



Trends in JAVA EE

EJB3.0, Annotations, JPA, Web
Services, JAXB

- Premier Java Developer Conference organized by Sun and Java technology vendors
- May 16-19, San Francisco, USA
- 15000+ attendees
- 350+ sessions, there is something for every Java developer
- Chance to learn upcoming technologies, meet interesting people, and see cool projects!

- EJB 3: An overview
- Annotations, Annotations, Annotations
- EJB 3: Java Persistence API
- EJB 3: Session Beans reloaded
- Annotations in WebServices, JAXB
- Other trends (JSF, AJAX, Scripting languages, etc..)

- EJBs are special java objects (components) managed by the EJB container.
- EJB container provides various services -Transactions, Persistence, Security, Resource Pooling, Remotability, etc. to the components.
- Types of EJBs – Entity Bean, Session Bean
- Entity Bean
 - Persistent components, typically persisted to a database.
 - Lacked rich class relationships like inheritance, collections.
- Session Bean
 - Interacts with entity bean to provide business functionality
- Complexity of developing was just too much.

EJB 2.1 Session Bean complexity

```
public class CustomerManagerBean implements javax.ejb.SessionBean {
    SessionContext ctx;
    CustomerLocalHome customerHome; //Entity bean reference

    public void ejbCreate() {
        Context initialCtx = new InitialContext();
        customerHome
            =(CustomerLocalHome)initialCtx.lookup("java:com/env/ejb/cust");
    }
    public void setCustomerStatus(Long id, String status) {
        customerHome.find(id).setCustomerStatus(status);
    }
    public void ejbActivate() {}
    public void ejbPassivate() {}
    public void ejbRemove() {}
    public void setSessionContext(SessionContext ctx) {
        this.ctx = ctx;
    }
}
```

EJB 2.1 Session Bean complexity

```
public interface CustomerManagerHome extends javax.ejb.EJBLocalHome {  
    // Required to instantiate component  
    public CustomerManager create() throws CreateException;  
}
```

```
public interface CustomerManager extends javax.ejb.EJBLocalObject {  
    public void setCustomerStatus(Long id, String status);  
}
```

ejb-jar.xml:

```
<session>  
    <ejb-name>CustomerManagerBean</ejb-name>  
    <local-home>com.example.CustomerManagerHome</local-home>  
    <local>com.example.CustomerManager</local>  
    <ejb-class>com.example.CustomerManagerBean</ejb-class>  
    <session-type>Stateless</session-type>  
    <transaction-type>Container</transaction-type>  
</session>
```

What is wrong with EJB 2.1?

- Tons of boiler plate code
 - Using InitialContext to lookup components
 - Creating home interface for Bean classes, implementing SessionBean, EJBLocalHome and EJBLocalObject classes
- XML to configure container provided services
- Alternatives emerged
 - Hibernate, JDO
 - Allowed Plain Old Java Objects (POJO) to be persisted to the database.
 - Easier and powerful than entity beans, but painful xml configuration
 - Spring Framework
 - Provides session bean type of functionality without boiler plate code, but again relied on xml configuration

- JSR 220: Enterprise JavaBeans 3.0
 - “The purpose of Enterprise JavaBeans (EJB) 3.0 is to improve the EJB architecture by **reducing its complexity from the developer's point of view.**”
- EJB 3.0 \approx EJB 2.1
 - XML configuration hell
 - Boiler plate EJB code
 - + Java 5 annotations
 - + Java Persistence API (Hibernate, JDO)
 - + Dependency injection
 - (inspired by Spring)
 - + Intelligent defaults (inspired by Rails)

- EJB 3: An overview
- Annotations, Annotations, Annotations
- EJB 3: Java Persistence API
- EJB 3: Session Beans reloaded
- Annotations in WebServices, JAXB
- Other trends (JSF, AJAX, Scripting Languages, etc..)

- “A metadata facility that would allow classes, interfaces, fields, and methods to be marked as having particular attributes.”
- Test framework example:

- Java 1.4 (using JUnit)

```
public class AdditionTest extends TestCase {  
    public void testAddition() {  
        assertEquals(2, 1+1);  
    }  
}
```

- Java 1.5 (Custom test framework using annotations):

```
public class AdditionTest extends TestCase {  
    @Test (id="10")  
    public void addition() {  
        assertEquals(2, 1+1);  
    }  
}
```

- **@Test annotation definition:**

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface Test {
    String id() default "[none]";
}
```

- **RetentionPolicy**

- **RUNTIME** - annotations available at compile and run time, so they may be read using Reflection API.
- **CLASS** - annotations available in the class file at compile time, but not runtime. Can be used for byte code manipulation.
- **SOURCE** – annotations discarded by the compiler.

- **@Target:** Method, Field, Class, Package, etc.

- Runtime annotation processing:

```
public class RunAdditionTest {  
    public static void main(String[] args) throws Exception {  
        for (Method m : AdditionTest.class.getMethods()) {  
            if (m.isAnnotationPresent(Test.class)) {  
                m.invoke(null); //Invoke the test method  
            }  
        }  
    }  
}
```

- Uses reflection API extensions to process Runtime annotations
- APT tool can be used to process compile time annotations.

- Pros
 - Standard, cleaner way to markup code
 - Framework usage doesn't require method name conventions like test*, extending framework classes or declaring xml configuration files
 - Annotation tags are checked at compile time - avoids having to struggle with xml configuration
 - Frameworks and tools can process metadata using standard annotation processing APIs/tools
- Cons
 - Lots of potential for abuse: runtime and byte code hooks could become the next bane of java development ?

- EJB 3: An overview
- Annotations, Annotations, Annotations
- EJB 3: Java Persistence API
- EJB 3: Session Beans reloaded
- Annotations in WebServices, JAXB
- Other trends (JSF, AJAX, Scripting Languages, etc..)

- Preferred persistence strategy, instead of Entity beans
- Standardize on Object Relational Mapping
 - Hibernate, JDO, Toplink implement JPA standards
 - Write Once Run Anywhere - ORM provider can be switched easily
 - Can run in J2SE or Java EE environment
- Persistence support for Plain Old Java Objects (POJO)
 - Includes one-one, one-many, many-many and inheritance relationships
- Annotations or XML to specify ORM metadata
 - Avoid xml configuration, to preserve your sanity
- Query language to query objects

@Entity

```
public class Customer {
```

```
    @Id @GeneratedValue(strategy=GenerationType.SEQUENCE,  
                        generator="IDSEQ")
```

```
    Long id;  
    String firstName;
```

```
    @Column(name="name2")  
    String lastname;
```

```
    @Version int version;  
    // Getter and setter methods
```

```
    @Transient String loginIp;
```

```
}
```

customer			
id	firstName	name2	version

- Fields are persistent by default (inspired by Rails)
- Table and Column names are configurable

```
@Entity public class Customer {
```

```
    @Id Long id;
```

```
    @OneToMany(cascade=CascadeType.ALL, fetch=FetchType.EAGER)
```

```
    Set<Order> orders;
```

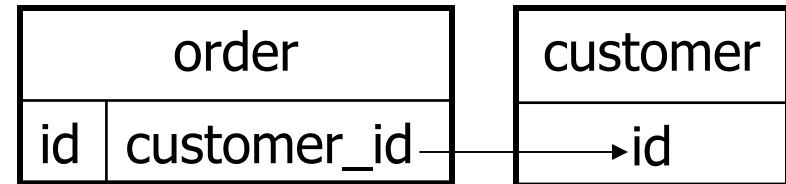
```
}
```

```
@Entity public class Order {
```

```
    @Id Long id;
```

```
    @ManyToOne Customer customer;
```

```
}
```



- **Cascade**

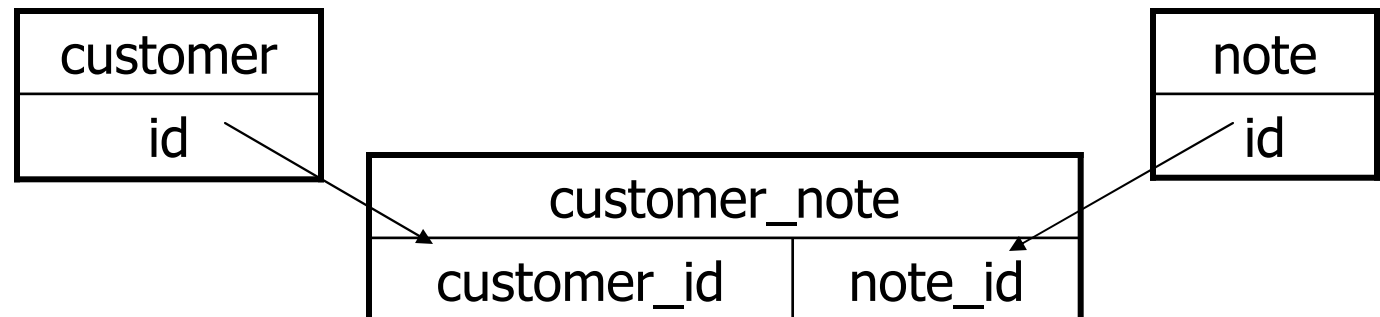
- Define how the persistence operation cascades to "orders"
- Cascade.ALL – cascade all persistence operations to "orders"
- Values - PERSIST, REMOVE, MERGE(update), REFRESH, ALL

- **Fetch strategy**

- Lazy – defers loading the "orders" from db, until it is accessed
- Eager – loads the "orders" collection always

```
@Entity public class Customer {
    @Id Long id;
    @ManyToMany
    Set<Note> phones;
}
```

```
@Entity public class Note {
    @Id Long id;
}
```



- Join table and field names are configurable using annotations

Inheritance mapping (Joined Strategy)

@Entity

@Inheritance(strategy=InheritanceType.JOINED)

```
public abstract class Customer {
```

```
    @Id Long id;
```

```
}
```

```
@Entity public class MailCustomer extends Customer{
```

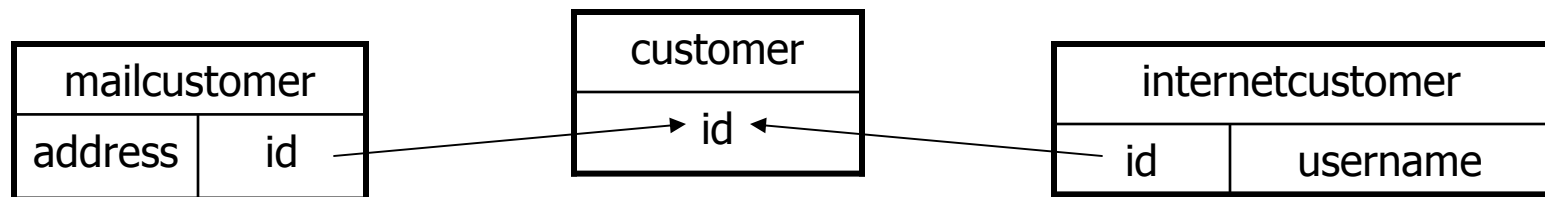
```
    String address;
```

```
}
```

```
@Entity public class InternetCustomer extends Customer{
```

```
    String userName;
```

```
}
```



Inheritance strategies compared

- Joined strategy
 - tables are normalized
 - allows polymorphic queries and dynamic resolution of class. For eg: a query to find all customers will return a collection of InternetCustomer/ MailCustomer objects
- Single table: table per class hierarchy
 - use when the child classes differ mostly in behaviour and not in data
 - Tables aren't normalized, but discriminator column ("type") is used to support polymorphic queries and dynamic class resolution

customer			
id	type	address	username

- Table Per Class
 - Use if you don't require polymorphic queries and don't care about column repetition in the child tables

mailcustomer	
address	id

internetcustomer	
id	username

- EntityListener callbacks to manage persistent entity lifecycle events

```
@Entity
```

```
@EntityListeners(value={CustomerListener.class})
```

```
public class Customer { ... }
```

```
public class CustomerListener {
```

```
    @PrePersist
```

```
    public void insertHistory(Customer customer){
```

```
        customer.addHistory(new CustomerHistory(new Date()));
```

```
    }
```

```
}
```

- @PrePersist, @PostPersist, @PreRemove,
 @PostRemove, @PreUpdate, @PostUpdate, @PostLoad

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence>
  <persistence-unit name="mycontext">
    <provider>org.hibernate.ejb.HibernatePersistence</provider>
    <class>test.Customer</class>
    <class>test.Order</class>
    <properties>
      <property name="hibernate.connection.url"
        value="jdbc:mysql://localhost/demodb"/>
      <property name="hibernate.dialect"
        value="org.hibernate.dialect.MySQLDialect"/>
      <property name="hibernate.connection.driver_class"
        value="com.mysql.jdbc.Driver"/>
      <property name="hibernate.connection.username" value="javauser"/>
      <property name="hibernate.connection.password" value="javadude"/>
    </properties>
  </persistence-unit>
</persistence>
```

Persistence Operations (using J2SE)

```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("mycontext");  
EntityManager em = emf.createEntityManager();  
em.getTransaction().begin();
```

```
em.persist(customer);
```

```
em.getTransaction().commit();  
em.close();
```

- `EntityManager.persist()`, `remove()`, `merge()` (update), `refresh()`

- Powerful query language with SQL like syntax

```
Query q = em.createQuery("select c from Customer c where  
                        c.firstName = :fname");
```

```
q.setParameter("fname", "Joe");
```

```
List<Customer> customers = (List<Customer>) q.getResultList();
```

- EJB 3: An overview
- Annotations, Annotations, Annotations
- EJB 3: Java Persistence API
- EJB 3: Session Beans reloaded
- Annotations in WebServices, JAXB
- Other trends (JSF, AJAX, Scripting Languages, etc..)

- **Dependency Injection**
 - Bean instance is supplied with references to resources, when instance is created
 - Inspired by Spring framework
- EJB container provided services are specified using annotations such as `@TransactionAttribute`, `@RolesAllowed`

EJB 3: Session Bean example

@Stateless

```
public class CustomerManagerBean implements CustomerManager{  
    @PersistenceContext  
    private EntityManager em;  
  
    @TransactionAttribute(REQUIRED)  
    public void updateCustomer(Customer customer){  
        em.merge(customer); }  
}
```

@Local

```
public interface CustomerManager {  
    public void updateCustomer(Customer customer);  
}
```

- EJB 3: An overview
- Annotations, Annotations, Annotations
- EJB 3: Java Persistence API
- EJB 3: Session Beans reloaded
- Annotations in WebServices, JAXB
- Other trends (JSF, AJAX, Scripting Languages, etc..)

```

package endpoint;

import java.rmi.*;

public class HelloServiceImpl
    implements HelloServiceSEI {

    public String sayHello(String param)
        throws java.rmi.RemoteException {
        return "Hello " + param;
    }
}

```

```

package endpoint;

import java.rmi.*;

public interface HelloServiceSEI
    extends java.rmi.Remote {

    public String sayHello(String param)
        throws java.rmi.RemoteException;
}

```

```

<?xml version='1.0' encoding='UTF-8' ?>
<webservises xmlns='http://java.sun.com/xml/ns/j2ee' version='1.1'>
  <webservice-description>
    <webservice-description-name>
      HelloService</webservice-description-name>
    <wsdl-file>
      WEB-INF/wsdl/HelloService.wsdl</wsdl-file>
    <jaxrpc-mapping-file>
      WEB-INF/HelloService-mapping.xml
    </jaxrpc-mapping-file>
    <port-component xmlns:wsdl-port_ns='urn:HelloService/wsdl'>
      <port-component-name>HelloService</port-component-name>
      <wsdl-port>wsdl-port_ns:HelloServiceSEIPort</wsdl-port>
      <service-endpoint-interface>
        endpoint.HelloServiceSEI</service-endpoint-interface>
      <service-impl-bean>
        <servlet-link>WSServlet_HelloService</servlet-link>
      </service-impl-bean>
    </port-component>
  </webservice-description>
</webservises>

```

```

<?xml version='1.0' encoding='UTF-8' ?>
<configuration
  xmlns='http://java.sun.com/xml/ns/jax-rpc/ri/config'>
  <service name='HelloService'
    targetNamespace='urn:HelloService/wsdl'
    typeNameSpace='urn:HelloService/types'
    packageName='endpoint'>
    <interface name='endpoint.HelloServiceSEI'
      servantName='endpoint.HelloServiceImpl'>
    </interface>
  </service>
</configuration>

```



Java EE Webservice (using annotations)

```
@WebService
public class Hello {

    public String sayHello(String param) {
        return "Hello " + param;
    }
}

public class HelloClient {

    @WebServiceRef(HelloService.class)
    private static Hello svc;
    public static void main(String[] argv) {
        System.out.println(svc.sayHello(argv[0]));
    }
}
```



```
@XmlAccessorType(FIELD)
@XmlType(name = "", propOrder = {"x", "y"})
@XmlRootElement(name = "point")
public class Point {

    protected float x;
    protected float y;

    public float getX() {
        return x;
    }

    public void setX(float value) {
        this.x = value;
    }

    public float getY() {
        return y;
    }

    public void setY(float value) {
        this.y = value;
    }
}
```

- 62 lines for
 - `<point><x>1</x><y>2</y></point>`
- 2 files
- 3KB of code total

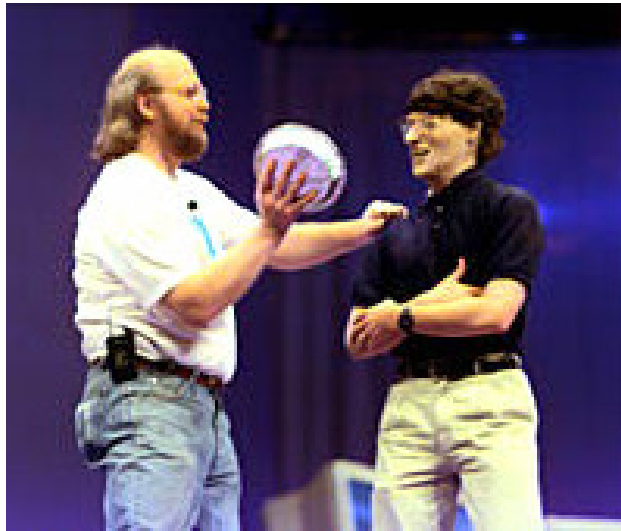
- EJB 3: An overview
- Annotations, Annotations, Annotations
- EJB 3: Java Persistence API
- EJB 3: Session Beans reloaded
- Annotations in WebServices, JAXB
- Other trends (JSF, AJAX, Scripting Languages, etc..)

- Sun and vendors want to push JSF
 - Lots of boiler plate code
 - To be simplified using annotations (unofficial). Check out Sun's test bed – Apache Shale Framework
 - If your project doesn't need JSF, don't learn it just yet!
- AJAX Frameworks – no leaders!
 - DOJO – Open source and Sun's preferred framework but poor documentation
 - Tons of commercial ones – Tibco, Backbase, etc. Have lots of fancy components.
 - Google Web Toolkit – looks cool, haven't dug deeper.
 - Write and test dynamic ui components using Java APIs and your favourite Java IDE.
 - Code generation tool to generate corresponding Javascript code
 - AJAX components can be written using toolkit's Remote APIs

- **Groovy – dynamic scripting language**
 - Inspired heavily by the popularity of Ruby
 - Selling points
 - Dynamic language, requires no typing
 - Java like syntax and runs on JVM
 - Extensive API additions
 - Metaprogramming – ability to add fields, methods at runtime
 - Still in its infancy, hard to say if it will get popular
 - Performance is a huge issue
 - Java Community Process request: JSR 241
- **Grails**
 - Web application development framework for JVM, based on Ruby on Rails
 - Still on Release 0.1, won't be production ready anytime soon

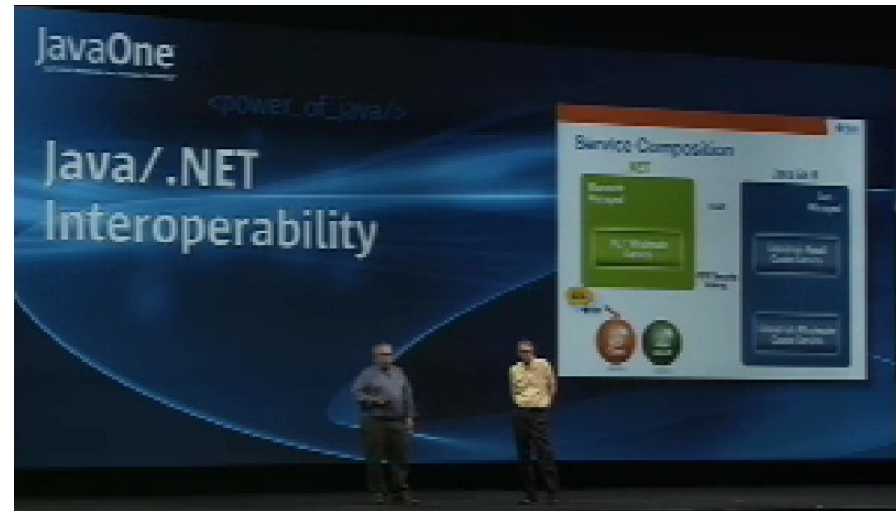
- Apache Derby (formerly IBM cloudscape)
 - Fully transactional, ANSI SQL99 compliant database written in Java
 - Requires no installation, just a file that is completely portable
 - Claims that performance is comparable to MySQL
 - Bundled with JDK 1.6 due this October. Potential to become popular.
- Glassfish
 - Complete Java EE application server that supports EJB 3, JPA, Web tier technologies
 - Base code for Sun's commercial Application server

JavaOne 1998



James Gosling hitting a pie on a person wearing Bill Gates mask

JavaOne 2006



Sun and Microsoft talk about their joint efforts to ensure interoperability of transactions and security across Java EE and .NET webservices

- Cooperating to settle Sun's anti trust claims
- Swing to use Windows native graphics rendering in Java 1.6, to provide the windows look and feel

Some interesting comments

- Swing API doubles as an IQ test – Swing Lead
- Struts is the next Cobol – Apache Struts committer
- We need a client side MVC, server side MVC like JSF can only take us only thus far – JSF spec co-lead
- BEA has a team working on supporting Rails in Weblogic platform – BEA chief architect
- Sun to open source Java, not a question of whether, but a question of how. Worried about compatibility issues – Sun VP
- Sun has no idea how to make money of Java – worried Sun employee

Key takeaways

- Annotations promise to simplify Java EE development. Watch out for “annotations abuse”
- Java Persistence API is here to stay, learn it well.
 - Hibernate (free, still in beta), Oracle Toplink (free for development, requires production license), Open JPA (free developer version)
- Migrate to latest version of Webservice and JAXB APIs for simplicity
- Wait for next revamp of JSF, annotations could simplify web development. Check out Apache Shale.
- Lots of buzz about AJAX, not sure if a developer friendly framework will emerge anytime soon ☹.
- JVM scripting using Groovy and Grails have a long way to go.

Acknowledgements

- JavaOne 2006 presentations
- <http://java.sun.com>
- <http://java.sun.com/javaee/>
- <http://dev2dev.bea.com/>



Contact Us

Telephone

(888) 594-6260 – Sales
(416) 304-0860 – Main

Fax

(416) 946-1929

Address

345 Adelaide Street W, 6th Floor
Toronto, Ontario
Canada
M5V 1R5

Email Contacts

General Info

jonahinfo@jonahgroup.com

Sales

jonahsales@jonahgroup.com