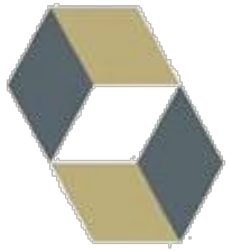


HIBERNATE

Trainer: Nilesh Ghule

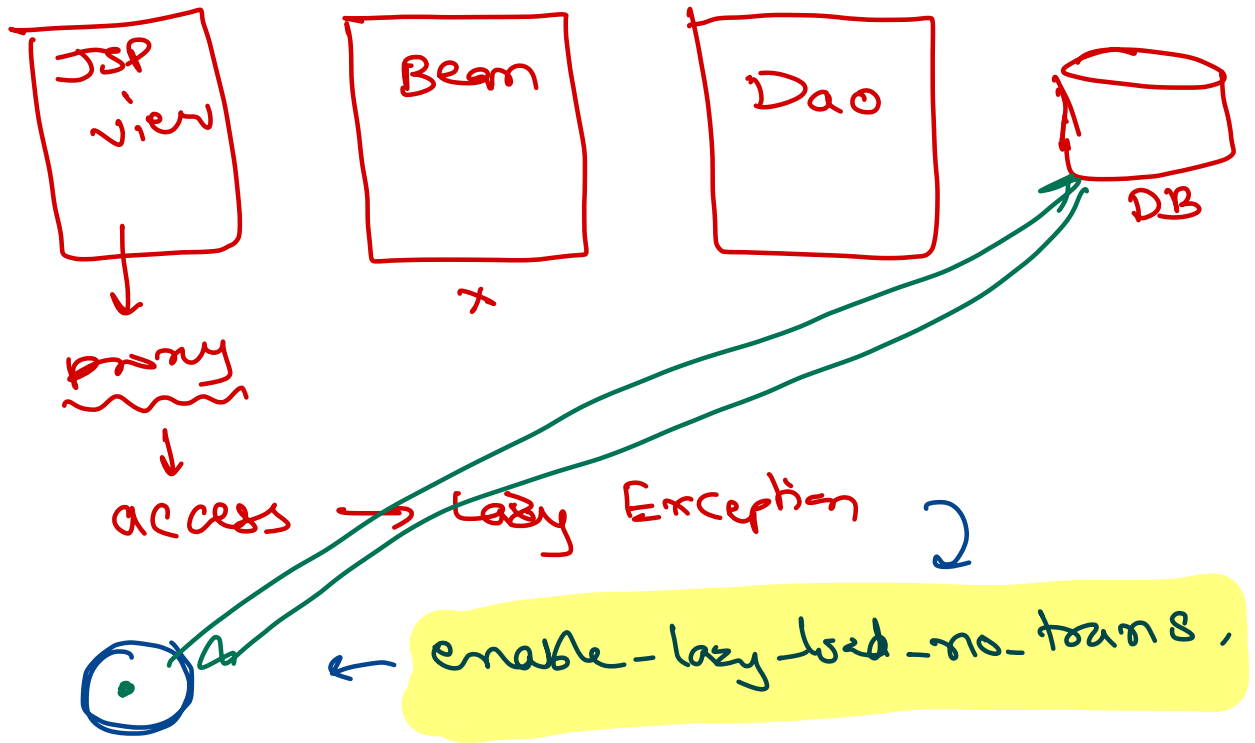
Wake up from Hibernate, Spring up!!!

Agenda

- Using Hibernate in Java EE application
  - OpenSessionInView
- N+1 problem
- JNDI connection pool
- Hibernate caching
  - SessionFactory cache
- JPA
  - EntityManager and EntityManagerFactory
  - Entity life cycle
- Criteria Queries

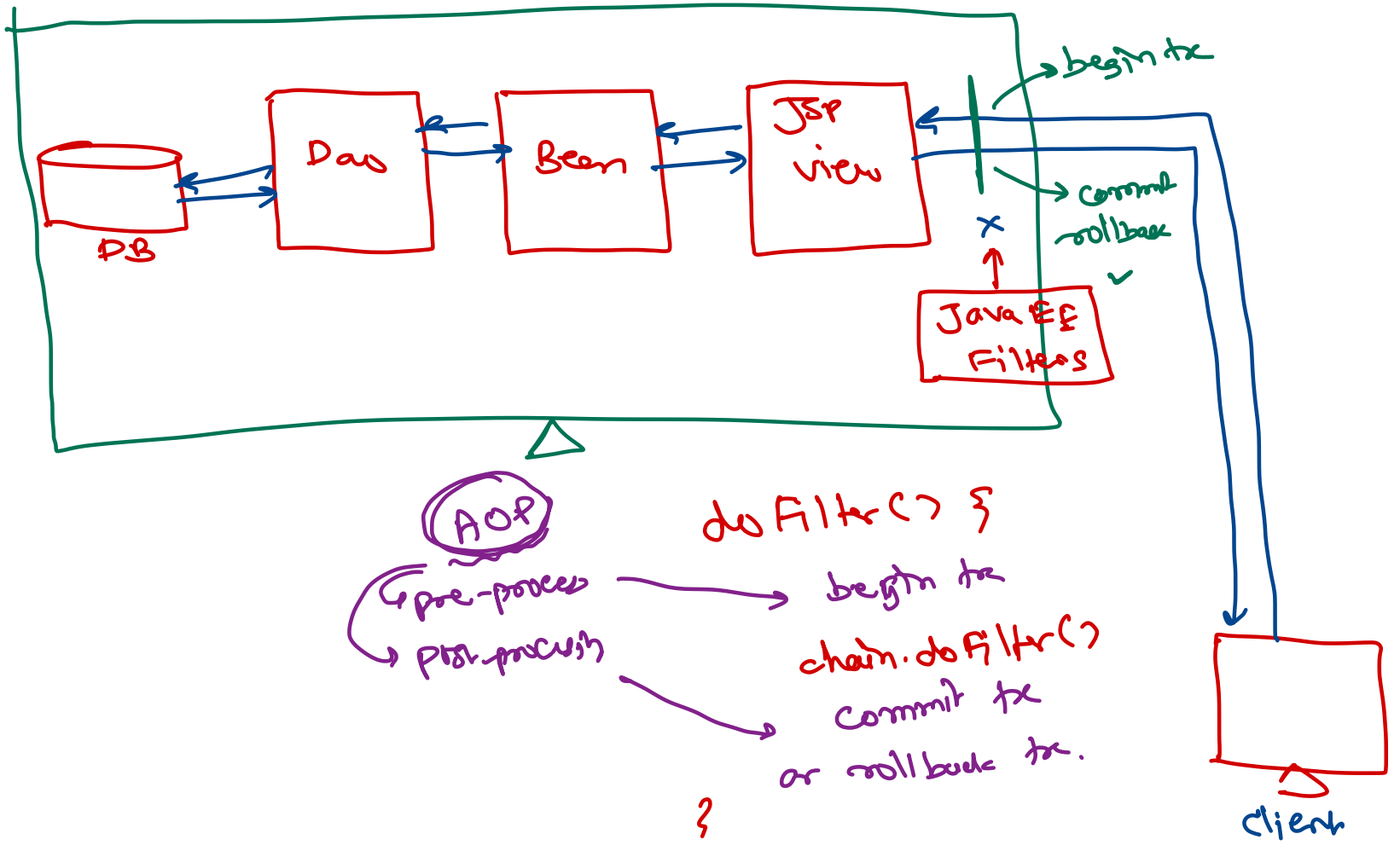
Using hibernate in Java EE application

- While dealing with single database, whole Java EE application should have single SessionFactory instance.
- This can be created as Singleton object in ServletContext.
- For each request new thread gets created.
- Hibernate session can be attached to thread session context, so that it can be accessed from any component.
- ORM is done using annotations on Entity objects and DAO layer is to be written using Hibernate (instead of JDBC).



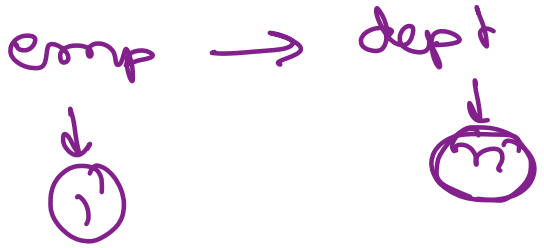
OpenSessionInView

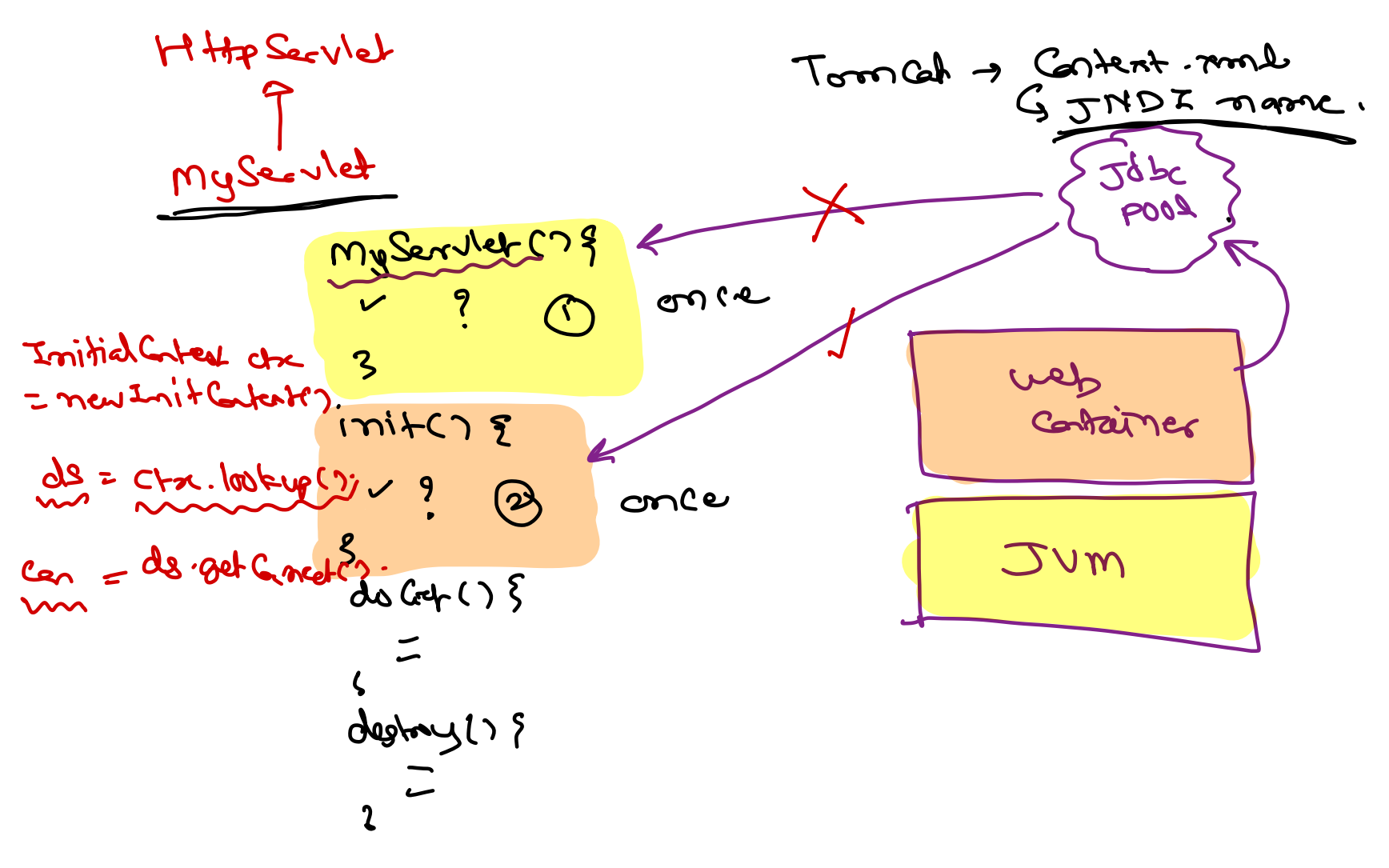
- Usually Tx is closed in dao/bean layer.
- If entity contains some lazily loaded collections and they are accessed from view (JSP page), then application fails with LazyInitializationExcpetion; because no active tx is present in view.
- To handle this session can be opened from view itself. This is done using
  - <property name="hibernate.enable_lazy_load_no_trans"> true </property>
- This is called as "OpenSessionInView" pattern.
- But now no clear separation in dao layer and view layer. View directly request to database. If it fails, exception cannot be handled. This is not recommended.


DB
Dao
Bean
JSP
view
begin tx
commit
rollback
x
Java EE
Filters
AOP
pre-process
post-process
doFilter() {
begin tx
chain.doFilter()
commit tx
or rollback tx.
}
client

N+1 Problem

- While fetching n records from a table, due to ManyToOne or OneToOne relation (eager fetch) parent entity will be fetched n times. This is referred as N+1 problem.
  - select e from Emp e;
- This problem can be handled by specifying fetch mode in the query.
  - select e from Emp e join fetch e.dept;
- N+1 problem is also observed when session factory cache is disabled, but query cache is enabled. In this case query cache save only ids and for each id data is re-fetched from database.




HttpServlet
MyServlet
Tomcat → Context.xml
↳ JNDI name.
Jdbc pool
MyServlet() {
✓ ? ①
3
once
InitialContext ctx
= new InitContext();
ds = ctx.lookup();
init() {
✓ ? ②
}
once
Con = ds.getConnection();
doGet() {
=
}
destroy() {
=
}
2
web container
JVM

JNDI

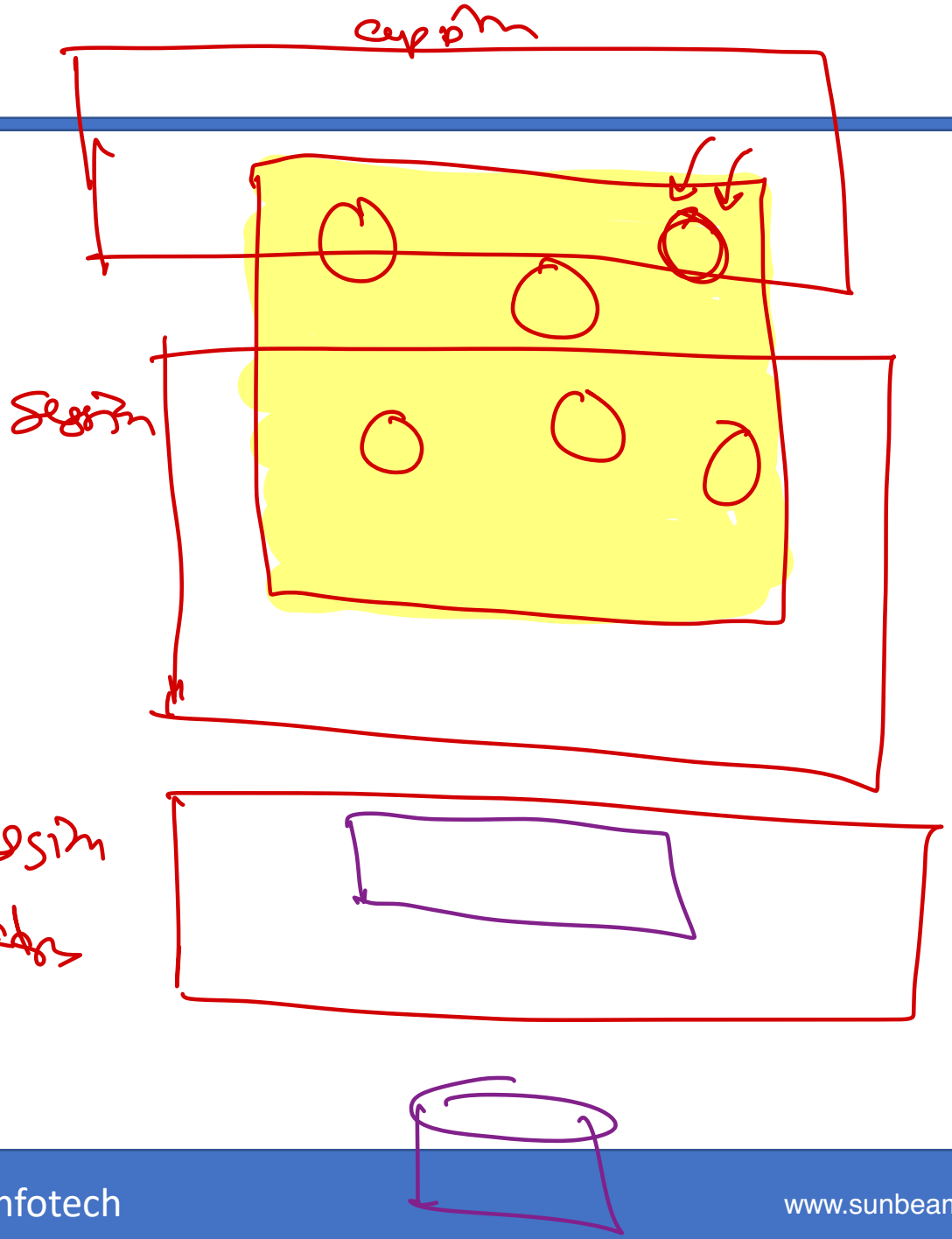
- JNDI is Java API used to discover and lookup resources by name in form of Java object.
- JNDI is independent of underlying implementation.
- JNDI is commonly used for RMI, JDBC (Web server), EJB and JMS (App server).
- JNDI names are in hierarchical fashion.
- Name is bound and looked up in some context.
- Typical example is web server bind some name with the JDBC connection pool and it can be looked up from the web applications.

Hibernate caching

- Hibernate caches are used to speed up execution of the program by storing data (objects) in memory and hence save time to fetch it from database repeatedly.
- There are two caches
  - Session cache (L1 cache)
  - SessionFactory cache (L2 cache)

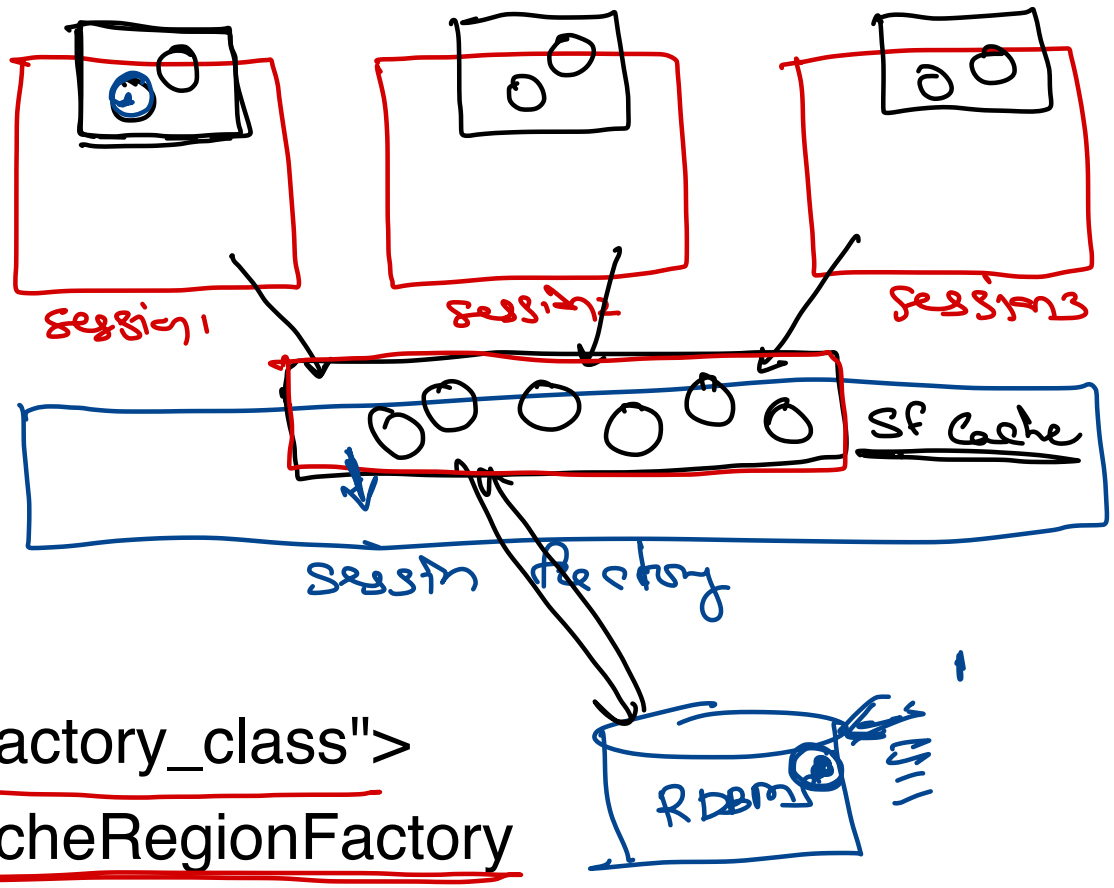
Hibernate session cache

- Collection of entities per session – persistent objects.
- Hibernate keep track of state of entity objects and update into database.
- Session cache cannot be disabled.
- If object is present in session cache, it is not searched into session factory cache or database.
- Use refresh() to re-select data from the database forcibly.



Hibernate session factory cache

- Collection of entities per session factory (usually single in appln).
- By default disabled, but can be enabled and configured into hibernate.cfg.xml
- Need to add respective additional second cache jars in project and optional cache config file (e.g. ehcache.xml).




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

↳ SF Cache manager

Hibernate session factory cache

- Four types of second level caches: EHCache, Swarm Cache, OS Cache, Tree Cache
- Use @Cache on entity class to cache its objects.
- Decide cache policy for those object and specify into its usage attribute: READ_ONLY, READ_WRITE, NONSTRICT_READWRITE, TRANSACTIONAL
- Stores the objects into the map (with its id as key). So lookup by key is very fast.

Hibernate session factory cache

- Cache Strategies (enum CacheConcurrencyStrategy)
  - READ_ONLY: Used only for entities that never change (exception is thrown if an attempt to update such an entity is made). It is very simple and performant. Very suitable for some static reference data that don't change.
  - NONSTRICT_READ_WRITE: Cache is updated after a transaction that changed the affected data has been committed. Thus, strong consistency is not guaranteed and there is a small time window in which stale data may be obtained from cache. This kind of strategy is suitable for use cases that can tolerate eventual consistency.
  - READ_WRITE: This strategy guarantees strong consistency which it achieves by using 'soft' locks: When a cached entity is updated, a soft lock is stored in the cache for that entity as well, which is released after the transaction is committed. All concurrent transactions that access soft-locked entries will fetch the corresponding data directly from database
  - TRANSACTIONAL: Cache changes are done in distributed XA transactions. A change in a cached entity is either committed or rolled back in both database and cache in the same XA transaction.

JTA

Hibernate session factory cache

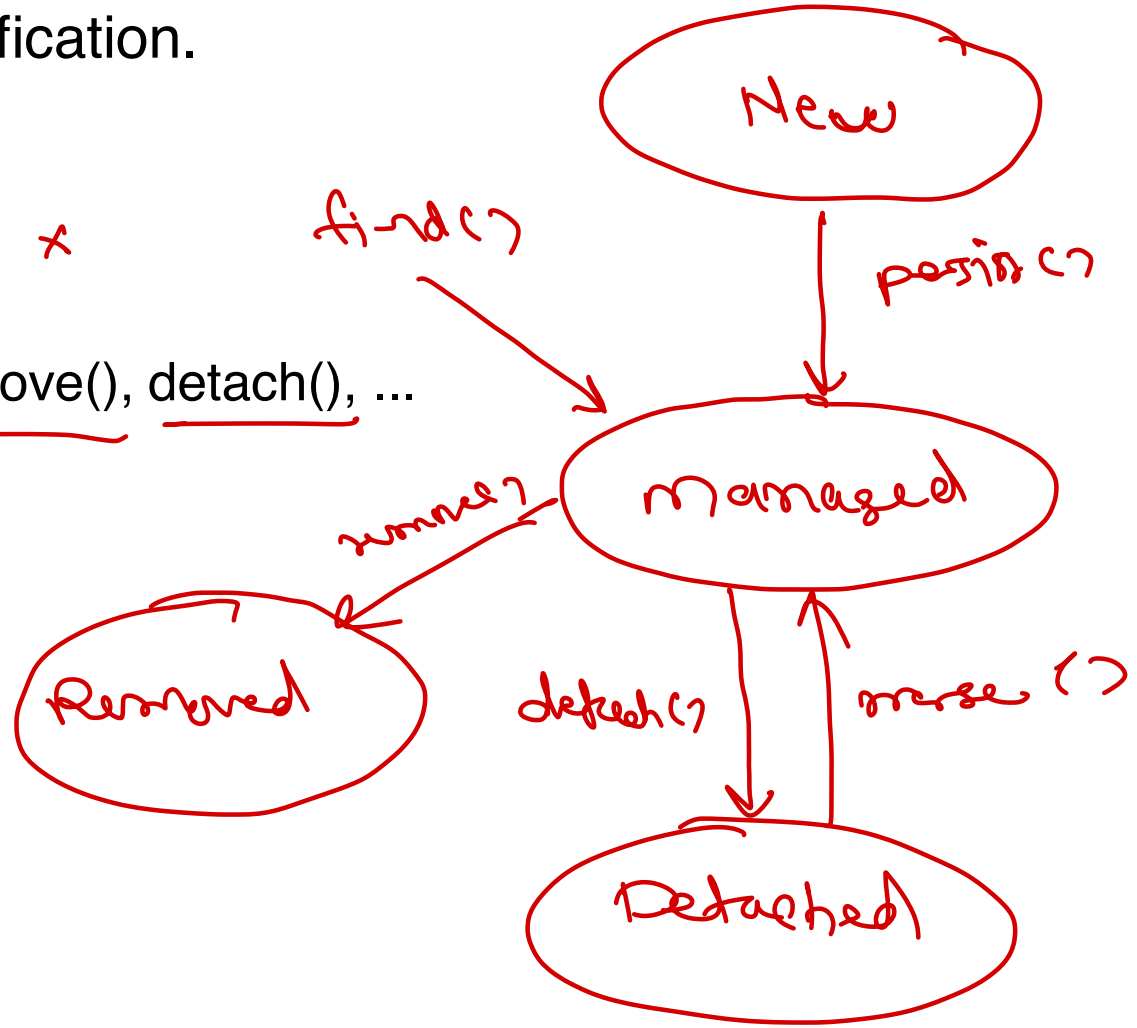
- Cache Types
  - EHCache: It can cache in memory or on disk and clustered caching and it supports the optional Hibernate query result cache.
  - Swarm Cache: A cluster cache based on JGroups. It uses clustered invalidation, but doesn't support the Hibernate query cache.
  - OSCache (Open Symphony Cache): Supports caching to memory and disk in a single JVM with a rich set of expiration policies and query cache support.
  - JBoss Tree Cache: A fully transactional replicated clustered cache also based on the JGroups multicast library. It supports replication or invalidation, synchronous or asynchronous communication, and optimistic and pessimistic locking.
  - Note that not all cache support all strategies.

Hibernate session factory cache

Cache	Read Only	NonStrict Read-Write	Read-Write	Transactional
EH Cache	Yes	Yes	Yes	No
OS Cache	Yes	Yes	Yes	No
Swarm Cache	Yes	Yes	No	No
Jboss Cache	Yes	No	No	Yes

JPA

- JPA is specification for ORM.
- All ORM implementations follow JPA specification.
  - Hibernate, Torque, iBatis, EclipseLink, ...
- Hibernate implements JPA specs.
  - SessionFactory → EntityManagerFactory
  - Session → EntityManager
    - find(), persist(), merge(), refresh(), remove(), detach(), ...
  - HQL → JPQL
  - hibernate.cfg.xml → persistence.xml
- JPA versions
  - 1.0, 1.1, 2.0, 2.1, 2.2
- JPA Entity life cycle
  - New (Transient)
  - Managed (Persistent)
  - Detached (Detached)
  - Removed (Removed)



JPA

- Hibernate Criteria
  - Only for SELECT operation
  - Deprecated
- Hibernate Query
  - HQL query
  - Parsed & converted to SQL
- CriteriaQuery
  - JPA complaint
  - Use builder design pattern
  - For SELECT operation
- CriteriaUpdate for update operation
- CriteriaDelete for delete operation

- CriteriaQuery

```
CriteriaBuilder builder = em.getCriteriaBuilder();
CriteriaQuery<Book> crQuery = builder.createQuery(Book.class);
Root<Book> table = crQuery.from(Book.class);
 crQuery .select(table)
     .where(builder.equal(table.get("author"), author));
TypedQuery<Book> q = em.createQuery(crQuery);
list = q.getResultList();
```

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>