Managing State

Lessons Learned

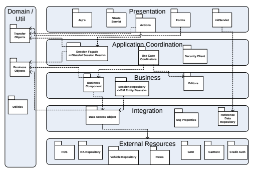
Agenda

- Introduction to the Architecture
- Where it didn't work and why
  - ✓ Putting State in Http Session
  - ✓ Putting State in Stateful Session Beans
  - ✓ Putting State in Entity Beans
- Putting State in JBoss Cache
  - ✓ What is JBoss Cache
  - ✓ Read-only Data using Hibernate 2nd level cache
  - ✓ How we managed Read/Write Data
  - ✓ Clustering and Failover
  - ✓ Using Thread Local Variables
- Why Seam solves all these problems more effectively

The Architecture

The Architecture

- This is a complex system
- A Use Case can interact with 14 external resources.
- Communication with external resources use JMS.
- Caching this information over multiple user interactions are required for system to scale and perform.

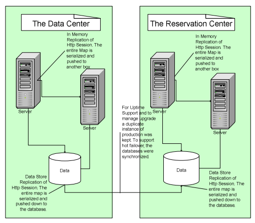


- 5+1 layered Architecture
  - ✓ 16 applications sharing single sign on and common look and feel
  - ✓ Foundational architecture and reusable components for J2EE applications in enterprise

Where it didn't work and why

HttpSession

- Http Session is copied.
  - ✓ Each Copy uses System resources.
  - ✓ The size of Http Session effects systems scalability
- JBoss TreeCache AOP can manage the size issue by replicating only the parts of the tree that are modified.


The Data Center
In Memory Replication of Http Session. The entire Map is serialized and pushed to another box
Server
Server
Data
Data Store Replication of Http Session. The entire map is serialized and pushed down to the database.
For Uptime Support and to manage upgrade a duplicate instance of production was kept. To support hot failover, the databases were synchronized.
The Reservation Center
In Memory Replication of Http Session. The entire Map is serialized and pushed to another box
Server
Server
Data
Data Store Replication of Http Session. The entire map is serialized and pushed down to the database.

Stateful Session Beans

- We attempted to use Stateful Session Beans.
  - ✓ Major Bugs
  - ✓ Not Supported or similar problems as HttpSession.
  - ✓ TreeCache AOP can be used to help manage state in Stateful Session Beans.
- Use Combination of Entity Beans and Stateful Session Beans.
  - ✓ Map stored in Session and passed to Components
  - ✓ Map contained handles or primary keys of Entity Beans

Map and Entity Beans Problems

- Passing Map along with the SessionKey and the Security Object reduced method cohesiveness.
- Map had no controls
  - ✓ Anyone could put anything in it, easily overwriting data put in by another component.
  - ✓ Could get cluttered and no way to clean it.
  - ✓ No way to track what was still needed and what was not.
  - ✓ A potential source of hard to track bugs.
- Couldn't share information between Session Beans.
- Our first app server didn't support clustered Stateful Session Beans.
- Entity Beans quickly became our number one problem with performance and scaling.

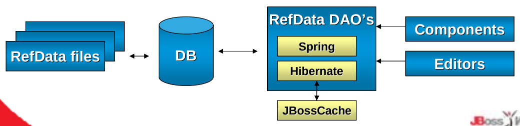
Putting State in JBoss Cache

Using JBoss Cache

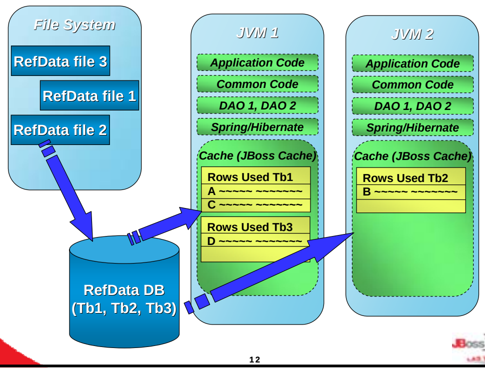
- The plan was to switch over to JBoss Cache in 4 Key areas.
  1. Replace the mostly read only reference data first
  2. Replace Entity Beans
     - ✓ Use first ones as examples
     - ✓ Have team complete the replacement as time allowed the replacement.
  3. Implement security Session caching
  4. Implement a combination of Aspects and Caching to inject stateful objects (eliminate Map)

1. RefData: The stack

- Files (for initial load and updates)
- DB2
- Hibernate
- JBossCache
- Spring

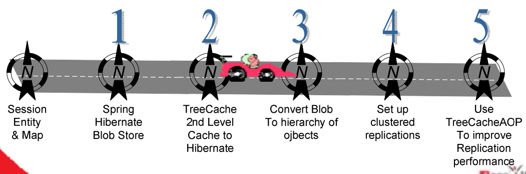


1. RefData: JVM level JBossCache



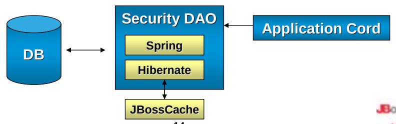
2. Replace Entity Beans

1. Replace Entity with Spring Hibernate/ store object hierarchy as serialized blob in DB
2. Add TreeCache as 2nd level cache to improve performance and manage memory usage
3. Convert blob to hierarchy of objects and tables
4. Setup Clustered replication between centers
5. Use TreeCache AOP to improve performance


1
2
3
4
5
Session Entity & Map
Spring Hibernate Blob Store
TreeCache 2nd Level Cache to Hibernate
Convert Blob To hierarchy of ojbects
Set up clustered replications
Use TreeCacheAOP To improve Replication performance

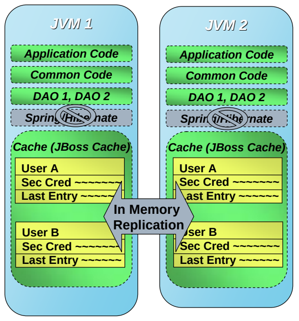
3. Security

- Every User Interaction with the system requires an update to the security DAO for timeout purposes.
  - ✓ When we initialized a new security system using the solution below, there were only 25 connections in our connection pool.
  - ✓ With an average of 1300 concurrent users, this quickly became a bottleneck that brought the system crashing down.
  - ✓ The number of connections were increased and the system now performs adequately.
- When the data is mostly read only we see a significant improvement in performance and scalability.
- When the data has significant updating, going to the database becomes a bottle neck.



3. Security

- Increasing the DB connection in the pool was short term.
- The best solution was to remove database updating every update
- Use in-memory replication to support failover and single sign on.



4. Injection of Stateful Objects

- Passing Map along with Session Key and Security Credentials around.
- Wrap our Application Coordinators (Stateful Session Beans) with an Aspect.
- This aspect identifies annotated objects in the Stateful Session Bean and inserts them as thread local variables before each method.
- These Objects are then injected into objects needing these variables where annotated.
- An aspect is written to look for objects with the annotation and wraps the methods to take the variables off the thread and put them in the object.
- Once the application Cord method is complete then the aspect will make sure the object on the Bean is updated with any changes.

JBoss World 2006 LAS VEGAS

Why Seam is more elegant

Seam Intro

```
@Name( "completedApplications" )
@Stateful
@Scope( ScopeType.CONVERSATION )
@Interceptors( SeamInterceptor.class )
public class ApplicationListBean implements
        ApplicationList
{

    @In( create = true )
    private ApplicationDao applicationDao;

    @In( create = true )
    private TagService tagService;

    @Out(scope = ScopeType.CONVERSATION,
        required= false, value="application")
    @DataModelSelection
    private Application application;

    @Begin
    @Factory("applicationList")
    public void initialize()
    {
        this.applicationList =
        applicationDao.getCompletedApplications();
    }
```

- The @Name annotation identifies this object as a seam component and gives it the name "completedApplications"
- The @Scope annotation puts the Seam component in the Conversation Context.
- Seam's Context are:
  - ✓ Stateless context
  - ✓ Event (or request) context
  - ✓ Page context
  - ✓ Conversation context
  - ✓ Session context
  - ✓ Business process context
  - ✓ Application context

Seam Intro

```
@Name( "completedApplications" )
@Stateful
@Scope( ScopeType.CONVERSATION )
@Interceptors( SeamInterceptor.class )
public class ApplicationListBean implements
        ApplicationList
{

    @In( create = true )
    private ApplicationDao applicationDao;

    @In( create = true )
    private TagService tagService;

    @Out(scope = ScopeType.CONVERSATION,
        required= false, value="application")
    @DataModelSelection
    private Application application;

    @Begin
    @Factory("applicationList")
    public void initialize()
    {
        this.applicationList =
        applicationDao.getCompletedApplications();
    }
```

- The @In annotation takes the object out of the bucket (context) and injects it.
- The @Out annotation puts the object into a bucket (context)
- The @Begin annotation begins a conversation. @End annotation will end a conversation.
- The @Factory annotation is the method used to initialize the object as it goes into the bucket (context)

Creating a Reservation

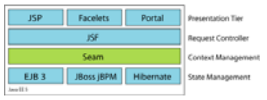
- Requirements are conversational.
- The Seam Context Model encourages thinking Conversationally.
- To Cache an object my developers don't think about
  - ✓ eviction policies
  - ✓ failover
  - ✓ how and when to talk to the API
  - ✓ Whether it's stored in the database
  - ✓ How updates happen

Actor	System
1. The user indicates the creation of a reservation.	2.The system responds with a reservation Create interface.
3. The user searches for a customer.	4. List's Customer's matching criteria.
5. The user identifies Customer.	6. The system associates customer information with new reservation
7. The user Searches for pickup location.	8. The system lists locations matching criteria
9. The user identifies pickup location and pickup time.	10. The system associates pickup location and time with the reservation.
11. The user searches for drop off location.	12. The system lists locations matching criteria.
13. The user identifies the drop off location and time.	14. The system associates the drop off location and time with the reservation.
15. The user searches for car class.	16. The system lists car class matching criteria.
17. The user identifies car class.	18. associate Car Class with Reservation

Taken from my article on Seam at http://www.devx.com/Java/Article/31327/0/page/1

Seam Highlights

- Takes the power of IOC and embraces Java EE 5.
- The page flow models provide tremendous power
- Programming that looks and feels like the requirements not the framework you are working in.
- Integrate JSF with EJB 3.0
- One Kind of "Stuff"
- Declarative State Management



- Bijection
- Workspace Management
- Integrate Business Process as a First Class Construct
- Annotated POJOs Everywhere
- Testability as a Core Feature

Take home Notes

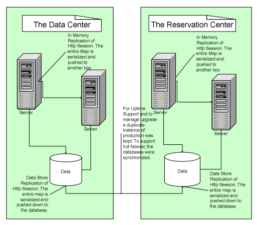
Not Part of Presentation

Http Session

- In the first application of the architecture we put the state in http Session.
- Long term this didn't work for us, all our front ends were not Struts or web based.
- As state grew the scalability of the solution was limited for clustering reasons.

Clustering Solutions

- JBoss TreeCache AOP can manage the size issue by replicating only the parts of the tree that are modified.
- Tree Cache is in memory replication.
- I don't have warm failover when switching between centers.
- Warm failover between centers is a goal but not a current requirement. We have not solved that problem yet.


The Data Center
In Memory Replication of Http Session. The entire Map is serialized and pushed to another box
Server
Server
Data
Data Store Replication of Http Session. The entire map is serialized and pushed down to the database.
For Uptime Support and to manage upgrade a duplicate instance of production was kept. To support hot failover, the databases were synchronized.
The Reservation Center
In Memory Replication of Http Session. The entire Map is serialized and pushed to another box
Server
Server
Data
Data Store Replication of Http Session. The entire map is serialized and pushed down to the database.

Http Session

- Struts Apps do make use of Http Session but for Front End Caching needs exclusively.
- To keep Http Session Clean we:
  - ✓ Use a wrapper for Http Session called *SessionManager*
  - ✓ Insert objects with an event associated.
  - ✓ The event causes all objects scoped at that event or below to be cleaned from session.
- Keep in mind for Seam section

```
protected XDelegate
getDelegate(HttpServletRequest request)
{
  Object delegate =
    SessionManager.getAttribute(request,
                                DELEGATE_KEY);
  if ((delegate == null) || (!(delegate
                      instanceof XDelegate)))
  {
    delegate = new XDelegate();
    SessionManager.setAttribute(request,
                  DELEGATE_KEY, delegate,
                  EventLevel.Screen);
  }
  return (XDelegate) delegate;
}


private static final EventLevel[] VALUES = {
Request, Page, Screen, UseCase, Application,
MajorFuctionality, SecurityEvent, LogOnOrOff };
```

25

Stateful Session Beans

- When we realized that Http Session would not work with state not related to front end work, we moved it to Stateful Session Beans.
- In our initial application server (not JBoss) we put a very large object tree in state, the app server would seize up with 3 or 4 concurrent users.
- Stateful Session Beans in clustered application servers either have the same problem as Http Session or it's simply not supported as clusterable.
- Again JBoss helps solve this problem by providing Tree Cache AOP as the back end solution for clustering Stateful Session EJB's which will update only the changed objects in a large object graph.

26

Stateful Session and Entity EJBs

- We decided to limit the size of data in Stateful Beans.
  - ✓ Created a Map object
  - ✓ Map contained handles or primary keys of Entity Beans
  - ✓ Map is passed into key methods behind the app cord layer.
  - ✓ The corresponding Components would use Map to find the cached data used for that Session.

27

Map and Entity Beans Problems

- Passing Map along with the SessionKey and the Security Object reduced The Cohesiveness of my methods.
- Map had no controls
  - ✓ Anyone could put anything in it, easily overwriting data put in by another component.
  - ✓ Could get cluttered and no way to clean it.
  - ✓ No way to track what was still needed and what was not.
  - ✓ A potential source of hard to track bugs.
- Couldn't share information between Session Beans.
- Our first app server didn't support clustered Stateful Session Beans.

28

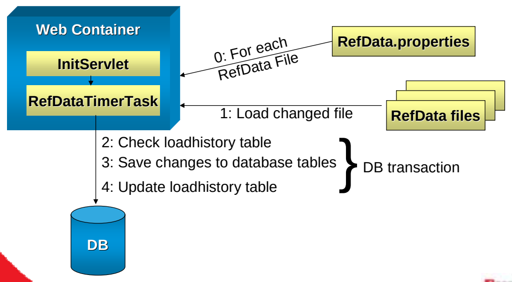
Map and Entity Beans Problems

- Didn't work for our Ref Data.
  - ✓ Built on singleton pattern
  - ✓ As the number and size of the ref data grew the amount of memory being used grew
  - ✓ It was using valuable memory and often not being used or needed.
- Entity Beans quickly became our number one problem with performance and scaling.
  - ✓ The Model is poorly thought out. Calling load and store all the time caused all sorts of problems.
  - ✓ Several developers coded around it, but for the most part, these routines were our biggest scalability problem.

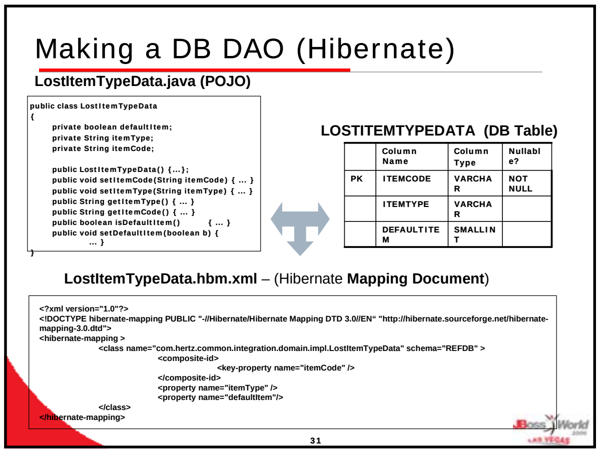
29

TimerTask & LoadHistory

Keeping the files in sync with the database



30



Making a DB DAO (Hibernate)

LostItemTypeData.java (POJO)

```
public class LostItemTypeData
{
        private boolean defaultItem;
        private String itemType;
        private String itemCode;

        public LostItemTypeData() {…};
        public void setItemCode(String itemCode) { … }
        public void setItemType(String itemType) { … }
        public String getItemType() { … }
        public String getItemCode() { … }
        public boolean isDefaultItem()        { … }
        public void setDefaultItem(boolean b) {
                … }
}
```

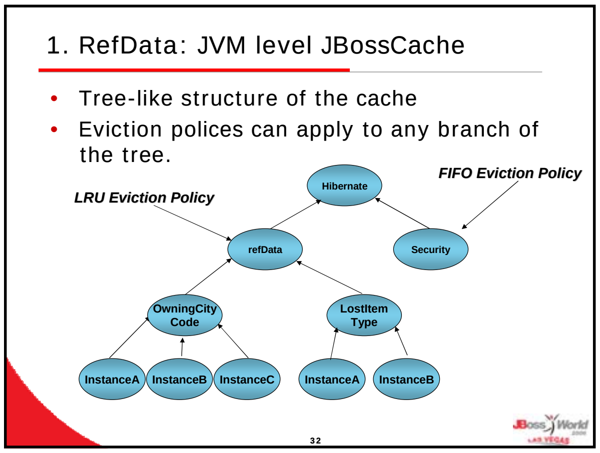
LOSTITEMTYPEDATA (DB Table)

	Column Name	Column Type	Nullable?
PK	ITEMCODE	VARCHAR	NOT NULL
	ITEMTYPE	VARCHAR	
	DEFAULTITEM	SMALLINT	

LostItemTypeData.hbm.xml – (Hibernate **Mapping Document**)

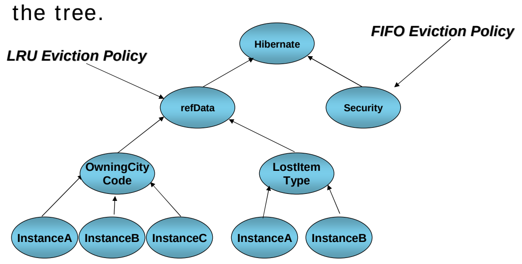
```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN“ "http://hibernate.sourceforge.net/hibernate-
mapping-3.0.dtd">
<hibernate-mapping >
            <class name="com.hertz.common.integration.domain.impl.LostItemTypeData" schema="REFDB" >
                        <composite-id>
                                    <key-property name="itemCode" />
                        </composite-id>
                        <property name="itemType" />
                        <property name="defaultItem"/>
            </class>
</hibernate-mapping>
```

31



1. RefData: JVM level JBossCache

- Tree-like structure of the cache
- Eviction polices can apply to any branch of the tree.



32