Hibernate性能调优

Robbin Fan

inverse = ?

JavaEye...

inverse=false(default)

- 用于单向one-to-many关联
- parent.getChildren().add(child) // insert child
- parent.getChildren().delete(child) // delete child

inverse=true

- 用于双向one-to-many关联
- child.setParent(parent); session.save(child) // insert child
- session.delete(child)

在分层结构的体系中

parentDao, childDao对于CRUD的封装导致往往直接通过session接口持久化对象，而很少通过关联对象可达性

one-to-many关系

单向关系还是双向关系？

- parent.getChildren().add(child)对集合的触及操作会导致lazy的集合初始化，在没有对集合配置二级缓存的情况下，应避免此类操作
- select * from child where parent_id = xxx;

性能口诀：

- 一般情况下避免使用单向关联，尽量使用双向关联
- 使用双向关联，inverse=“true”
- 在分层结构中通过DAO接口用session直接持久化对象，避免通过关联关系进行可达性持久化

many-to-one关系

单向many-to-one表达了外键存储方

灵活运用many-to-one可以避免一些不必要的性能问题

many-to-one表达的含义是：0..n : 1，many可以是0，可以是1，也可以是n，也就是说many-to-one可以表达一对多，一对一，多对一关系

因此可以配置双向many-to-one关系，例如：

- 一桌四人打麻将，麻将席位和打麻将的人是什么关系？是双向many-to-one的关系

one-to-one

通过主键进行关联

相当于把大表拆分为多个小表

例如把大字段单独拆分出来，以提高数据库操作的性能

Hibernate的one-to-one似乎无法lazy，必须通过bytecode enhancement

集合List/Bag/Set

one-to-many

- List需要维护index column，不能被用于双向关联，必须inverse=“false”，被谨慎的使用在某些稀有的场合
- Bag/Set语义上没有区别
- 我个人比较喜欢使用Bag

many-to-many

- Bag和Set语义有区别
- 建议使用Set

集合的过滤

- children = session.createFilter(parent.getChildren(), “where this.age > 5 and this.age < 10”).list()

针对一对多关联当中的集合元素非常庞大的情况，特别适合于庞大集合的分页：

- session.createFilter(parent.getChildren(), “”).setFirstResult(0).setMaxResults(10).list();

继承关系当中的隐式多态

HQL: from Object

- 将把所有数据库表全部查询出来
- polymorphism="implicit"(default)将当前对象，和对象所有继承子类全部一次性取出
- polymorphism="explicit"，只取出当前查询对象

Hibernate二级缓存

著名的n+1问题：from Child，然后在页面上面显示每个子类的父类信息，就会导致n条对parent表的查询：

- select * from parent where id = ?
- …
- select * from parent where id = ?

解决方案

- eager fetch
- 二级缓存

inverse和二级缓存的关系

当使用集合缓存的情况下：

- inverse=“false”，通过parent.getChildren()来操作，Hibernate维护集合缓存
- inverse=“true”，直接对child进行操作，未能维护集合缓存！导致缓存脏数据
- 双向关联，inverse=“true”的情况下应避免使用集合缓存

Hibernate二级缓存是提升web应用性能的法宝

OLTP类型的web应用，由于应用服务器端可以进行群集水平扩展，最终的系统瓶颈总是逃不开数据库访问；

哪个框架能够最大限度减少数据库访问，降低数据库访问压力，哪个框架提供的性能就更高；

针对数据库的缓存策略：

- 对象缓存：细颗粒度，针对表的记录级别，透明化访问，在不改变程序代码的情况下可以极大提升web应用的性能。对象缓存是ORM的制胜法宝。
- 对象缓存的优劣取决于框架实现的水平，Hibernate是目前已知对象缓存最强大的开源ORM
- 查询缓存：粗颗粒度，针对查询结果集，应用于数据实时化要求不高的场合

应用场合决定了系统架构

一、是否需要ORM

Hibernate or iBATIS?

二、采用ORM决定了数据库设计

Hibernate：

- 倾向于细颗粒度的设计，面向对象，将大表拆分为多个关联关系的小表，消除冗余column，通过二级缓存提升性能*（DBA比较忌讳关联关系的出现，但是ORM的缓存将突破关联关系的性能瓶颈）*；Hibernate的性能瓶颈不在于关联关系，而在于大表的操作

iBATIS：

- 倾向于粗颗粒度设计，面向关系，尽量把表合并，通过表column冗余，消除关联关系。无有效缓存手段。iBATIS的性能瓶颈不在于大表操作，而在于关联关系。

性能口诀

1、使用双向一对多关联，不使用单向一对多

2、灵活使用单向多对一关联

3、不用一对一，用多对一取代

4、配置对象缓存，不使用集合缓存

5、一对多集合使用Bag，多对多集合使用Set

6、继承类使用显式多态

7、表字段要少，表关联不要怕多，有二级缓存撑腰