REliable, INtelligent & Scalable Systems

互联网应用开发技术

Web Application Development

第8课
WEB后端-访问关系型数据库2

Episode Eight

Access to RDBMS With JPA 2

Web Application Development

陈昊鹏

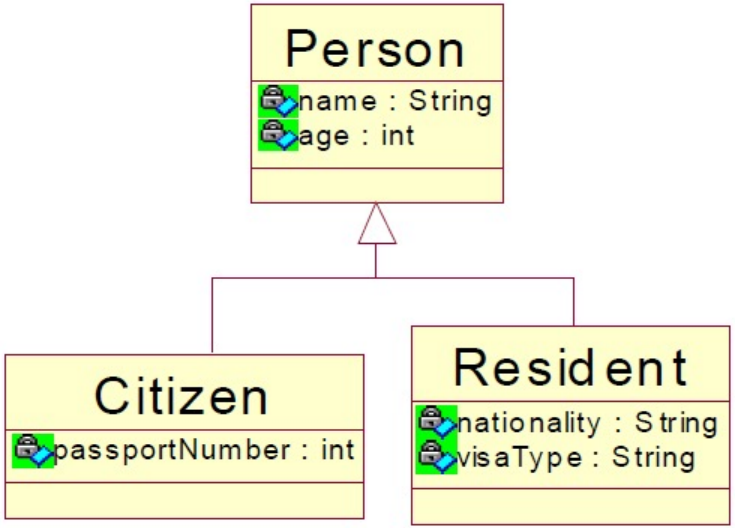
chen-hp@sjtu.edu.cn

Modeling Inheritance

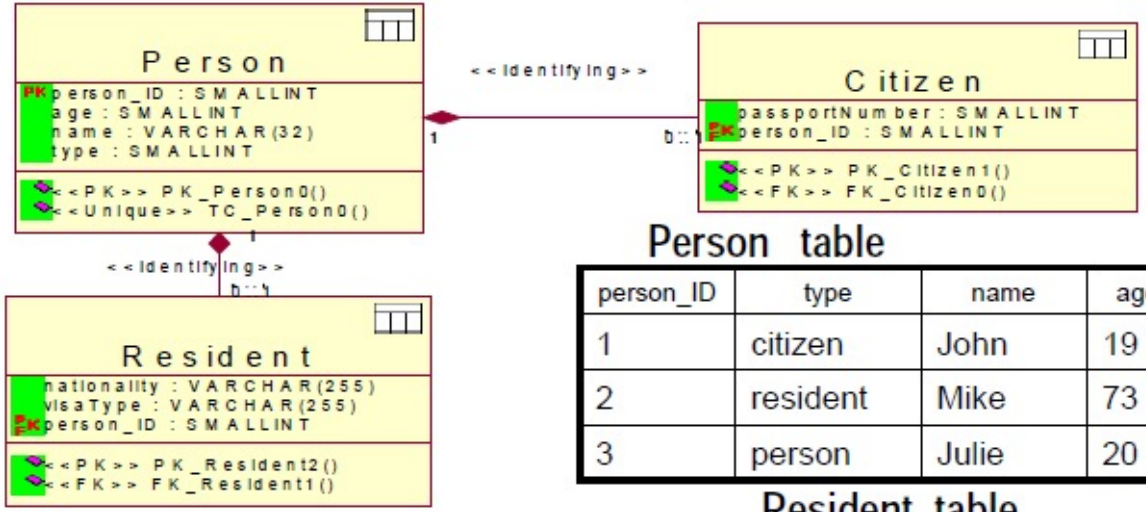
- A data model does not support modeling inheritance in a direct way

- MappedSuperclass
  - Inheritance is implemented in the domain model only without reflecting it in the database schema.
- Single table
  - The domain model class hierarchy is materialized into a single table which contains entities belonging to different class types.
- Joined table
  - The base class and all the subclasses have their own database tables and fetching a subclass entity requires a join with the parent table as well.
- Table per class
  - Each subclass has its own table containing both the subclass and the base class properties.

Modeling Inheritance



Joined Table



Person table

person_ID	type	name	age
1	citizen	John	19
2	resident	Mike	73
3	person	Julie	20

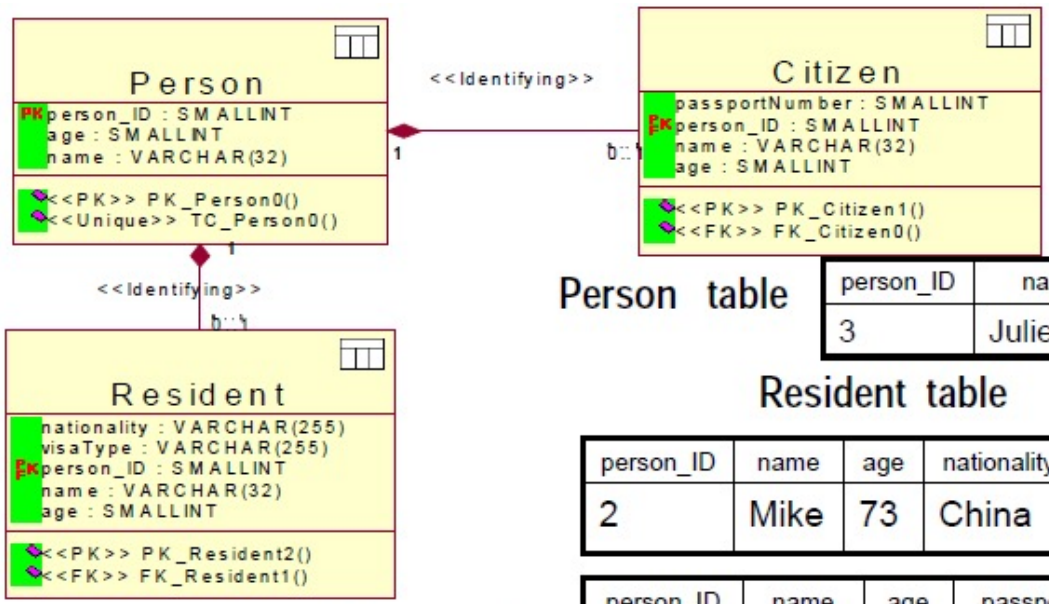
Citizen table

person_ID	passportNumber
1	123-456

Resident table

person_ID	nationality	visaType
2	China	H-1

Table per Class



Person table

person_ID	name	age
3	Julie	20

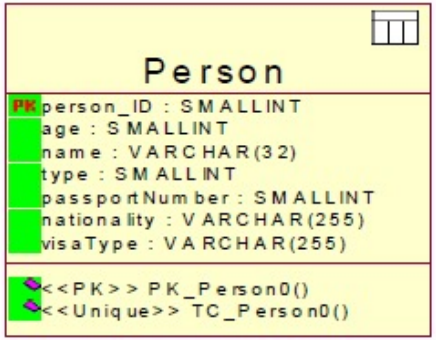
Resident table

person_ID	name	age	nationality	visaType
2	Mike	73	China	H-1

Citizen table

person_ID	name	age	passportNumber
1	John	19	123-456

Single Table



Person table

person_ID	type	name	age	passportNumber	nationality	visaType
1	citizen	John	19	123-456		
2	resident	Mike	73		China	H-1
3	person	Julie	20			

Single Table

planes
*ID
DISC
type
manufacturer
capacity
comfort

ID	DISC	type	manufacturer	capacity	comfort
1	2	320	Airbus	320	NULL
2	1	777	Boeing	NULL	Great
3	0	78	II	NULL	NULL

Single Table

- Plane.java

```
@Entity
@Table(name = "plane")
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name = "disc", discriminatorType = DiscriminatorType.STRING)
@DiscriminatorOptions(force=true)
@DiscriminatorValue(value = "0")
public class Plane {
   private Long id;
   private String type;
   private String manufacturer;

   public Plane() {
   }

   @Id
   @GeneratedValue(generator = "increment")
   @GenericGenerator(name = "increment",
                     strategy = "increment")
   public Long getId() {
      return id;
   }

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

   public String getType() {
      return type;
   }

   public void setType(String type) {
      this.type = type;
   }

   public String getManufacturer() {
      return manufacturer;
   }

   public void setManufacturer(String manufacturer) {
      this.manufacturer = manufacturer;
   }

}
```

Single Table

- Airbus.java

```
@Entity
@DiscriminatorValue(value = "1" )
public class Airbus extends Plane{
   private String capacity;
   public Airbus() {}

   public String getCapacity() { return capacity; }
   public void setCapacity(String capacity) { this.capacity = capacity; }
}
```

- Boeing.java

```
@Entity
@DiscriminatorValue(value = "2" )
public class Boeing extends Plane{
   private String comfort;
   public Boeing() {}

   public String getComfort() { return comfort; }
   public void setComfort(String comfort) { this.comfort = comfort; }
}
```

Single Table

- PlaneServlet.java

```
@WebServlet("/Plane")
public class PlaneServlet extends HttpServlet {
  private static final long serialVersionUID = 1L;

  public PlaneServlet() {
    super();
  }

  protected void processRequest(HttpServletRequest request,
      HttpServletResponse response)
      throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    try {
      out.println("<html lang=\"en\">");
      out.println("<head>");
      out.println("<title>Servlet PlaneServlet</title>");
      out.println("</head>");
      out.println("<body>");

      String manufacturer = (String) request.getParameter("manufacturer");
      System.out.println(manufacturer);
```

Single Table

- PlaneServlet.java

```
Session session = HibernateUtil.getSessionFactory().getCurrentSession();
session.beginTransaction();

Airbus airbus = new Airbus();
airbus.setManufacturer("Airbus");
airbus.setType("320");
airbus.setCapacity("320");
session.save(airbus);

Boeing boeing = new Boeing();
boeing.setManufacturer("Boeing");
boeing.setType("777");
boeing.setComfort("Great");
session.save(boeing);

Plane plane = new Plane();
plane.setManufacturer("Il");
plane.setType("78");
session.save(plane);
```

Single Table

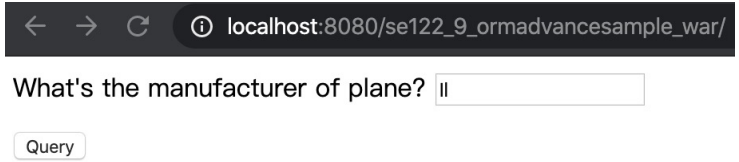
- PlaneServlet.java

```
List planes = session.createQuery("from Plane as p where p.manufacturer = :manu").
        setParameter("manu", manufacturer).list();
for (int i = 0; i < planes.size(); i++) {
  Plane thePlane = (Plane) planes.get(i);
  out.println("id: " + thePlane.getId() + "<br>" + "type: "
      + thePlane.getManufacturer() + thePlane.getType() + "<br>"
      + "manufacturer: " + thePlane.getManufacturer() + "<br><br>");
}

  session.getTransaction().commit();

  out.println("</body>");
out.println("</html>");

} catch(Exception e){
  e.printStackTrace();
}
finally {
  out.close();
}
}
```

Single Table

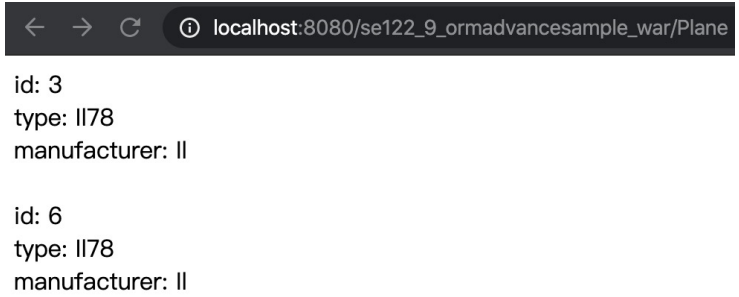
- index.html

```
<HTML>
<BODY>
<FORM METHOD=POST ACTION="Plane">
  What's the manufacturer of plane? <INPUT TYPE=TEXT NAME=manufacturer SIZE=20><BR>
  <P><INPUT TYPE=SUBMIT value="Query">
</FORM>
</BODY>
</HTML>
```



- hibernate.cfg.xml

```
<!-- Names the annotated entity class -->
<mapping class="org.reins.orm.Entity.Plane"/>
<mapping class="org.reins.orm.Entity.Airbus"/>
<mapping class="org.reins.orm.Entity.Boeing"/>
```



Joined Table

CATS
*UID
birthday
color
sex
weight

DOMESTIC_CATS
*CAT
*name

UID	birthday	color	sex	weight
3	2017-04-16 00:00:00	blue	male	20

CAT	name
3	Doraemon

Joined Table

- Cat.java

```
@Id
@GeneratedValue(generator = "increment")
@GenericGenerator(name = "increment", strategy = "increment")
public Long getId() {
    return id;
}

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

public Date getBirthday() {
    return birthday;
}

public void setBirthday(Date birthday) {
    this.birthday = birthday;
}

public String getColor() {
    return color;
}

public void setColor(String color) {
    this.color = color;
}

public String getSex() {
    return sex;
}

public void setSex(String sex) {
    this.sex = sex;
}

public int getWeight() {
    return weight;
}

public void setWeight(int weight) {
    this.weight = weight;
}

}
```

Joined Table

- DomesticCat.java

```
@Entity
@Table(name = "domestic_cat")
public class DomesticCat extends Cat {
  private String name;

   public DomesticCat() {}

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

- hibernate.cfg.xml

```
<!-- Names the annotated entity class -->
<mapping class="org.reins.orm.Entity.Cat"/>
<mapping class="org.reins.orm.Entity.DomesticCat"/>
```

Joined Table

- CatServlet.java

```
@WebServlet("/Cat")
public class CatServlet extends HttpServlet {
  private static final long serialVersionUID = 1L;

  public CatServlet() {
     super();
  }

  protected void processRequest(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
     response.setContentType("text/html;charset=UTF-8");
     PrintWriter out = response.getWriter();
     try {
        out.println("<html lang=\"en\">");
        out.println("<head>");
        out.println("<title>Servlet CatServlet</title>");
        out.println("</head>");
        out.println("<body>");

        Session session = HibernateUtil.getSessionFactory().getCurrentSession();
        session.beginTransaction();
```

Joined Table

- CatServlet.java

```
DomesticCat domesticCat = new DomesticCat();
Date birthday = new Date();
domesticCat.setId(Long.valueOf(3));
domesticCat.setBirthday(birthday);
domesticCat.setColor("blue");
domesticCat.setName("Doraemon");
domesticCat.setSex("male");
domesticCat.setWeight(20);
session.save(domesticCat);

List cats = session.createQuery("from Cat").list();
for (int i = 0; i < cats.size(); i++) {
  Cat theCat = (Cat)cats.get(i);
  out.println("id: " + theCat.getId() + "<br>" +
          "birthday: " + theCat.getBirthday() + "<br>" +
         "color: " + theCat.getColor() + "<br>" +
         "weight: " + theCat.getWeight() + "<br>");
  if (theCat instanceof DomesticCat) {
    DomesticCat aCat = (DomesticCat)theCat;
    out.println("name: " + aCat.getName() + "<br>");
  }
  out.println("<br>");
}
session.getTransaction().commit();
```

Joined Table

- CatServlet.java

```
        out.println("</body>");
        out.println("</html>");

      } catch(Exception e){
        e.printStackTrace();
      }
      finally {
        out.close();
      }
    }

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
      processRequest(request, response);
    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
      processRequest(request, response);
    }

  }
```

localhost:8080/se122_9_ormadvancesample_war/Cat

```
id: 1
birthday: 2020-02-29 19:44:20.0
color: blue
weight: 20
name: Doraemon

id: 2
birthday: 2020-02-29 19:45:44.0
color: blue
weight: 20
name: Doraemon
```

Table per Class

FRUITS
*UID
shape
flavor
color

APPLES
*UID
shape
flavor
color
weight

UID	shape	flavor	color
NULL	NULL	NULL	NULL

UID	weight	shape	flavor	color
5	9	round	so-so	yellow

Table per Class

- Fruit.java

```
@Entity
@Table(name = "fruit")
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
public class Fruit {
   private Long id;
   private String shape;
   private String flavor;
   private String color;

   public Fruit() {
   }

   @Id
   @GeneratedValue(generator = "increment")
   @GenericGenerator(name = "increment",
                     strategy = "increment")
   public Long getId() {
      return id;
   }

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

   public String getShape() {
      return shape;
   }

   public void setShape(String shape) {
      this.shape = shape;
   }

   public String getFlavor() {
      return flavor;
   }

   public void setFlavor(String flavor) {
      this.flavor = flavor;
   }

   public String getColor() {
      return color;
   }

   public void setColor(String color) {
      this.color = color;
   }
}
```

Table per Class

- Apple.java

```
@Entity
@Table(name = "apple")
public class Apple extends Fruit {

    private int weight;

    public Apple() {}

    public int getWeight() { return weight; }
    public void setWeight(int weight) { this.weight = weight; }

}
```

- hibernate.cfg.xml

```
<!-- Names the annotated entity class -->
<mapping class="org.reins.orm.Entity.Fruit"/>
<mapping class="org.reins.orm.Entity.Apple"/>
```

Table per Class

- FruitServlet.java

```
@WebServlet("/Fruit")
public class FruitServlet extends HttpServlet {
  private static final long serialVersionUID = 1L;

   public FruitServlet() {
      super();
   }

   protected void processRequest(HttpServletRequest request,
         HttpServletResponse response)
         throws ServletException, IOException {
      response.setContentType("text/html;charset=UTF-8");
      PrintWriter out = response.getWriter();
      try {
         out.println("<html lang=\"en\">");
         out.println("<head>");
         out.println("<title>Servlet FruitServlet</title>");
         out.println("</head>");
         out.println("<body>");

         Session session = HibernateUtil.getSessionFactory().getCurrentSession();
         session.beginTransaction();
```

Table per Class

- FruitServlet.java

```
Apple apple = new Apple();
apple.setId(new Long(5));
apple.setShape("round");
apple.setFlavor("so-so");
apple.setColor("yellow");
apple.setWeight(9);
session.save(apple);

List fruits = session.createQuery("from Fruit").list();

session.getTransaction().commit();

for (int i = 0; i < fruits.size(); i++) {
  Fruit theFruit = (Fruit)fruits.get(i);
  out.println("id: " + theFruit.getId() + "<br>" +
          "shape: " + theFruit.getShape() + "<br>" +
         "color: " + theFruit.getColor() + "<br>" +
         "flavor: " + theFruit.getFlavor() + "<br>");
  if (theFruit instanceof Apple) {
    Apple aFruit = (Apple)theFruit;
    out.println("weight: " + aFruit.getWeight() + "<br>");
  }
  out.println("<br>");
}
```

Table per Class

- FruitServlet.java

```
        out.println("</body>");
        out.println("</html>");

      } catch(Exception e){
        e.printStackTrace();
      }
      finally {
        out.close();
      }
    }

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
      processRequest(request, response);
    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
      processRequest(request, response);
    }

  }
```

localhost:8080/se122_9_ormadvancesample_war/Fruit

id: 1
shape: round
color: yellow
flavor: so–so
weight: 9

Hibernate object states

- Hibernate defines and supports the following object states:
  - *Transient*
    - an object is transient if it has just been instantiated using the new operator, and it is not associated with a Hibernate Session.
  - *Persistent*
    - a persistent instance has a representation in the database and an identifier value.
  - *Detached*
    - a detached instance is an object that has been persistent, but its Session has been closed.

Making objects persistent


```
DomesticCat fritz = new DomesticCat();
fritz.setColor(Color.GINGER);
fritz.setSex('M');
fritz.setName("Fritz");
Long generatedId = (Long) sess.save(fritz);

DomesticCat pk = new DomesticCat();
pk.setColor(Color.TABBY);
pk.setSex('F');
pk.setName("PK");
pk.setKittens( new HashSet() );
pk.addKitten(fritz);
sess.save( pk, new Long(1234) );
```

Or

```
sess.persist( pk, new Long(1234) );
```

Loading an object


```
Cat fritz = (Cat) sess.load(Cat.class, generatedId);

// you need to wrap primitive identifiers
long id = 1234;
DomesticCat pk = (DomesticCat) sess.load( DomesticCat.class, new Long(id) );

Cat cat = new DomesticCat();
// load pk's state into cat
sess.load( cat, new Long(pkId) );
Set kittens = cat.getKittens();

Cat cat = (Cat) sess.get(Cat.class, id);
 if (cat==null) {
   cat = new Cat();
   sess.save(cat, id);
}
return cat;

sess.save(cat);
sess.flush(); //force the SQL INSERT
sess.refresh(cat); //re-read the state (after the trigger executes)
```

Executing queries


```
 List cats = session.createQuery(
    "from Cat as cat where cat.birthdate < ?")
    .setDate(0, date)
    .list();
List mothers = session.createQuery(
    "select mother from Cat as cat join cat.mother
                    as mother where cat.name = ?")
    .setString(0, name)
    .list();
List kittens = session.createQuery(
    "from Cat as cat where cat.mother = ?")
    .setEntity(0, pk)
    .list();
Cat mother = (Cat) session.createQuery(
    "select cat.mother from Cat as cat where cat = ?")
    .setEntity(0, izi)
    .uniqueResult();
Query mothersWithKittens = (Cat) session.createQuery(
    "select mother from Cat as mother left join fetch mother.kittens");
Set uniqueMothers = new HashSet(mothersWithKittens.list());
```

Iterating results


```
Iterator kittensAndMothers = sess.createQuery(
    "select kitten, mother from Cat kitten join kitten.mother mother")
   .list()
   .iterator();
while ( kittensAndMothers.hasNext() ) {
   Object[] tuple = (Object[]) kittensAndMothers.next();
   Cat kitten = (Cat) tuple[0];
   Cat mother = (Cat) tuple[1];
   ....
}

Iterator results = sess.createQuery(
    "select cat.color, min(cat.birthdate), count(cat) from Cat cat " + "group by cat.color")
    .list()
    .iterator();
while ( results.hasNext() ) {
   Object[] row = (Object[]) results.next();
   Color type = (Color) row[0];
   Date oldest = (Date) row[1];
   Integer count = (Integer) row[2];
   .....
}
```

Bind parameters


```
//named parameter (preferred)
Query q = sess.createQuery("from DomesticCat cat
                    where cat.name = :name");
q.setString("name", "Fritz");
Iterator cats = q.iterate();

//positional parameter
Query q = sess.createQuery("from DomesticCat cat
                    where cat.name = ?");
q.setString(0, "Izi");
Iterator cats = q.iterate();

//named parameter list
List names = new ArrayList();
names.add("Izi");
names.add("Fritz");
Query q = sess.createQuery("from DomesticCat cat
                    where cat.name in (:namesList)");
q.setParameterList("namesList", names);
List cats = q.list();
```

Pagination and Scrollable iteration


```
Query q = sess.createQuery("from DomesticCat cat");
q.setFirstResult(20);
q.setMaxResults(10);
List cats = q.list();

Query q = sess.createQuery("select cat.name, cat from DomesticCat cat " + "order by cat.name");
ScrollableResults cats = q.scroll();
if ( cats.first() ) {
   // find the first name on each page of an alphabetical list of cats by name
   firstNamesOfPages = new ArrayList();
   do {
      String name = cats.getString(0);
      firstNamesOfPages.add(name);
   } while ( cats.scroll(PAGE_SIZE) );
   // Now get the first page of cats
   pageOfCats = new ArrayList();
   cats.beforeFirst();
   int i=0;
   while( ( PAGE_SIZE > i++ ) && cats.next() )
     pageOfCats.add( cats.get(1) );
}
cats.close()
```

Criteria queries and native SQL


```
Criteria crit = session.createCriteria(Cat.class);
crit.add( Restrictions.eq( "color", eg.Color.BLACK ) );
crit.setMaxResults(10);
List cats = crit.list();

List cats = session.createSQLQuery("SELECT {cat.*} FROM CAT {cat} WHERE
ROWNUM<10")
  .addEntity("cat", Cat.class)
  .list();
List cats = session.createSQLQuery(
  "SELECT {cat}.ID AS {cat.id}, {cat}.SEX AS {cat.sex}, " +
  "{cat}.MATE AS {cat.mate}, {cat}.SUBCLASS AS {cat.class}, ... " +
  "FROM CAT {cat} WHERE ROWNUM<10")
  .addEntity("cat", Cat.class)
  .list()
```

Modifying objects


```
DomesticCat cat = (DomesticCat) sess.load( Cat.class, new Long(69) );
cat.setName("PK");
sess.flush(); // changes to cat are automatically detected and persisted

// in the first session
Cat cat = (Cat) firstSession.load(Cat.class, catId);
Cat potentialMate = new Cat();
firstSession.save(potentialMate);

// in a higher layer of the application
cat.setMate(potentialMate);

// later, in a new session
secondSession.update(cat); // update cat
secondSession.update(mate); // update mate
or
secondSession.merge(cat); // merge cat
secondSession.merge(mate); // merge mate
```

Automatic state detection


```
// in the first session
Cat cat = (Cat) firstSession.load(Cat.class, catID);

// in a higher tier of the application
Cat mate = new Cat();
cat.setMate(mate);

// later, in a new session
secondSession.saveOrUpdate(cat);
// update existing state (cat has a non-null id)
secondSession.saveOrUpdate(mate);
// save the new instance (mate has a null id)
```

Deleting persistent objects


```
session.delete(cat);
```

Replicating object


```
//retrieve a cat from one database
Session session1 = factory1.openSession();
Transaction tx1 = session1.beginTransaction();
Cat cat = session1.get(Cat.class, catId);
tx1.commit();
session1.close();

//reconcile with a second database
Session session2 = factory2.openSession();
Transaction tx2 = session2.beginTransaction();
session2.replicate(cat, ReplicationMode.LATEST_VERSION);
tx2.commit();
session2.close();
```

- ReplicationMode:
  - IGNORE
  - OVERWRITE
  - EXCEPTION
  - LATEST_VERSION

Flushing the Session

- *flush*, occurs by default at the following points:
  - before some query executions
  - from `org.hibernate.Transaction.commit()`
  - from `Session.flush()`

- The SQL statements are issued in the following order:
  - all entity insertions in the same order the corresponding objects were saved using `Session.save()`
  - all entity updates
  - all collection deletions
  - all collection element deletions, updates and insertions
  - all collection insertions
  - all entity deletions in the same order the corresponding objects were deleted using `Session.delete()`

Flushing the Session


```
sess = sf.openSession();
Transaction tx = sess.beginTransaction();
sess.setFlushMode(FlushMode.COMMIT);
// allow queries to return stale state

Cat izi = (Cat) sess.load(Cat.class, id);
izi.setName(iznizi);

// might return stale data
sess.find("from Cat as cat left outer join cat.kittens kitten");

// change to izi is not flushed!
...
tx.commit(); // flush occurs
sess.close();
```

Transitive persistence

- For each basic operation of the Hibernate session there is a corresponding cascade style
- including `persist()`, `merge()`, `saveOrUpdate()`, `delete()`, `lock()`, `refresh()`, `evict()`, `replicate()`
  - `CascadeType.PERSIST`
  - `CascadeType.MERGE`
  - `CascadeType.REMOVE`
  - `CascadeType.REFRESH`
  - `CascadeType.DETACH`
  - `CascadeType.ALL`

Transitive persistence


```
@Entity
public class Customer {
   private Set<Order> orders;
   @OneToMany(cascade=CascadeType.ALL, orphanRemoval=true)
   public Set<Order> getOrders() { return orders; }
   public void setOrders(Set<Order> orders) {
     this.orders = orders;
  }
  ...
}

@Entity
public class Order { ... }

Customer customer = em.find(Customer.class, 1l);
Order order = em.find(Order.class, 1l);
customer.getOrders().remove(order);
//order will be deleted by cascade
```

References

- Hibernate ORM - Getting started with Hibernate ORM
  - https://hibernate.org/orm/documentation/getting-started/
- Hibernate ORM – Final User Guide
  - https://docs.jboss.org/hibernate/orm/current/userguide/html_single/Hibernate_User_Guide.html
- The Java EE 8 Tutorial – Introduction to the Java Persistence API
  - https://javaee.github.io/tutorial/persistence-intro.html#BNBPZ

- *Web开发技术*
- *Web Application Development*

Thank You!