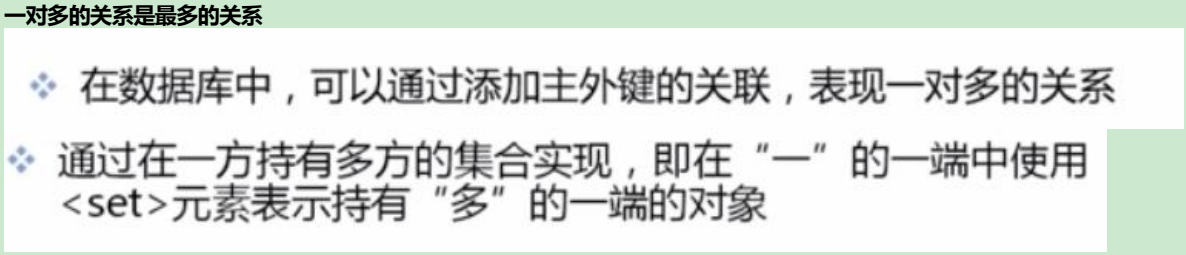
# Hibernate初探一对多映射

一对多 ( one to many )
多对一 ( many-to-one )
一对一 ( one-to-one )
多对多 ( many-to-many )

**主要还是**
**一对多 ( one to many )**
**多对一 ( many-to-one )**

**一对多的关系是最多的关系**




```
//读取hibernate.cfg.xml 文件，完成初始化
Configuration config = new Configuration().configure();

ServiceRegistryBuilder srb = new ServiceRegistryBuilder().applySettings(config.getProperties());

ServiceRegistry sr = srb.buildServiceRegistry();

sessionFactory = config.buildSessionFactory(sr);
```

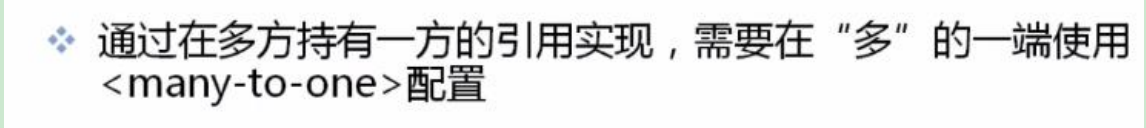
创建数据库文件：
语句格式：

```
create table grade
(
gid int primary key,
gname varchar(20) not null,
gdesc varchar(50));
```

**set元素的常用属性**

| 属性 | 含义和作用 | 必须 | 默认值 |
|---|---|---|---|
| name | 映射类属性的名称 | Y | |
| table | 关联类的目标数据库表 | N | |
| lazy | 指定关联对象是否使用延迟加载 | N | proxy |
| inverse | 标识双向关联中被动的一方 | N | false |

多对一的关系和关系数据库中的外键参照关系最匹配，即在己方的表中的一个外键参照另一个表的主键




单向多对一的配置：
1.在多方定义一放的引用

```
private Classes classes
```

get，set方法
2.配置映射文件
<many-to-one name="classes" class="com.pro.domain.Classes" column="gid"></many-to-one>

需要一个单独的属性作为外键，只要设置了这个外键，就可以达到外键连接的目的
**配置一对多关联关系**
<set></set>是配置一对多关联关系
<key></key>指关联的外键链
<one-to-many />是配置一对多关系
<set name="students" table="student">
<key column="gid"></key> column为数据库单独对应的列名
<one-to-many class="com.xxx.entity.Student"/>
session.save(g);
session.save(stu1);
session.save(stu2);
**key是指关联的外键链**
**配置多对一关联关系**
**<many-to-one></many-to-one>**
<many-to-one name="grade" class="com.xxx.entity.Grade" column="gid" cascade="all"></many-to-one> column为数据库单独对应的列名
stu1.setGrade(g);
stu2.setGrade(g);

session.save(g);
session.save(stu1);
session.save(stu2);

同时也需要在代码中添加的信息

**inverse属性**

- <set>节点的inverse属性指定关联关系的控制方向，默认由one方来维护
- 关联关系中，inverse=“false” 则为主动方，由主动方负责维护关联关系
- 在一对多关联中，只能设置one 方的inverse为true，这将有助于性能的改善

cascade属性也是在set标签中的，**代表的是级联属性**，例子：我们在班级中已经添加了学生，如果我们保存的时候，正常的说，会分别保存班级和学生，有了cascade这个属性，我们就可以只保存班级就好了，学生会**自动帮我们保存的**，他的属性有四个，all，savae-update,delete,none(默认);
**casccade属性的设置会带来性能上的变动，需谨慎设置**

| 属性值 | 含义和作用 |
| --- | --- |
| all | 对所有操作进行级联操作 |
| save-update | 执行保存和更新操作时进行级联操作 |
| delete | 执行删除操作时进行级联操作 |
| none | 对所有操作不进行级联操作 |

```
<!--配置单向的一对多的关联关系
    设置inverse对象为true 由多方维护关联关系
    当进行保存和更新操作时级联操作所关联的对象
-->
```

```
<set name="addressSet" table="address" inverse="true" cascade="all">
        <key column="id"></key><!--这是外键对应的-->
<one-to-many class="com.Hibernate.Dao.entity.Address"/>
    </set>
</class>
```

```
<!--配置多对一映射关系-->
<many-to-one name="addressAccount" class="com.Hibernate.Dao.entity.Account" column="id"  cascade="all" />
```

myeclipse与hibernate可以生成自动映射文件

多对一关联映射与一对一关联映射类似，只是在多对一的指向端可以存在多个对象，在指向端加载的时候，同时加载被指向端。
对比一对多关联映射和我们之前讲的多对一关联映射，可以发现两种映射原理是一致的，都是在多的一端加入一个外键，指向一的一端。它们的区别在于维护的关系不同，多对一维护的是多指向一的关系，有了此关系，在加载多的时候可以将一加载上来，一对多维护的是一指向多的关系，有了此关系，在加载一的时候可以将多加载上来。

关联关系的本质是将关联关系映射到数据库中。关联关系在对象模型中体现为内存中的一个或多个引用。 一对多关系： 一对多关系 分为“单向一对多 / 多对一关系”和“双向多对一”关系。
- “单向一对多 / 多对一关系”只需在“一” / “多”方进行配置，
- “双向多对一关系”需要在关联双方均加以配置。 双向多对一关联就是在多方和一方都进行配置，并在“一” 方通过属性inverse="true"设置控制关系反转

注：单向多对一关联是最常见的单向关联关系。
注：双向多对一关联是最常见的双向关联关系。双向多对一关联实际上是“多对
一”与“一对多”关联的组合。

多对一及一对多关联映射的区别（单向）：

不管多对一还是一对多，都是在"多"的一端添加一个外键指向"一"的一端，只不过是多对一是在多的一端为其自己添外键，而一对多则是在一的一端为多的一端添加外主键。

模型：一个用户组（group）对应多个用户（user）

多对一关联映射

多对一及一对多关联映射的区别（单向）：

不管多对一还是一对多，都是在"多"的一端添加一个外键指向"一"的一端，只不过是多对一是在多的一端为其自己添外键，而一对多则是在一的一端为多的一端添加外主键。

模型：一个用户组（group）对应多个用户（user）。

多对一关联映射：是在"多"的一端添加一个外键指向"一"的一端，它维护的关系为多到一的关系，如：当载入一个用户（user）时将会同时载入组（group）的信息。它的关系映射将写在多的一端（user）：

一对多关联映射

一对多关联映射：是在"多"的一端添加一个外键指向"一"的一端，它维护的关系为一到多的关系，如：当载入一个组（group）时将会同时载入此组用户（user）的信息。它的关系映射将写在一的一端

总之，一对多和多对一的映射策略是一致的，都是通过在"多"的一端添加一个外键指向"一"的一端，只是站的角度不同。

<hibernate-mapping package="【对应持久化类所在包名】" schema="【指定属于数据库某个（schema）模式下】" catalog="【指定属于某个（名）数据库下】" default-cascade="【默认的级联风格】" default-access="【默认是property（使用get,set访问属性）】" default-lazy="【默认开启延迟加载】" auto-import="【是否允许使用非全限定（类似具体路径）类名（如果2个持久化类映射，类名一样，处于不同包下，则设置false，否则无法识别具体）】"><class name="【持久化类名，】"（这里也可以指定schema,catalog,lazy相关属性来覆盖上级元素指定的默认行为） table="【持久化类映射的数据库

表明，默认以持久化类的类名作为表名】" discriminator-value="【继承映射关系方面的，子类标示符】" mutable="【该类的实例是否可变，默认为false,该类更新(update)会被忽略，只允许增加和删除】" proxy="【指定一个接口，在延迟装载时作为代理使用。 】" dynamic-update="【指定更新记录的update语句是否在运行时动态生成，只更新改变过的字段】" dynamic-insert="【指定用于INSERT的 SQL是否在运行时动态生成，只包含那些非空值字段】" select-before-update="【指定Hibernate除非确定对象真正被修改了（如果该值为true－译注），否则不会执行SQL UPDATE操作。在特定场合（实际上，它只在一个瞬时对象（transient object）关联到一个 新的session中时执行的update()中生效），这说明Hibernate会在UPDATE 之前执行一次额外的SQL SELECT操作，来决定是否应该执行 UPDATE。】" polymorphism="【implicit，界定是隐式还是显式的使用多态查询，当指定polymorphism为true(默认)时，如果查询的话，会返回该类及子类的实例】" where="【指定一个附加的SQLWHERE 条件， 在抓取这个类的对象时会一直增加这个条件】" persister="【指定一个定制的ClassPersister （暂不理解】" batch-size="【指定一个用于 根据标识符（identifier）抓取实例时使用的 batch size（批次抓取数量），打个比方 读取数据每次从数据库里取出多少条记录 】" optimistic-lock="version【乐观锁】，适合高并发，相比悲观锁，放松了加锁机制，读取出数据时，将此版本号一同读出，之后更新时，对此版本号加一。此时，将提交数据的版本数据与数据库表对应记录的当前版本信息进行比对，如果提交的数据版本号大于数据库表当前版本号，则予以更新，否则认为是过期数据。" check="【这是一个SQL表达式， 用于为自动生成的schema添加多行（multi-row）约束检查】" rowid="【Hibernate可以使用数据库支持的所谓的ROWIDs，例如： Oracle数据库，如果你设置这个可选的rowid， Hibernate可以使用额外的字段rowid实现快速更新。ROWID是这个功能实现的重点， 它代表了一个存储元组（tuple）的物理位置】" subselect="【它将一个不可变（immutable）并且只读的实体映射到一个数据库的 子查询中，变成持久化对象，它用于实现一个视图代替一张基本表】"> ..... </class></hibernate-mapping>