河北师范大学软件学院
Software College of Hebei Normal University

HIBERNATE

第九讲
Hibernate高级配置

BIG DATA Java与大数据分析

Java与移动智能设备开发

知识回顾

- Hibernate检索方式
  - 根据OID检索、HQL、QBC、本地SQL
- Hibernate检索策略
  - 立即检索、延迟检索、迫切左外连接检索

目录

Hibernate二级缓存机制

- Session的缓存
  - 一块内存空间，其中存放了相互关联的Java对象。
  - 处在Session缓存中的对象被称为持久化对象。
- Session缓存是内置的不能被卸载的，被称为Hibernate的一级缓存
- SessionFactory有一个内置缓存（实现机制跟Session缓存类似）和一个可以配置的缓存插件被称为外置缓存

SessionFactory的缓存

- SessionFactory的外置缓存被称为Hibernate的二级缓存

内置缓存
• 存储Hibernate配置信息和映射元数据信息
• 物理介质是内存

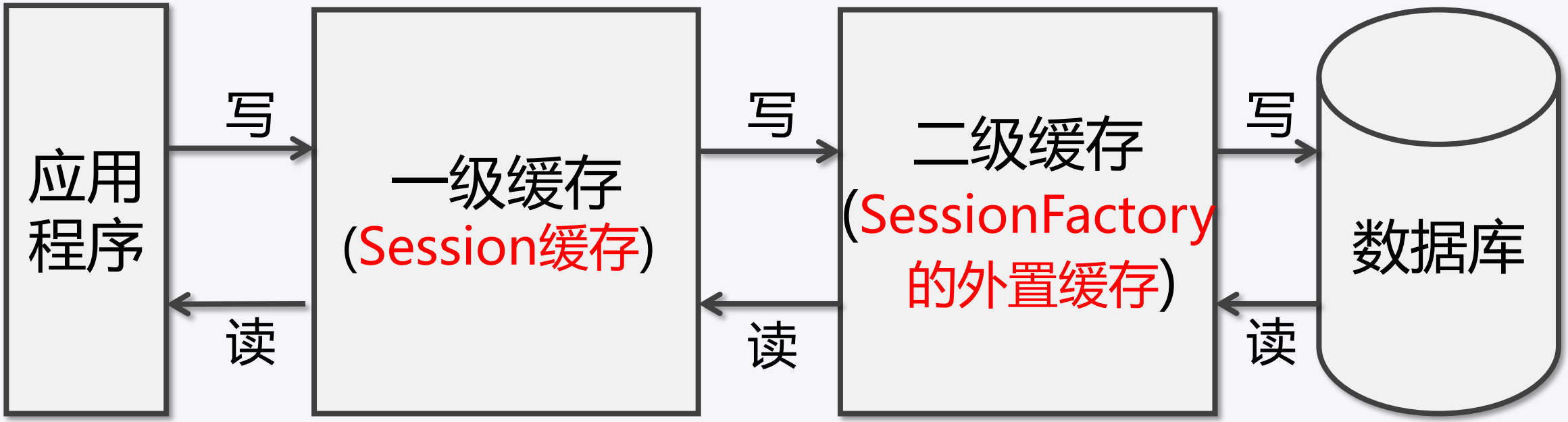
外置缓存
• 是一个可配置的缓存插件，可存放大量数据库数据的拷贝
• 物理介质是内存或硬盘

Hibernate二级缓存机制

- 位于持久化层，存放数据库数据的拷贝的缓存被称为持久化层缓存
  - Hibernate的一级缓存、二级缓存都是持久化层缓存
- 持久化层缓存的分类
  - 事务范围缓存
  - 进程范围缓存
  - 群集范围缓存
- 缓存的范围决定了缓存的生命周期以及缓存可以被谁访问

Hibernate二级缓存机制

- 持久化层的二级缓存，在查询时先在事务缓存中查找，如果没有查询到相应数据，再到进程范围或集群范围缓存中查找，如果还没找到再数据库中查找。



Hibernate二级缓存机制

- Session缓存是事务范围的缓存。
- SessionFactory缓存是进程范围或者集群范围的缓存。
- 进程范围和集群范围的缓存可能被进程内的多个事务并发访问，因此需要采取必要的事务隔离机制。

持久化层缓存并发访问策略

- 事务型
  - 提供Repeatable Read事务隔离级别，适用于经常被修改的数据。
- 读写型
  - 提供Read Commited事务隔离级别，只适用于进程范围缓存中经常被读，但很少被修改的数据。

持久化层缓存并发访问策略

- 非严格读写型
  - 不保证缓存中的数据与数据库中数据的一致性，可能出现脏读。
  - 适用于不要求准确性的数据读取。
- 只读型
  - 对于从来不被修改的数据可以使用此策略。
  - 适用于只读型数据。

二级缓存适合存储的数据

- 很少被修改的数据
- 不是很重要的数据，允许出现偶尔的并发问题
- 不会被并发访问的数据
- 参考数据

目录

Hibernate第二级缓存配置

- Hibernate的第二级缓存是可配置的缓存插件，允许选用以下类型的缓存插件，表中给出了各个缓存插件支持的并发访问策略。

缓存插件	只读型	非严格读写型	读写型	事务型
EHCache	支持	支持	支持	不支持
OSCache	支持	支持	支持	不支持
SwarmCache	支持	支持	不支持	不支持
JBossCache	支持	不支持	不支持	支持

Hibernate第二级缓存配置

- 选择EHCache缓存插件的配置步骤如下：
- 1.在Hibernate.cfg.xml配置文件中配置
  - cache.use_second_level_cache设为true，打开二级缓存。
  - cache.region.factory_class配置缓存插件提供商。

```
<property name="cache.use_second_level_cache">true
    </property>
<property name="cache.region.factory_class">
org.hibernate.cache.ehcache.EhCacheRegionFactory</property>
```

Hibernate第二级缓存配置

- 2.设置实体映射配置文件中<class>或<set>元素的<cache>子元素的usage属性
  - transactional：一般缓存插件没有此策略（除jboss-cache）
  - nonstrict-read-write：不严格的读写（如：帖子的回帖量）
  - read-write：严格的读写（如：银行系统数据）
  - read-only ： 只读策略（对象不能修改，修改会报异常），效率最高

```
<class name="User">
    <cache usage="read-write"/>
    ……
</class>
```

Hibernate第二级缓存配置

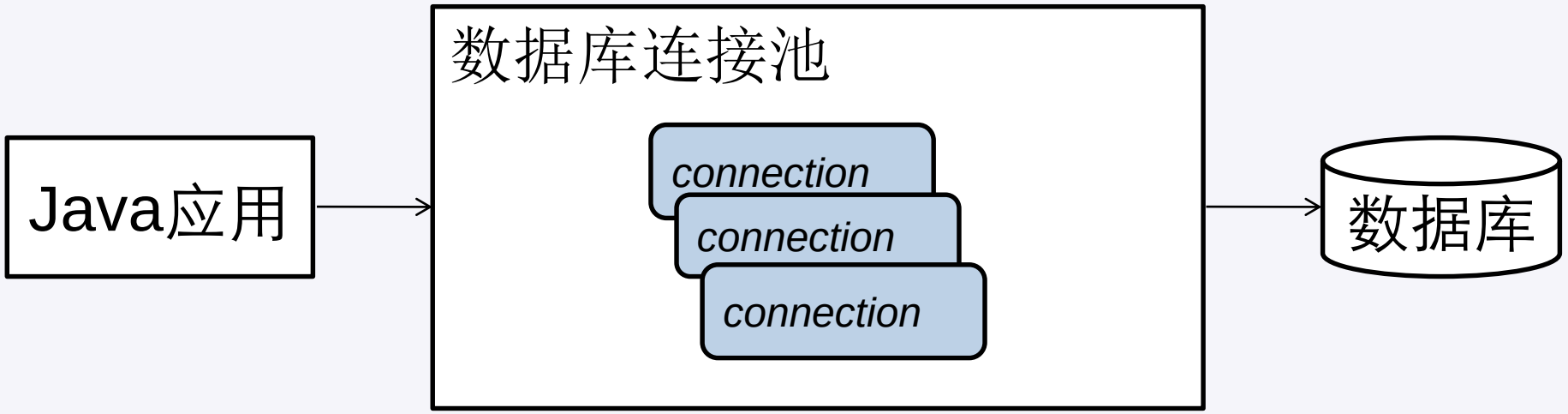
- 3.引入二级缓存插件jar包
- 4.编写二级缓存插件的配置文件ehcache.xml

```
<ehcache>
    <diskStore path="d:/cache/"/>
    <defaultCache
        maxElementsInMemory="10"
        eternal="false"
        timeToIdleSeconds="120"
        timeToLiveSeconds="120"
        overflowToDisk="true"/>
</ehcache>
```

目录

Java应用程序访问数据库

- Java应用程序最终通过JDBC API访问数据库
  - 执行数据库事务时获取一个JDBC Connection ，执行完事务关闭Connection，频繁的创建数据库连接消耗大量的系统资源，降低系统的性能。
  - 为了解决以上问题，采用数据库连接池技术。



数据库连接池的实现

- 一种是从头自己实现
- 另一种是使用第三方提供的连接池产品

名称	供应商
C3P0	开源软件
DBCP	Jakarta
Proxool	开源软件
Poolman	开源软件
Expresso	Jcorporate
JDBCPool	开源软件

Hibernate数据库连接池配置

- 在Hibernate的Java应用中，Java应用不会直接访问数据库连接池，而是通过Hibernate访问数据库连接池
- Hibernate获取数据库连接池的几种方式
  - 使用默认的数据库连接池
  - 使用配置文件指定的数据库连接池
  - 在受管环境中，从容器中获得标准的数据源

使用默认的数据库连接池

- Hibernate提供了默认的连接池实现，它的实现类为DriverManagerConnectionProvider
- 在Hibernate配置文件中如果没有明确配置任何连接池，会使用默认的连接池

Hibernate默认数据库连接池缺陷

- Hibernate内置的数据库连接池性能不佳，且存在诸多BUG
- 官方也只是建议仅在开发环境下使用

使用配置文件指定数据库连接池

- Hibernate中配置c3p0数据库连接池，并添加相应的jar包

```
<property name="hibernate.c3p0.max_size">2</property>
<property name="hibernate.c3p0.min_size">2</property>
<property name="hibernate.c3p0.timeout">120</property>
<property name="hibernate.c3p0.max_statements">100</property>
<property
name="hibernate.c3p0.idle_test_period">3000</property>
<property name="hibernate.c3p0.acquire_increment">2</property>
<property name="hibernate.c3p0.validate">false</property>
<property
name="hibernate.connection.provider_class">org.hibernate.connec
tion.C3P0ConnectionProvider</property>
```

本章小结

- Hibernate二级缓存机制
- Hibernate第二级缓存配置
- Hibernate数据库连接池配置

THANK YOU