無料電子ブック

学習

hibernate

Free unaffiliated eBook created from
Stack Overflow contributors.

#hibernate

1: をする

`SessionFactory` Beanは、 `TransactionManager`がをするすべてのデータベースセッションの、、、およびフラッシュをいます。だからたちは`SessionFactory`をDAOにautowireし、それをしてすべてのクエリをします。

しいHibernateユーザーがねるのの1つは、「いつのがコミットされるのですか」です。そのえは、 `TransactionManager`が`SesisonFactory`とどのようにするかをえるときにがあります。`@Transactional`でをけられたサービスメソッドをすると、データベースのはフラッシュされ、コミットされ`@Transactional`。このは、トランザクションは、れていないのの「」をすことになっているということです。ユニットにがしたは、ユニットがしたとみなされ、すべてのがロールバックされます。したがって、 `SessionFactory`は、々びされたサービスメソッドをするとセッションをフラッシュしてクリアします。

それは、トランザクションがにセッションをフラッシュしてクリアしないとっているわけではありません。たとえば、5つのオブジェクトのコレクションをし、データベースのオブジェクトのをすサービスメソッドをびすと、 `SessionFactory`はクエリ `SELECT COUNT(*)` がされたをにすることをします。カウントクエリをするに5つのオブジェクトのをフラッシュします。はのようになります。

バージョン

バージョン	ドキュメントリンク	
4.2.0	http://hibernate.org/orm/documentation/4.2/	2013-03-01
4.3.0	http://hibernate.org/orm/documentation/4.3/	2013-12-01
5.0.0	http://hibernate.org/orm/documentation/5.0/	2015-09-01

Examples

XMLをしたHibernateのセットアップ

クラスパスのどこかに`database-servlet.xml`というファイルをします。

のファイルはのようになります

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:jdbc="http://www.springframework.org/schema/jdbc"
xmlns:tx="http://www.springframework.org/schema/tx"
xsi:schemaLocation="http://www.springframework.org/schema/beans
```

```
http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
    http://www.springframework.org/schema/jdbc
http://www.springframework.org/schema/jdbc/spring-jdbc-3.2.xsd
    http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-
tx-3.2.xsd">

</beans>
```

txとjdbc Springがインポートされていることがtxます。これは、このファイルでかなりにするためです。

まず、アノテーションベースのトランザクション @Transactionalトランザクションをにし @Transactional。 々がSpringでHibernateをするなは、Springがすべてのトランザクションをするためです。ファイルにのをします。

```
<tx:annotation-driven />
```

データソースをするがあります。データソースは、に、Hibernateがオブジェクトをするためにするデータベースです。に、1つのトランザクション・マネージャには1つのデータ・ソースがあります。 Hibernateがのデータソースとするようにするには、のトランザクションマネージャがです。

```
<bean id="dataSource"
    class="org.springframework.jdbc.datasource.DriverManagerDataSource">
    <property name="driverClassName" value="" />
    <property name="url" value="" />
    <property name="username" value="" />
    <property name="password" value="" />
</bean>
```

このBeanのクラスは、 javax.sql.DataSourceをするものであればでもかまいません。このサンプルクラスはSpringによってされていますが、のスレッドプールをちません。なは、Apache Commons org.apache.commons.dbcp.BasicDataSource 、にもあります。のプロパティについてします。

- *driverClassName* JDBCドライバのパス。これはクラスパスでできるデータベースの JARです。のバージョンをしていることをしてください。 Oracleデータベースをしているは、OracleDriverがです。 MySQLデータベースをおちのは、MySQLDriverがです。あなたがここになドライバをつけることができるかどうかをしますが、クイックグーグルはしいドライバをするがあります。

- *url* データベースへのURL。、これはjdbc\:oracle\:thin\:\path\to\your\databaseまたは jdbc:mysql://path/to/your/database。 あなたがしているデータベースのデフォルトのについてgoogleでした、これがであるべきかをることができます。 org.hibernate.HibernateException: Connection cannot be null when 'hibernate.dialect' not setというメッセージでHibernateExceptionしたorg.hibernate.HibernateException: Connection cannot be null when 'hibernate.dialect' not set 、このガイドにっている org.hibernate.HibernateException: Connection cannot be null when 'hibernate.dialect' not

set 。 あなたのURLがっているが5、データベースがしておらず、ユーザー/パスワードがっているが5です。

- *username* データベースでするときにするユーザー。
- パスワード データベースでするときにするパスワード。

に、 SessionFactoryをします。これは、トランザクションのとにHibernateがするもので、にはデータベースとします。にするいくつかのオプションがあります。

```
<bean id="sessionFactory"
    class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">
    <property name="dataSource" ref="dataSource" />
    <property name="packagesToScan" value="au.com.project />
    <property name="hibernateProperties">
        <props>
            <prop key="hibernate.use_sql_comments">true</prop>
            <prop key="hibernate.hbm2ddl.auto">validate</prop>
        </props>
    </property>
</bean>
```

- *dataSource* データソースBean。 dataSourceのIDをしたは、ここでします。
- *packagesToScan* JPAアノテートされたオブジェクトをつけるためにスキャンするパッケージ。これらは、セッションファクトリがするがあるオブジェクトであり、にはPOJOであり、 @Entityでがけられ@Entity。 Hibernateでオブジェクトリレーションシップをするのについては、こちらをしてください 。
- *annotatedClasses* せずHibernateがじパッケージにすべてまれていない、スキャンするクラスのリストをすることもできます。 packagesToScanまたはannotatedClassesいずれかをするがありpackagesToScanが、をすることはできません。はのようになります。

```
<property name="annotatedClasses">
    <list>
        <value>foo.bar.package.model.Person</value>
        <value>foo.bar.package.model.Thing</value>
    </list>
</property>
```

- *hibernateProperties* ここには、これらのすべてがをってかれています 。あなたがするなものはのりです
- *hibernate.hbm2ddl.auto* もホットなHibernateのの1つは、このプロパティのです。 はこちらをください 。はにvalidateをい、SQLスクリプトin-memoryをしてデータベースをセットアップするか、あらかじめデータベースのデータベースをします。
- *hibernate.show_sql* ブールフラグtrueのHibernateは、したすべてのSQLをstdoutます。 log4j.logger.org.hibernate.type=TRACE log4j.logger.org.hibernate.SQL=DEBUGをログマネージャにすることによって、クエリにバインドされているをするようにロガーをすることもできますlog4jをします。

- *hibernate.format_sql* ブールフラグ。HibernateはあなたのSQLをstdoutにきれいにします。
- *hibernate.dialect* なでされていませんいチュートリアルのくは、データベースとのにするHibernateのをするをしています。 Hibernate は 、しているJDBCドライバについて、するをできます。3つのなるOracleのと5つのなるMySQLのがあるので、はこのをHibernateにせます。 Hibernateがサポートしているのなリストについては、 こちらをしてください 。

するがあるの2つのBeanはのとおりです。

```
<bean class="org.springframework.dao.annotation.PersistenceExceptionTranslationPostProcessor"
    id="PersistenceExceptionTranslator" />

<bean id="transactionManager"
    class="org.springframework.orm.hibernate4.HibernateTransactionManager">
    <property name="sessionFactory" ref="sessionFactory" />
</bean>
```

`PersistenceExceptionTranslator`は、データベースの`HibernateException`または`SQLExceptions`を、アプリケーションコンテキストができるSpringにします。

`TransactionManager` Beanはトランザクションとロールバックをするものです。

`SessionFactory` BeanをDAOにオートワイヤリングするがあります。

XMLレスのHibernate

このはここからられています

```
package com.reborne.SmartHibernateConnector.utils;

import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class LiveHibernateConnector implements IHibernateConnector {

    private String DB_DRIVER_NAME = "";
    private String DB_URL = "jdbc:h2:~/liveDB;MV_STORE=FALSE;MVCC=FALSE";
    private String DB_USERNAME = "sa";
    private String DB_PASSWORD = "";
    private String DIALECT = "org.hibernate.dialect.H2Dialect";
    private String HBM2DLL = "create";
    private String SHOW_SQL = "true";

    private static Configuration config;
    private static SessionFactory sessionFactory;
    private Session session;

    private boolean CLOSE_AFTER_TRANSACTION = false;

    public LiveHibernateConnector() {

        config = new Configuration();
```

```
        config.setProperty("hibernate.connector.driver_class",         DB_DRIVER_NAME);
        config.setProperty("hibernate.connection.url",                 DB_URL);
        config.setProperty("hibernate.connection.username",         DB_USERNAME);
        config.setProperty("hibernate.connection.password",         DB_PASSWORD);
        config.setProperty("hibernate.dialect",                     DIALECT);
        config.setProperty("hibernate.hbm2dll.auto",                 HBM2DLL);
        config.setProperty("hibernate.show_sql",                    SHOW_SQL);

        /*
         * Config connection pools
         */

        config.setProperty("connection.provider_class",
"org.hibernate.connection.C3P0ConnectionProvider");
        config.setProperty("hibernate.c3p0.min_size", "5");
        config.setProperty("hibernate.c3p0.max_size", "20");
        config.setProperty("hibernate.c3p0.timeout", "300");
        config.setProperty("hibernate.c3p0.max_statements", "50");
        config.setProperty("hibernate.c3p0.idle_test_period", "3000");


        /**
         * Resource mapping
         */

//        config.addAnnotatedClass(User.class);
//        config.addAnnotatedClass(User.class);
//        config.addAnnotatedClass(User.class);

        sessionFactory = config.buildSessionFactory();
    }


    public HibWrapper openSession() throws HibernateException {
        return new HibWrapper(getOrCreateSession(), CLOSE_AFTER_TRANSACTION);
    }


    public Session getOrCreateSession() throws HibernateException {
        if (session == null) {
            session = sessionFactory.openSession();
        }
        return session;
    }

    public void reconnect() throws HibernateException {
        this.sessionFactory = config.buildSessionFactory();
    }


}
```

のHibernateでは、このはうまくいきませんHibernate 5.2リリースではこのがです

XMLをしたなの

にXMLをしてなのプロジェクトをセットアップするには、3つのファイル、hibernate.cfg.xml、エンティティのPOJO、およびエンティティのEntityName.hbm.xmlがです。にMySQLをしたをしま

す。

hibernate.cfg.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
   <session-factory>
   <property name="hibernate.dialect">
      org.hibernate.dialect.MySQLDialect
   </property>
   <property name="hibernate.connection.driver_class">
      com.mysql.jdbc.Driver
   </property>

   <property name="hibernate.connection.url">
      jdbc:mysql://localhost/DBSchemaName
   </property>
   <property name="hibernate.connection.username">
      testUserName
   </property>
   <property name="hibernate.connection.password">
      testPassword
   </property>

   <!-- List of XML mapping files -->
   <mapping resource="HibernatePractice/Employee.hbm.xml"/>

</session-factory>
</hibernate-configuration>
```

DBSchemaName、testUserName、およびtestPasswordはすべてきえられます。パッケージにまれているは、ずフルリソースをしてください。

Employee.java

```
package HibernatePractice;

public class Employee {
    private int id;
    private String firstName;
    private String middleName;
    private String lastName;

    public Employee(){

    }
    public int getId(){
        return id;
    }
    public void setId(int id){
        this.id = id;
    }
    public String getFirstName(){
        return firstName;
```

```
    }
    public void setFirstName(String firstName){
        this.firstName = firstName;
    }
    public String getMiddleName(){
        return middleName;
    }
    public void setMiddleName(String middleName){
        this.middleName = middleName;
    }
    public String getLastName(){
        return lastName;
    }
    public void setLastName(String lastName){
        this.lastName = lastName;
    }
}
```

Employee.hbm.xml

```
<hibernate-mapping>
   <class name="HibernatePractice.Employee" table="employee">
      <meta attribute="class-description">
         This class contains employee information.
      </meta>
      <id name="id" type="int" column="empolyee_id">
         <generator class="native"/>
      </id>
      <property name="firstName" column="first_name" type="string"/>
      <property name="middleName" column="middle_name" type="string"/>
      <property name="lastName" column="last_name" type="string"/>
   </class>
</hibernate-mapping>
```

クラスがパッケージにあるは、なクラスpackageName.classNameもします。

これら3つのファイルをしたら、プロジェクトでhibernateをするがいました。

オンラインでをするをむ https://riptutorial.com/ja/hibernate/topic/907/をする

2: HibernateとJPA

Examples

HibernateとJPAとの

HibernateはJPAのです。そのため、すべてがHibernateにもてはまるとわれました。

HibernateはJPAをいくつかしています。また、JPAプロバイダをするはプロバイダです。このドキュメントのセクションには、Hibernateにのものだけがまれていなければなりません。

オンラインでHibernateとJPAをむ https://riptutorial.com/ja/hibernate/topic/6313/hibernateとjpa

3: HQL

き

HQLはHibernate Query Languageであり、SQLにづいており、そのではSQLにされますが、はなります。テーブルではなくエンティティ/クラスをし、ではなくフィールドをします。また、くのがです。

hqlをするときにえておくべきなは、SQLでされているテーブルとカラムのわりに、クラスとフィールドをすることです。

Examples

テーブルをする

```
hql = "From EntityName";
```

のをする

```
hql = "Select id, name From Employee";
```

Whereをめる

```
hql = "From Employee where id = 22";
```

する

```
hql = "From Author a, Book b Where a.id = book.author";
```

オンラインでHQLをむ https://riptutorial.com/ja/hibernate/topic/9388/hql

4: SQLログをIにする

これらのクエリのロギングはく 、はHibernateよりもいです。また、のログをします。パフォーマンスがなシナリオでは、ログをしないでください。これは、Hibernateがにするクエリをテストするにのみしてください。

Examples

ロギングファイルの

ロギングファイルでは、のパッケージのロギングをのレベルにします。

```
# log the sql statement
org.hibernate.SQL=DEBUG
# log the parameters
org.hibernate.type=TRACE
```

おそらくいくつかのロガーのがです。

Log4j

```
log4j.logger.org.hibernate.SQL=DEBUG
log4j.logger.org.hibernate.type=TRACE
```

Springブート`application.properties`

```
logging.level.org.hibernate.SQL=DEBUG
logging.level.org.hibernate.type=TRACE
```

`logback.xml`

```
<logger name="org.hibernate.SQL" level="DEBUG"/>
<logger name="org.hibernate.type" level="TRACE"/>
```

Hibernateプロパティの

これによりされたSQLはされますが、クエリにまれるはされません。

```
<bean id="sessionFactory"
    class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">
    <property name="hibernateProperties">
        <props>
            <!-- show the sql without the parameters -->
            <prop key="hibernate.show_sql">true</prop>
            <!-- format the sql nice -->
            <prop key="hibernate.format_sql">true</prop>
            <!-- show the hql as comment -->
```

```
            <prop key="use_sql_comments">true</prop>
        </props>
    </property>
</bean>
```

デバッグでSQLログを/にする

Hibernateをするのアプリケーションは、アプリケーションのになのSQLをします。によっては、デバッグにのポイントでSQLログを/にするがよいもあります。

にするには、アプリケーションをデバッグするときにIDEでこのコードをします。

```
org.apache.log4j.Logger.getLogger("org.hibernate.SQL")
    .setLevel(org.apache.log4j.Level.DEBUG)
```

にするには

```
org.apache.log4j.Logger.getLogger("org.hibernate.SQL")
    .setLevel(org.apache.log4j.Level.OFF)
```

オンラインでSQLログを/にするをむ https://riptutorial.com/ja/hibernate/topic/3548/sqlログを-にする

5: アノテーションをしたHibernateエンティティリレーションシップ

パラメーター

`@OneToOne`	するオブジェクトとの11のをします。
`@OneToMany`	くのオブジェクトにマップするのオブジェクトをします。
`@ManyToOne`	のオブジェクトにマップされるオブジェクトのコレクションをします。
`@Entity`	データベーステーブルにマップするオブジェクトをします。
`@Table`	このオブジェクトがマップするデータベーステーブルもします。
`@JoinColumn`	foreginキーがされているをします。
`@JoinTable`	キーをするテーブルをします。

Examples

ユーザーのジョイン·テーブル·オブジェクトをして、の

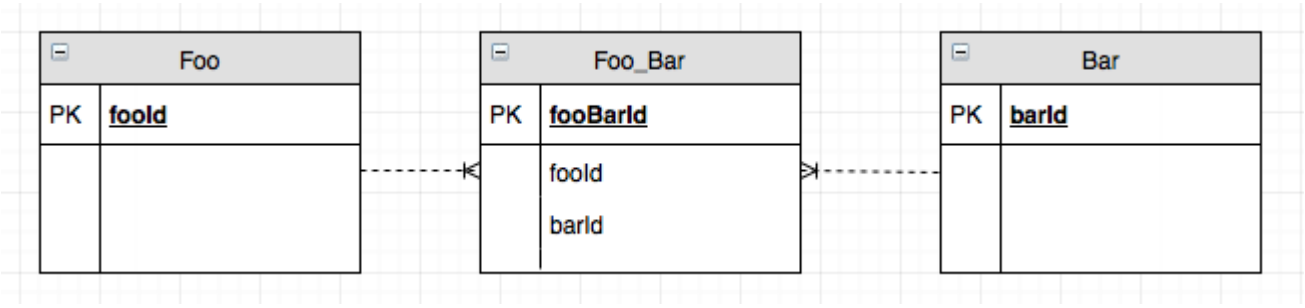
```
@Entity
@Table(name="FOO")
public class Foo {
    private UUID fooId;

    @OneToMany(mappedBy = "bar")
    private List<FooBar> bars;
}

@Entity
@Table(name="BAR")
public class Bar {
    private UUID barId;

    @OneToMany(mappedBy = "foo")
    private List<FooBar> foos;
}

@Entity
@Table(name="FOO_BAR")
public class FooBar {
    private UUID fooBarId;

    @ManyToOne
```

```
    @JoinColumn(name = "fooId")
    private Foo foo;

    @ManyToOne
    @JoinColumn(name = "barId")
    private Bar bar;

    //You can store other objects/fields on this table here.
}
```

ユーザーがするテーブルをして、くのFooオブジェクトとのBarオブジェクトとののをします。

Fooオブジェクトは FOOとばれるテーブルにとしてされFOO。 Barオブジェクトは、 BARというテーブルにとしてされます。 Fooオブジェクトと Barオブジェクトのは、 FOO_BARというテーブルにされます。アプリケーションのとして FooBarオブジェクトがあります。

リレーションシップがされたなど、のをジョイン·オブジェクトにするに、にされます。

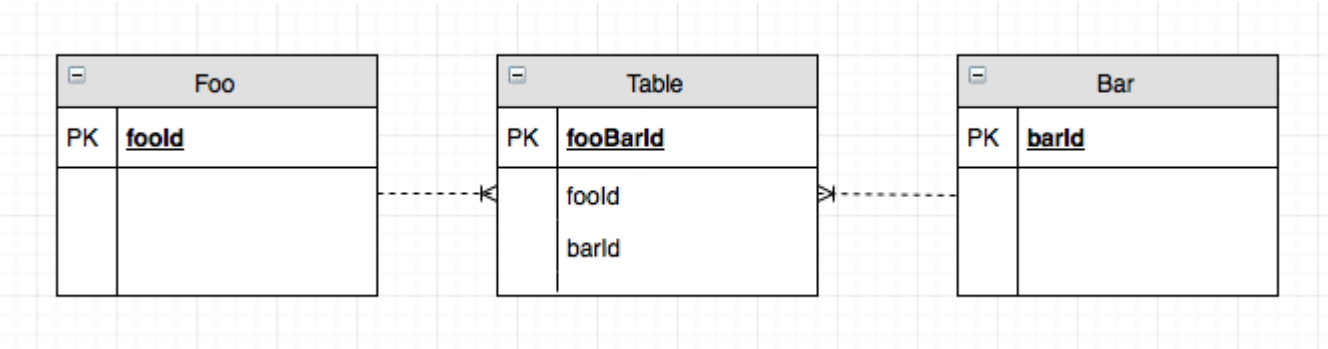


Hibernateマネージド·ジョイン·テーブルをするの

```
@Entity
@Table(name="FOO")
public class Foo {
    private UUID fooId;

    @OneToMany
    @JoinTable(name="FOO_BAR",
        joinColumns = @JoinColumn(name="fooId"),
        inverseJoinColumns = @JoinColumn(name="barId"))
    private List<Bar> bars;
}

@Entity
@Table(name="BAR")
public class Bar {
    private UUID barId;

    @OneToMany
    @JoinTable(name="FOO_BAR",
        joinColumns = @JoinColumn(name="barId"),
        inverseJoinColumns = @JoinColumn(name="fooId"))
    private List<Foo> foos;
}
```

Hibernateがするテーブルをして、くのFooオブジェクトとくのBarオブジェクトとのをします。

`Foo`オブジェクトは`FOO`とばれるテーブルにとしてされ`FOO`。 `Bar`オブジェクトは、 `BAR`というテーブルにとしてされます。 `Foo`オブジェクトと`Bar`オブジェクトのは、 `FOO_BAR`というテーブルにされます。しかし、これは、アプリケーションのとして`FooBar`オブジェクトがしないことをします。

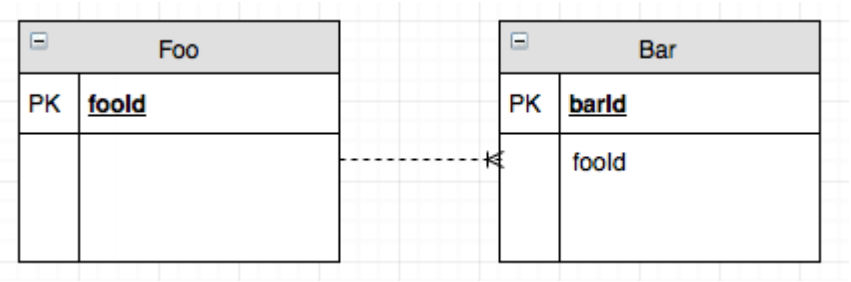


キーマッピングをした1リレーションシップ

```
@Entity
@Table(name="FOO")
public class Foo {
    private UUID fooId;

    @OneToMany(mappedBy = "bar")
    private List<Bar> bars;
}

@Entity
@Table(name="BAR")
public class Bar {
    private UUID barId;

    @ManyToOne
    @JoinColumn(name = "fooId")
    private Foo foo;
}
```

`Foo`オブジェクトとキーをするくの`Bar`オブジェクトとののをします。

`Foo`オブジェクトは`FOO`とばれるテーブルにとしてされ`FOO`。 `Bar`オブジェクトは、 `BAR`というテーブルにとしてされます。キーは、 `fooId`というの`BAR`テーブルにされます。



Foo.classによってされるのの

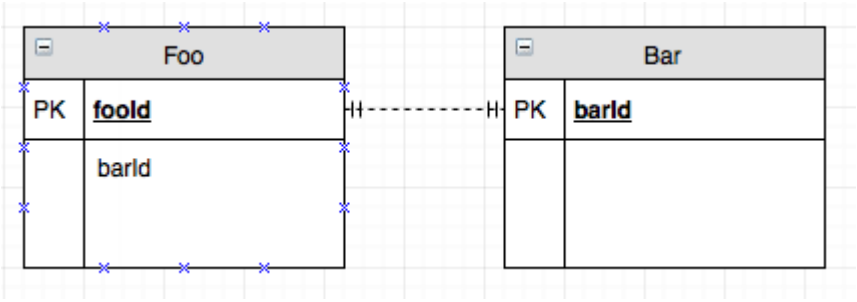
```
@Entity
@Table(name="FOO")
```

```
public class Foo {
    private UUID fooId;

    @OneToOne(cascade = CascadeType.ALL)
    @JoinColumn(name = "barId")
    private Bar bar;
}

@Entity
@Table(name="BAR")
public class Bar {
    private UUID barId;

    @OneToOne(mappedBy = "bar")
    private Foo foo;
}
```

キーをして1つの`Foo`オブジェクトと1つの`Bar`オブジェクトののをします。

`Foo`オブジェクトは`FOO`とばれるテーブルにとしてされ`FOO`。 `Bar`オブジェクトは、 `BAR`というテーブルにとしてされます。キーは、 `barId`というの`FOO`テーブルにされ`FOO`。

`mappedBy`は、オブジェクトのフィールドであり、ではありません。



ユーザーのジョイン·テーブルをした1リレーションシップ

```
@Entity
@Table(name="FOO")
public class Foo {
    private UUID fooId;

    @OneToMany
    @JoinTable(name="FOO_BAR",
        joinColumns = @JoinColumn(name="fooId"),
        inverseJoinColumns = @JoinColumn(name="barId", unique=true))
    private List<Bar> bars;
}

@Entity
@Table(name="BAR")
public class Bar {
    private UUID barId;

    //No Mapping specified here.
}

@Entity
```

```
@Table(name="FOO_BAR")
public class FooBar {
    private UUID fooBarId;

    @ManyToOne
    @JoinColumn(name = "fooId")
    private Foo foo;

    @ManyToOne
    @JoinColumn(name = "barId", unique = true)
    private Bar bar;

    //You can store other objects/fields on this table here.
}
```

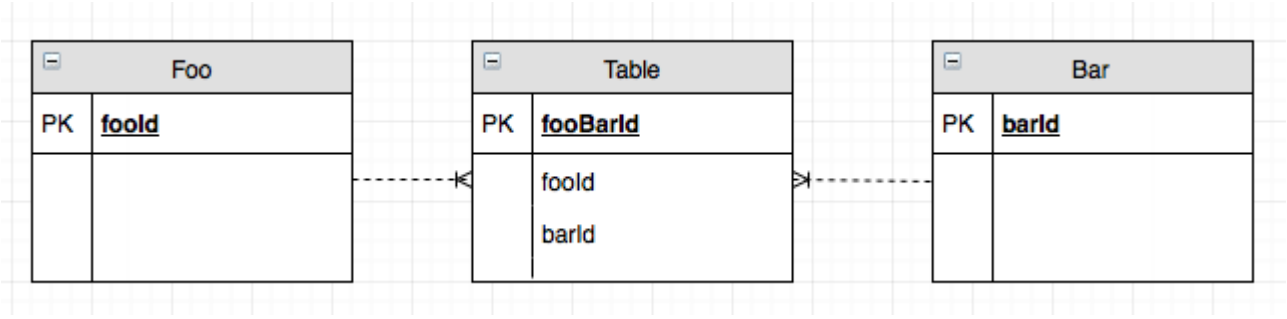
ユーザーがするテーブルをして、1つの`Foo`オブジェクトとの`Bar`オブジェクトとののをします。

これは`ManyToMany`にていますが、ターゲットキーに`unique`をすると、 `OneToMany`ことをでき`OneToMany`。

`Foo`オブジェクトは`FOO`とばれるテーブルにとしてされ`FOO`。 `Bar`オブジェクトは、 `BAR`というテーブルにとしてされます。 `Foo`オブジェクトと`Bar`オブジェクトのは、 `FOO_BAR`というテーブルにされます。アプリケーションのとして`FooBar`オブジェクトがあります。

`Foo`オブジェクトにる`Bar`オブジェクトのマッピングがないことにしてください。 `Bar`オブジェクトは`Foo`オブジェクトにをえずににできます。

できる`Role`のリストをつ`User`オブジェクトをするとき、Spring Securityでよくされます。あなたは、カスケードをにすることなく、ユーザーにをおよびすることができます`Role`さんを。



のの

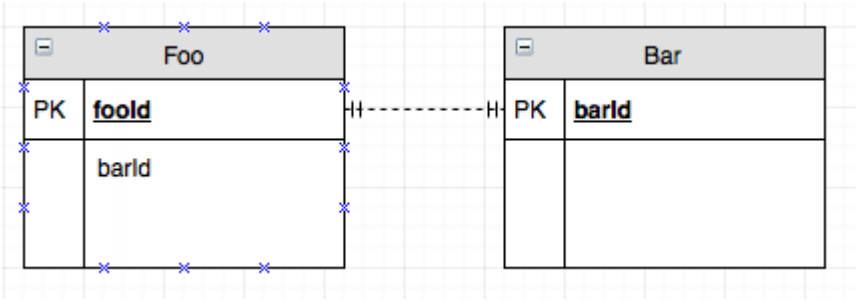
```
@Entity
@Table(name="FOO")
public class Foo {
    private UUID fooId;

    @OneToOne
    private Bar bar;
}

@Entity
@Table(name="BAR")
public class Bar {
```

```
    private UUID barId;
    //No corresponding mapping to Foo.class
}
```

1つのFooオブジェクトと1つのBarオブジェクトののをします。

Fooオブジェクトは`FOO`とばれるテーブルにとしてされ`FOO`。 Barオブジェクトは、 `BAR`というテーブルにとしてされます。

Fooオブジェクトにる`Bar`オブジェクトのマッピングがないことにしてください。 Barオブジェクトは`Foo`オブジェクトにをえずににできます。



オンラインでアノテーションをしたHibernateエンティティリレーションシップをむ
https://riptutorial.com/ja/hibernate/topic/5742/アノテーションをしたhibernateエンティティリレーションシップ

6: エンティティのアソシエーションマッピング

Examples

OneToManyアソシエーション

OneToManyをするために、2つのエンティティ、たとえばとがです。あるにはのがあります。

のCountryEntityには、Countryのセットがされています。

```
@Entity
@Table(name = "Country")
public class CountryEntity implements Serializable
{
   private static final long serialVersionUID = 1L;

   @Id
   @Column(name = "COUNTRY_ID", unique = true, nullable = false)
   @GeneratedValue(strategy = GenerationType.SEQUENCE)
   private Integer           countryId;

   @Column(name = "COUNTRY_NAME", unique = true, nullable = false, length = 100)
   private String            countryName;

   @OneToMany(mappedBy="country", fetch=FetchType.LAZY)
   private Set<CityEntity> cities = new HashSet<>();

   //Getters and Setters are not shown
}
```

、の。

```
@Entity
@Table(name = "City")
public class CityEntity implements Serializable
{
   private static final long serialVersionUID = 1L;

   @Id
   @Column(name = "CITY_ID", unique = true, nullable = false)
   @GeneratedValue(strategy = GenerationType.SEQUENCE)
   private Integer           cityId;

   @Column(name = "CITY_NAME", unique = false, nullable = false, length = 100)
   private String            cityName;

   @ManyToOne(optional=false, fetch=FetchType.EAGER)
   @JoinColumn(name="COUNTRY_ID", nullable=false)
   private CountryEntity country;

   //Getters and Setters are not shown
}
```

XMLをした1のけ

これは、XMLをして1のマッピングをするのです。とをとしてし、がくのをいているとしますが、には1のしかいません。

クラス

```
public class Author {
    private int id;
    private String firstName;
    private String lastName;

    public Author(){

    }
    public int getId(){
        return id;
    }
    public void setId(int id){
        this.id = id;
    }
    public String getFirstName(){
        return firstName;
    }
    public void setFirstName(String firstName){
        this.firstName = firstName;
    }
    public String getLastName(){
        return lastName;
    }
    public void setLastName(String lastName){
        this.lastName = lastName;
    }
}
```

ブッククラス

```
public class Book {
    private int id;
    private String isbn;
    private String title;
    private Author author;
    private String publisher;

    public Book() {
        super();
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getIsbn() {
        return isbn;
    }
    public void setIsbn(String isbn) {
```

```
        this.isbn = isbn;
    }
    public String getTitle() {
        return title;
    }
    public void setTitle(String title) {
        this.title = title;
    }
    public Author getAuthor() {
        return author;
    }
    public void setAuthor(Author author) {
        this.author = author;
    }
    public String getPublisher() {
        return publisher;
    }
    public void setPublisher(String publisher) {
        this.publisher = publisher;
    }
}
```

Author.hbm.xml

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd" >

<hibernate-mapping>
   <class name="Author" table="author">
      <meta attribute="class-description">
         This class contains the author's information.
      </meta>
      <id name="id" type="int" column="author_id">
         <generator class="native"/>
      </id>
      <property name="firstName" column="first_name" type="string"/>
      <property name="lastName" column="last_name" type="string"/>
   </class>
</hibernate-mapping>
```

Book.hbm.xml

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd" >

<hibernate-mapping>
    <class name="Book" table="book_title">
      <meta attribute="class-description">
         This class contains the book information.
      </meta>
        <id name="id" type="int" column="book_id">
            <generator class="native"/>
        </id>
        <property name="isbn" column="isbn" type="string"/>
        <property name="title" column="title" type="string"/>
```

```
        <many-to-one name="author" class="Author" cascade="all">
            <column name="author"></column>
        </many-to-one>
       <property name="publisher" column="publisher" type="string"/>
    </class>
</hibernate-mapping>
```

1のをるのは、BookクラスにAuthorがまれ、XMLに<many-to-one>タグがあることです。カスケードをすると、エンティティの/をできます。

オンラインでエンティティのアソシエーションマッピングをむ
https://riptutorial.com/ja/hibernate/topic/6165/エンティティのアソシエーションマッピング

7: カスタムネーミング

Examples

カスタムImplicitNamingStrategyのと

カスタム`ImplicitNamingStrategy`することで、Hibernateが、キー、キー、、など、でないの`Entity`にをりてるをすることができます。

たとえば、デフォルトでは、Hibernateはハッシュされたキーをし、のようになります。

```
FKe6hidh4u0qh8y1ijy59s2ee6m
```

これはしばしばではありませんが、そのようなは、

```
FK_asset_tenant
```

これは、カスタムの`ImplicitNamingStrategy`でにうことができます。

このでは`ImplicitNamingStrategyJpaCompliantImpl`していますが、にじて`ImplicitNamingStrategy`をすることもできます。

```
import org.hibernate.boot.model.naming.Identifier;
import org.hibernate.boot.model.naming.ImplicitForeignKeyNameSource;
import org.hibernate.boot.model.naming.ImplicitNamingStrategyJpaCompliantImpl;

public class CustomNamingStrategy extends ImplicitNamingStrategyJpaCompliantImpl {

    @Override
    public Identifier determineForeignKeyName(ImplicitForeignKeyNameSource source) {
        return toIdentifier("FK_" + source.getTableName().getCanonicalName() + "_" +
source.getReferencedTableName().getCanonicalName(), source.getBuildingContext());
    }

}
```

どの`ImplicitNamingStrategy`をするかをHibernateにするには、のように、 `persistence.xml`または`hibernate.cfg.xml`ファイルの`hibernate.implicit_naming_strategy`プロパティをします。

```
<property name="hibernate.implicit_naming_strategy"
                  value="com.example.foo.bar.CustomNamingStrategy"/>
```

あるいは、のように`hibernate.properties`ファイルでプロパティをすることもできます

```
hibernate.implicit_naming_strategy=com.example.foo.bar.CustomNamingStrategy
```

このでは、にされた`name`をたないすべてのキーが、 `CustomNamingStrategy`からをするようになりま

した。

カスタムネーミング

たちのエンティティをデータベースのテーブルにマッピングするとき、たちは`@Table`アノテーションにします。しかし、データベーステーブルのをっているは、`@Table`アノテーションをにすることなく、エンティティのにづいてテーブルをするようにhibernateにするために、カスタムのをできます。とのマッピングもじです。

たとえば、エンティティはのとおりです。

```
ApplicationEventLog
```

たちのテーブルは

```
application_event_log
```

たちのなは、キャメルケースのエンティティからスネークケースのdbテーブルにするがあります。これをするには、Hibernateの`PhysicalNamingStrategyStandardImpl`します。

```
import org.hibernate.boot.model.naming.Identifier;
import org.hibernate.boot.model.naming.PhysicalNamingStrategyStandardImpl;
import org.hibernate.engine.jdbc.env.spi.JdbcEnvironment;

public class PhysicalNamingStrategyImpl extends PhysicalNamingStrategyStandardImpl {

    private static final long serialVersionUID = 1L;
    public static final PhysicalNamingStrategyImpl INSTANCE = new
PhysicalNamingStrategyImpl();

    @Override
    public Identifier toPhysicalTableName(Identifier name, JdbcEnvironment context) {
        return new Identifier(addUnderscores(name.getText()), name.isQuoted());
    }

    @Override
    public Identifier toPhysicalColumnName(Identifier name, JdbcEnvironment context) {
        return new Identifier(addUnderscores(name.getText()), name.isQuoted());
    }

    protected static String addUnderscores(String name) {
        final StringBuilder buf = new StringBuilder(name);
        for (int i = 1; i < buf.length() - 1; i++) {
            if (Character.isLowerCase(buf.charAt(i - 1)) &&
                    Character.isUpperCase(buf.charAt(i)) &&
                    Character.isLowerCase(buf.charAt(i + 1))) {
                buf.insert(i++, '_');
            }
        }
        return buf.toString().toLowerCase(Locale.ROOT);
    }
}
```

たちは、メソッドのデフォルトのをオーバーライドしている`toPhysicalTableName`と`toPhysicalColumnName`たちデシベルのをします。

カスタムをするには、 `hibernate.physical_naming_strategy`プロパティをし、`PhysicalNamingStrategyImpl`クラスのをえるがあります。

```
hibernate.physical_naming_strategy=com.example.foo.bar.PhysicalNamingStrategyImpl
```

このようにして`@Table`と`@Column`アノテーションからコードをすることができるので、エンティティクラス

```
@Entity
public class ApplicationEventLog {
    private Date startTimestamp;
    private String logUser;
    private Integer eventSuccess;

    @Column(name="finish_dtl")
    private String finishDetails;
}
```

dbテーブルにしくマップされます

```
CREATE TABLE application_event_log (
  ...
  start_timestamp timestamp,
  log_user varchar(255),
  event_success int(11),
  finish_dtl varchar(2000),
  ...
)
```

のにられるように、たちはなにってらかのでにdbオブジェクトのを`@Column(name="finish_dtl")`することができます `@Column(name="finish_dtl")`

オンラインでカスタムネーミングをむ https://riptutorial.com/ja/hibernate/topic/3051/カスタムネーミング

8: キャッシング

Examples

WildFlyでのHibernateキャッシングの

WildFlyでHibernateの2レベルキャッシングをにするには、このプロパティを`persistence.xml`ファイルにします。

```
<property name="hibernate.cache.use_second_level_cache" value="true"/>
```

また、このプロパティでクエリキャッシングをにすることもできます。

```
<property name="hibernate.cache.use_query_cache" value="true"/>
```

WildFlyは、デフォルトでInfinispanがされるため、Hibernateのセカンドレベルキャッシュをにするときにキャッシュプロバイダをするはありません。ただし、のキャッシュ·プロバイダをするは、`hibernate.cache.provider_class`プロパティをします。

オンラインでキャッシングをむ https://riptutorial.com/ja/hibernate/topic/3462/キャッシング

9: ネイティブSQLクエリ

Examples

なクエリ

Hibernate `Session`オブジェクトこの、 `session`というのハンドルがあるとします。

```
List<Object[]> result = session.createNativeQuery("SELECT * FROM some_table").list();
for (Object[] row : result) {
    for (Object col : row) {
        System.out.print(col);
    }
}
```

これにより、 `some_table`のすべてのがされ、 `result`に`some_table`され、すべてのがされます。

のをるための

```
Object pollAnswered = getCurrentSession().createSQLQuery(
        "select * from TJ_ANSWERED_ASW where pol_id = "+pollId+" and prf_log =
'"+logid+"'").uniqueResult();
```

このクエリでは、クエリのがににになることがわかっているときに、のがられます。

クエリでのがされた、がします

    org.hibernate.NonUniqueResultException

あなたはまた、このリンクのをよりにチェックします

したがって、クエリがのをすことがわかっていることをしてください

オンラインでネイティブSQLクエリをむ https://riptutorial.com/ja/hibernate/topic/6978/ネイティブsqlクエリ

10: マッピングアソシエーション

Examples

11ハイバネーションマッピング

すべてのには1つあります。すべてのには1つのがあります。

Country.java

```
package com.entity;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.OneToOne;
import javax.persistence.Table;

@Entity
@Table(name = "countries")
public class Country {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @Column(name = "name")
    private String name;

    @Column(name = "national_language")
    private String nationalLanguage;

    @OneToOne(mappedBy = "country")
    private Capital capital;

    //Constructor

    //getters and setters

   }
```

Capital.java

```
package com.entity;

import javax.persistence.CascadeType;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.OneToOne;
import javax.persistence.Table;
```

```
@Entity
@Table(name = "capitals")
public class Capital {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    private String name;

    private long population;

    @OneToOne(cascade = CascadeType.ALL)
    @JoinColumn(name = "country_id")
    private Country country;

    //Constructor

    //getters and setters

}
```

HibernateDemo.java

```
package com.entity;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class HibernateDemo {

public static void main(String ar[]) {
    SessionFactory sessionFactory = new Configuration().configure().buildSessionFactory();
    Session session = sessionFactory.openSession();
    Country india = new Country();
    Capital delhi = new Capital();
    delhi.setName("Delhi");
    delhi.setPopulation(357828394);
    india.setName("India");
    india.setNationalLanguage("Hindi");
    delhi.setCountry(india);
    session.save(delhi);
    session.close();
  }

}
```

オンラインでマッピングアソシエーションをむ https://riptutorial.com/ja/hibernate/topic/6478/マッピングアソシエーション

11: レイジーロードとEagerロード

Examples

レイジーロードとEagerロード

データのまたはみみは、に、eagerとlazyの2にされます。

Hibernateをするには、pom.xmlファイルのセクションにバージョンをしてください

```
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>5.2.1.Final</version>
</dependency>
```

1. Eager Loadingと**Lazy Loading**

たちがここですべきのことは、なみみとなみみがであるかです

Eager Loadingは、そのでデータのがわれるデザインパターンです。これは、がフェッチされたでコレクションがにフェッチされるすぐにフェッチされることをします。

レイジーローディングは、オブジェクトのをなまでするためにされるデザインパターンです。これにより、アプリケーションのパフォーマンスににできます。

2.さまざまなのみみをする

レイジーローディングは、のXMLパラメータをしてにできます。

```
lazy="true"
```

をりげてみましょう。まず、Userクラスがあります。

```
public class User implements Serializable {

    private Long userId;
    private String userName;
    private String firstName;
    private String lastName;
    private Set<OrderDetail> orderDetail = new HashSet<>();

    //setters and getters
    //equals and hashcode
    }
```

たちがっているOrderDetailのセットをてください。 **OrderDetail**クラスをてみましょう

```
public class OrderDetail implements Serializable {

    private Long orderId;
    private Date orderDate;
    private String orderDesc;
    private User user;

    //setters and getters
    //equals and hashcode
}
```

`UserLazy.hbm.xml`ロードのにするなは`UserLazy.hbm.xml`です。

```
<set name="orderDetail" table="USER_ORDER" inverse="true" lazy="true" fetch="select">
    <key>
        <column name="USER_ID" not-null="true" />
    </key>
   <one-to-many class="com.baeldung.hibernate.fetching.model.OrderDetail" />
</set>
```

これは、みみがになるです。みみをにするには、に`lazy = "false"`をするだけです。これによってにみみがになります。は、のファイルUser.hbm.xmlにeager loadingをするです。

```
<set name="orderDetail" table="USER_ORDER" inverse="true" lazy="false" fetch="select">
    <key>
        <column name="USER_ID" not-null="true" />
    </key>
   <one-to-many class="com.baeldung.hibernate.fetching.model.OrderDetail" />
</set>
```

これら2つのデザインでんでいないにとっては、けでありするのはの*SessionFactory*の セッション にあります。 *Eager*はにすべてをロードします。つまり、フェッチするためにかをびすはありません。しかし、フェッチでは、マップされたコレクション/オブジェクトをするためのアクションがです。これはセッションのでなフェッチをするのがになることがあります。たとえば、マップされたPOJOのをすビューがあります。

```
@Entity
public class User {
    private int userId;
    private String username;
    @OneToMany
    private Set<Page> likedPage;

    // getters and setters here
}

@Entity
public class Page{
    private int pageId;
    private String pageURL;

    // getters and setters here
}

public class LazzyTest{
```

```
    public static void main(String...s){
        SessionFactory sessionFactory = new SessionFactory();
        Session session = sessionFactory.openSession();
        Transaction transaction = session.beginTransaction();

        User user = session.get(User.class, 1);
        transaction.commit();
        session.close();

        // here comes the lazy fetch issue
        user.getLikedPage();
    }
}
```

セッションのでlazyをフェッチしようとすると、 *lazyinitializeException*がします。デフォルトですべてのoneToManyやのはであるオンデマンドDBへのびしとセッションをじているとき、あなたはデ―タベ―スとするためにのをっていないためのをフェッチするためです。そのため、たちのコ―ドは*favoritePage*のコレクションをしようとしますが、DBをレンダリングするためのするセッションがないためがスロ―されます。

これをするには、をします。

1. ビュ―のセッションをく - レンダリングされたビュ―でもセッションをいたままにします。
2. セッションをじるに`Hibernate.initialize(user.getLikedPage())` - これは、コレクションをするようにhibernateにします

オンラインでレイジ―ロ―ドとEagerロ―ドをむ https://riptutorial.com/ja/hibernate/topic/7249/レイジ―ロ―ドとeagerロ―ド

12: でのフェッチ

き

JPAJava Persistence APIではフェッチがにです。 JPAでは、HQLHibernate Query LanguageとJPQLJava Persistence Query Languageをしてエンティティをそのについてフェッチします。ネイティブSQLをしてむものをるためには、にくのクエリとサブクエリをするよりもれていますが、JPAでエンティティをするはとしてにアプリケーションのパフォーマンスにします。

Examples

FetchType.LAZYをすることをおめします。がなときにフェッチしてします。

は、Employerエンティティクラスで、テーブルのにりてられています。ごのとおり、 **fetch = FetchType.EAGER**のわりにfetch = FetchType.LAZYをしました。がLAZYをしているは、がでくのをっているがあり、のすべてのをるがないたびに、すべてのをロードするとがとなります。

```
@Entity
@Table(name = "employer")
    public class Employer
    {
        @Id
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        private Long id;

        @Column(name = "name")
        private String Name;

        @OneToMany(mappedBy = "employer", fetch = FetchType.LAZY,
               cascade = { CascadeType.ALL }, orphanRemoval = true)
        private List<Employee> employees;

        public Long getId() {
            return id;
        }

        public void setId(Long id) {
            this.id = id;
        }

        public String getName() {
            return name;
        }

        public void setName(String name) {
            this.name = name;
        }

        public List<Employee> getEmployees() {
            return employees;
        }
```

```
        public void setEmployees(List<Employee> employees) {
            this.employees = employees;
        }
    }
```

ただし、LAZYフェッチのけでは、されていないプロキシがLazyInitializationExceptionにつながることがあります。この、にHQL / JPQLでJOIN FETCHをしてLazyInitializationExceptionをできます。

```
SELECT Employer employer FROM Employer
       LEFT JOIN FETCH employer.name
       LEFT JOIN FETCH employer.employee employee
       LEFT JOIN FETCH employee.name
       LEFT JOIN FETCH employer.address
```

オンラインでででのフェッチをむ https://riptutorial.com/ja/hibernate/topic/9475/でのフェッチ

13: と

Examples

をしたリスト

CityというのTravelReviewテーブルが "title"というであるとすると、

```
 Criteria criteria =
    session.createCriteria(TravelReview.class);
  List review =
    criteria.add(Restrictions.eq("title", "Mumbai")).list();
  System.out.println("Using equals: " + review);
```

にをえるには、のようにさせます。

```
List reviews = session.createCriteria(TravelReview.class)
   .add(Restrictions.eq("author", "John Jones"))
   .add(Restrictions.between("date",fromDate,toDate))
   .add(Restrictions.ne("title","New York")).list();
```

プロジェクションの

いくつかのカラムだけをしたいは、Projectionsクラスをします。たとえば、のコードはtitleをします。

```
 // Selecting all title columns
  List review = session.createCriteria(TravelReview.class)
        .setProjection(Projections.property("title"))
    .list();
  // Getting row count
  review = session.createCriteria(TravelReview.class)
    .setProjection(Projections.rowCount())
    .list();
  // Fetching number of titles
  review = session.createCriteria(TravelReview.class)
    .setProjection(Projections.count("title"))
    .list();
```

フィルタをする

`@Filter`は`WHERE`キャンプとしてされ、ここではいくつかのがあります

```
@Entity
@Table(name = "Student")
public class Student
{
    /*...*/
```

```
    @OneToMany
    @Filter(name = "active", condition = "EXISTS(SELECT * FROM Study s WHERE state = true and
s.id = study_id)")
    Set<StudentStudy> studies;

    /* getters and setters methods */
}
```

```
@Entity
@Table(name = "Study")
@FilterDef(name = "active")
@Filter(name = "active", condition="state = true")
public class Study
{
    /*...*/

    @OneToMany
    Set<StudentStudy> students;

    @Field
    boolean state;

    /* getters and setters methods */
}
```

StudentStudyエンティティ

```
@Entity
@Table(name = "StudentStudy")
@Filter(name = "active", condition = "EXISTS(SELECT * FROM Study s WHERE state = true and s.id
= study_id)")
public class StudentStudy
{
    /*...*/

    @ManytoOne
    Student student;

    @ManytoOne
    Study study;

    /* getters and setters methods */
}
```

このように、「アクティブ」フィルタがになるたびに、

- たちがのでうすべてのは、 すべてのに`state = true`のみをし`state = true`

- エンティティでうすべてのクエリは、 すべての `state = true`をす

- StudentStudy entiyでうすべてのクエリは、 `state = true`もののみをします。Study relationship

Plsは、study_idがsql StudentStudyテーブルのフィールドのであることにしてください

オンラインでとをむ https://riptutorial.com/ja/hibernate/topic/3939/と

14:

Examples

EAGERフェッチタイプはしないでください

Hibernateは、2つのエンティティ `EAGER`と`LAZY`のをマッピングするときに、2のフェッチをできます。

に、 `EAGER`フェッチタイプはいえではありません。なぜなら、このデ―タがでないときでも、JPAがにデ―タをフェッチするようにするからです。

のように、あなたが`Person`エンティティをっていて、のような`Address`とのがある

```
@Entity
public class Person {

  @OneToMany(mappedBy="address", fetch=FetchType.EAGER)
  private List<Address> addresses;

}
```

`Person`にするたびに、この`Person`の`Address`のリストもされます。

したがって、あなたのエンティティをのものとマッピングするのではなく、

```
@ManyToMany(mappedBy="address", fetch=FetchType.EAGER)
```

つかいます

```
@ManyToMany(mappedBy="address", fetch=FetchType.LAZY)
```

すべきもうつのことは`@OneToOne`と`@ManyToOne@ManyToOne`。 どちらもデフォルトではEAGER です 。したがって、アプリケ―ションのパフォ―マンスがされるは、このタイプののフェッチをするがあります。

```
@ManyToOne(fetch=FetchType.LAZY)
```

そして

```
@OneToOne(fetch=FetchType.LAZY)
```

のわりにをする

Hibernateにはいくつかのがあります。 `JOINED`は、エンティティとエンティティのでJOINをしま

す。

このアプローチのは、Hibernateがにするすべてのテーブルのデータをにちむことです。

たとえば、 `JOINED`をするエンティティ`Bicycle`および`MountainBike`があるは、のようになります。

```
@Entity
@Inheritance(strategy = InheritanceType.JOINED)
public abstract class Bicycle {

}
```

そして

```
@Entity
@Inheritance(strategy = InheritanceType.JOINED)
public class MountainBike extends Bicycle {

}
```

`MountainBike`をったJPQLクエリは`Bicycle`データをし、のようなSQLクエリをします。

```
select mb.*, b.* from MountainBike mb JOIN Bicycle b ON b.id = mb.id WHERE ...
```

`Bicycle`ののをっているとして`Transport`ように、のクエリはこのからのデータもし、なJOINをいます。

ごのとおり、これは`EAGER`マッピングのです。このストラテジをして`MountainBike`テーブルのデータのみをするはありません。

パフォーマンスにとってののは、のわりにのです。

これをするには、 `MountainBike`エンティティをフィールド`bicycle`マッピングすることができます。

```
@Entity
public class MountainBike {

    @OneToOne(fetchType = FetchType.LAZY)
    private Bicycle bicycle;

}
```

そして`Bicycle`

```
@Entity
public class Bicycle {

}
```

すべてのクエリはデフォルトで`MountainBike`データのみをちます。

オンライ ンでをむ https://riptutorial.com/ja/hibernate/topic/2326/

クレジット

S. No		Contributors
1	をする	Community, JamesENL, Michael Piefel, Naresh Kumar, Reborn, user7491506
2	HibernateとJPA	Michael Piefel
3	HQL	Daniel Käfer, user7491506
4	SQLログを/にする	Daniel Käfer, Dherik, JamesENL, Michael Piefel
5	アノテーションをしたHibernateエンティティリレーションシップ	Aleksei Loginov, JamesENL
6	エンティティのアソシエーションマッピング	StanislavL, user7491506
7	カスタムネーミング	Mitch Talmadge, Naresh Kumar, veljkost
8	キャッシング	Mitch Talmadge
9	ネイティブSQLクエリ	Daniel Käfer, Nathaniel Ford, Sandeep Kamath
10	マッピングアソシエーション	Dherik, omkar sirra
11	レイジーロードとEagerロード	BELLIL, Pramod, Pritam Banerjee, vicky
12	でのフェッチ	rObOtAndChalie
13	と	Saifer, Sameer Srivastava
14		Dherik, Michael Piefel