

The real benefit of GORM

A comparison to native JPA

Kurt Munter

Mapping



- No annotations
 - No XML descriptors
 - Mapping by convention
 - Validation by constraints (lots of useful validators)
-
- but circumstantial Many-To-Many mapping

Mapping samples (class)



Grails Group Switzerland

JPA

```
@Entity
public class Person
    implements Serializable {
    private long id;
    private String name;

    public Person() {}
    @Id
    @GeneratedValue
    public long getId() {...}
    public String getName() {
        ...
    }
    public void setId() {...}
    public void setName(...) {
        ..}
}
```

GORM

```
class Person {
    String name
}
```

Mapping samples



Grails Group Switzerland

JPA: OneToMany bidir.

```
@Entity
public class Person
    implements Serializable {
    ...
    private Set<Car> cars
        = new HashSet<Car>();

    @OneToMany (
        mappedBy="owner")
    public Set<Car> getCars() {
        return cars;
    }
}
```

```
@Entity
public class Car
    implements Serializable {
    ...
    private Person person;

    @ManyToOne
    @JoinColumn (name="perid")
    public Person getPerson() {
        return person;
    }
}
```

Mapping samples



Grails Group Switzerland

GORM: OneToMany bidir.

```
class Person {  
    ...  
    static hasMany = [cars:Car]  
}
```

```
class Car {  
    Person person  
}
```

Mapping samples



Grails Group Switzerland

JPA: ManyToMany bidir.

```
@Entity
public class Person
    implements Serializable {
    ...
    private Set<Department>
        depts = new
            HashSet<Department>();

    @ManyToMany
    @JoinTable(name="per_dept",
        joinColumns=
            @JoinColumn(name="deptid"),
        inverseJoinColumns=
            @JoinColumn(name="perid"))
    public Set<Department> getDepts(){
        return depts;
    }
}
```

```
@Entity
public class Department
    implements Serializable {
    ...
    private Set<Person> persons =
        new HashSet<Person>();

    @ManyToMany(mappedBy="depts")
    public Set<Person> getPersons() {
        return persons;
    }
}
```

Mapping samples



Grails Group Switzerland

GORM: ManyToMany bidir.

```
class Person {  
    ...  
    static hasMany =  
        [deptAssigns:DeptAssign]  
}
```

```
class Department {  
    ...  
    static hasMany =  
        [deptAssigns:DeptAssign]  
}
```

Mapping samples



Grails Group Switzerland

GORM: ManyToMany bidir.

```
class DeptAssign {
    Person person
    Department department

    static DeptAssign link(person, department) {
        def da = DeptAssign.findByPersonAndDepartment (
            person, department)
        if (!da) {
            da = new DeptAssign()
            person?.addToDeptAssigns(da)
            department?.addToDeptAssigns(da)
            da.save()
        }
        return da
    }
    ...
}
```

Mapping samples



Grails Group Switzerland

GORM: ManyToMany bidir.

```
...

static void unlink(person, department) {
    def da = DeptAssign.findByPersonAndDepartment(
        person, department)
    if (da) {
        person?.removeFromDeptAssigns(da)
        department?.removeFromDeptAssigns(da)
        da.delete()
    }
}
}
```

Mapping samples



Grails Group Switzerland

GORM: ManyToMany bidir. -> shorter solution

```
class DeptAssign {
  Person person
  Department department

  static mapping = {
    table 'person_department'
    version false
    id composite: ['person', 'department']
  }

  static DeptAssign create(Person person, Department
    department, boolean flush = false) {
    def da =
      new DeptAssign(person:person, department:department)
    da.save(flush:flush, insert:true)
    return da
  }
}
```

Mapping samples



Grails Group Switzerland

GORM: ManyToMany bidir. -> shorter solution

```
class Person {
    String name
    Set<Department> getDepartments() {
        DeptAssign.findAllByPerson(this).collect
            {it.department} as Set
    }
}

class Department {
    String name
    Set<Person> getPersons() {
        DeptAssign.findAllByDepartment(this).collect
            {it.person} as Set
    }
}
```

Mapping samples



Grails Group Switzerland

GORM: ManyToMany bidir. -> use of DeptAssign

```
class PersonController {
  def createDeptAssign = {
    def person = Person.get(params.perId)
    def department = Department.get(params.deptId)
    DeptAssign.create person, department
  }
}
```

Think about Sets



- hasMany uses a Set-Implementation
 - `org.hibernate.collection.PersistentSet`
- Adding a department to a person requires loading all instances of the persons departments (for uniqueness check) and all other persons who already belongs to that department
- This because Grails maps both collections for you and populates both. `person.addToDepartment` and `department.addToPersons` are equivalent because it's bidirectional
- Be careful with big amount of objects

See <http://burtbeckwith.com/blog/files/169/gorm%20grails%20meetup%20presentation.pdf>

Object handling



- Object oriented approach -> ability added to the class by runtime meta programming
- `DomainClass.get()`
- `DomainClass.read()`
- `DomainClass.findByPxAndPyOrPz(px,py,pz)`
- Sort by `static mapping` per `DomainClass` or relation attributes
- In Java? Have fun while implementing services

Java object handling



Grails Group Switzerland

```
public class PersonServiceImpl implements PersonService {

    EntityManager em;

    public Person getPersonById(long id) {
        return em.get(Person.class, id);
    }

    public List<Person> getPersonByNamePattern(String pa) {
        Query q = em.createQuery("from Person p where p.name
            like '%" + pa + "%'");
        return q.getResultList();
    }

}
```

Grails: use of dynamically added methods



Grails Group Switzerland

```
class PersonController {
  def filterList = {
    def persons = Person.findByNameLike (params.namepattern)
    [persList:persons]
  }

  def showPerson = {
    [person:Person.get (params.id) ]
  }

  def removePerson = {
    def person = Person.get (params.id)
    person.delete ()
  }
  ...
}
```

Grails: use of dynamically added methods



Grails Group Switzerland

```
class PersonController {
  def addToDepartment = {
    def person = Person.get(params.id)
    def dept = Department.get(params.deptId)
    dept.addToPersons(person)
    dept.save()
  }

  def removeFromDepartment = {
    def person = Person.get(params.id)
    def dept = Department.get(params.deptId)
    dept.removeFromPersons(person)
    dept.save()
  }
}
```

Summary



- JPA was a real benefit compared to a manual implementation of DAO-Pattern
- JPA mapping annotations are simple and easy to use, there remains a lot of overhead
- JPA object handling is inflexible.
- GORM mapping is more simple than JPA
- ManyToMany remains "complex"
- GORM object handling is just cool
- Be careful with big amount of objects in hasMany mappings