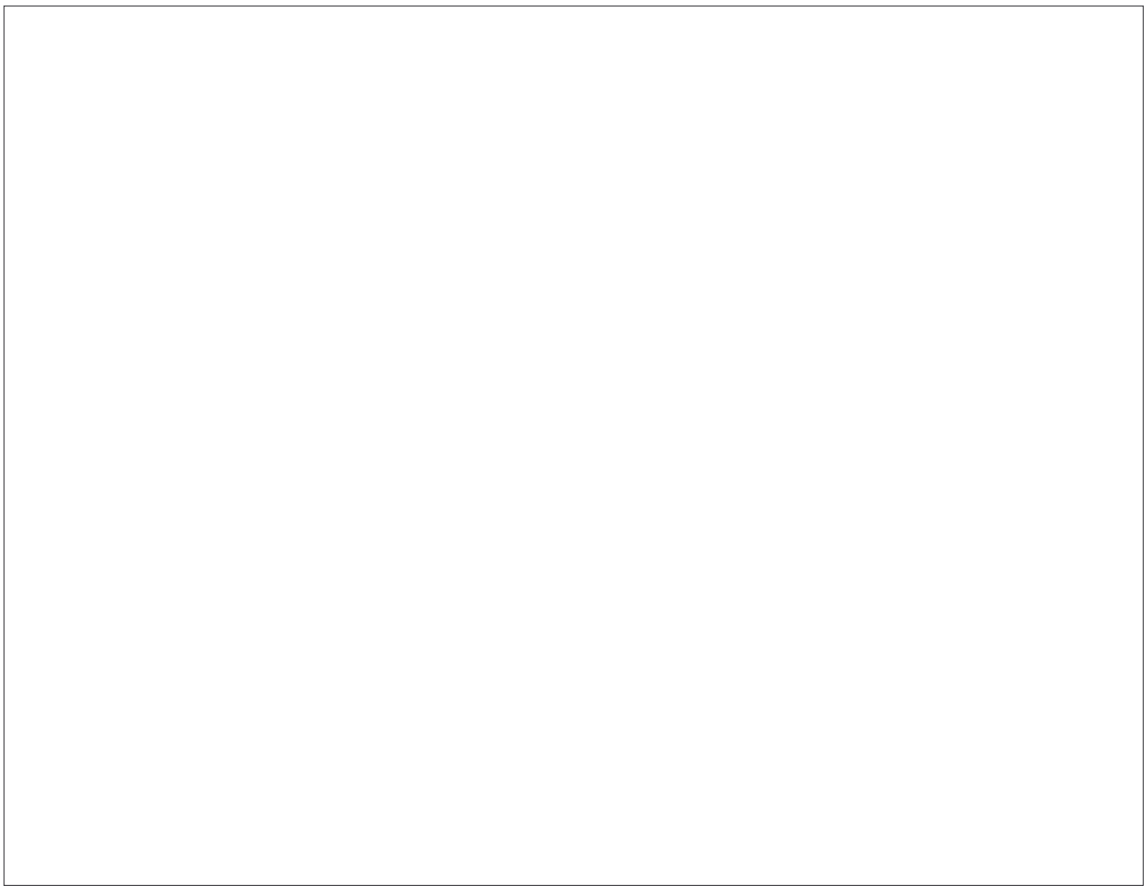
WebSphere Application Server v5.0

Developing and Deploying Web Applications - Day 2



EJB Review

dragonSlayer Team

What are Enterprise JavaBeans?

- EJBs are "Beans" in the sense that they get runtime services from a container but they are not JavaBeans
- A component architecture for distributed object systems
- A framework for creating extensible, scalable, object-oriented applications

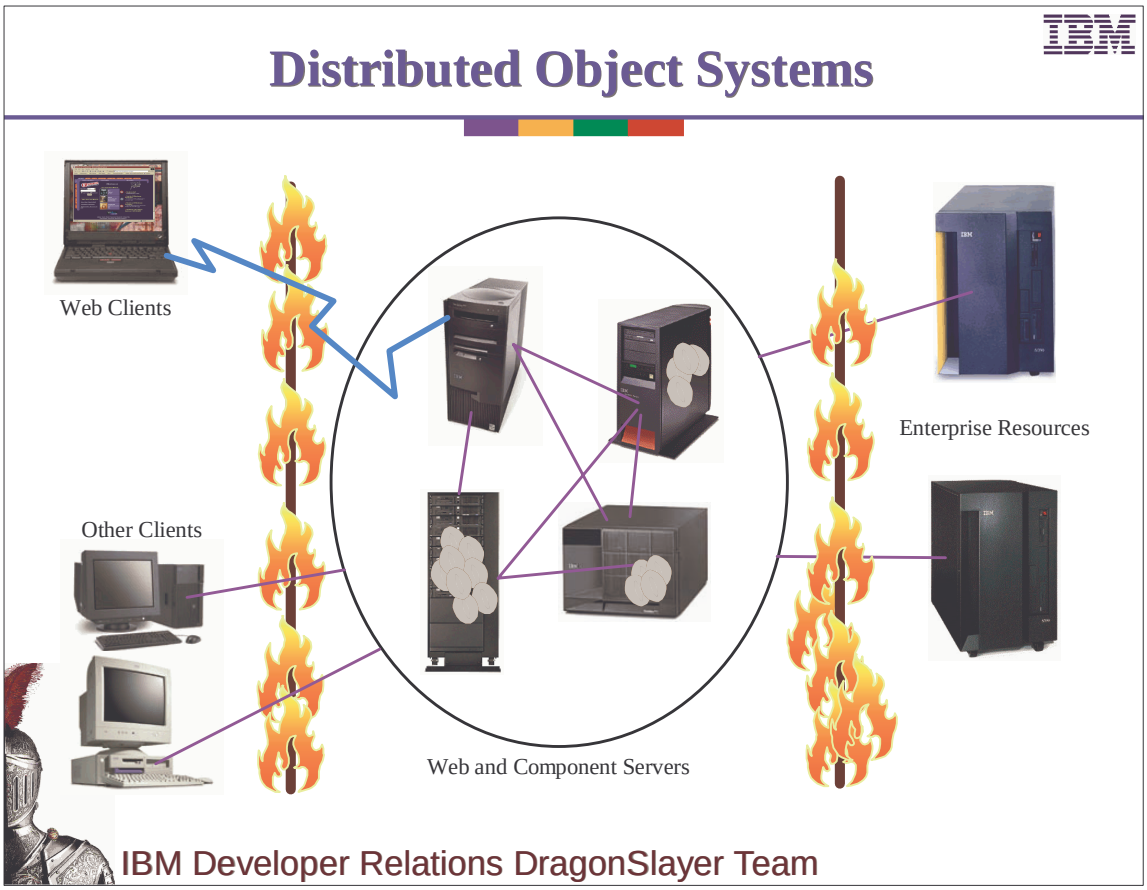
IBM Developer Relations DragonSlayer Team

JavaBeans and **Enterprise JavaBeans** are both specifications for component models.

An EJB is not technically a JavaBean, purely because of the differences in the lifecycle requirements between the two specifications.

JavaBeans must define a public no-argument constructor. On the other hand, the client of an EJB never sees (and is guaranteed not to) any constructor, since object creation is performed by a factory (via the enterprise bean’s home interface).

EJBs are distributed objects (by specification).



EJBs are a perfect fit for assembling distributed applications. This capability takes a whole new meaning when you run EJBs on WebSphere as it allows you to run your applications across a multitude of different platforms.

Thanks to this ability you can now scale your applications horizontally and vertically as much as you need to. The advantage of EJBs is that they are written entirely in Java therefore you can run them on any platform. The advantage of WebSphere is that it runs on the most popular platforms (Windows, Linux, AIX, Solaris, NetWare, OS/2, HP-UX, OS/400, and OS/390) giving your application access to as much computing power as there is available in the market.

EJBs running on WebSphere integrate very well with enterprise-level resources. IBM provides a wide assortment of connectors to enterprise resources and legacy systems. Some of them are provided to you as features in Application Developer.

Simplified Programming Model

- EJBs are written in Java
- They follow a common programming model
- EJBs possess high-level APIs for services
- The focus of EJBs is on business logic

IBM Developer Relations DragonSlayer Team

Enterprise JavaBeans is a programming model for development of reusable, portable Java components. It is really nothing more than a Java class that uses certain programming and naming conventions and can be customized through properties or its customizer.

EJBs are special, non-visual Java components that run on a server. The programmer doesn't have to implement multithreading, concurrency, resource pooling, security, or transaction management. These are provided by the server runtime.

Characteristics of EJBs

- Managed at runtime by a container
- Can be customized at deployment time
- Client access is mediated by the EJB container and the server
- Can be deployed in any EJB compliant container
- Can be included in a composite application without changing the source code or recompiling
- A client's view of an EJB is defined by the bean developer

IBM Developer Relations DragonSlayer Team

The term deploy has a specific meaning in the Enterprise JavaBean paradigm. We'll describe this in more detail when we talk about containers.

EJB Community's Goals for EJB Architecture

- Provide a distributed, OO-based component architecture
- Shield complex system-level issues
- Write once, run anywhere
- Provide interoperability with non-Java applications
- Be compatible with CORBA

IBM Developer Relations DragonSlayer Team

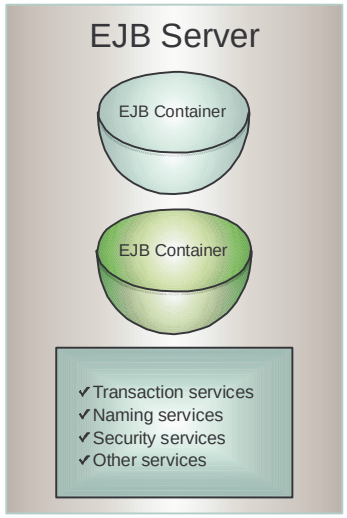
The EJB Architecture

- To help meet the goals of the architecture, the specification describes:
  - EJB Servers
  - EJB Containers
  - EJB classes and instances
  - EJB home and remote interfaces
  - EJB clients

IBM Developer Relations DragonSlayer Team

EJB Servers

- An EJB Server is a process that:
  - Manages EJB Containers (which manage the beans)
  - Provides access to system services
  - Provides Java-related services, specifically
    - A name space accessible via JNDI
    - An OTS-based transaction service



IBM Developer Relations DragonSlayer Team

An **EJB server** may provide access to vendor-specific services. In the illustration, "Other Services" means things like JNDI (the Java Naming and Directory Service) and JTS (the Java Transaction Service). These are used by EJBs but aren't part of the EJB specification.

An EJB server may also provide access to special, local services.

EJB Containers

- Manage EJB classes and instances
  - Generate code to implement access to the beans
  - Enforce transactional requirements
  - Create, initialize, and destroy beans
  - Manage persistence of durable data
- Deliver EJB server services available to the beans
- Containers are transparent to clients

IBM Developer Relations DragonSlayer Team

The word **deploy** has a specific meaning in the EJB paradigm.

With Enterprise JavaBeans, deploy means telling the container to generate the code it needs to supply the services the bean needs.

The code you, the application programmer, write is portable. But to run a bean in a specific server and container, additional code must be created. This code is generated in the deployment step. Thus, you deploy a bean to a server and container.

You can think of it as installing the bean into the environment. There is no client-side API to determine a bean's container. Vendors may distribute beans to containers however they like. This is entirely implementation-specific.

Containers in an EJB Server

- Containers are currently provided by the EJB server itself
  - The container-to-server interface is not specified in the 2.0 or 1.1 EJB specifications
  - Vendors can implement server and container responsibilities as they see fit

IBM Developer Relations DragonSlayer Team

From the outside looking in, this nesting of beans in containers in servers is transparent; it's simply a Java process.

Programmers have no control over any of this. The administrative tools in WebSphere can give you a hint about how WebSphere does this, but the relationship between beans, containers and servers is implementation-specific.

Vendors will be able to write containers for other vendors' servers when the container-server interface is specified.

Container Services: Persistence

- Management can be delegated to the container
  - The bean programmer doesn't need to write "broker" code
  - The container generates the necessary classes and code
- Management can be handled by the programmer if needed

IBM Developer Relations DragonSlayer Team

EJBs offer two ways to handle persistence. It can be delegated to the container, known as container-managed persistence (CMP), or it can be handled by code explicitly included in the bean by the developer, called bean-managed persistence (BMP). Both of these will be discussed in more detail later.

A broker is a common piece of software developed to serve as the "data access" layer for persistent objects. The broker is responsible for saving and restoring the state of an object from the backing store and for keeping the "live" object consistent with its persistent state.

Container Services: Transactions

- Management can be delegated to the container
  - The container is told the transactional requirements of business methods
  - The container provides the transaction control code
- Management can also be handled by the programmer if required

IBM Developer Relations DragonSlayer Team

EJBs offer two ways to handle transactions. They can be handled by the container, known as container-managed transactions, or they can be explicitly handled by the bean developer, using bean-managed transactions.

Container Services: Messaging

- EJB 2.0 requires that containers are able to respond to asynchronous messages from a Java Messaging Service (JMS) Provider
  - The container is told what JMS resources to listen to (queue or topics) and how to respond to messages
  - The container will invoke the appropriate resources when a message arrives

IBM Developer Relations DragonSlayer Team

J2EE 1.3 also requires that all J2EE 1.3 compliant application servers have a JMS Provider

Container Services: Other

- Containers offer other services in addition to persistence and transactions including
  - Naming
  - Security
- Delegating responsibility for these services to the container reduces the burden on application programmers

IBM Developer Relations DragonSlayer Team

Client Access to EJBs

- There are 2 ways that clients can access EJBs
  - Remotely - where the client can reside in the same JVM as the EJB Container or in a different JVM on a different host
  - Locally - the client must be running in the same JVM as the EJB container

IBM Developer Relations DragonSlayer Team

Remote access is more flexible. Clients can be anywhere on the network

Local access is more efficient because both client and EJB are in the same JVM and none of the overhead that's required to handle remote clients is necessary

EJB Classes and Instances

- Construction of EJB applications involves code from three sources
  - Developer-provided code
  - Classes and Interfaces defined by the EJB API
  - Generated code provided by the container(s)
- Developer-provided code includes:
  - An EJB class (a.k.a. "the bean class")
  - One or both of
    - Interfaces for remote client access
      - Home interface (used to find or create beans)
      - Remote interface (used to access bean's business logic)
    - Interfaces for local client access
      - LocalHome interface (used to find or create beans)
      - Local interface (used to access bean's business logic)
  - Other pieces depending on the bean

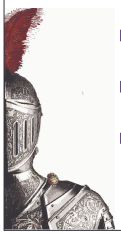
IBM Developer Relations DragonSlayer Team

The EJB API, contained in the Java package javax.ejb, provides the starting point, with interfaces and classes like javax.ejb.EJBHome and javax.ejb.EJBObject. Developers build on this basic API, extending interfaces and implementing classes, to write the code that comprises the actual enterprise bean.

Finally, to run the bean in a specific server and container, we need to generate the implementation classes for the runtime environment. This also allows the server/container to provide the requested services to an enterprise bean.

Types of Enterprise JavaBeans

- Session beans
  - Associated with a particular client
  - Created and destroyed by a client (transient)
  - Do not survive system shutdown
- Entity beans
  - Shared by multiple clients
  - Persist across multiple invocations
  - Survive system shutdown
- Message Driven Beans
  - Associated with a JMS resource (queue or topic)
  - Container invokes when a message arrives
  - No client interface
  - Usually calls other EJBs (Session or Entity)



IBM Developer Relations DragonSlayer Team

This slide shows the three bean types, session beans, entity and message driven beans.

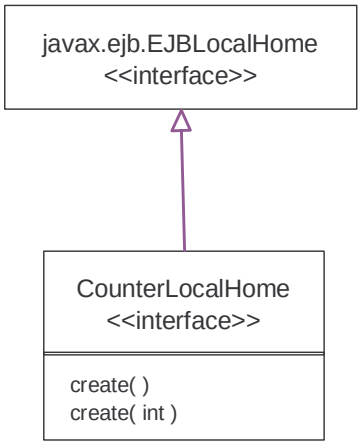
Session beans are always associated with a particular client. This association may last only for the duration of a single method call, or it may last for several method calls.

Entity beans, on the other hand, may be used by multiple clients. The container handles multithreading issues for you, though.

Message Driven Beans have no client interface. They are associated with a JMS resource and are invoked by the container when a message arrives on that resource. They tyoically call Session Beans or Entity beans to perform business logic when invoked

Local client access - Local Home Interfaces

- All EJBs that require local client access have local home interfaces
  - Extend javax.ejb.EJBLocalHome
  - Define create() methods
  - May define remove and finder methods
- Implementation EJBLocalHome object is provided by container

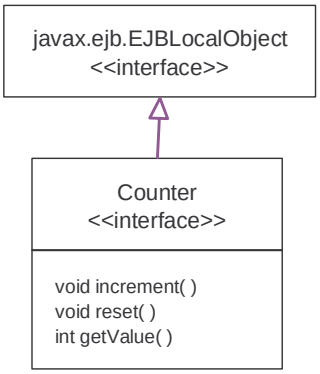


IBM Developer Relations DragonSlayer Team

Clients get a proxy that implements javax.ejb.EJBLocalHome. The container creates a server-side object that implements this interface; this is what the client-side proxy calls.

Local client access - Local Interfaces

- All EJBs that require local client access have local interfaces
  - Define the business logic of a bean
  - Extend the interface:
  - javax.ejb.EJBLocalObject
  - Used by a local client to access the logic of an EJB
- Implementation is provided by the container



IBM Developer Relations DragonSlayer Team

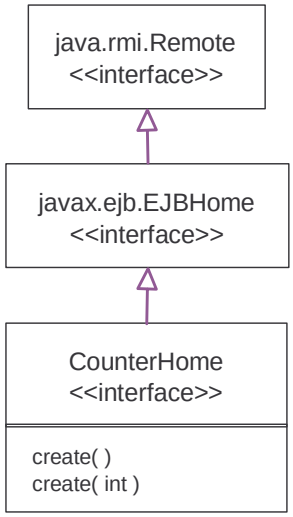
When you write a local interface for an enterprise bean, you extend the javax.ejb.EJBLocalObject interface.

When your bean is deployed, the container implements this interface in a server-side class, EJBLocalObject. An instance of this class is generally referred to as "an EJB object."

The local interface is the piece that truly defines the local client's view of the bean. This interface provides the client with access to the application logic. In fact, local clients cannot call any business method that isn't defined in the local interface.

Remote Client Access - Home Interfaces

- All EJBs that require remote client access have home interfaces
  - Extend javax.ejb.EJBHome
  - Define create() methods
  - May define remove and finder methods
- Implementation EJBHome object is provided by container



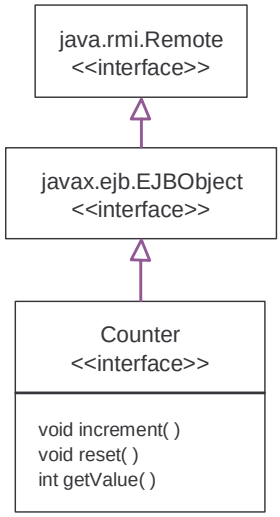
IBM Developer Relations DragonSlayer Team

Clients get a proxy that implements javax.ejb.EJBHome. The container creates a server-side object that implements this interface; this acts as the "subject" to which the client-side proxy connects.

Remote Client access- Remote Interfaces

- All EJBs that require remote client access have remote interfaces
  - Define the business logic of a bean
  - Extend the interface:
  - javax.ejb.EJBObject
  - Used by a remote client to access the logic of an EJB
- Implementation is provided by the container

IBM Developer Relations DragonSlayer Team



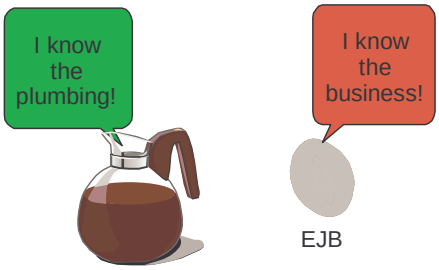
When you write a remote interface for an enterprise bean, you extend the javax.ejb.EJBObject interface.

When your bean is deployed, the container implements this interface in a server-side class, EJBObject. An instance of this class is generally referred to as "an EJB object."

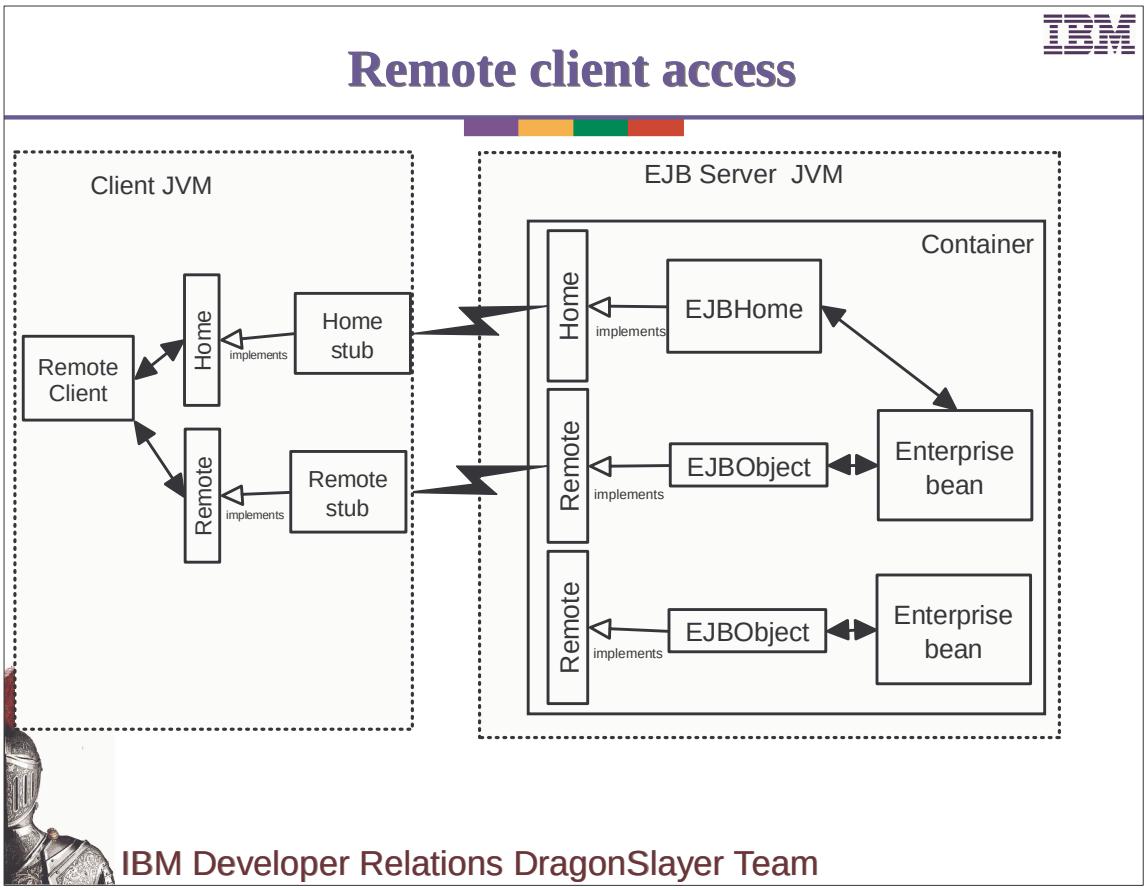
The remote interface is the piece that truly defines the remote client's view of the bean. This interface provides the client with access to the application logic. In fact, clients cannot call any business method that isn't defined in the remote interface.

Server Architecture

- EJBs live and run inside intelligent containers or servers
- EJB containers provide all of the essential services and handle the plumbing following a well-defined contract



IBM Developer Relations DragonSlayer Team



Now we are going to look a little bit deeper at the EJB server architecture.

Notice that a client uses a local "proxy" or "stub" to communicate with a remote object. One of the special remote objects provided in EJB environments is called a "Home".

Clients find the home in a location-independent fashion using the Java Naming and Directory Interface (or JNDI). The purpose of the home is to provide a mechanism for finding and creating remote EJBs. The client provides the home with a unique identifier that distinguishes the target EJB in the server environment.

After the client receives an EJB proxy back from the home, it uses it to communicate with the EJB itself. There are several things about this programming model that should be mentioned.

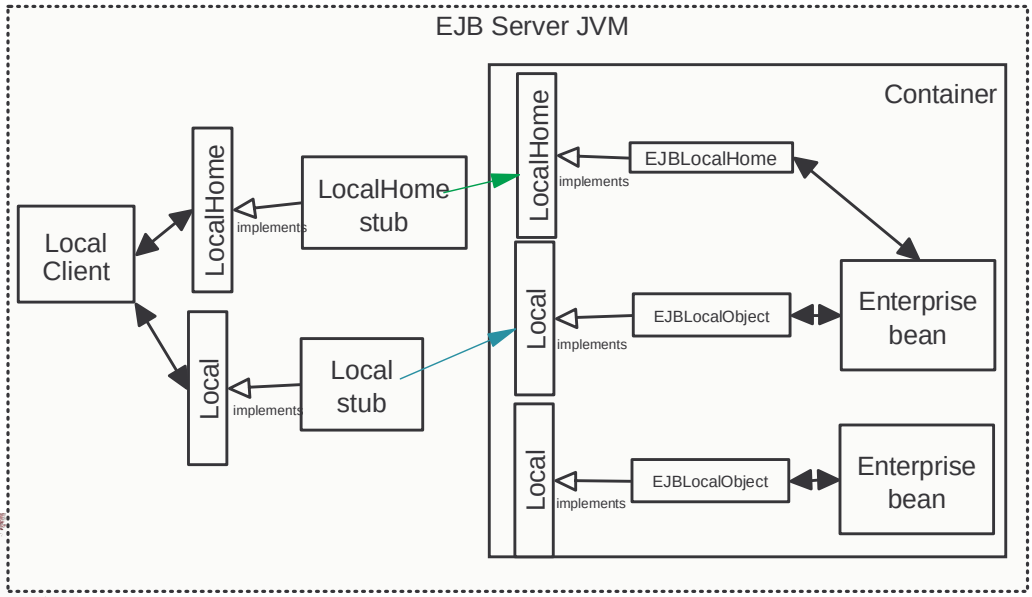
Our clients do not know what kind of persistence mechanisms or resource managers provide the backing store for the EJB. A given client accesses all remote EJBs in the same consistent way.

Clients never worry about reading rows from tables or any other backend system APIs. Clients also have an illusion that EJBs are always in memory.

It might be necessary for the home and container to bring an EJB into storage but this is hidden from clients. We call this a "single-level store" semantic.

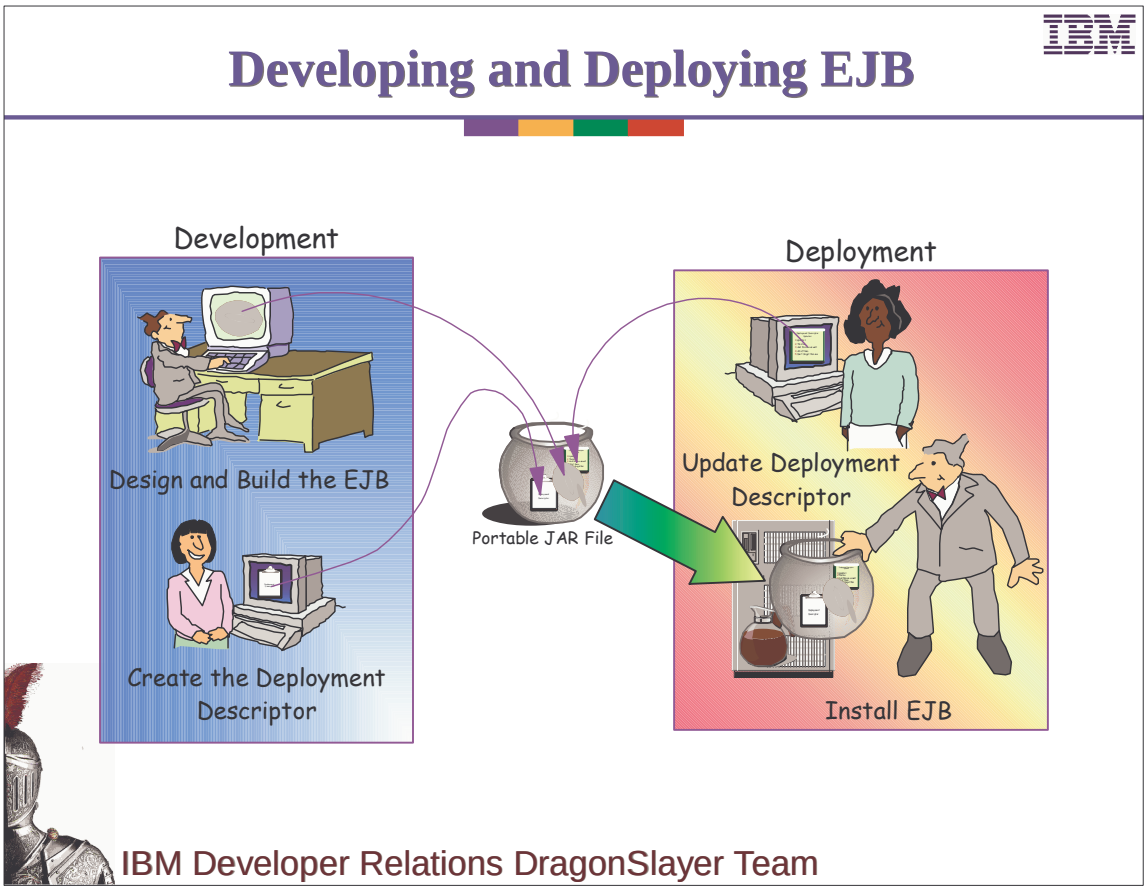
Finally, all clients, whether they be Java clients, ActiveX clients, or something else, follow this same consistent approach. This is a simple and highly rational approach to programming and it promises to be extremely productive.

Local client access



IBM Developer Relations DragonSlayer Team

With Local Access there is no RMI/IIOP involved. Client stubs can call directly into the containers implementation classes



Development activities are shown here on the left. We start by creating the EJB's **bean class**, **home interface**, and **remote interface**. Here's where we will implement the bean's business logic.

Incorporating the EJB into a production environment is what we call a **deployment activity**. This involves additional updates to the deployment descriptor to add things like access control lists and JNDI location information related to the home, and there are other things done here that can only be decided during actual deployment (such as setting environment properties).

The deployment descriptor technique is giving us a "declarative" approach to development. We can literally customize applications without requiring access to the underlying source code. This declarative approach with deployment-time configuration is one of the things that makes EJBs simple to develop. It also makes EJBs more reusable by avoiding a lot of rigid hard-coding.

Summary

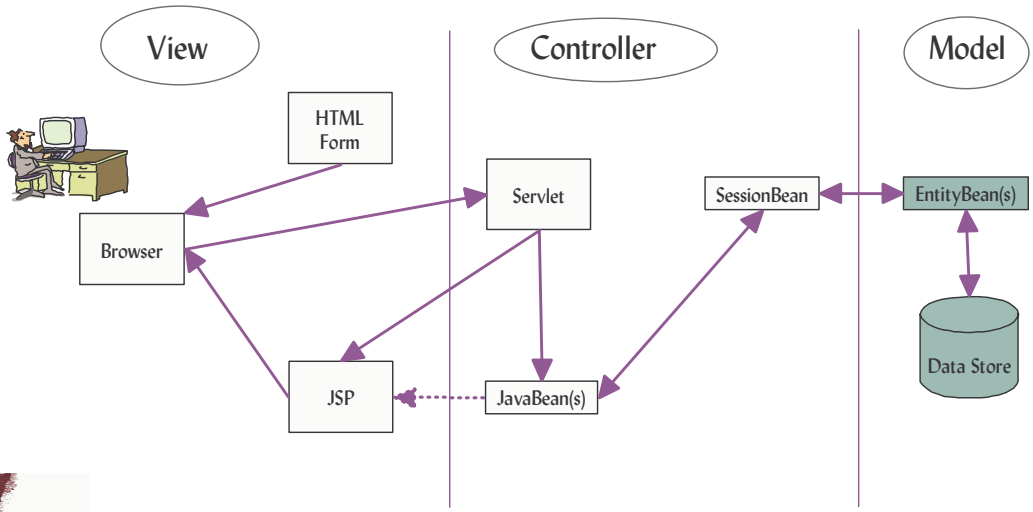
- We've seen:
  - What EJBs are
  - How they work
  - What the development steps are

IBM Developer Relations DragonSlayer Team

Entity Beans

dragonSlayer Team

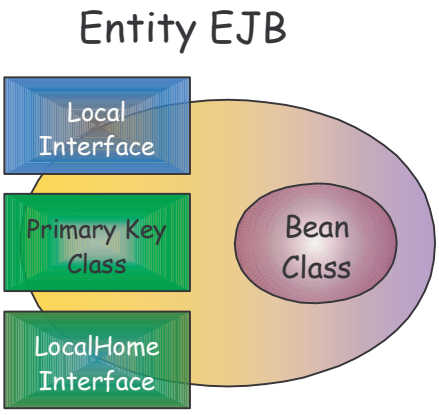
Entity Beans



IBM Developer Relations DragonSlayer Team

Overview

- When the Session Facade pattern is used, every entity bean requires
  - A local home interface
  - A local interface
  - A bean class
  - A primary key class
- Entity beans can take advantage of container-managed persistence



IBM Developer Relations DragonSlayer Team

When the Session Facade pattern is used, writing an entity bean consists of writing four components:
✓The local home interface
✓The local interface
✓The bean class
✓The primary key class.

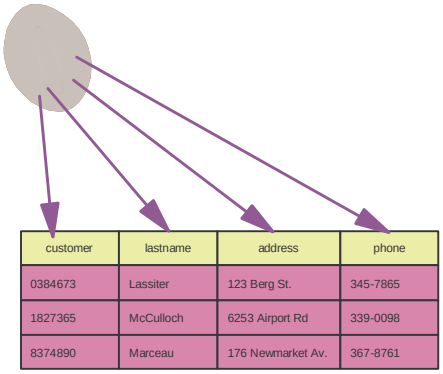
The **local home interface** allows a user to create or find instances of the bean.
The **local interface** gives access to the bean's business logic.
The **bean class** is where the business logic is implemented.
The **primary key class** provides a way to identify beans for future lookup.

Entity beans represent persistent data, and such beans fall into two classes, based on how they managed persistence. The recommended approach, for simplicity and portability, is to delegate management of persistence to the container. These beans are said to make use of **container-managed persistence, or CMP**. The beans discussed in this unit use CMP.

The other option, which requires much more coding on the part of the application programmer, is to write the methods in the bean class to manage persistence. This is called **bean-managed persistence, or BMP**. Bean-managed persistence requires more specialized coding, and such beans are strongly tied to their backing data stores, reducing portability.

General Characteristics

- An entity bean represents a persistent business object
- Multiple clients can concurrently access an entity bean
- Entity beans have a lifetime beyond the client and server process



customer	lastname	address	phone
0384673	Lassiter	123 Berg St.	345-7865
1827365	McCulloch	6253 Airport Rd	339-0098
8374890	Marceau	176 Newmarket Av.	367-8761

IBM Developer Relations DragonSlayer Team

The container transparently manages concurrency, transactions, persistence and other services for the beans that live in it.

Entity beans are persistent; that is, they are stored in some sort of database. They will survive both a crash and a deliberate shutdown of the server. This is one of the ways in which they differ from session beans.

Entity Beans and Persistence

- Container-Managed Persistence (CMP)
  - The container is responsible for saving state
  - You specify the container managed fields
  - Persistence is independent of the data source
- Bean-Managed Persistence (BMP)
  - You write the code to save the bean's state
  - The container doesn't need to generate DB calls
  - Can exploit existing, more powerful persistence frameworks
  - Less adaptable: persistence is hard-coded

IBM Developer Relations DragonSlayer Team

EJB QL

- EJB QL is a portable object query specification language
  - Can write queries that are independent of the underlying database schema
  - Similar to SQL, applies to CMP entity EJBs
  - Allows querying on CMP fields
  - Applies to finder and select methods of CMP Entity beans
- EJB QL can be used for two types of methods:
  - Finder methods
    - Defined in local and remote homes
    - Return EJB local or remote interfaces or collections of those
    - There is no need to provide EJB QL for findByPrimaryKey() method
  - Select methods
    - Not for client use, they are meant to be called by the bean class itself
    - Can return interfaces, but also individual CMP fields (or collections)
- Bean developers defines abstract finder and/or select methods
  - Specifies EJB QL query statement in the deployment descriptor
  - Container tools to generate query implementation

IBM Developer Relations DragonSlayer Team

LocalHome Interfaces for Entity Beans

- The local home interface for an entity bean:
  - Allows a client to:
    - Create new EJBLocalObject instances (with multiple create methods)
    - Look up existing EJBLocalObject (with finder methods)
    - Remove EJBLocalObject instances
  - Extends the EJBLocalHome class

IBM Developer Relations DragonSlayer Team

A local home interface consists most importantly of methods to create new beans and to find existing ones.

Each of the bean-creation methods must be named create(...), and each create(...) method in an entity bean's local homeinterface corresponds to an ejbCreate(...) method with the same parameter set in the bean class.

You can write create(...) methods that fill in all, some, or none of the fields in an entity bean. There is one default finder method, findByPrimaryKey(key), that must be defined in the local homeinterface. Definition of others is optional; their names must begin with the word "find." A local home interface can also define bean-removal methods.

Each create(...) method in the local homeinterface can throw javax.ejb.CreateException, which indicates that the creation
failed. Each finder method can throw javax.ejb.FinderException to indicate that the lookup failed.

Errand Home Example

```
public interface ErrandLocalHome extends javax.ejb.EJBLocalHome {

  Errand create(int argErrandId)
     throws javax.ejb.CreateException,;

  Errand create(com.goforit.errand.ErrandDataBean errand)
     throws javax.ejb.CreateException;

  java.util.Collection findByUserId(String userId)
     throws  javax.ejb.FinderException;

  Errand findByPrimaryKey(ErrandKey key)
     throws javax.ejb.FinderException;

  Errand findNewestErrand()
     throws javax.ejb.FinderException;
}
```

IBM Developer Relations DragonSlayer Team

This is the complete local home interface for the GoForIt Errand entity bean.

Note that all create(...) methods throw the same exceptions and all the finder method throw the same exceptions.

The home interface has two create(...) methods, one that takes only the primary key of the ERRAND table (errand id) and one that takes an ErrandDataBean object.

There are three finder methods, the default findByPrimaryKey and two application-specific ones- findNewestErrand and findByUserId. The first two return an Errand object.

findByUserId(...) returns a Collection of the remote interfaces of all the Errands that are requested by a given user.

The key thing to remember is that this interface specifies what needs to be done, not how. The container or the bean determines the how.

Finder Methods

- A local home interface defines one or more finder methods to look up EJBLocalObjects
  - Every home interface must define the method findByPrimaryKey()
  - Beans can offer lookup by other criteria
- The name of a finder method always starts with the word "find"
  - Can return the local-interface type or a collection of local-interface types

IBM Developer Relations DragonSlayer Team

The findByPrimaryKey() method is automatically generated by the "Create EJB" wizard from WSAD. You must implement any other finder methods, if used.

Finder Method Implementation

- The container cannot fully implement finders other than findByPrimaryKey
  - In CMP beans, you must provide EJB QL in the deployment descriptor to enable the container building the necessary helper class(es)
  - In BMP beans the finders are implemented by the bean provider

IBM Developer Relations DragonSlayer Team

A local home interface can define finders other than the default method, findByPrimaryKey.

The container can implement the default method, but additional methods require an understanding of the semantics of the application. The EJB 2.0 specification describes a variant of SQL called EJBQL to tell the container how to implement the finder. EJBQL is defined in terms of the EJB's methods so that the finders will be completely independent of the underlying database schema

Implementing a Finder

- Suppose we defined a finder method for an Errand bean called findByUserId(String userId)
  - The container can't implement this without a hint
  - In EJB 2.0 you provide the hint in the EJB's deployment descriptor as an EJB QL statement

```
select object(o) from Errand o where o.requester = ?1
```

  - So when the finder is invoked (i.e. findByUserId("joeg")) the container will execute an SQL query like:

```
SELECT * FROM ERRAND WHERE USERID='joeg'
```

IBM Developer Relations DragonSlayer Team

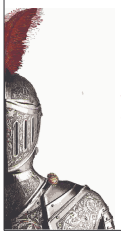
The container implements the default method (findByPrimaryKey) but additional methods require an understanding of the semantics of the application, which means we must provide a hint to the container.

In this example our finder method, findByUserId(), takes one argument and we put that argument into the query string by using a question mark. These will be substituted, from left to right, with values from the argument list when the method is called.

Local Interfaces for Entity Beans

- The local interface of an entity bean:
  - Allows a client to invoke the business methods of the bean
  - Extends the EJBLocalObject class
- The Errand local interface

```
public interface Errand extends EJBLocalObject {
   public String getStatus();
   public void setStatus(String newStatus) ;
   .
   .
   .
}
```



IBM Developer Relations DragonSlayer Team

The second component of an entity bean is the **local interface**.

A local nterface defines the business methods that will be implemented in the bean class. It extends the EJBLocalObject class, and each method within the interface must declare any application specific exceptions that it throws- this example does not.

Writing the Bean Class

- Implements javax.ejb.EntityBean
- Declare the CMP fields as abstract getter/setter methods
- Implements:
  - Any lifecycle management methods the bean needs
  - The business methods defined in the local interface
  - Any other methods the bean requires

IBM Developer Relations DragonSlayer Team

The third major component of an entity bean is the **bean class itself**. This is where the programmer-supplied code resides. The bean class extends EntityBean (rather than Session Bean) and implements functions that support both the home/local home and remote/local interfaces and provides any internal helper functions that are needed by the bean class itself.

The bean class is also where the bean's CMP fields are declare as abstract getter/setter pairs. As you will see later, there is nothing in the declaration that indicates whether the variables are persistent or not. The deployment descriptor allows you to indicate the variables subject to container-managed persistence; any others are considered by the container to be ephemeral. If you wish to make them persistent explicitly in the bean (bean-managed persistence), you can, but the container will not manage persistence of those variables for you.

CMP fields in the Bean Class

```
public class ErrandBean implements EntityBean {
    // CMP Fields
      /**
       * Get accessor for persistent attribute: id
       */
      public abstract java.lang.Long getId();
      /**
       * Set accessor for persistent attribute: id
       */
      public abstract void setId(java.lang.Long newId);
      /**
       * Get accessor for persistent attribute: category
       */
      public abstract java.lang.String getCategory();
      /**
       * Set accessor for persistent attribute: category
       */
      public abstract void setCategory(java.lang.String newCategory);

      ...
}
```

IBM Developer Relations DragonSlayer Team

This excerpt from the bean class shows that the bean class implements the javax.ejb.EntityBean interface. It also shows theCMP fields declared in the bean class as abstract getter/setter pairs. Notice that there's nothing here that indicates which of these are to be container-managed for persistence.

Indicating Persistent Fields

- The container needs to know which fields in each bean it will manage
  - This is done in the deployment descriptor
  - Nothing in the bean class declaration indicates persistence
- This information is used to generate the deployment classes for persistence

IBM Developer Relations DragonSlayer Team

Lifecycle Methods

- The bean-creation methods, ejbCreate(...) and ejbPostCreate(...)
  - Correspond to create(...) methods in the local home interface
  - Argument list must match across triads
  - Initialize persistent variables in ejbCreate(...)
- The bean-management methods, for example, ejbActivate(), ejbPassivate(), ejbLoad() and ejbStore()

IBM Developer Relations DragonSlayer Team

The lifecycle methods in the bean class consist of creation and lifecycle management.

In the local home interface, you defined create(...) methods. In the bean class, you will define a pair of methods that correspond to each create(...) method, called ejbCreate(...) and ejbPostCreate(...). The argument lists for each triad of create, ejbCreate and ejbPostCreate must match. Of course, the argument lists across triads can vary.

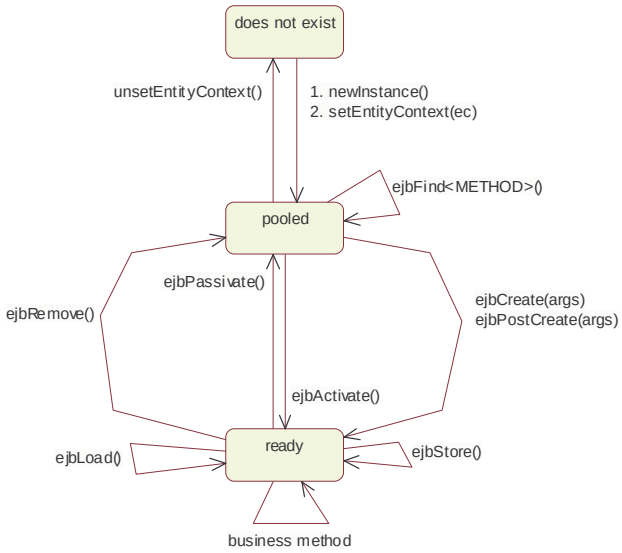
For Errand, we have two create(...) methods in the home interface, so we will have two corresponding pairs of ejbCreate and ejbPostCreate in the bean class.

Any variables that are subject to container-managed persistence should be initialized in the ejbCreate(...) methods. After calling ejbCreate, the container insert the values into the persistent store, and then it calls ejbPostCreate. In most cases, ejbPostCreate(...) will probably be an empty method.

If you are using a development tool like WSAD, it will generate the necessary empty methods for you, so you will generally be able to ignore the ejbPostCreate methods.

The other kind of lifecycle methods you must provide in the bean class are those inherited from the EntityBean interface: ejbActivate, ejbPassivate, ejbRemove, ejbLoad, ejbStore, setEntityContext, and unsetEntityContext. Again, a good development tool will generate empty methods, which you can modify as necessary. In many cases, the only ones you will want to modify are the two dealing with contexts.

Entity State Diagram



IBM Developer Relations DragonSlayer Team

Business Methods

- The bean class also implements business logic
  - The local interface defines the methods
  - The bean class implements the methods
- The bean class can also implement private methods (for example getters and setters)
  - Not defined in an interface
  - Not directly available to clients
  - Used by the bean to do application-specific work

IBM Developer Relations DragonSlayer Team

In addition to the lifecycle-management methods (which support the local home interface), the bean class implements the business methods defined in the local interface as well as any private methods that the bean class needs. These methods are not part of any interface publicized to clients, but they may be used in implementing the methods of the interfaces.

Primary Key

- The bean must designate a field (or fields) to act as a primary key
- The primary key:
  - Must uniquely identify a bean instance
  - Must be serializable
  - Must offer equality and hashing methods
    - equals(Object o);
    - hashcode();
  - May be supplied in a class

IBM Developer Relations DragonSlayer Team

This is the last aspect of writing an entity bean: providing a **primary key**.

Entity beans, unlike session beans, are uniquely identifiable, and the primary key helps identify a bean. So far, the components of entity beans have matched those of session beans (home and remote interfaces, bean classes), but session beans do not have primary, or any other, keys.

A PrimaryKey class must implement the Serializable interface and the methods equals(Object) and hashCode(). These two methods are needed to implement the isIdentical() method of the EJBObject class. Two beans are considered identical if they have the same primary key and the same home interface.

In practice, your PrimaryKey class will probably be generated by WSAD, and then you will modify it if necessary. Note that you don't really need a primary key class if you choose as your primary key a single field whose type already is serializable and implements equals() and hashCode() methods. For example, a field of type String would work as an out-of-the-box primary key. UserEJB uses an integer field for its customer key, and VisualAge generates a simple primary key class for it.

Data Mapping with CMP

- Container-managed persistence supports various bean-to-data relationships
  - The most basic approach maps each persistent instance variable to a column in a table
    - Each bean maps to a single table
    - The table is created during deployment
    - Each field maps to a simple, fixed SQL type
  - This is the most common, simple support offered by most EJB containers

IBM Developer Relations DragonSlayer Team

When the container manages persistence for you, it does so by using a simple mapping of instance variables to SQL data types. If portability is important, it is best to use these mappings if possible. These beans can be deployed into any EJB server.

Another option for more complex mapping than this is writing the persistence code yourself, using bean-managed persistence. This code will be tied closely to the backing data store, which may limit portability. It is also much harder to write beans with BMP than those with CMP.

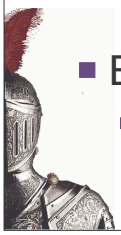
Container Managed Relationships

- Allows multiple entity beans to have relationships among themselves
- Container implements and supports the relationship
  - One-to-One, One-to-Many and Many-to-Many relationships
  - Uni- and bi-directional relationships
  - Can model RDMS foreign key relationships
  - CMR relationships are defined in the deployment descriptor
    - Defined in terms of the local interfaces of the related beans

IBM Developer Relations DragonSlayer Team

ejbSelect... methods

- Similar to finders in that they are described using EJB QL in the deployment descriptor
- Declared as abstract in the Bean class
- Can return interfaces, collections or a single CMP field
- Not exposed to the client, can only be used in the Bean class



- Example
  - select o.id from Errand o where o.id > 10

IBM Developer Relations DragonSlayer Team

ejbHome ... methods

- Methods on the local or remote home that are not specific to a particular instance
  - Saves clients the additional step of getting the remote interface
  - Must be declared as public
  - Must not be declared as static

IBM Developer Relations DragonSlayer Team

Summary

- We've seen:
  - The basics of entity beans with local interfaces
  - Customer finders with EJB QL
  - Declaring CMP fields
  - ejbSelect... and ejbHome... methods

IBM Developer Relations DragonSlayer Team

Developing EJBs using WebSphere Application Developer

dragonSlayer Team

EJB Development Environment

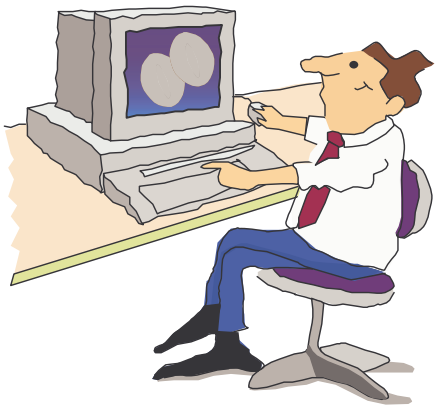
- Application Developer provides an EJB development environment that includes:
  - Wizards for creating entity beans, session beans and message driven beans
  - Management of dependencies between the EJB implementation classes and the home/remote and local home/local class interfaces
  - Generation of EJS Deployment code
  - Universal Test Client
  - Visual Editor for EJB Deployment Descriptor
  - Product CD contains the WebSphere Embedded JMS Provider for testing Message Driven Beans
  - Support for both EJB 1.1 and EJB 2.0 development

IBM Developer Relations DragonSlayer Team

EJBs are no different from other objects. You take advantage of the integrated nature of WSAD as you edit, compile, and debug new EJBs.

Developing EJBs in Application Developer

- Create EJB Project
- Create EJB
- Code methods in generated bean
- Promote methods to local/local home and/or remote/home interfaces
- Generate Deployment classes
- Test EJB



IBM Developer Relations DragonSlayer Team

EJB Project

- An EJB Project corresponds to an EJB jar file used to deploy into an Application Server
  - Can contain one or more EJBs
  - Is contained within an Enterprise Application
  - Contains a deployment descriptor

IBM Developer Relations DragonSlayer Team

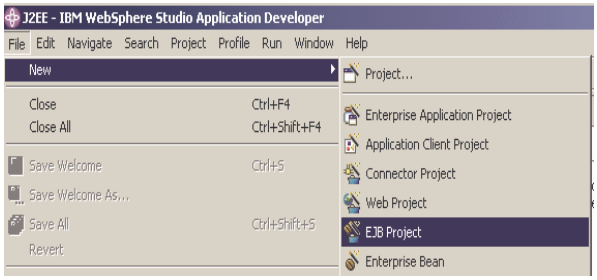
WSAD has a special editor for editing the deployment descriptor. This allows developers to make changes without knowing the details of the XML Schema that is defined for a DD in the EJB specification.

Creating EJB Projects

- Select

  File->New->EJB Project

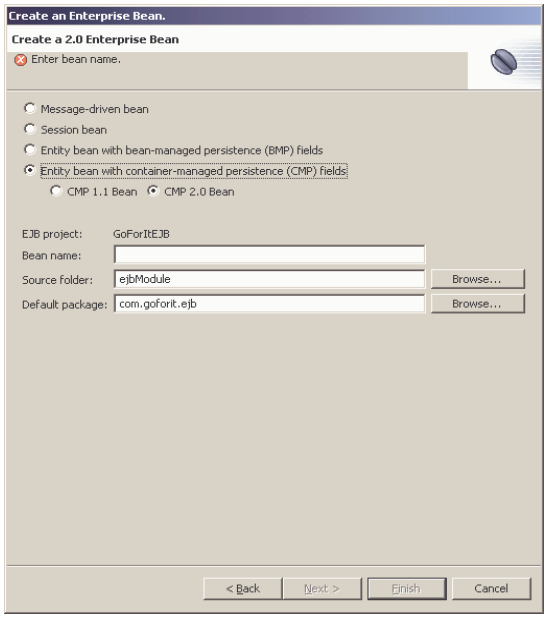
  from the Application Developer menu



IBM Developer Relations DragonSlayer Team

Adding EJBs

- Select **File-> New -> Enterprise Bean** from the main menu
- This brings up a wizard that creates a Session Bean or a Message Driven Bean or Entity Bean including:
  - A Bean class
  - A Local Home and/or Home Interface
  - A Local and/or Remote Interface
  - Container managed fields for CMP bean



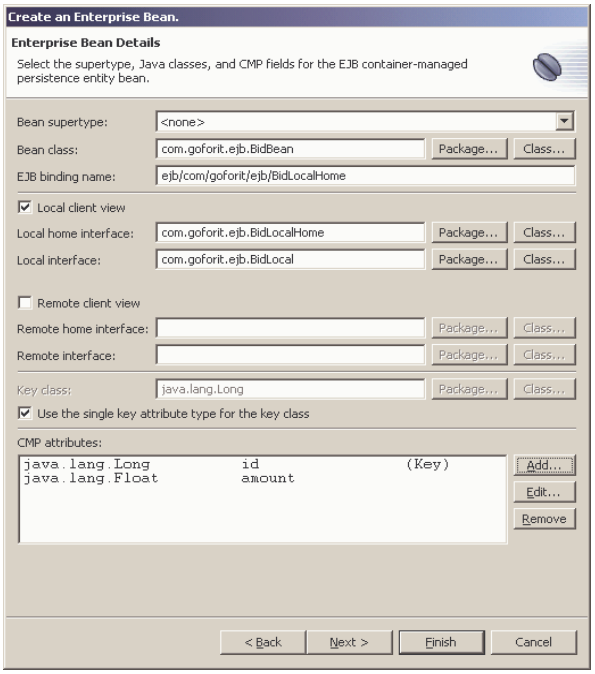
IBM Developer Relations DragonSlayer Team

This first page allows you to name a bean, set the bean type, and choose a project and package for the generated code.

Defining Bean Class Attributes and Interfaces

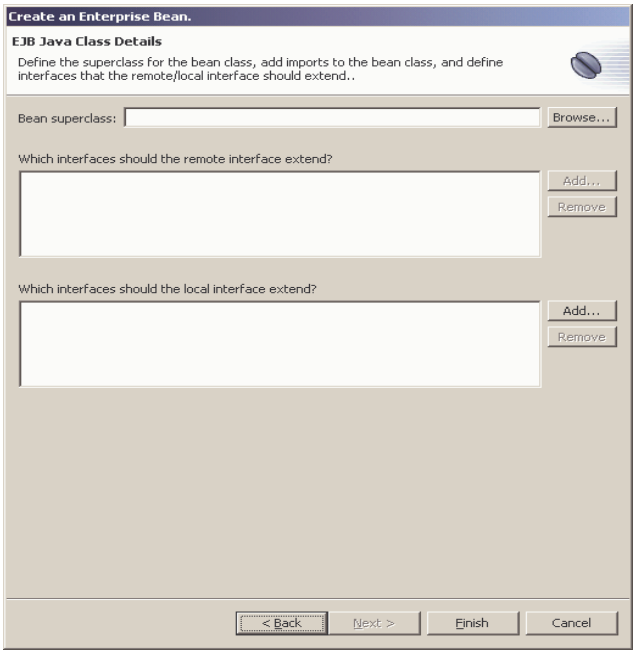
- Validate names for home/remote and/or local home/local interfaces
- Add CMP fields if needed
- Indicate superclass if using inheritance

IBM Developer Relations DragonSlayer Team



Defining Bean Class Interfaces and Imports

- Define other interfaces the bean should implement

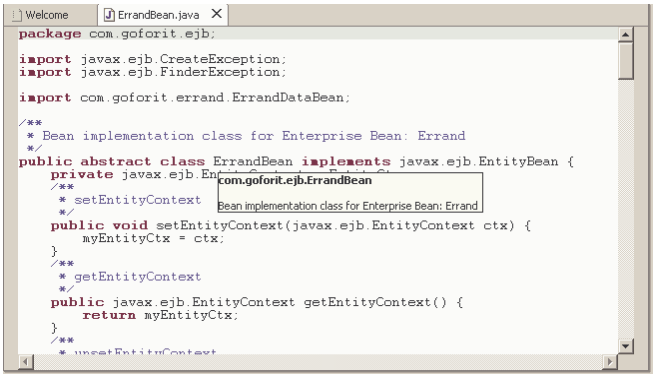


IBM Developer Relations DragonSlayer Team

You can also add import statements and add additional interfaces for your Bean class -- note that you DO NOT need your bean class to implement your Remote interface -- the framework handles that. When you press "Finish", the classes and interfaces you have chosen will be generated by the wizard and you will see them displayed in the Navigator view in WSAD.

Adding methods and fields

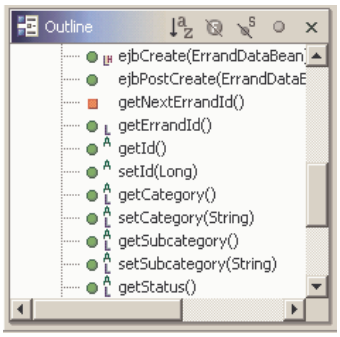
- The next step is to write your business logic
  - Add methods to the Bean class
  - Use the standard Java Editor to do this



IBM Developer Relations DragonSlayer Team

Promoting Methods

- The EJB Tooling is centered around the Bean class
  - Clients only see the Remote and Home Interfaces or the Local or LocalHome Interfaces
- Write methods in the bean class and promote them to the Home, Remote, LocalHome or Local Interface



IBM Developer Relations DragonSlayer Team

For remote clients, promoting works in two ways:
(1) If you have a regular method that you want to declare in a Remote interface, just use the "Enterprise Bean->Promote to Remote Interface" menu selection in the Outline view

(2) If you want to create a new create() method in your Home Interface, start by adding a corresponding ejbCreate(your parameters here) method in the bean class. Then select "Enterprise Bean->Promote toHome Interface" and the create() method will be added.

Note that these two selections are mutually exclusive based on the method you have selected.
Once a method has been "promoted" the "promotion" icon (an H or an R) will be displayed to the left of the method name.

This is the way you want to build Remote and Home Interfaces -- you should probably not try to work backwards from the Remote or Home Interface to the Bean Class. However, if you do this, warnings will show you what methods need to be implemented.

For local clients, promoting also works in two ways:
(1) If you have a regular method that you want to declare in a Local interface, just use the "Enterprise Bean->Promote to Local Interface" menu selection in the Outline view

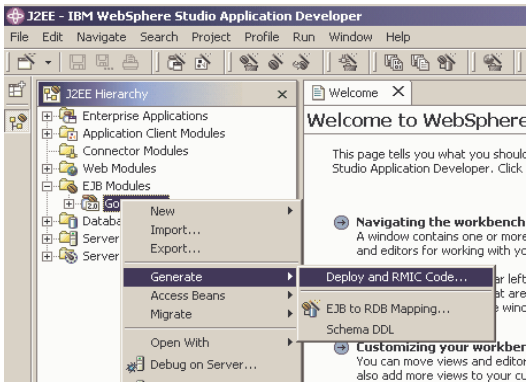
(2) If you want to create a new create() method in your Local Home Interface, start by adding a corresponding ejbCreate(your parameters here) method in the bean class. Then select "Enterprise Bean->Promote to Local Home Interface" and the create() method will be added.

Note that these two selections are mutually exclusive based on the method you have selected.
Once a method has been "promoted" the "promotion" icon (an LH or an L) will be displayed to the left of the method name.

This is the way you want to build Local and LocalHome Interfaces -- you should probably not try to work backwards from the Local o rLoca Home Interface to the Bean Class. However, if you do this, warnings will show you what methods need to be implemented.

Generating deployment code

- After finishing your EJB code, generate the deployment code
  - This will include all the stubs and EJS classes
  - Use "Generate Deploy and RMIC Code" from the J2EE or Navigator View context menu
  - Errors that prevent generation will show up in the Tasks view



IBM Developer Relations DragonSlayer Team

Testing your EJBs

- Application Developer includes all of the WebSphere 4.0 and 5.0 EJS code
  - You can start the processes and an EJS server within Application Developer
  - All accessible through Server Configuration pane in the Server Perspective
- When you create a new Enterprise Application you can add it to a server
  - Use "Add" from the Server Configuration pane in the Server Perspective

IBM Developer Relations DragonSlayer Team

Starting servers

- Servers can be started implicitly by selecting the EJB Module in the J2EE View of the J2EE Perspective and selecting "Run on Server"
- Can be started/stopped explicity in the Servers pane of the Server Perspective

IBM Developer Relations DragonSlayer Team

Since all of the EJB Projects will be deployed to the same naming servic in the Test Environment, you can have session beans from one EJB Project use entity beans in another project for instance.

The first option starts the Test Client also. The second option doesn't.

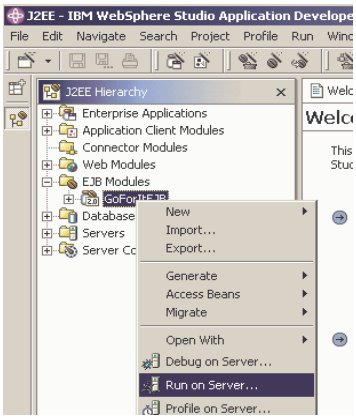
EJB Tests

- You can either run your own EJB client or use the Application Developer test client
  - This is a "default client" that allows you to send messages to an EJB.
    - Written as a Web Application
    - You use a browser based UI
- In either case watch the Server Console for messages from the EJB server

IBM Developer Relations DragonSlayer Team

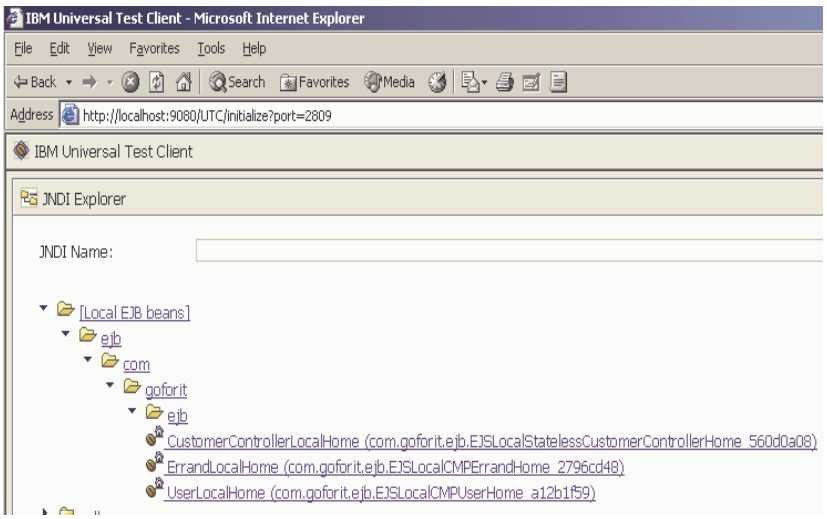
The Test Client

- To run the test client use "Run on Server" from the J2EE Hierarchy or Navigator view of the J2EE Perspective



IBM Developer Relations DragonSlayer Team

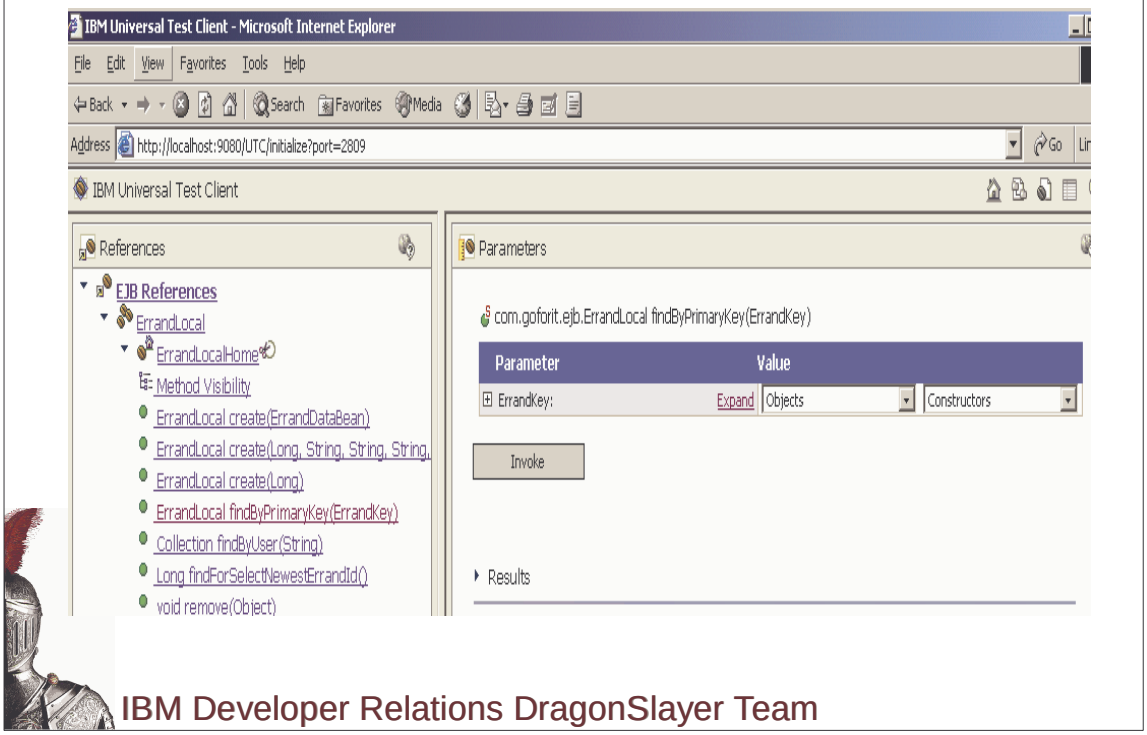
Testing the Connection



IBM Developer Relations DragonSlayer Team

When you start the test client for an EJB Project (as opposed to an individual EJB) the browser . comes up with the Test Client homepage. Clicking on the JNDI Explorer link will show you a dump of the JNDI namepace and allow you to connect to an individual EJB by clicking on its JNDI entry.

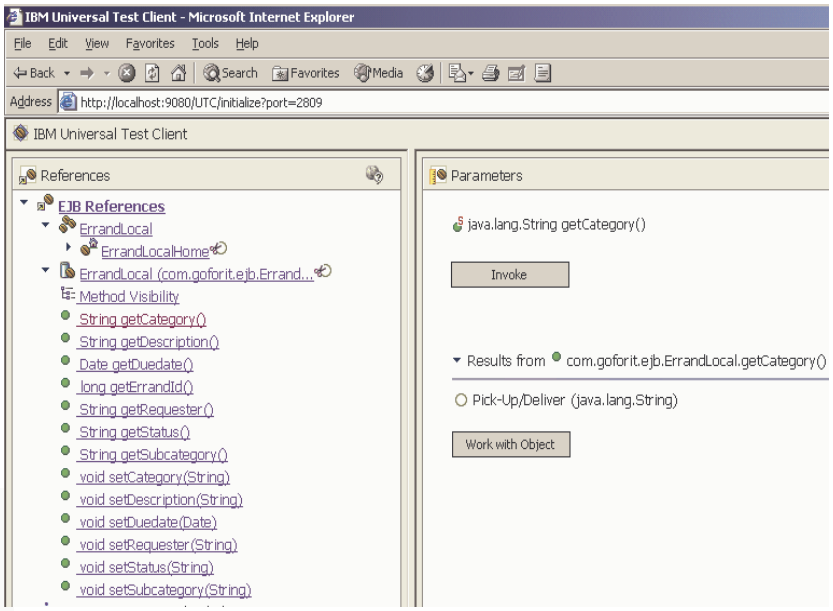
Testing the Home/Local Home Interface



IBM Developer Relations DragonSlayer Team

The methods in your Home/Local Home interface (create(), remove(), finders) will be shown under **EJB Reference**. You fill in parameters in the Parameters frame and then click on "Invoke". Once you have successfully executed a method on the Home/Local Home, you can click on "Work with Object" and the returned Remote/Local Interface(s) will show up on the left under EJB References.

Testing the Local/Remote Interface



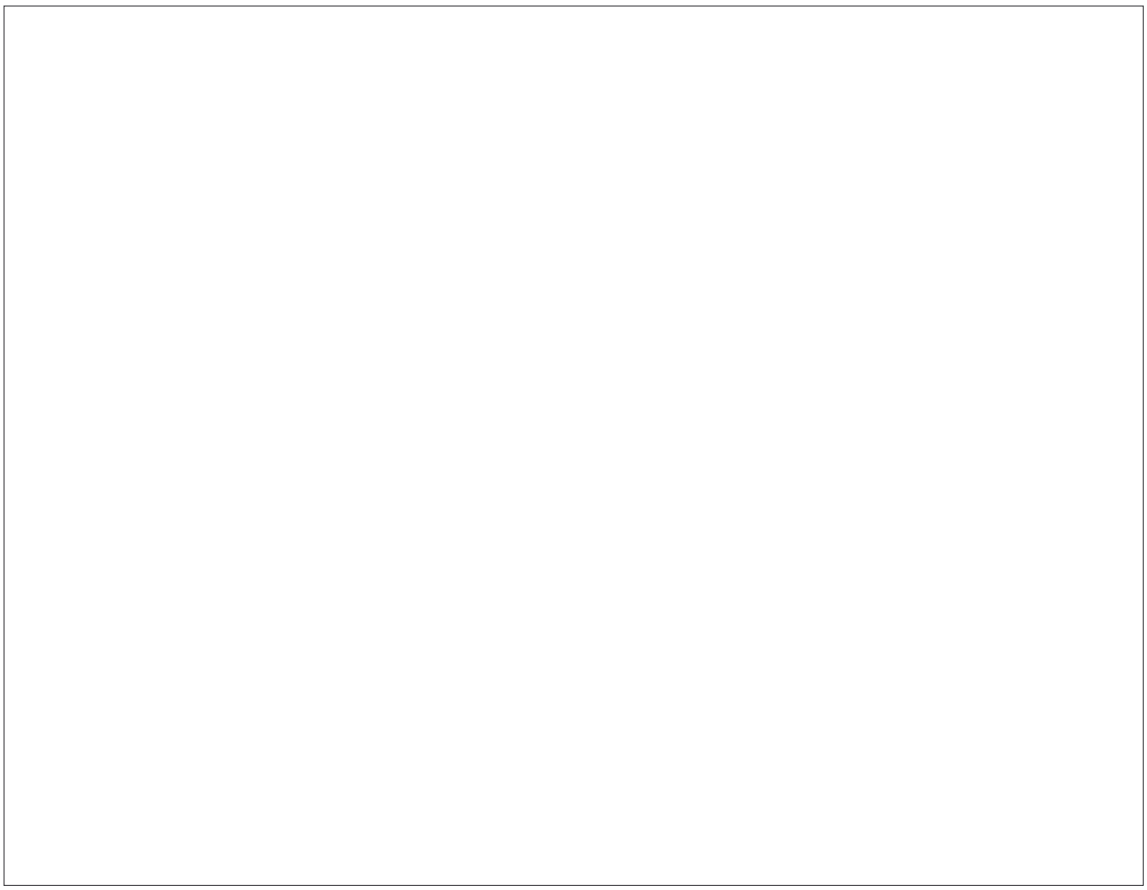
IBM Developer Relations DragonSlayer Team

Again, in this page you can select a method, and then add parameters and send the message. If you send a message that returns a primitive the value will be shown under Last results. If the method call returns a class, click on "Work with Object" and the object will appear on the left under Object references.

Summary

- We've seen:
  - Creating EJB Projects
  - Writing EJBs
  - Deploying EJBs into the WebSphere Test Environment
  - Testing EJBs using the Test Client

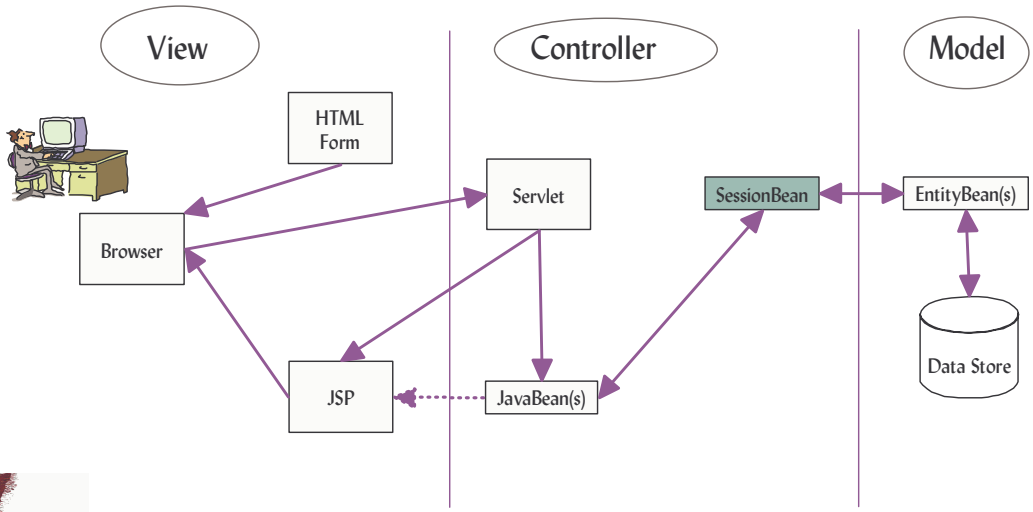
IBM Developer Relations DragonSlayer Team



Session Beans and Message Driven Beans

dragonSlayer Team

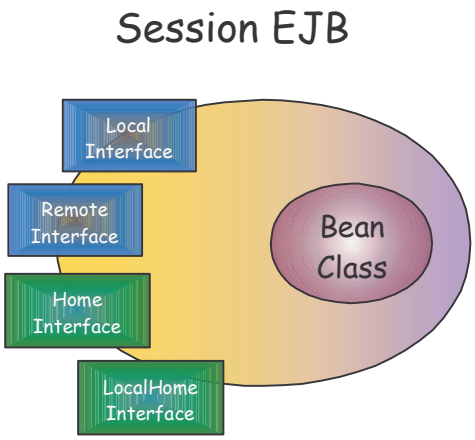
Session Beans



IBM Developer Relations DragonSlayer Team

Session Beans

- Session beans come in two flavors
  - Stateless
  - Stateful
- In any case, a session bean is not persistent
  - Not saved in database



IBM Developer Relations DragonSlayer Team

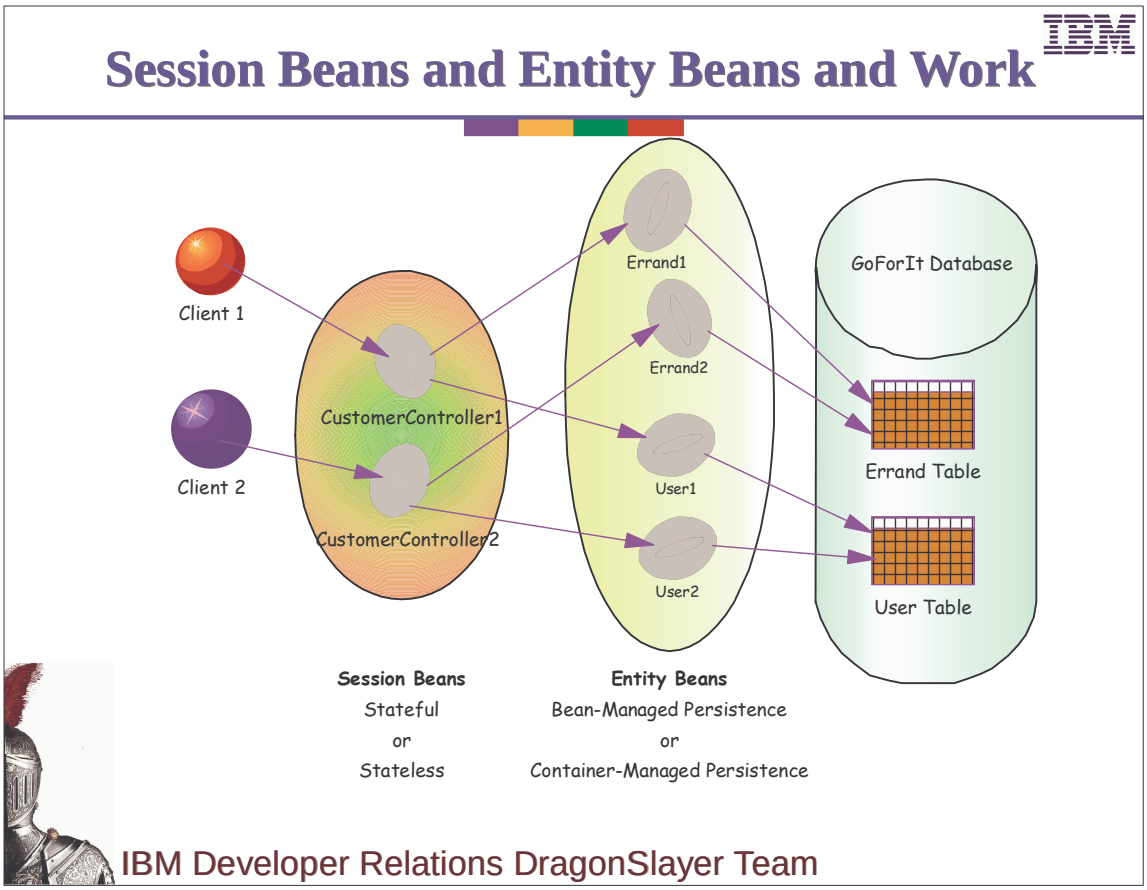
Session Beans as Clients

- A session bean is often a client of an entity bean
- Can manipulate several entity bean instances using the local interface of the entity bean instances
- Simplify client interface by doing transactions across different entity bean instances

IBM Developer Relations DragonSlayer Team

A **session bean** typically contacts one or more **entity beans**, which manipulate recoverable data on its behalf. Thus, they are clients of the entity beans just like a Java client or a servlet is a client of the session bean.

When clients are remote, Session-bean access to entity beans is faster than client access since session beans live on the server. Session beans can do complex manipulations of several entity-bean instances using local interfaces in a single session-bean method.



Now let's look a little bit closer at the issue of persistence.

As we saw in previous slides there are actually several "flavors" of EJBs and each exhibit different persistence characteristics depending on deployment tradeoffs and the underlying requirements of the business domain.

In this example, we have a couple of EJBs that we call CustomerController1 and CustomerController2. They are instances of a Session Bean called CustomerController.

Notice that we have two clients, each communicating with a different CustomerController bean. These clients send requests to the CustomerController beans to create errands on behalf of the clients. The CustomerController beans then communicate with the appropriate Errand and User entity beans to manage errands and users.

Stateless Session Beans

- Used for task oriented objects where:
  - All parameters required are supplied with method calls
  - No data is required to be kept between method calls
- The container can:
  - Choose any available instance to execute a method
  - Create and remove methods as needed for current load

IBM Developer Relations DragonSlayer Team

The idea is that **stateless session beans** of any type are all equivalent.

Any method invocation can run on any available instance. The container keeps a pool of several instances of a particular session bean class. When a method call comes in, an instance is selected out of the pool and used.

The size of the pool and how long each instance remains in the pool is determined by the container; this is all transparent to client code. You should not care when or if this happens.

Building Stateless Session Beans

- The bean should implement the javax.ejb.SessionBean interface
- The bean should have a single ejbCreate() method with no parameters
- You mark the bean as STATELESS in the deployment descriptor

IBM Developer Relations DragonSlayer Team

When using Application Developer to build stateless session beans you can mark the bean as STATELESS in either the properties sheet for the bean, or in the create-a-bean wizard. This information becomes part of the deployment descriptor.

Stateful Session Beans

- Used for task-oriented objects where state needs to be kept between invocations
- The container is responsible for:
  - Routing a method invocation to the appropriate instance
  - Serializing non-transient state variables when passivating
  - Restoring non-transient state when activating

IBM Developer Relations DragonSlayer Team

When a client has a stateful session bean, it is guaranteed that, if it sends a message to that bean, then all method invocations will go to the same bean. This is handled by the EJB framework; your client code does not have to do anything special to guarantee that this happens.

Activation and passivation happen when the container decides that it is holding "too many" session beans in memory. The container implements some policy that says (in effect) that if a bean has not been used (received a message) within a certain amount of time, it is "passivated" or stored to persistent storage (for example, to a serialized file).

Implementing a Session Bean

- Implementing a session bean involves
  - Implementing a SessionBean interface
  - Implementing your business logic
  - Implementing the bean's lifecycle methods
- Once you are done, you can deploy and go!

IBM Developer Relations DragonSlayer Team

The lifecycle methods are methods defined in the SessionBean interface. You need to implement all of them, but what you do in each method depends on the bean; in many cases, these methods will be empty. The next two slides discuss these methods in more detail.

Lifecycle Methods for Session Beans

- ejbActivate()
  - called when bean is activated
- ejbPassivate()
  - called when bean is passivated
- ejbRemove()
  - called when bean is destroyed
- setSessionContext(SessionContext ctx)
  - Called by container to give the bean a context

IBM Developer Relations DragonSlayer Team

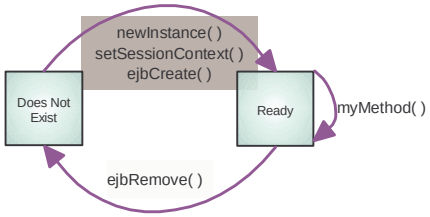
The ejbActivate() and ejbPassivate() methods must be implemented, but they do not need to do anything.

When a bean is selected for passivation, the ejbPassivate() method will be called in the bean before the bean is serialized to a file. Likewise, after a bean is loaded from a serialized file, the ejbActivate() method is called in that bean.

These methods are generally used to set up and tear down instance variables that cannot be serialized, either variables that contain transient information (like caches) or non-serializable objects (like AWT objects or TCP connections).

Stateless Session Bean Lifecycle

- Does not exist
  - newInstance(): matched default constructor
  - setSessionContext(sc): may be empty if context not needed
  - ejbCreate(); handles one time initialization of resources
- Ready (able to handle method calls)
  - myMethod(...): implementation of remote bean methods
  - ejbRemove(): releases resources obtained on create



IBM Developer Relations DragonSlayer Team

Creating Stateful Beans

- You can create a stateful bean with multiple create( ) methods in the home interface
  - Use any number of parameters
- Each create( ) method in the home or local home interface corresponds to an ejbCreate( ) in the bean, for example,
  - public Counter create(int arg1) // defined in home
  - public void ejbCreate(int arg1) // defined in bean

IBM Developer Relations DragonSlayer Team

The Counter bean is a stateful session bean that comes with WebSphere as part of the example code. It simply keeps a count for its client.

Passivation/Activation

- An EJB Server has the right to manage its working set
  - Passivation: saves state of a bean to persistent storage, then swaps it out
  - Activation: restores state of a bean from persistent storage
- Not necessary for stateless session beans

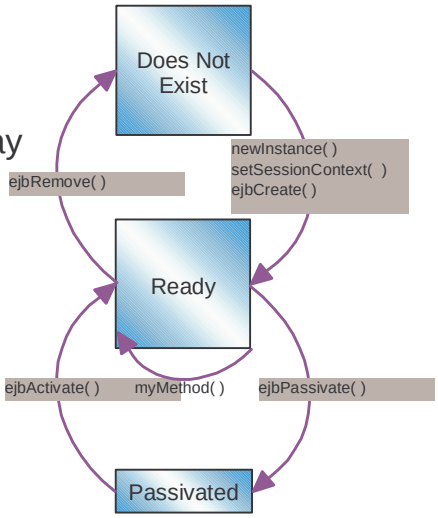
IBM Developer Relations DragonSlayer Team

Stateful session beans are one of the reasons we need to be able to swap beans in and out; they take up room since you must have many instances (one per client) running around. You don't want them all in memory at once!

This is known as the "conversational state" of the Session bean.

Stateful Session Bean Lifecycle

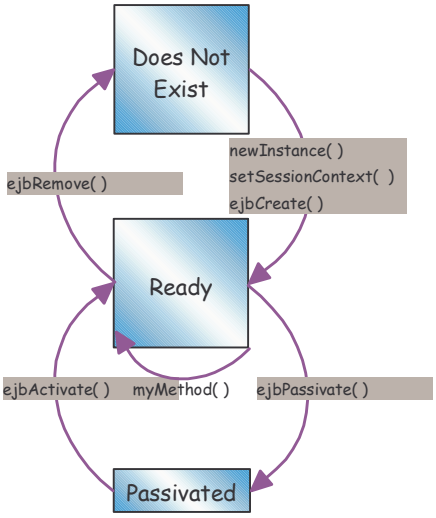
- Does not exist
  - newInstance(): matched default constructor
  - setSessionContext(sc): may be empty if context not needed
  - ejbCreate(); handles one time initialization of resources



IBM Developer Relations DragonSlayer Team

Stateful Session Bean Lifecycle (continued)

- Ready (able to handle method calls)
  - <<method>>(...): implementation of remote bean methods
  - ejbRemove(): releases resources obtained on create



IBM Developer Relations DragonSlayer Team

Message Driven Beans

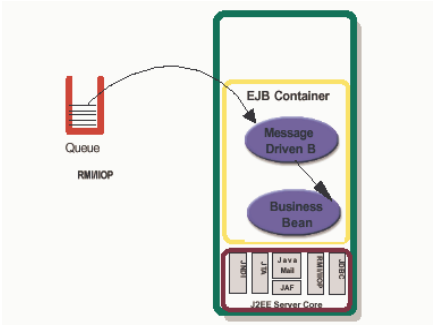
- MDB Basics:
  - Stateless enterprise beans, server side components
  - Transactional
  - Point-to-point and Pub/Sub supported
  - No remote interface, no remote home
    - Container activates MDBs as needed
- Bean Provider responsibilities
  - Implement javax.jms.MessageListener interface
    - onMessage(msg) method performs necessary message processing actions
- Application Deployer responsibilities
  - Associate bean with JMS destinations at deployment
    - Deployment descriptor holds association information

IBM Developer Relations DragonSlayer Team

Message-driven beans (MDBs) are stateless, server-side, transaction-aware components for processing asynchronous JMS messages. It supports the two messaging models namely, Point-to-Point and Publish/Subscibe.
A message-driven bean is a complete enterprise bean, just like a session or entity bean, but there are some important differences. While a message-driven bean has a bean class and XML deployment descriptor, it does not have component interfaces. The component interfaces are absent because the message-driven bean is not accessible via the Java RMI API; it responds only to asynchronous messages.
It's important to understand that MDBs are not for "client use". There fore they have no interface and no home, they cannot be "looked up" by a client.
The Bean Provider writes the application code for the Message Driven Bean. All Message driven beans must implement the javax.jms.MessageListener interface. The onMessage method contains the business logic that handles the processing of the messages. This method is called by the container when a message has arrived for the bean to service.
It is the Application Deployer's responsibility to associate the Message driven bean with the appropriate JMS destinations. This association is done at deployment time.

Benefits of Message Driven Beans

- Automatic consumption of messages
  - No polling needed in the application code
- Reduce application code
- Leave JMS resource management to the container
  - Configuration of JMS destinations and providers

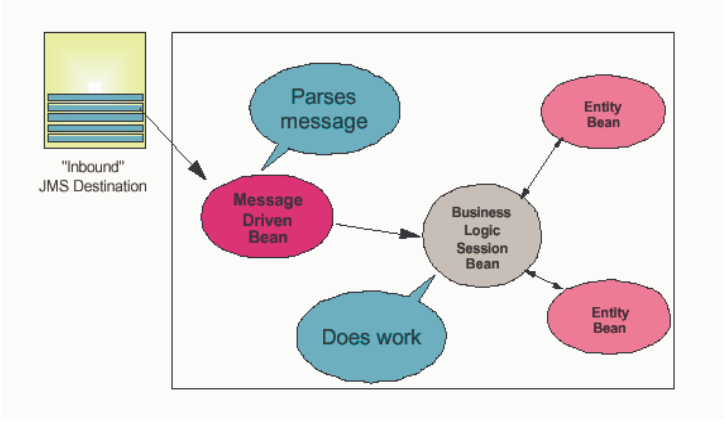


IBM Developer Relations DragonSlayer Team

Message Driven Beans offer a standard way to create a message consumer that is fully managed by the container. The bean provider only needs to concentrate on writing the logic that performs the parsing and processing of the message. Typically, the MDB will delegate the execution of business logic to some other EJB - a Session EJB in most cases. However, no coding needs to be done to retrieve the message or poll the JMS destination - no specific coding is needed to provide quality of service (failover, parallel sessions, etc.) - all this is up to the container to implement and provide.

Message Processing and Business Logic

- Provide separation between message processing and business logic



IBM Developer Relations DragonSlayer Team

By providing a clear separation between message and business processing, it is easier to implement the Message Driven Beans. Ideally, the Message driven bean parses the message and then delegates the work to be done to a Business logic session bean. This design pattern promotes components reusability because the business logic session bean can be used by a variety of other clients.

MDB onMessage() example

```
public void onMessage(javax.jms.Message msg) {
    try {
        // Retrieve message
        ObjectMessage message = (ObjectMessage) msg;
        ErrandDataBean errand = (ErrandDataBean)message.getObject();

        // Perform business logic
        CustomerControllerLocal customerController =
            customerControllerHome.create();
        customerController.insertErrand(errand);

    } catch (Exception e) {
        throw new EJBException(e.getMessage());
    }
}
```

IBM Developer Relations DragonSlayer Team

This is an example of a Message Driven Bean's onMessage method. The message driven bean parses the message and then delegates the work to a business logic EJB.

Summary

- You've seen:
  - The basics of session beans
  - The difference between stateless and stateful beans
  - The basics of Message Driven Beans

IBM Developer Relations DragonSlayer Team

Workload Management

dragonSlayer Team

Workload Management (WLM)

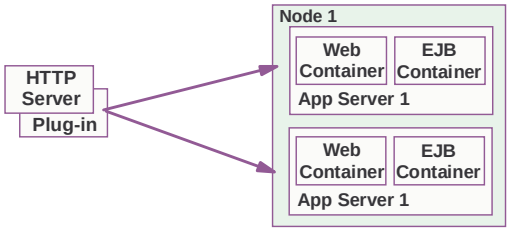
- What is WLM?
  - Sharing of work over multiple resources
  - Improves performance, scalabilty and reliability
  - Provides failover capability
  - Can be on single machine or across multiple machines
  - Centralizes administration

IBM Developer Relations DragonSlayer Team

WLM is only available in the Network Deploy version, not the Base version

Vertical Scaling

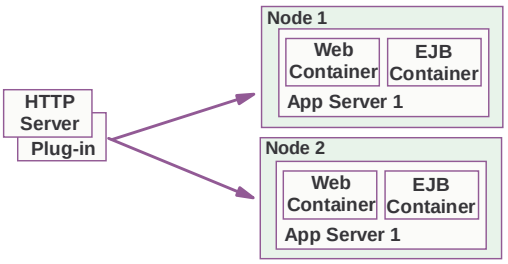
- Vertical Scaling
  - Multiple copies of an application server on the same physical machine
  - Allows for more efficient allocation of machine's processing power



IBM Developer Relations DragonSlayer Team

Horizontal Scaling

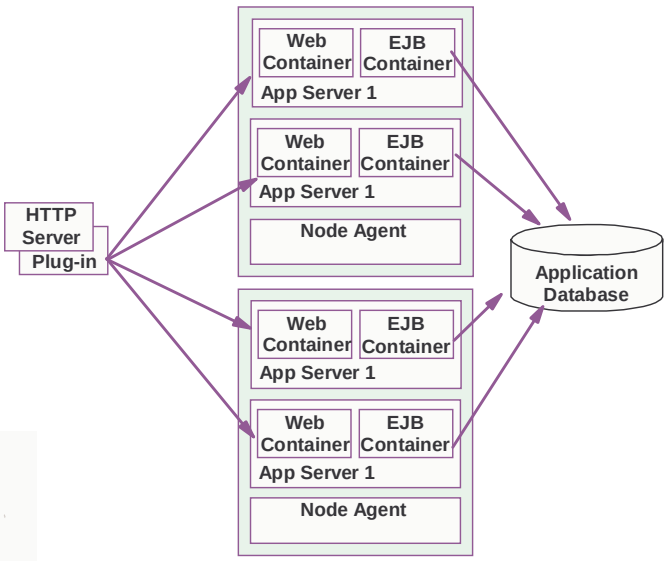
- Horizontal Scaling
  - Copies of an application server on multiple physical machines
  - Useful in an environment with several smaller, less powerful machines
  - Failover support



IBM Developer Relations DragonSlayer Team

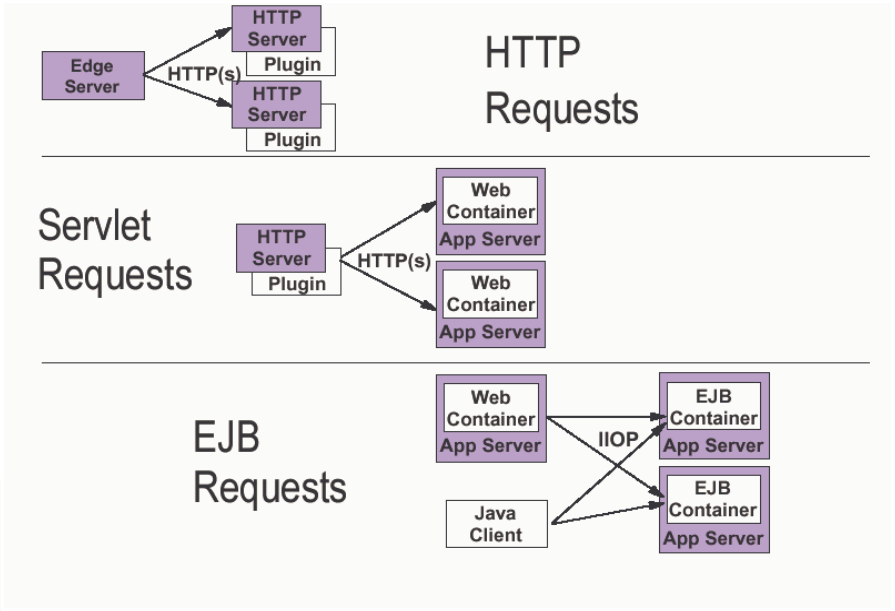
Horizontal scaling is restricted to machines on the same platform
Vertical and horizontal scaling can be combined for even better performance

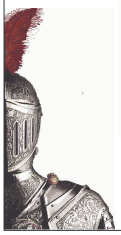
Vertical and Horizontal Scaling



IBM Developer Relations DragonSlayer Team

What can be Workload Managed ?



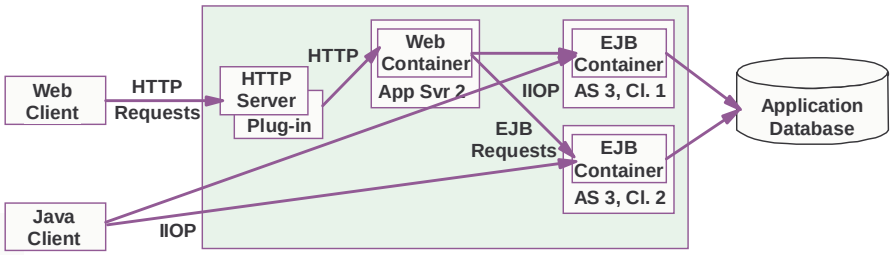


IBM Developer Relations DragonSlayer Team

Three types of requests can be workload managed in WebSphere v5.0.
HTTP Requests can be shared across multiple HTTP Servers.
This requires a TCP/IP sprayer to take the incoming requests and distribute them.
There are both hardware and software products available to spray TCP/IP requests.
Network Dispatcher is a software solution that is part of the WebSphere Edge Server.
Network Dispatcher applies intelligent load balancing to HTTP requests.
Servlet Requests can be shared across multiple Web Containers.
The WebSphere Plugin to the HTTP Server distributes Servlet requests.
Web Containers can be configured on the same machine or multiple machines.
EJB Requests can be shared across multiple EJB Containers.
The Workload Management Plugin to the Object Request Broker (ORB) distributes EJB requests.
EJB requests can come from Servlets, Java client applications, or other EJBs.

EJS WLM

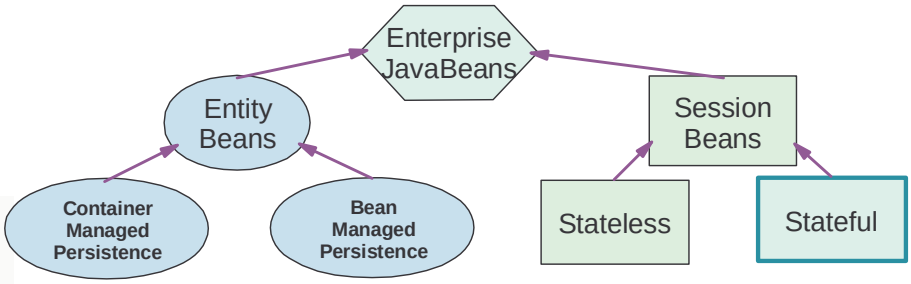
- EJB requests are routed to available EJB server clones, using IIOP
- EJB clients are servlets, Java clients or other EJBs
- Requests are routed based on Workload Management Selection Policy



IBM Developer Relations DragonSlayer Team

What is Workload Managed with EJS WLM?

- Homes of Entity or Session Beans
- Instances of Entity Beans
- Instances of Stateless Session Beans
- Calls to Home Interface of Stateful Session Beans

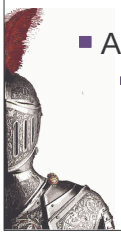


IBM Developer Relations DragonSlayer Team

The creation of Stateful Session Beans is Workload Managed (the calls to the create(...) methods of the Home interface), but all subsequent calls to the same instance will not be Workload Managed.

Cluster Management

- Definition of a Cluster
  - Clusters are a set of servers having the same applications installed, and grouped logically for Workload Management
- Purpose of the Cluster
  - Not all application servers are cluster members, but all cluster members are application servers
  - Cluster members:
    - run the same applications
    - share workload
    - can be centrally administered
- Creation of a Cluster
  - Start with an existing server configuration
    - that server may become the first cluster member
    - Development/Test environment suggestion: Leave first server out of the cluster
- Additional servers are created from templates
  - i.e., copies of existing servers

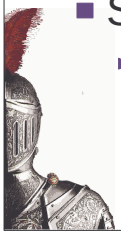


IBM Developer Relations DragonSlayer Team

By default, you can only install one copy of the application server binaries on a machine. Once those binaries are installed, you can have multiple application servers configured - the data needed for each additional server is stored in several XML files, and uses up about 50 K of disk space.
Several application servers can run on a single machine - but there is no requirement that they all be in the same cluster. Clustering is a logical grouping, not a physical one. All members of a cluster are nearly identical 'clones' of a common ancestor.
When you create a cluster, you make copies of an existing template. That template will probably be an application server that you have configured. You are offered the option of making that server a member of the cluster. It may be wise to keep that server available only as a template, because the only way to remove a cluster member is to delete it - keeping the original intact allows you to reuse that configuration to rebuild from.

Installing/Updating applications to a cluster

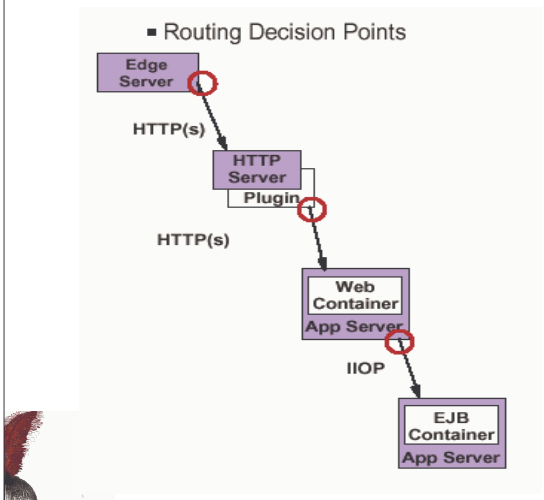
- Same steps as Installing to base server except:
  - select a Cluster as the target, rather than select a server
- Nothing changes until Save
- Application files (binaries and configuration files) copied at next synchronization
  - Behavior can be changed in the console
- Servers may be ripple-started
  - Restarts cluster members one at a time



IBM Developer Relations DragonSlayer Team

Updating applications to a cluster is done in the same manner as updating applications to a stand-alone server.

Basic WLM Routing



- Edge Server
  - Routing decision table stored internally
  - Configurable with NDAdmin tool
  - Multiple intelligent routing options
- HTTP Server Plugin
  - Routing table part of plugin-cfg.xml
  - Configured with Admin web app or wsadmin scripting tool
- WLM-aware Client
  - Includes Web Container, Java client, EJB
  - Routing table supplied by LSD
  - Configured with Admin web app or wsadmin scripting tool

IBM Developer Relations DragonSlayer Team

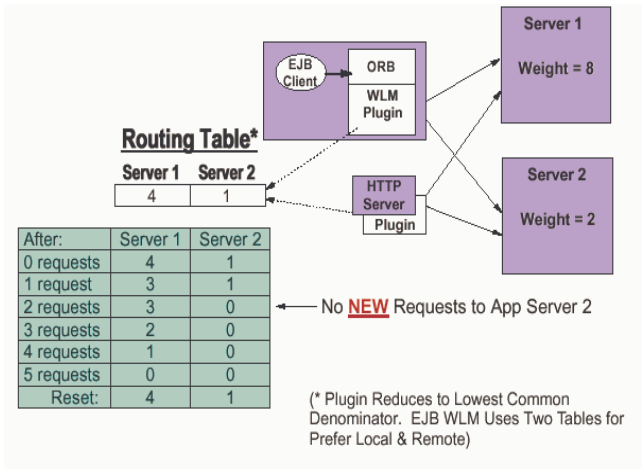
Now we'll address request routing.
Let's first look at the Fair Weather scenario - Assuming that everything works. We'll address failover scenarios in a few slides.

Edge Server's Network Dispatcher product is an IP sprayer that makes intelligent load balancing decisions. Using the NDAdmin tool, you can set it up to route to your HTTP servers based on Round Robin, Statistical Round Robin, Best, Custom Advisor, or Content Based Routing.

Once the request arrives at an HTTP server, the routing is Weighted Round Robin - the only configuration option is how much 'weight' to give each server. The routing information, the list of available servers and their weights, is ultimately stored by the Deployment Manager. The Node Agent is responsible for storing that information on the local drive for the server process to use in writing the plugin-cfg.xml file for the HTTP server plugin to read.

WLM-aware Clients include the Web Container, and stand-alone Java clients and EJBs running from WebSphere containers. The Location Service Daemon runs in the Node Agent and is responsible for providing clients with the routing table for EJB Containers. Again, the information comes from the Deployment Manager, and is configured in the Admin Console. Server weights and the Prefer Local are the configurables.

Weighted routing example



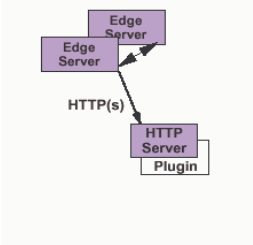
After:	Server 1	Server 2
0 requests	4	1
1 request	3	1
2 requests	3	0
3 requests	2	0
4 requests	1	0
5 requests	0	0
Reset:	4	1

IBM Developer Relations DragonSlayer Team

Weighted routing is fundamentally the same concept for HTTP requests, or for EJB client requests, so the two are combined on this slide.
When the HTTP Server plugin is generated, servlet request routing weights are written into the plugin-cfg.xml file, which the HTTP server will reload at configurable intervals.
There is a distinct Routing Table for each cluster. When a client requests an IOR for an EJB, the Location Service Daemon returns the IOR and a copy of the routing table. The client uses two in-memory copies of the table - one static, one dynamic.
The table illustrated here is the dynamic one - it has two entries, Server 1 and Server 2. The initial values are 5 and 3. The first new request will be sent to Server 1, and the counter for Server 1 will be decremented by one. The next request will be routed to Server 2, and the count for server 2 will be decremented.
Basically, the routing is Round Robin for all servers with non-zero table values.

Edge Server Failover

- Network Dispatcher can be paired with a backup machine
- Topology is 'Active/Standby'
- One machine does all the work
  - The other waits for a failure to begin handling routing

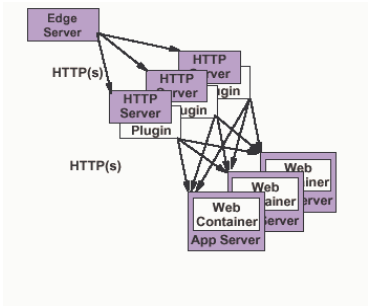


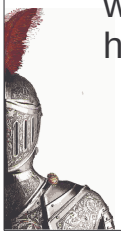
IBM Developer Relations DragonSlayer Team

Now let's look at what happens when things are not running so smoothly. Failover is one reason to implement workload management; it effects all the places where routing decisions are made.
The first routing decision is made outside the WebSphere Application Server's footprint - in the Edge Server (or other IP spraying technology).
Edge Server can be configured on two machines - they share a heartbeat, and if that heartbeat fails, the backup server becomes the primary and handles the load.
Having eliminated that as a Single Point of Failure, let's move next to the HTTP server.

HTTP Server Failover

- Multiple HTTP Servers provide coverage
- Edge Server can route around failed HTTP server
- HTTP Plugin
  - Every plugin knows about all web containers
  - Session key contains address of server
  - Sessions get properly routed
- Topology is 'Active/Active', with all HTTP servers handling load before failover





IBM Developer Relations DragonSlayer Team

Typically, a production environment will have multiple HTTP servers; each of those HTTP servers will route to multiple WebSphere Application Server instances.
Each plugin knows about all the servers; it has a server list and a back-up server list. If all the servers in the server list are unavailable, it will route to the backup list.
If any HTTP server fails, the Edge Server will simply route around it. The plugin reads the cloneID from the session key, and can route the request to it's originating server.

Web/Servlet Container Failover

- HTTP Server Plugin Detects Failure
  - Marks Container as unavailable
  - Tries next Cluster member in the Cluster
- What about In-flight sessions?
  - Sessions may be persisted to database
  - Sessions may be replicated in memory

IBM Developer Relations DragonSlayer Team

What happens if a server process dies? The HTTP server notes the failure and marks that application server as unavailable, then routes the request to the next cluster member.

Sessions already in progress will have a server ID for that failed server; the HTTP server routes them to the next server... What about the session? We can handle that two ways. Session Persistence to a Database, or internal messaging of session information.

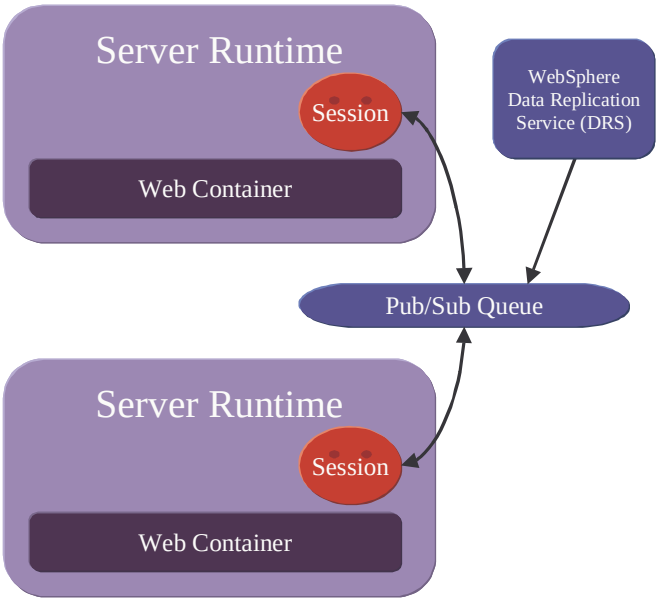
Database session persistence functions largely as it did in version 4.0. WebSphere Internal Messaging is new in 5.0,

HTTP Session Management

- Multiple mechanisms are provided to manage HTTP Session State
  - in-memory session
  - persistent http session to a database
  - JMS pub/sub based memory replication of HTTP Session state between clustered Application servers
- Address different topology requirements
  - failover, performance, cluster size

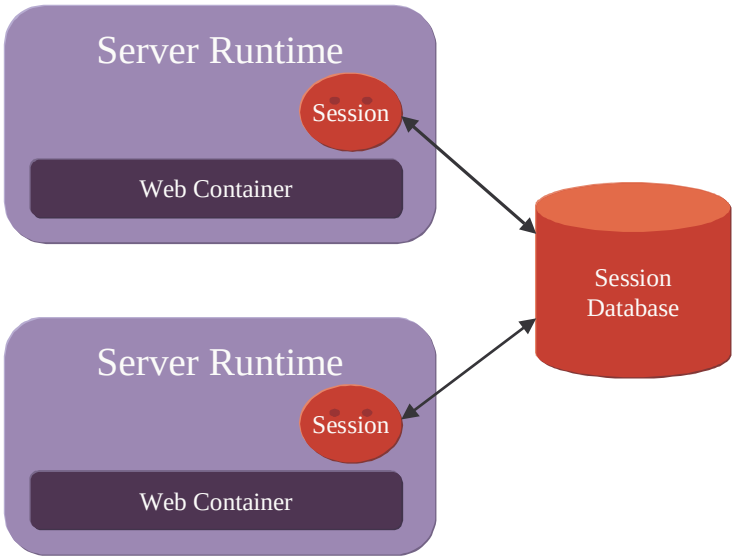
IBM Developer Relations DragonSlayer Team

HTTP Session: JMS pub/sub based memory replication



IBM Developer Relations DragonSlayer Team

HTTP Session: Persistent to Database

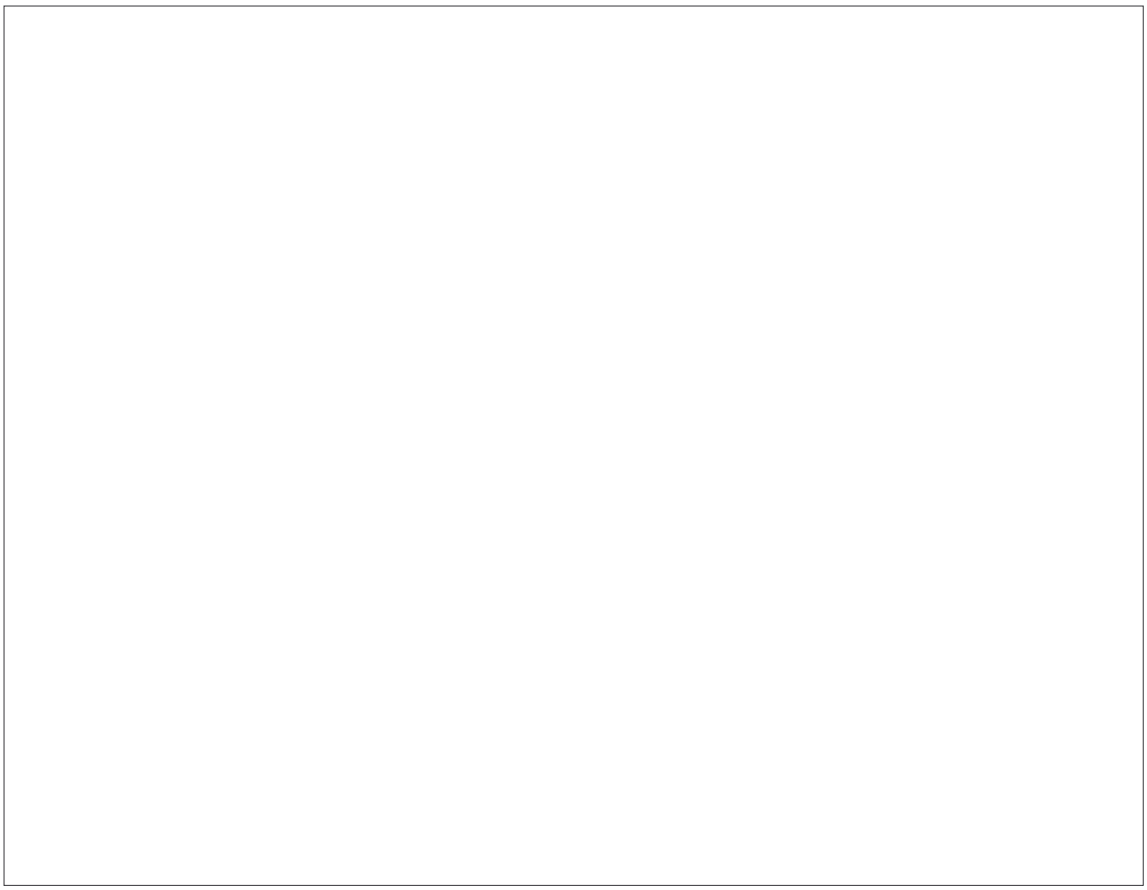


IBM Developer Relations DragonSlayer Team

Summary

- Overview of Workload Management
- What can be Workload Managed
- Clusters
- Weighted Workload Management
- Failover
- Session replication options

IBM Developer Relations DragonSlayer Team



Web Services

dragonSlayer Team

What is a Web Service ?

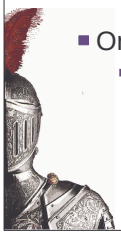
- WSDL described interface that defines a collection of network accessible operations
  - Shared, open, emerging technology standards - SOAP, UDDI, WSDL, WSIL etc.
  - Modular
  - Self-described
  - Published
  - Independently deployable
  - Loosely coupled
  - Simple
  - Business driven
  - Provide access to business services

IBM Developer Relations DragonSlayer Team

So, what is a Web Service? Definitions will vary, but here is ours. Simply, an "Interface that describes a collection of networked accessible operations". For developers, one way to think of Web Services is as a component model. A fairly coarse grained component model that gives us a building block from which to build solutions from. It's not a subroutine library or replacing Java collection classes or C++ libraries or anything else you use. It's function. Could be a stock quote service, could be something more complicated like a credit authorization, or perhaps an entire business process like loan authorization, a service composed of other nested services.
The other way to think about Web Services is how most people talk about them, that is the mechanism itself..., XML messages sent over the HTTP transport protocol.
Modular by design because inherently they are interface oriented.
Described using a service description language. WSDL (Web Services Description Language).
Published by making its description available to potential users.
Found by sending queries to that registry and receiving the binding details of the service(s) that fit the parameters of the query.
Bound by using the information contained in the service description to customize the connection to be created.
Invoked over a network by using the information contained in the binding details of the service description and possibly composed with other services into new services.

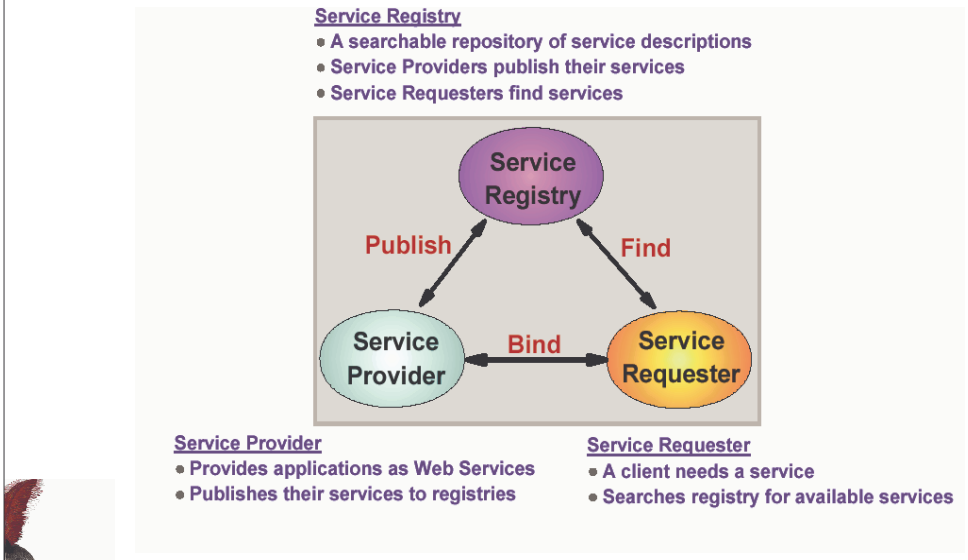
Benefits of Web Services

- Revenue benefits
  - Improved relationships with customers/partners
  - Work force productivity
  - Innovation and reduced cycle times
  - Share processes without sharing technology
- Cost reduction
  - Deliver new business solutions faster
  - Better information flow and knowledge
  - Consistent infrastructure
  - Based on industry standards avoiding costly proprietary implementation
  - Standards hide underlying implementation allowing for more efficient debugging/testing
- Unification of applications
  - New ways of accessing old applications
  - Applications can be integrated without regard to implementation details
  - Applications can dynamically navigate, discover, and interact over the Internet (loosely coupled)
- Organized information
  - Targeted, more relevant



IBM Developer Relations DragonSlayer Team

Roles and Functions



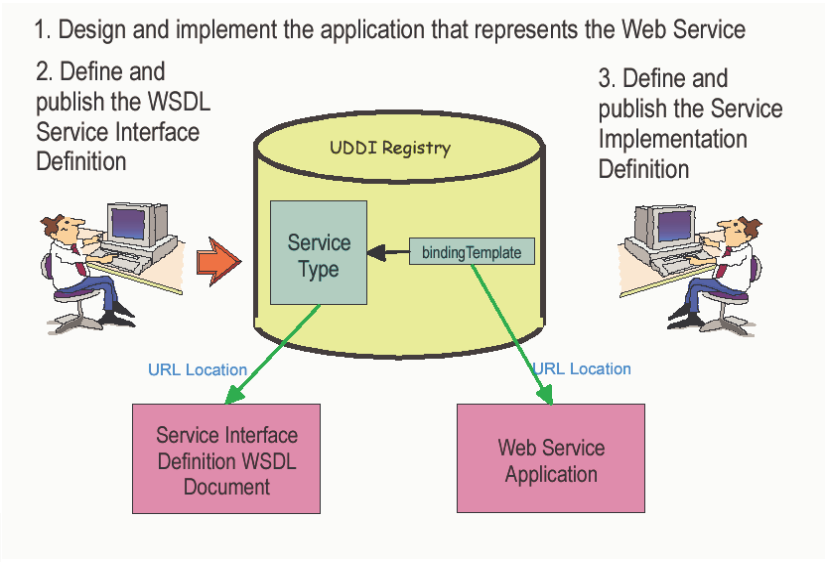
IBM Developer Relations DragonSlayer Team

The Service Registry role owns a directory of all of the services available. The Service Registry represents a new, potentially very lucrative business model. The Registry owner could charge a fee for the use of their directory. Service Providers lists (or advertise) the Service offering in the registry. Service Requesters on the other hand query the Service Registry about the services available. Once Service Registry provides the binding information to the requester, it is no longer involved in the communications between the provider and requester.

The Service Provider has developed services that they make available as Web Service. These services will be hosted on their Application Server. A service is invoked by a requester through an XML message. These XML messages are generally carried across the Internet through a network-neutral standard protocol called Simple Object Access Protocol (SOAP). The Service Provider describes the service they are making available with a standard encoding called Web Services Description Language (WSDL).

The Service Requester is the business that requires a certain business function to be fulfilled. From an architectural perspective, this is the application that is looking for (queries the Service Registry) and then binds to and invokes the service. The requester has to find the service before invoking it - this process of discovery involves accessing a directory where the infomation of what the service does and how to invloke it resides. UDDI addresses this capability

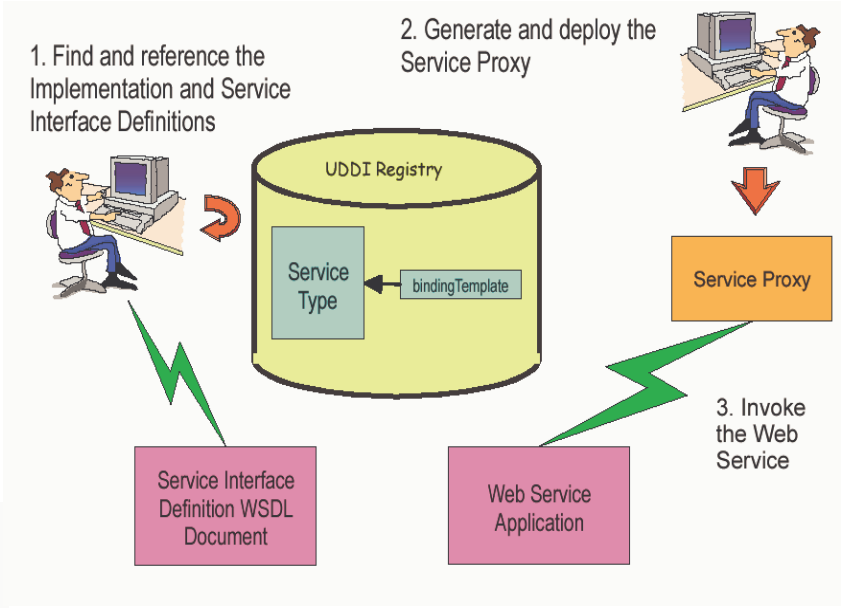
Developing an Application to provide a Web Service



IBM Developer Relations DragonSlayer Team

Step 1
This involves the design and coding required to implement the Web Service
Step 2
The WSDL Service Interface Definition document defines the interface and mechanics of the service interaction (e.g. message structures, data types)
This step involves publishing a Service Type definition to the UDDI Registry (stored in a tModel structure). The tModel contains an element called overviewDoc which references a URL at which the WSDL Service Interface Definition document is located (the WSDL file is not stored in the Registry directly). Note that this step might not necessarily be performed by the service provider. For example, a standards body may publish a Service Type representing a standard web service interface (a hotel booking interface for example).
Step 3 involves publishing a Service to the UDDI Registry (stored in a businessService structure within a businessEntity structure). The Service registry entry contains a bindingTemplate structure whose tModelInstanceInfo element references the Service Type created in Step 2

Developing an Application to access a Web Service



IBM Developer Relations DragonSlayer Team

Step 1
The developer interrogates the UDDI Registry to find:
a. the Service entry
b. the associated WSDL Service Interface definition (via the referenced Service Type)
c. the network location of the service (this is contained in the accessPoint element within the bindingTemplate)
Step 2
The Service Proxy contains all of the code that is required to access and invoke a Web Service. Typically, the development tooling will generate this automatically
The Service Proxy is deployed with a client application
Step 3
The client application is run and uses the Service Proxy to invoke the Web Service
Note: It is possible that the service interface definition is located at run time (called Runtime Dynamic Binding) rather than at build time as above (Static Binding)

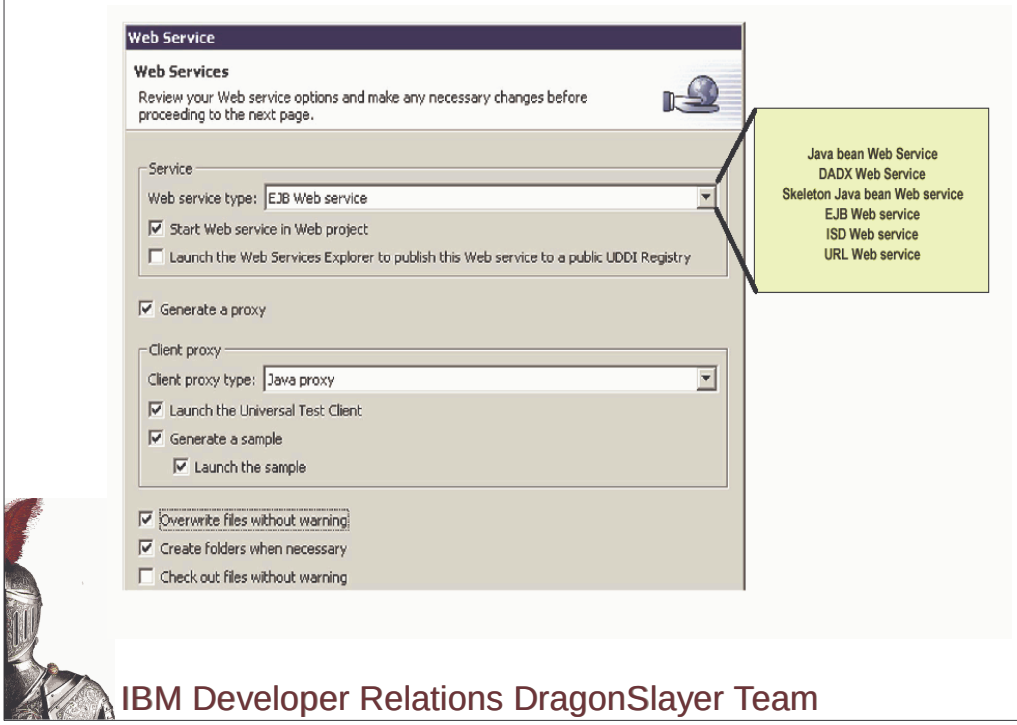
Web Services Tooling

- WebSphere Studio Application Developer V5 provides following actions to assist with Web Services development
  - Discover - Browse UDDI Registries to locate existing Web Services
  - Create or Transform - Create Web Services from existing artifacts
  - Build - Web Services wizards assist you in generating a SOAP proxy to Web services described in WSDL and in generating Java bean skeletons from WSDL.
  - Deploy - Deploy Web Services into the WebSphere Application Server 5.0 or Tomcat test environments using Server Tools.
  - Test - Test Web Services running locally or remotely in order to get instant feedback.
  - Develop - Generate sample applications
  - Publish - Publish Web Services to public or test UDDI registries

IBM Developer Relations DragonSlayer Team

WebSphere Studio Application Developer V5 provides a feature-rich Web Services tooling environment. Through wizards, many of the tasks are greatly simplified.

Web Services Wizard



IBM Developer Relations DragonSlayer Team

The Web Services wizard is used to both create and consume Web Services.
Popup actions (on a context menu) provide entry points into the Web Services wizard

Web Services Explorer

- Web Services Explorer (was IBM UDDI Explorer in previous versions of Application Developer)
  - Assists you in discovering and publishing your Web service descriptions
  - No configuration necessary
  - Supports UDDI V2 for public, private (intra-enterprise, private marketplace), and test registries
  - Based on DB2 or Cloudscape for use in unit test environment
  - Multiple language support for names and descriptions
  - Allows Web Services Inspection Language (WSIL) document browsing
- WebSphere UDDI Registry
  - Standalone UDDI registry
  - One per workspace
  - Allows you to work solely within one tool throughout Web Service development/testing cycle

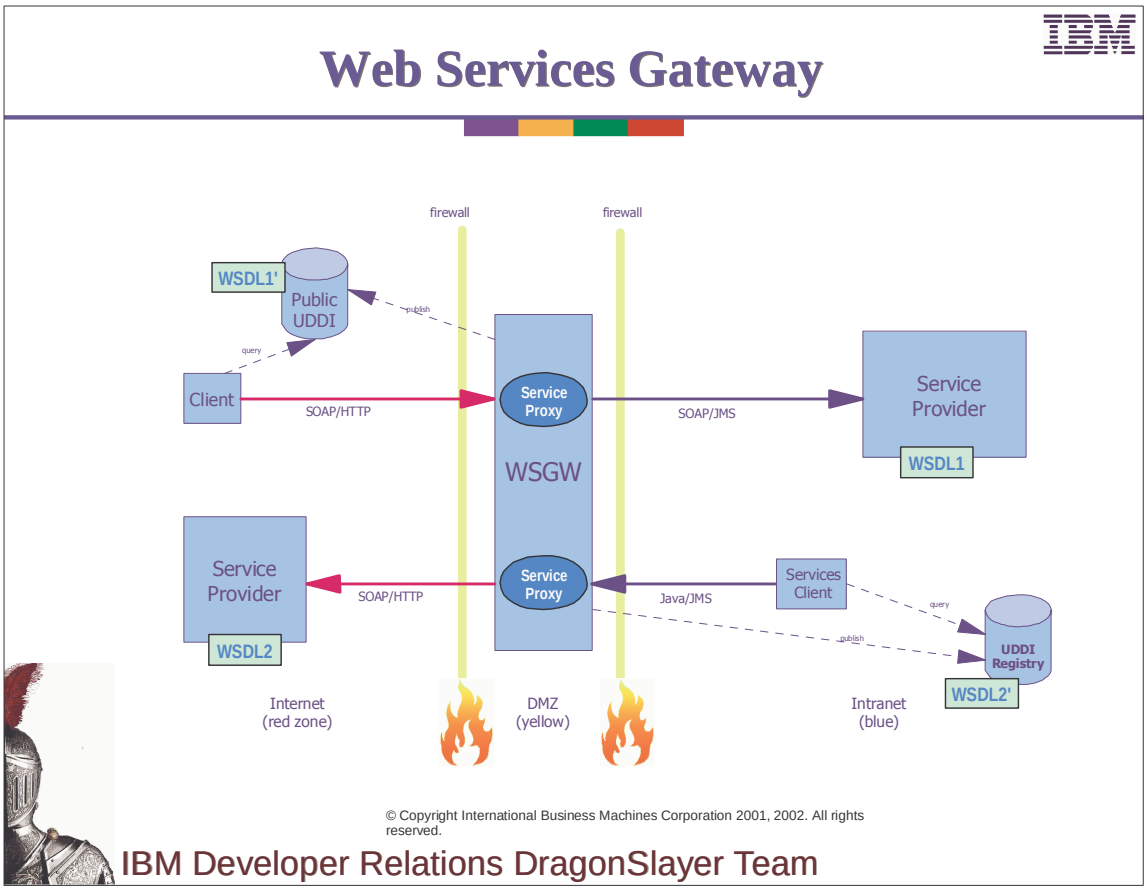
IBM Developer Relations DragonSlayer Team

The Web Services Explorer does not support V1 Registries and is not expected to do so as defined by the UDDI specifications available at http://www.uddi.org.
The Web Services Explorer is part of WebSphere Studio Application Developer's web services plugins. There is no configuration necessary to use it. You can simply launch the explorer, connect to a registry and pick one of the favorites or select a private registry. Additional information on Web Services Inspection Language can be found on IBM's developerWorks. The version of WSIL that is supported within the Application Developer is 1.0.

Web Services Gateway

- Ships with Network Deploy version
  - Middleware component that provides framework between Internet and intranet environment during Web Services invocations
  - Can be used to subset exposure of Enterprise Web Services to internet (proxy gateway)
  - Support for multiple transports and protocols
    - SOAP/HTTP, SOAP/JMS, Direct Java via RMI-IIOP, Java over JMS
  - Benefits
    - J2EE application
    - Application server hosts the service proxy
    - Provides centralized management of Web Services
    - Handles protocol translation

IBM Developer Relations DragonSlayer Team



This chart shows the final result of deploying your binding information to the Web Services Gateway.
In the top half of the chart, you have created a Web Service internally and have deployed the binding information to the Web Services Gateway. The gateway in turn creates a service proxy and a description file that is made available in a public UDDI. When the client (external in this case) requests that specific service, it retrieves the binding information from the public UDDI registry and invokes the service based on the description. The client does not have to be concerned with the implementation details as the Web Services Gateway hides the implementation.
The gateway handles the Security aspects of this call. There are three primary authentication mechanisms: HTTP(S) based authentication, servlet-based authentication, and SOAP authentication done by document signing and verification.
The bottom half of the charts illustrates how an external service could be exposed as an internal service.

Here is a brief description of the protocols illustrated:
1) SOAP/JMS: The JMS message body is a text message, whose text contains a SOAP message which encodes the input and output messages for the service invocation.
2) Java/JMS: The JMS message body is an object message, and the input and output messages for the service invocation are written into the message body as serialized java objects.

Web Services Technology Preview

- JAX-RPC (JSR 101)
  - Standard programming model
  - XML-based RPC
  - Client stub generation and programming model
  - Standard mappings from WSDL to Java and from Java to WSDL
- Web Services for J2EE (JSR 109)
  - J2EE standard implementation and deployment model for Web Services
  - Web Services as resources, J2EE client access
  - Axis runtime - Stateless Session EJB or Servlet endpoint
- Standard Programming Model
  - portable Web Services applications
  - clients: EJB, servlet, application client as client to Web Service
  - servers: Stateless Session Bean and Java Bean
- Web Services Security

IBM Developer Relations DragonSlayer Team

WebSphere Application Server V5 has the following support for Web Services:

- Apache SOAP 2.3 runtime
- DOM-based parsing
- Relies on Web Services Toolkit or WSAD for emitters (no tooling in the runtime to help write stubs)
- Proprietary programming model

Web Services Technical Preview

- Based on Apache Axis, with IBM extensions
- SAX-based parsing
- Includes command line tooling
- Conforms to Java Web Services standards
- Standards-based programming model
- Applications portable across Application Servers

So, the technology preview items bring performance (Axis, SAX), the J2EE programming model, and a solid security foundation to the product.

Summary

- Web Services offer many benefits
- Industry standards used - XML, SOAP, WSDL, UDDI, WSIL
- Wizard decreases development time and ease the creation and publication of Web Services
- Application Developer tooling allows for working with all Web Service Roles (Service Registry, Service Provider, and Service Requester)
- The WebSphere UDDI Registry allows testing UDDI access with a self contained, fully functional UDDI registry
- Web Services Gateway provides acts as a proxy and protocol converter bewteen intranet and internet Web Services clients and servers

IBM Developer Relations DragonSlayer Team