntityManagerFactoryInfo

---

```
public class LocalContainerEntityManagerFactoryBean
extends AbstractEntityManagerFactoryBean
implements ResourceLoaderAware, LoadTimeWeaverAware
```

FactoryBean that creates a JPA EntityManagerFactory according to JPA's standard *container* bootstrap contract. This is the most powerful way to set up a shared JPA EntityManagerFactory in a Spring application context; the EntityManagerFactory can then be passed to JPA-based DAOs via dependency injection. Note that switching to a JNDI lookup or to a LocalEntityManagerFactoryBean definition is just a matter of configuration!

As with LocalEntityManagerFactoryBean, configuration settings are usually read in from a `META-INF/persistence.xml` config file, residing in the class path, according to the general JPA configuration contract. However, this FactoryBean is more flexible in that you can override the location of the `persistence.xml` file, specify the JDBC DataSources to link to, etc. Furthermore, it allows for pluggable class instrumentation through Spring's LoadTimeWeaver abstraction, instead of being tied to a special VM agent specified on JVM startup.

Internally, this FactoryBean parses the `persistence.xml` file itself and creates a corresponding PersistenceUnitInfo object (with further configuration merged in, such as JDBC DataSources and the Spring LoadTimeWeaver), to be passed to the chosen JPA PersistenceProvider. This corresponds to a local JPA container with full support for the standard JPA container contract.

The exposed EntityManagerFactory object will implement all the interfaces of the underlying native EntityManagerFactory returned by the PersistenceProvider, plus the EntityManagerFactoryInfo interface which exposes additional metadata as assembled by this FactoryBean.

NOTE: Spring's JPA support requires JPA 2.0 or higher, as of Spring 4.0. JPA 1.0 based applications are still supported; however, a JPA 2.0/2.1 compliant persistence provider is needed at runtime. Spring's persistence unit bootstrapping automatically detects JPA 2.0 vs 2.1 through checking the JPA API on the classpath.

Since:

2.0

Author:

Juergen Hoeller, Rod Johnson

See Also:

setPersistenceXmlLocation(java.lang.String), AbstractEntityManagerFactoryBean.setJpaProperties(java.util.Properties), AbstractEntityManagerFactoryBean.setJpaVendorAdapter(org.springframework.orm.jpa.JpaVendorAdapter), setLoadTimeWeaver(org.springframework.instrument.classloading.LoadTimeWeaver), setDataSource(javax.sql.DataSource), EntityManagerFactoryInfo, LocalEntityManagerFactoryBean, SharedEntityManagerBean, PersistenceProvider.createContainerEntityManagerFactory(javax.persistence.spi.PersistenceUnitInfo, java.util.Map), Serialized Form

Field Summary

Fields inherited from class org.springframework.orm.jpa.AbstractEntityManagerFactoryBean

logger

Constructor Summary

Constructors

Constructor and Description
`LocalContainerEntityManagerFactoryBean()`

Method Summary

All Methods Instance Methods Concrete Methods

Modifier and Type	Method and Description
protected `EntityManagerFactory`	`createNativeEntityManagerFactory()` Subclasses must implement this method to create the EntityManagerFactory that will be returned by the `getObject()` method.
protected `PersistenceUnitInfo`	`determinePersistenceUnitInfo(PersistenceUnitManager persistenceUnitManager)` Determine the PersistenceUnitInfo to use for the EntityManagerFactory created by this bean.
`DataSource`	`getDataSource()` Return the JDBC DataSource that this EntityManagerFactory obtains its JDBC Connections from.
`PersistenceUnitInfo`	`getPersistenceUnitInfo()` Return the PersistenceUnitInfo used to create this EntityManagerFactory, if the in-container API was used.
`String`	`getPersistenceUnitName()` Return the name of the persistence unit used to create this EntityManagerFactory, or `null` if it is an unnamed default.
protected void	`postProcessEntityManagerFactory(EntityManagerFactory emf, PersistenceUnitInfo pui)` Hook method allowing subclasses to customize the EntityManagerFactory after its creation via the PersistenceProvider.
void	`setDataSource(DataSource dataSource)` Specify the JDBC DataSource that the JPA persistence provider is supposed to use for accessing the database.
void	`setJtaDataSource(DataSource jtaDataSource)` Specify the JDBC DataSource that the JPA persistence provider is supposed to use for accessing the database.
void	`setLoadTimeWeaver(LoadTimeWeaver loadTimeWeaver)` Specify the Spring LoadTimeWeaver to use for class instrumentation according to the JPA class transformer contract.
void	`setMappingResources(String... mappingResources)` Specify one or more mapping resources (equivalent to `<mapping-file>` entries in `persistence.xml`) for the default persistence unit.
void	`setPackagesToScan(String... packagesToScan)` Set whether to use Spring-based scanning for entity classes in the classpath instead of using JPA's standard scanning of jar files with `persistence.xml` markers in them.
void	`setPersistenceUnitManager(PersistenceUnitManager persistenceUnitManager)` Set the PersistenceUnitManager to use for obtaining the JPA persistence unit that this FactoryBean is supposed to build an EntityManagerFactory for.
void	`setPersistenceUnitName(String persistenceUnitName)` Uses the specified persistence unit name as the name of the default persistence unit, if applicable.
void	`setPersistenceUnitPostProcessors(PersistenceUnitPostProcessor... postProcessors)`

	Set the PersistenceUnitPostProcessors to be applied to the PersistenceUnitInfo used for creating this EntityManagerFactory.
void	**setPersistenceUnitRootLocation**(**String** defaultPersistenceUnitRootLocation) Set a persistence unit root location for the default persistence unit.
void	**setPersistenceXmlLocation**(**String** persistenceXmlLocation) Set the location of the `persistence.xml` file we want to use.
void	**setResourceLoader**(**ResourceLoader** resourceLoader) Set the ResourceLoader that this object runs in.
void	**setSharedCacheMode**(**SharedCacheMode** sharedCacheMode) Specify the JPA 2.0 shared cache mode for this persistence unit, overriding a value in `persistence.xml` if set.
void	**setValidationMode**(**ValidationMode** validationMode) Specify the JPA 2.0 validation mode for this persistence unit, overriding a value in `persistence.xml` if set.

Methods inherited from class org.springframework.orm.jpa.AbstractEntityManagerFactoryBean

afterPropertiesSet, createEntityManagerFactoryProxy, destroy, getBeanClassLoader, getBootstrapExecutor, getEntityManagerInterface, getJpaDialect, getJpaPropertyMap, getJpaVendorAdapter, getNativeEntityManagerFactory, getObject, getObjectType, getPersistenceProvider, isSingleton, setBeanClassLoader, setBeanFactory, setBeanName, setBootstrapExecutor, setEntityManagerFactoryInterface, setEntityManagerInterface, setJpaDialect, setJpaProperties, setJpaPropertyMap, setJpaVendorAdapter, setPersistenceProvider, setPersistenceProviderClass, translateExceptionIfPossible, writeReplace

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

LocalContainerEntityManagerFactoryBean

```
public LocalContainerEntityManagerFactoryBean()
```

Method Detail

setPersistenceUnitManager

```
public void setPersistenceUnitManager(PersistenceUnitManager persistenceUnitManager)
```

Set the PersistenceUnitManager to use for obtaining the JPA persistence unit that this FactoryBean is supposed to build an EntityManagerFactory for.

The default is to rely on the local settings specified on this FactoryBean, such as "persistenceXmlLocation", "dataSource" and "loadTimeWeaver".

For reuse of existing persistence unit configuration or more advanced forms of custom persistence unit handling, consider defining a separate PersistenceUnitManager bean (typically a DefaultPersistenceUnitManager instance) and linking it in here. `persistence.xml` location, DataSource configuration and LoadTimeWeaver will be defined on that separate DefaultPersistenceUnitManager bean in such a scenario.

See Also:

setPersistenceXmlLocation(java.lang.String), setDataSource(javax.sql.DataSource), setLoadTimeWeaver(org.springframework.instrument.classloading.LoadTimeWeaver), DefaultPersistenceUnitManager

setPersistenceXmlLocation

```
public void setPersistenceXmlLocation(String persistenceXmlLocation)
```

Set the location of the `persistence.xml` file we want to use. This is a Spring resource location.

Default is "classpath:META-INF/persistence.xml".

NOTE: Only applied if no external PersistenceUnitManager specified.

Parameters:

`persistenceXmlLocation - a Spring resource String identifying the location of the persistence.xml file that this LocalContainerEntityManagerFactoryBean should parse`

See Also:

`setPersistenceUnitManager(org.springframework.orm.jpa.persistenceunit.PersistenceUnitManager)`

setPersistenceUnitName

```
public void setPersistenceUnitName(String persistenceUnitName)
```

Uses the specified persistence unit name as the name of the default persistence unit, if applicable.

NOTE: Only applied if no external PersistenceUnitManager specified.

Overrides:

`setPersistenceUnitName in class AbstractEntityManagerFactoryBean`

See Also:

`DefaultPersistenceUnitManager.setDefaultPersistenceUnitName(java.lang.String)`

setPersistenceUnitRootLocation

```
public void setPersistenceUnitRootLocation(String defaultPersistenceUnitRootLocation)
```

Set a persistence unit root location for the default persistence unit.

Default is "classpath:", that is, the root of the current classpath (nearest root directory). To be overridden if unit-specific resolution does not work and the classpath root is not appropriate either.

NOTE: Only applied if no external PersistenceUnitManager specified.

Since:

`4.3.3`

See Also:

`DefaultPersistenceUnitManager.setDefaultPersistenceUnitRootLocation(java.lang.String)`

setPackagesToScan

```
public void setPackagesToScan(String... packagesToScan)
```

Set whether to use Spring-based scanning for entity classes in the classpath instead of using JPA's standard scanning of jar files with `persistence.xml` markers in them. In case of Spring-based scanning, no `persistence.xml` is necessary; all you need to do is to specify base packages to search here.

Default is none. Specify packages to search for autodetection of your entity classes in the classpath. This is analogous to Spring's component-scan feature (`ClassPathBeanDefinitionScanner`).

Note: There may be limitations in comparison to regular JPA scanning. In particular, JPA providers may pick up annotated packages for provider-specific annotations only when driven by `persistence.xml`. As of 4.1, Spring's scan can detect annotated packages as well if supported by the given `JpaVendorAdapter` (e.g. for Hibernate).

If no explicit `mapping resources` have been specified in addition to these packages, Spring's setup looks for a default `META-INF/orm.xml` file in the classpath, registering it as a mapping resource for the default unit if the mapping file is not co-located with a `persistence.xml` file (in which case we assume it is only meant to be used with the persistence units defined there, like in standard JPA).

NOTE: Only applied if no external PersistenceUnitManager specified.

Parameters:

packagesToScan - one or more base packages to search, analogous to Spring's component-scan configuration for regular Spring components

See Also:

setPersistenceUnitManager(org.springframework.orm.jpa.persistenceunit.PersistenceUnitManager), DefaultPersistenceUnitManager.setPackagesToScan(java.lang.String...)

setMappingResources

```
public void setMappingResources(String... mappingResources)
```

Specify one or more mapping resources (equivalent to <mapping-file> entries in persistence.xml) for the default persistence unit. Can be used on its own or in combination with entity scanning in the classpath, in both cases avoiding persistence.xml.

Note that mapping resources must be relative to the classpath root, e.g. "META-INF/mappings.xml" or "com/mycompany/repository/mappings.xml", so that they can be loaded through ClassLoader.getResource.

If no explicit mapping resources have been specified next to packages to scan, Spring's setup looks for a default META-INF/orm.xml file in the classpath, registering it as a mapping resource for the default unit if the mapping file is not co-located with a persistence.xml file (in which case we assume it is only meant to be used with the persistence units defined there, like in standard JPA).

Note that specifying an empty array/list here suppresses the default META-INF/orm.xml check. On the other hand, explicitly specifying META-INF/orm.xml here will register that file even if it happens to be co-located with a persistence.xml file.

NOTE: Only applied if no external PersistenceUnitManager specified.

See Also:

setPersistenceUnitManager(org.springframework.orm.jpa.persistenceunit.PersistenceUnitManager), DefaultPersistenceUnitManager.setMappingResources(java.lang.String...)

setSharedCacheMode

```
public void setSharedCacheMode(SharedCacheMode sharedCacheMode)
```

Specify the JPA 2.0 shared cache mode for this persistence unit, overriding a value in persistence.xml if set.

NOTE: Only applied if no external PersistenceUnitManager specified.

Since:

4.0

See Also:

PersistenceUnitInfo.getSharedCacheMode(), setPersistenceUnitManager(org.springframework.orm.jpa.persistenceunit.PersistenceUnitManager)

setValidationMode

```
public void setValidationMode(ValidationMode validationMode)
```

Specify the JPA 2.0 validation mode for this persistence unit, overriding a value in persistence.xml if set.

NOTE: Only applied if no external PersistenceUnitManager specified.

Since:

4.0

See Also:

PersistenceUnitInfo.getValidationMode(), setPersistenceUnitManager(org.springframework.orm.jpa.persistenceunit.PersistenceUnitManager)

setDataSource

```
public void setDataSource(DataSource dataSource)
```

Specify the JDBC DataSource that the JPA persistence provider is supposed to use for accessing the database. This is an alternative to keeping the JDBC configuration in `persistence.xml`, passing in a Spring-managed DataSource instead.

In JPA speak, a DataSource passed in here will be used as "nonJtaDataSource" on the PersistenceUnitInfo passed to the PersistenceProvider, as well as overriding data source configuration in `persistence.xml` (if any). Note that this variant typically works for JTA transaction management as well; if it does not, consider using the explicit `setJtaDataSource(javax.sql.DataSource)` instead.

NOTE: Only applied if no external PersistenceUnitManager specified.

See Also:

`PersistenceUnitInfo.getNonJtaDataSource()`, `setPersistenceUnitManager(org.springframework.orm.jpa.persistenceunit.PersistenceUnitManager)`

setJtaDataSource

```
public void setJtaDataSource(DataSource jtaDataSource)
```

Specify the JDBC DataSource that the JPA persistence provider is supposed to use for accessing the database. This is an alternative to keeping the JDBC configuration in `persistence.xml`, passing in a Spring-managed DataSource instead.

In JPA speak, a DataSource passed in here will be used as "jtaDataSource" on the PersistenceUnitInfo passed to the PersistenceProvider, as well as overriding data source configuration in `persistence.xml` (if any).

NOTE: Only applied if no external PersistenceUnitManager specified.

See Also:

`PersistenceUnitInfo.getJtaDataSource()`, `setPersistenceUnitManager(org.springframework.orm.jpa.persistenceunit.PersistenceUnitManager)`

setPersistenceUnitPostProcessors

```
public void setPersistenceUnitPostProcessors(PersistenceUnitPostProcessor... postProcessors)
```

Set the PersistenceUnitPostProcessors to be applied to the PersistenceUnitInfo used for creating this EntityManagerFactory.

Such post-processors can, for example, register further entity classes and jar files, in addition to the metadata read from `persistence.xml`.

NOTE: Only applied if no external PersistenceUnitManager specified.

See Also:

`setPersistenceUnitManager(org.springframework.orm.jpa.persistenceunit.PersistenceUnitManager)`

setLoadTimeWeaver

```
public void setLoadTimeWeaver(LoadTimeWeaver loadTimeWeaver)
```

Specify the Spring LoadTimeWeaver to use for class instrumentation according to the JPA class transformer contract.

It is a not required to specify a LoadTimeWeaver: Most providers will be able to provide a subset of their functionality without class instrumentation as well, or operate with their VM agent specified on JVM startup.

In terms of Spring-provided weaving options, the most important ones are InstrumentationLoadTimeWeaver, which requires a Spring-specific (but very general) VM agent specified on JVM startup, and ReflectiveLoadTimeWeaver, which interacts with an underlying ClassLoader based on specific extended methods being available on it (for example, interacting with Spring's TomcatInstrumentableClassLoader).

NOTE: As of Spring 2.5, the context's default LoadTimeWeaver (defined as bean with name "loadTimeWeaver") will be picked up automatically, if available, removing the need for LoadTimeWeaver configuration on each affected target bean. Consider using the `context:load-time-weaver` XML tag for creating such a shared LoadTimeWeaver (autodetecting the environment by default).

NOTE: Only applied if no external PersistenceUnitManager specified. Otherwise, the external `PersistenceUnitManager` is responsible for the weaving configuration.

Specified by:

`setLoadTimeWeaver` in interface `LoadTimeWeaverAware`

Parameters:

loadTimeWeaver - the LoadTimeWeaver instance (never null)

See Also:

InstrumentationLoadTimeWeaver, ReflectiveLoadTimeWeaver, TomcatInstrumentableClassLoader

setResourceLoader

```
public void setResourceLoader(ResourceLoader resourceLoader)
```

Description copied from interface: ResourceLoaderAware

Set the ResourceLoader that this object runs in.

This might be a ResourcePatternResolver, which can be checked through `instanceof ResourcePatternResolver`. See also the `ResourcePatternUtils.getResourcePatternResolver` method.

Invoked after population of normal bean properties but before an init callback like InitializingBean's `afterPropertiesSet` or a custom init-method. Invoked before ApplicationContextAware's `setApplicationContext`.

Specified by:

setResourceLoader in interface ResourceLoaderAware

Parameters:

resourceLoader - ResourceLoader object to be used by this object

See Also:

ResourcePatternResolver,
ResourcePatternUtils.getResourcePatternResolver(org.springframework.core.io.ResourceLoader)

createNativeEntityManagerFactory

```
protected EntityManagerFactory createNativeEntityManagerFactory()
                                                    throws PersistenceException
```

Description copied from class: AbstractEntityManagerFactoryBean

Subclasses must implement this method to create the EntityManagerFactory that will be returned by the `getObject()` method.

Specified by:

createNativeEntityManagerFactory in class AbstractEntityManagerFactoryBean

Returns:

EntityManagerFactory instance returned by this FactoryBean

Throws:

PersistenceException - if the EntityManager cannot be created

determinePersistenceUnitInfo

```
protected PersistenceUnitInfo determinePersistenceUnitInfo(PersistenceUnitManager persistenceUnitManager)
```

Determine the PersistenceUnitInfo to use for the EntityManagerFactory created by this bean.

The default implementation reads in all persistence unit infos from `persistence.xml`, as defined in the JPA specification. If no entity manager name was specified, it takes the first info in the array as returned by the reader. Otherwise, it checks for a matching name.

Parameters:

persistenceUnitManager - the PersistenceUnitManager to obtain from

Returns:

the chosen PersistenceUnitInfo

postProcessEntityManagerFactory

```
protected void postProcessEntityManagerFactory(EntityManagerFactory emf,
                                               PersistenceUnitInfo pui)
```

Hook method allowing subclasses to customize the EntityManagerFactory after its creation via the PersistenceProvider.

The default implementation is empty.

Parameters:

`emf` - the newly created EntityManagerFactory we are working with

`pui` - the PersistenceUnitInfo used to configure the EntityManagerFactory

See Also:

`PersistenceProvider.createContainerEntityManagerFactory(javax.persistence.spi.PersistenceUnitInfo, java.util.Map)`

getPersistenceUnitInfo

```
public PersistenceUnitInfo getPersistenceUnitInfo()
```

Description copied from interface: `EntityManagerFactoryInfo`
Return the PersistenceUnitInfo used to create this EntityManagerFactory, if the in-container API was used.

Specified by:

`getPersistenceUnitInfo` in interface `EntityManagerFactoryInfo`

Overrides:

`getPersistenceUnitInfo` in class `AbstractEntityManagerFactoryBean`

Returns:

the PersistenceUnitInfo used to create this EntityManagerFactory, or null if the in-container contract was not used to configure the EntityManagerFactory

getPersistenceUnitName

```
public String getPersistenceUnitName()
```

Description copied from interface: `EntityManagerFactoryInfo`
Return the name of the persistence unit used to create this EntityManagerFactory, or `null` if it is an unnamed default.

If `getPersistenceUnitInfo()` returns non-null, the result of `getPersistenceUnitName()` must be equal to the value returned by `PersistenceUnitInfo.getPersistenceUnitName()`.

Specified by:

`getPersistenceUnitName` in interface `EntityManagerFactoryInfo`

Overrides:

`getPersistenceUnitName` in class `AbstractEntityManagerFactoryBean`

See Also:

`EntityManagerFactoryInfo.getPersistenceUnitInfo()`, `PersistenceUnitInfo.getPersistenceUnitName()`

getDataSource

```
public DataSource getDataSource()
```

Description copied from interface: `EntityManagerFactoryInfo`
Return the JDBC DataSource that this EntityManagerFactory obtains its JDBC Connections from.

Specified by:

`getDataSource` in interface `EntityManagerFactoryInfo`

Overrides:

`getDataSource` in class `AbstractEntityManagerFactoryBean`

Returns:

the JDBC DataSource, or null if not known

org.springframework.orm.jpa

Class LocalEntityManagerFactoryBean

java.lang.Object
  org.springframework.orm.jpa.AbstractEntityManagerFactoryBean
    org.springframework.orm.jpa.LocalEntityManagerFactoryBean

All Implemented Interfaces:

Serializable, Aware, BeanClassLoaderAware, BeanFactoryAware, BeanNameAware, DisposableBean, FactoryBean<EntityManagerFactory>, InitializingBean, PersistenceExceptionTranslator, EntityManagerFactoryInfo

---

```
public class LocalEntityManagerFactoryBean
extends AbstractEntityManagerFactoryBean
```

FactoryBean that creates a JPA EntityManagerFactory according to JPA's standard *standalone* bootstrap contract. This is the simplest way to set up a shared JPA EntityManagerFactory in a Spring application context; the EntityManagerFactory can then be passed to JPA-based DAOs via dependency injection. Note that switching to a JNDI lookup or to a LocalContainerEntityManagerFactoryBean definition is just a matter of configuration!

Configuration settings are usually read from a `META-INF/persistence.xml` config file, residing in the class path, according to the JPA standalone bootstrap contract. Additionally, most JPA providers will require a special VM agent (specified on JVM startup) that allows them to instrument application classes. See the Java Persistence API specification and your provider documentation for setup details.

This EntityManagerFactory bootstrap is appropriate for standalone applications which solely use JPA for data access. If you want to set up your persistence provider for an external DataSource and/or for global transactions which span multiple resources, you will need to either deploy it into a full Java EE application server and access the deployed EntityManagerFactory via JNDI, or use Spring's LocalContainerEntityManagerFactoryBean with appropriate configuration for local setup according to JPA's container contract.

Note: This FactoryBean has limited configuration power in terms of what configuration it is able to pass to the JPA provider. If you need more flexible configuration, for example passing a Spring-managed JDBC DataSource to the JPA provider, consider using Spring's more powerful LocalContainerEntityManagerFactoryBean instead.

NOTE: Spring's JPA support requires JPA 2.0 or higher, as of Spring 4.0. JPA 1.0 based applications are still supported; however, a JPA 2.0/2.1 compliant persistence provider is needed at runtime.

Since:

2.0

Author:

Juergen Hoeller, Rod Johnson

See Also:

AbstractEntityManagerFactoryBean.setJpaProperties(java.util.Properties), AbstractEntityManagerFactoryBean.setJpaVendorAdapter(org.springframework.orm.jpa.JpaVendorAdapter), JpaTransactionManager.setEntityManagerFactory(javax.persistence.EntityManagerFactory), LocalContainerEntityManagerFactoryBean, JndiObjectFactoryBean, SharedEntityManagerBean, Persistence.createEntityManagerFactory(java.lang.String), PersistenceProvider.createEntityManagerFactory(java.lang.String, java.util.Map), Serialized Form

Field Summary

Fields inherited from class org.springframework.orm.jpa.AbstractEntityManagerFactoryBean

logger

Constructor Summary

Constructors

Constructor and Description
LocalEntityManagerFactoryBean()

Method Summary

All Methods | Instance Methods | Concrete Methods

Modifier and Type	Method and Description
protected EntityManagerFactory	createNativeEntityManagerFactory() Initialize the EntityManagerFactory for the given configuration.

Methods inherited from class org.springframework.orm.jpa.AbstractEntityManagerFactoryBean

afterPropertiesSet, createEntityManagerFactoryProxy, destroy, getBeanClassLoader, getBootstrapExecutor, getDataSource, getEntityManagerInterface, getJpaDialect, getJpaPropertyMap, getJpaVendorAdapter, getNativeEntityManagerFactory, getObject, getObjectType, getPersistenceProvider, getPersistenceUnitInfo, getPersistenceUnitName, isSingleton, setBeanClassLoader, setBeanFactory, setBeanName, setBootstrapExecutor, setEntityManagerFactoryInterface, setEntityManagerInterface, setJpaDialect, setJpaProperties, setJpaPropertyMap, setJpaVendorAdapter, setPersistenceProvider, setPersistenceProviderClass, setPersistenceUnitName, translateExceptionIfPossible, writeReplace

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

LocalEntityManagerFactoryBean

```
public LocalEntityManagerFactoryBean()
```

Method Detail

createNativeEntityManagerFactory

```
protected EntityManagerFactory createNativeEntityManagerFactory()
                                                    throws PersistenceException
```

Initialize the EntityManagerFactory for the given configuration.

Specified by:

`createNativeEntityManagerFactory` in class `AbstractEntityManagerFactoryBean`

Returns:

EntityManagerFactory instance returned by this FactoryBean

Throws:

`PersistenceException` - in case of JPA initialization errors