JPA 2 and Annotation Processing

Hardy Ferentschik, RedHat

About me

- Member of the Hibernate Team w/ focus on Validator and Search
- Over ten years experience in software development
  - ➡ Worked for small (10), medium (100) and big (3000+) companies
  - ➡ Have been everything from Team Lead to System Administrator
  - ➡ Developed in C++, perl, ... and of course Java
- Software Craftsman

What's new in JPA 2?

Standardized Properties

```
<persistence version="2.0">
    <persistence-unit name="default">
        <properties>
            <property name="hibernate.connection.driver_class"
                      value="org.h2.Driver"/>
            <property name="hibernate.connection.username"
                      value="foo"/>
            <property name="hibernate.connection.password"
                      value="bar"/>
            <property name="hibernate.connection.url"
                      value="jdbc:h2:mem:db1;DB_CLOSE_DELAY=-1;MVCC=TRUE"/>
            ...
        </properties>
        ...
    </persistence-unit>
</persistence>
```

JBoss Community

Standardized Properties

```
<persistence version="2.0">
    <persistence-unit name="default">
        <properties>
            <property name="javax.persistence.jdbc.driver"
                      value="org.h2.Driver"/>
            <property name="javax.persistence.jdbc.user"
                      value="foo"/>
            <property name="javax.persistence.jdbc.password"
                      value="bar"/>
            <property name="javax.persistence.jdbc.url"
                      value="jdbc:h2:mem:db1;DB_CLOSE_DELAY=-1;MVCC=TRUE"/>
            ...
        </properties>
        ...
    </persistence-unit>
</persistence>
```

JBoss Community

New mappings

JBoss Community

@OrderColumn

```
@Entity
public class PrintQueue {
   @Id
   private String name;

   @OneToMany
   @OrderColumn(name="PRINT_ORDER")
   private List<PrintJob> jobs;
   ...
}
```

```
@Entity
public class PrintJob {
   @Id
   private int id;


   @ManyToOne
   private PrintQueue queue;
   ...
}
```

JBoss Community

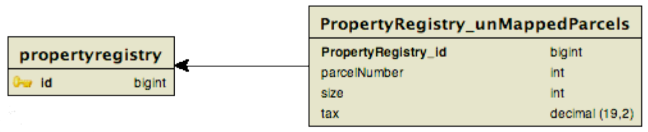
For a list of *n* elements, each element added required *n* additional SQL updates

Always turn on SQL debug log
and/or
use p6spy

@ElementCollection

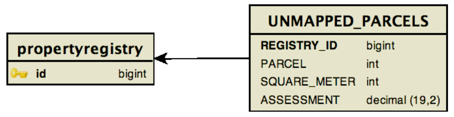
```
@Entity
public class PropertyRegistry {
   @Id @GeneratedValue
   public Long id;

   @ElementCollection
   public Set<PropertyInfo> unMappedParcels;
   ...
}
```

```
@Embeddable
public class PropertyInfo {
  public Integer parcelNumber;
  public Integer size;
  public BigDecimal tax;
  ...
}
```



@ElementCollection

```
@Entity
public class PropertyRegistry {
   @Id @GeneratedValue
   public Long id;

   @ElementCollection(targetClass = PropertyInfo.class)
   @CollectionTable(name = "UNMAPPED_PARCELS",
      joinColumns = @JoinColumn(name ="REGISTRY_ID"))
   @AttributeOverrides({
      @AttributeOverride(name = "parcelNumber",
         column = @Column(name = "PARCEL")),
      @AttributeOverride(name = "size",
         column = @Column(name = "SQUARE_METER")),
      @AttributeOverride(name = "tax",
         column = @Column(name = "ASSESSMENT"))
    public Set unMappedParcels;
   ...
})
```

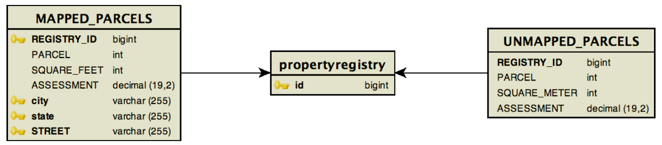


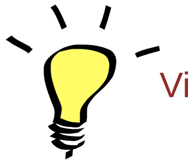
@ElementCollection

```
@Entity
public class PropertyRegistry {
   @Id @GeneratedValue
   public Long id;

   @ElementCollection
   @CollectionTable(name = "MAPPED_PARCELS",
      joinColumns = @JoinColumn(name = "REGISTRY_ID"))
   @AttributeOverrides({
      @AttributeOverride(name = "key.street",
         column = @Column(name = "STREET")),
      @AttributeOverride(name = "value.parcelNumber",
         column = @Column(name = "PARCEL")),
      @AttributeOverride(name = "value.size",
         column = @Column(name = "SQUARE_FEET")),
     @AttributeOverride(name = "value.tax",
         column = @Column(name = "ASSESSMENT"))
    })
   public Map<Address, PropertyInfo> parcels;
   ...
}
```

```
@Embeddable
public class Address {
  public String street;
  public String city;
  public String state;
  ...
}
```

@ElementCollection





Visualize your database structure,
e.g. with DbVisualizer

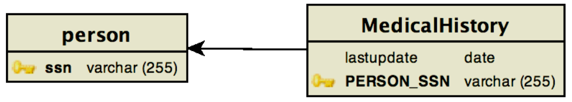
@AccessType

- Mix and match access modes in hierarchy and within single class

```
@Entity
@AccessType("field")
public class Furniture {
    @Id @GeneratedValue
    private Integer id;

    public long weight;

   @AccessType("property")
    public long getWeight() {
        convertWeight(weight);
    }

    public void setWeight(long weight) {
        this.weight = weight + 1;
    }
}
```

Derived Identifiers

When an identifier in one entity includes a foreign key to another entity, we call it a derived entity

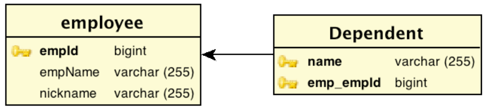


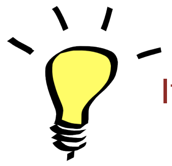
Derived Identifiers

```
@Entity
public class Employee {
   @Id
   long empId;
   String empName;
...
}
```

```
@Entity
@IdClass(DependentId.class)
public class Dependent {
   @Id String name;

   @Id @ManyToOne
   Employee emp;
...
}
```

```
public class DependentId {
   String name;
   long emp;
}
```





If an entity has multiple id attributes an IdClass must be used

New APIs

Cache API

- First attempt to specify second level cache API
- Result rudimentary set of cache control methods
- `EntityManagerFactory.getCache()`

```
public class Cache {
   public boolean contains(Class cls, Object pk);
   public void evict(Class cls, Object pk);
   public void evict(Class cls);
   public void evictAll();
}
```

Cache Configuration

- `javax.persistence.sharedCache.mode`
  - ➡ NOT_SPECIFIED
  - ➡ ALL
  - ➡ NONE
  - ➡ DISABLE_SELECTIVE
  - ➡ ENABLE_SELECTIVE
- Last two modes in conjunction with `@Cacheable`

New locking capabilities

- Many methods overloaded with new lock mode parameter, eg `find()`, `refresh()`, `lock()`
- `LockModeTypes`
  - ➡ OPTIMISTIC
  - ➡ OPTIMISTIC_FORCE_INCREMENT
  - ➡ PESSIMISTIC_READ
  - ➡ PESSIMISTIC_WRITE
  - ➡ PESSIMISTIC_FORCE_INCREMENT
- Additional properties `javax.persistence.lock.scope` and `javax.persistence.lock.timeout`

Locking example

```
@Transactional
public void updateEmployeeVacation(int id) {
   Employee emp = em.find(Employee.class, id);
   EmployeeStatus status = emp.getStatus();
   double earnedVacationDays = calcualteVacationDays(status);
   if(earnedVacationDays > 0) {
      em.lock(emp, LockModeType.PESSIMISTIC_WRITE);
      emp.setVacationDays(emp.getVacationDays() + earnedVacationDays);
   }
}
```

Locking example - improved

```
@Transactional
public void updateEmployeeVacation(int id) {
   Employee emp = em.find(Employee.class, id);
   EmployeeStatus status = emp.getStatus();
   double earnedVacationDays = calcualteVacationDays(status);
   if(earnedVacationDays > 0) {
      em.refresh(emp, LockModeType.PESSIMISTIC_WRITE);
      if(status != emp.getStatus())) {
        earnedVacationDays = calcualteVacationDays(emp.getStatus());
      }
      if(earnedVacationDays > 0) {
        emp.setVacationDays(emp.getVacationDays() + earnedVacationDays);
      }
   }
}
```

Criteria Query

Criteria API overview

- Most vendors already had OO query API. Just needed to find a standard
- `CriteriaQuery` objectification of JPQL
- Choose between string based and strongly typed approach
- Open the doors for dynamic query generation without string manipulation
- Entry point is the `QueryBuilder`

The Canonical Metamodel

```
@Entity
public class Item {
  @Id @GeneratedValue public Long getId() {}
  public Boolean isShipped() {}
  public String getName() {}
  public BigDecimal getPrice() {}
  @OneToMany public Map<String, Photo> getPhotos() {}
  @ManyToOne public Order getOrder() {}
  @ManyToOne public Product getProduct() {}
}

@StaticMetamodel(Item.class)
public class Item_ {
  public static SingularAttribute<Item, Long> id;
  public static SingularAttribute<Item, Boolean> shipped;
  public static SingularAttribute<Item, String> name;
  public static SingularAttribute<Item, BigDecimal> price;
  public static MapAttribute<Item, String, Photo> photos;
  public static SingularAttribute<Item, Order> order;
  public static SingularAttribute<Item, Product> product;
}
```

Typesafe Criteria query

```
CriteriaQuery<Vendor> q = cb.createQuery(Vendor.class);
Root<Employee> emp = q.from(Employee.class);
Join<ContactInfo, Phone> phone = emp
    .join(Employee_.contactInfo)
    .join(ContactInfo_.phones);
q.where(cb.equal(emp.get(Employee_.contactInfo)
                    .get(ContactInfo_.address)
                    .get(Address_.zipcode),
                 "95054"))
.select(phone.get(Phone_.vendor));
```

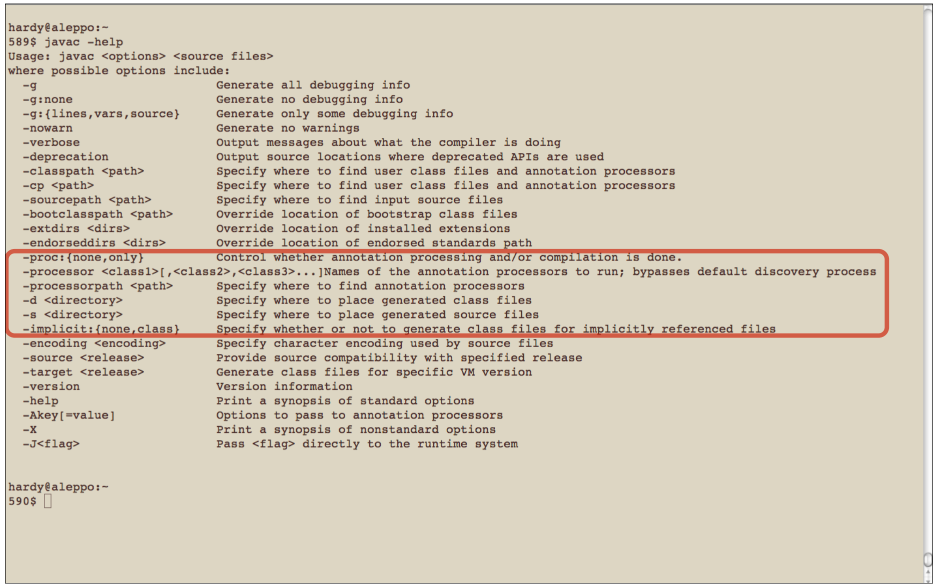
```
SELECT p.vendor
FROM Employee e JOIN e.contactInfo.phones p
WHERE e.contactInfo.address.zipcode = '95054'
```

JBoss Community

Pluggable Annotation Processing API (JSR 269)

- Successor of *apt* tool in JDK 5
- In JDK 6 command line options for *javac*
- Core packages in
  - javax.lang.model.*
  - javax.annotation.processing

JBoss Community

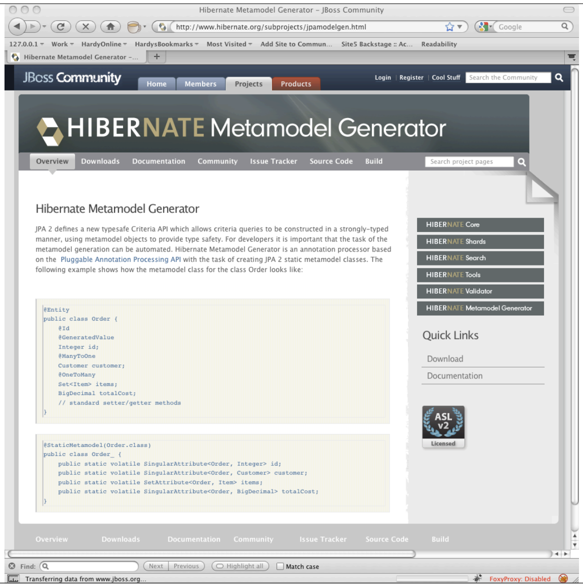


```
hardy@aleppo:~
589$ javac -help
Usage: javac <options> <source files>
where possible options include:
  -g                         Generate all debugging info
  -g:none                    Generate no debugging info
  -g:{lines,vars,source}     Generate only some debugging info
  -nowarn                    Generate no warnings
  -verbose                   Output messages about what the compiler is doing
  -deprecation               Output source locations where deprecated APIs are used
  -classpath <path>          Specify where to find user class files and annotation processors
  -cp <path>                 Specify where to find user class files and annotation processors
  -sourcepath <path>         Specify where to find input source files
  -bootclasspath <path>      Override location of bootstrap class files
  -extdirs <dirs>            Override location of installed extensions
  -endorseddirs <dirs>       Override location of endorsed standards path
  -proc:{none,only}          Control whether annotation processing and/or compilation is done.
  -processor <class1>[,<class2>,<class3>...]Names of the annotation processors to run; bypasses default discovery process
  -processorpath <path>      Specify where to find annotation processors
  -d <directory>             Specify where to place generated class files
  -s <directory>             Specify where to place generated source files
  -implicit:{none,class}     Specify whether or not to generate class files for implicitly referenced files
  -encoding <encoding>       Specify character encoding used by source files
  -source <release>          Provide source compatibility with specified release
  -target <release>          Generate class files for specific VM version
  -version                   Version information
  -help                      Print a synopsis of standard options
  -Akey[=value]              Options to pass to annotation processors
  -X                         Print a synopsis of nonstandard options
  -J<flag>                   Pass <flag> directly to the runtime system


hardy@aleppo:~
590$ 
```

Start with extending *AbstractProcessor*

```
@SupportedAnnotationTypes("javax.persistence.Entity")
public class JPAMetaModelEntityProcessor extends AbstractProcessor {

   public void init(ProcessingEnvironment env) {
      super.init( env );
        ...
   }

   @Override
   public boolean process(final Set<? extends TypeElement> annotations,
                          final RoundEnvironment roundEnvironment) {
        ...
   }
...
}
```

JBoss Community



Hibernate Metamodel Generator

JPA 2 defines a new typesafe Criteria API which allows criteria queries to be constructed in a strongly-typed manner, using metamodel objects to provide type safety. For developers it is important that the task of the metamodel generation can be automated. Hibernate Metamodel Generator is an annotation processor based on the Pluggable Annotation Processing API with the task of creating JPA 2 static metamodel classes. The following example shows how the metamodel class for the class Order looks like:

```
@Entity
public class Order {
    @Id
    @GeneratedValue
    Integer id;
    @ManyToOne
    Customer customer;
    @OneToMany
    Set<Item> items;
    BigDecimal totalCost;
    // standard setter/getter methods
}
```

```
@StaticMetamodel(Order.class)
public class Order_ {
    public static volatile SingularAttribute<Order, Integer> id;
    public static volatile SingularAttribute<Order, Customer> customer;
    public static volatile SetAttribute<Order, Item> items;
    public static volatile SingularAttribute<Order, BigDecimal> totalCost;
}
```

Transferring data from www.jboss.org...

Demo

JBoss Community

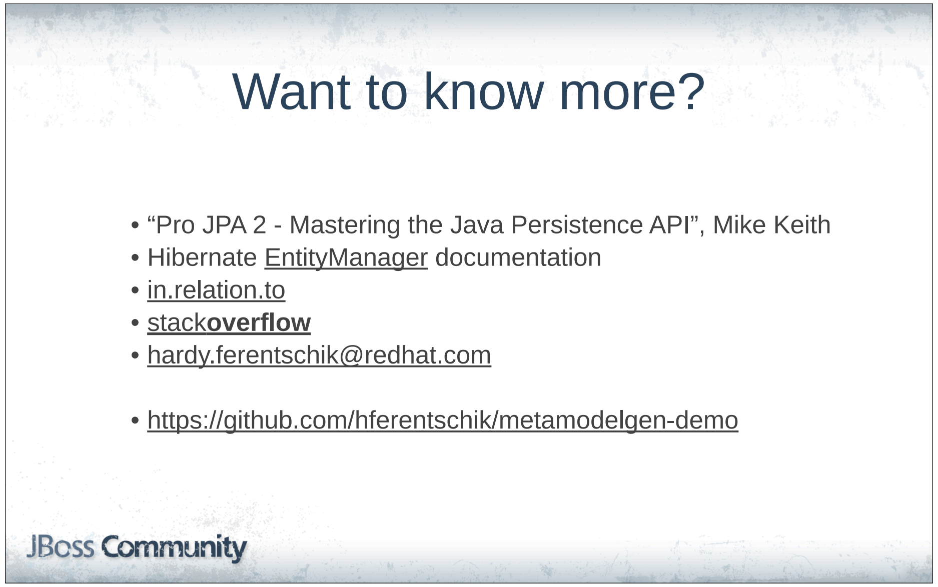
Problems!?

- Visitor/Mirror API needs to get used to
- Documentation is sparse
- Older IDEs don’t offer configuration options
- Maven integration - MCOMPILER-62, MCOMPILER-66
  ➡ You get it to work with additional maven plugins, eg maven-annotation-plugin

JBoss Community

Q + A

JBoss Community



Want to know more?

- “Pro JPA 2 - Mastering the Java Persistence API”, Mike Keith
- Hibernate EntityManager documentation
- in.relation.to
- stack**overflow**
- hardy.ferentschik@redhat.com

- https://github.com/hferentschik/metamodelgen-demo

JBoss Community