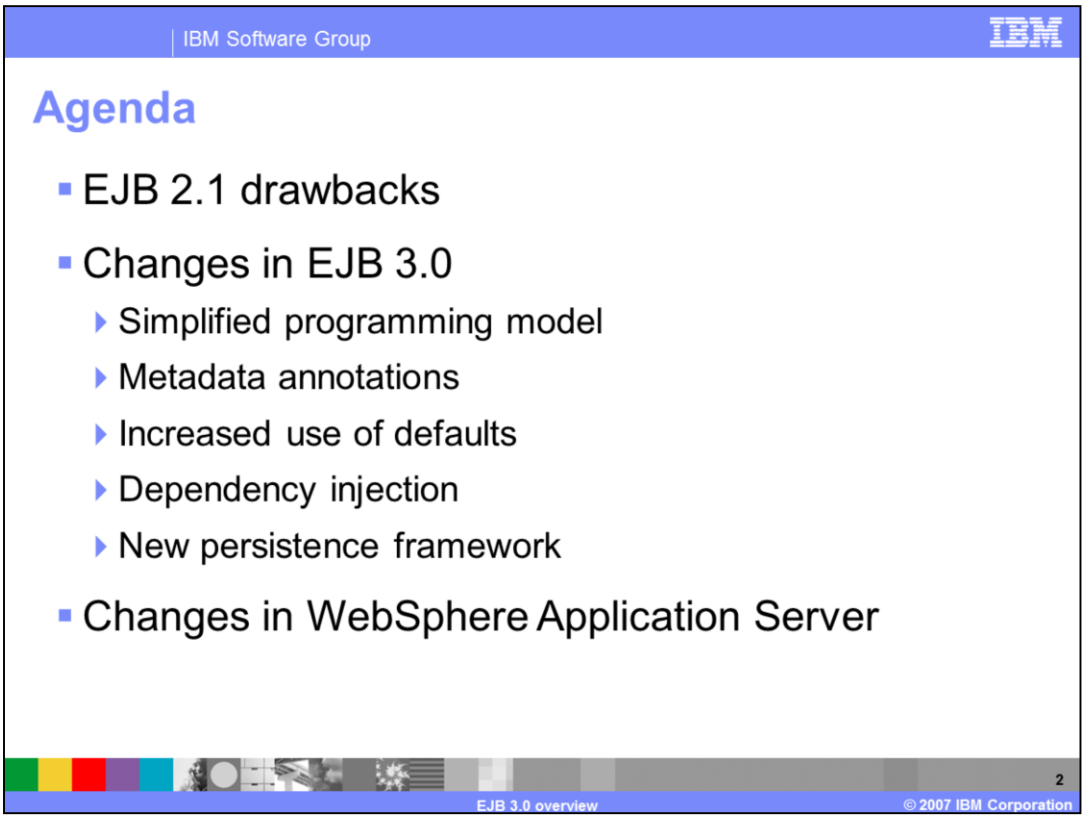
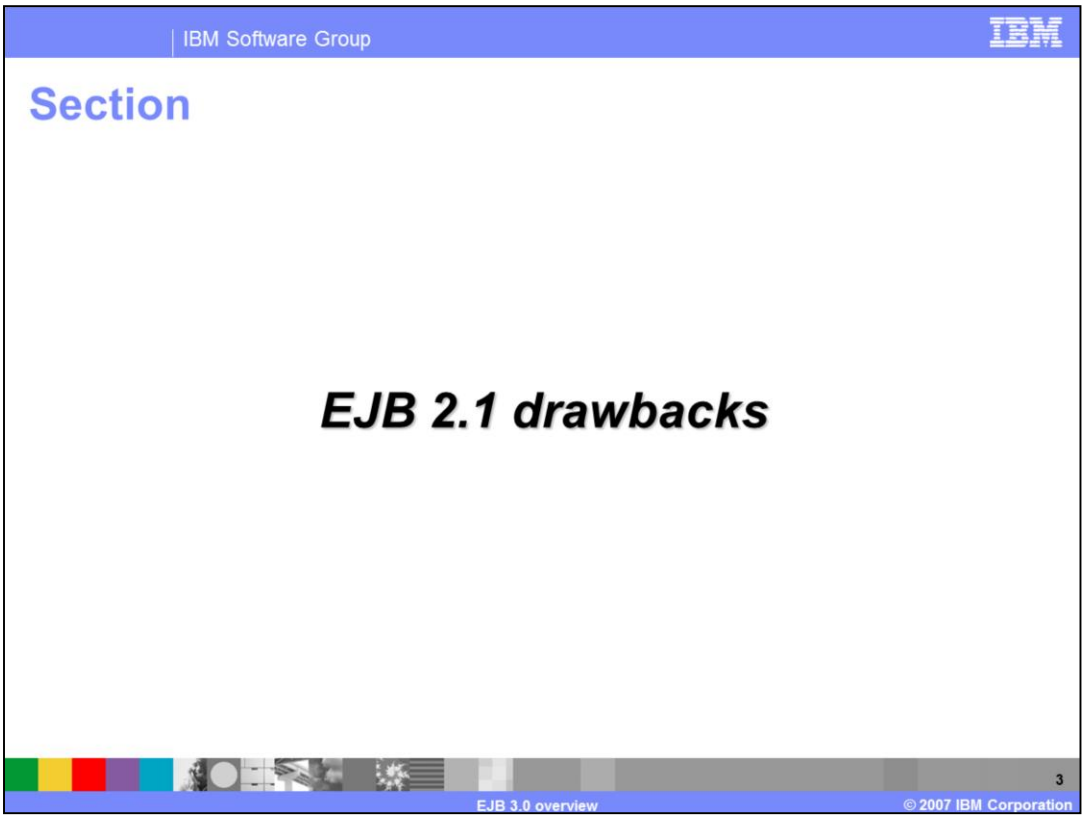
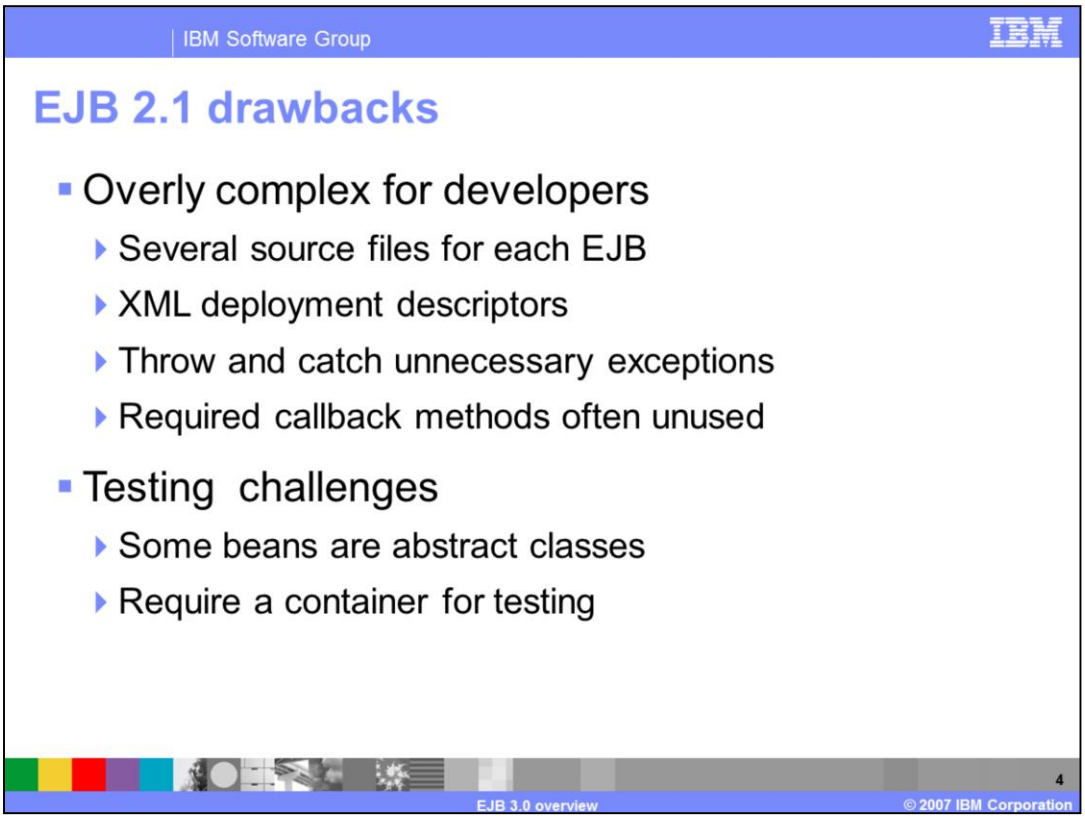
This presentation gives an overview of the WebSphere® Application Server V6.1 Feature Pack for EJB 3.0.



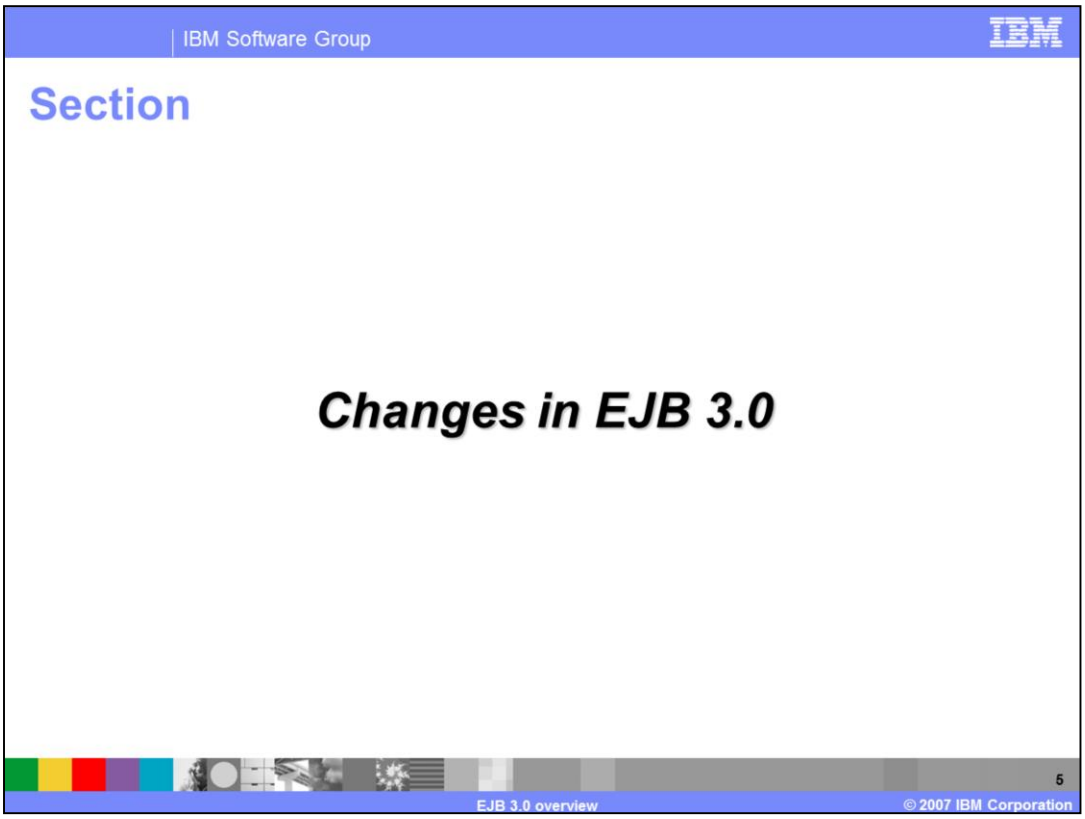
This presentation will first describe some of the drawbacks of EJB 2.1 that provided the motivation for EJB 3.0. Then it will give an overview of the major changes in EJB 3.0, and some changes that were made in WebSphere Application Server itself.



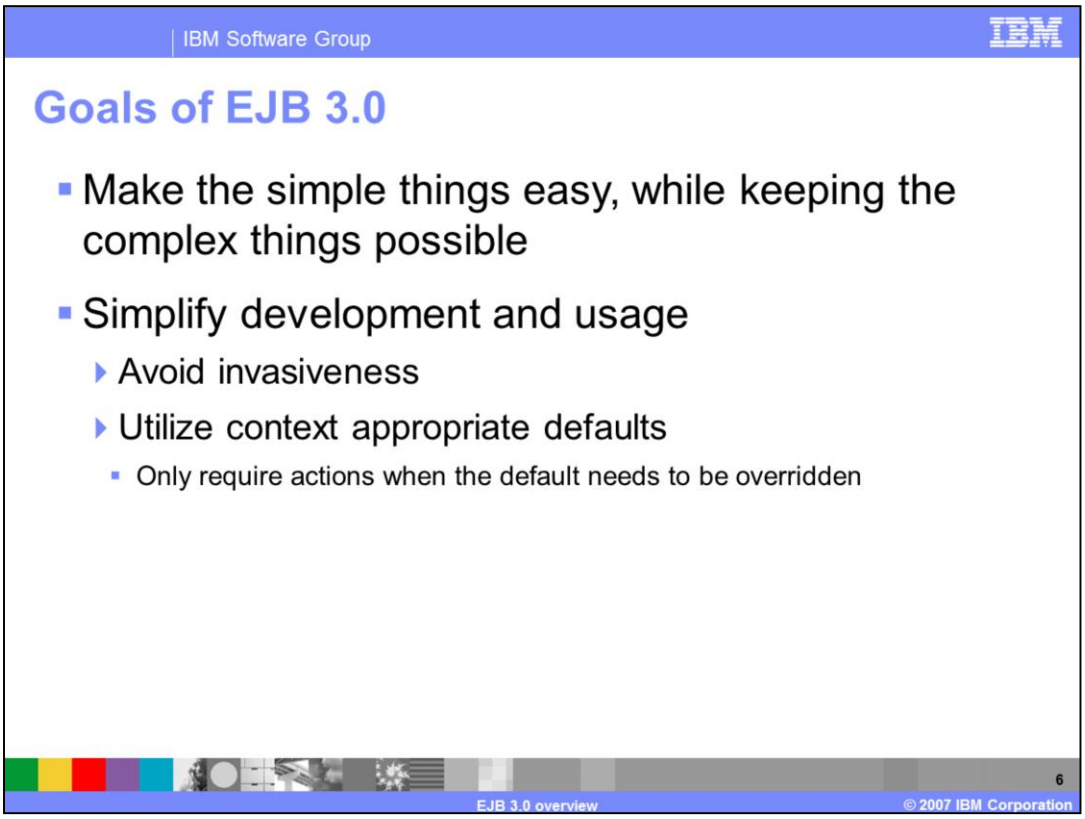
This section describes some of the drawbacks in EJB 2.1.



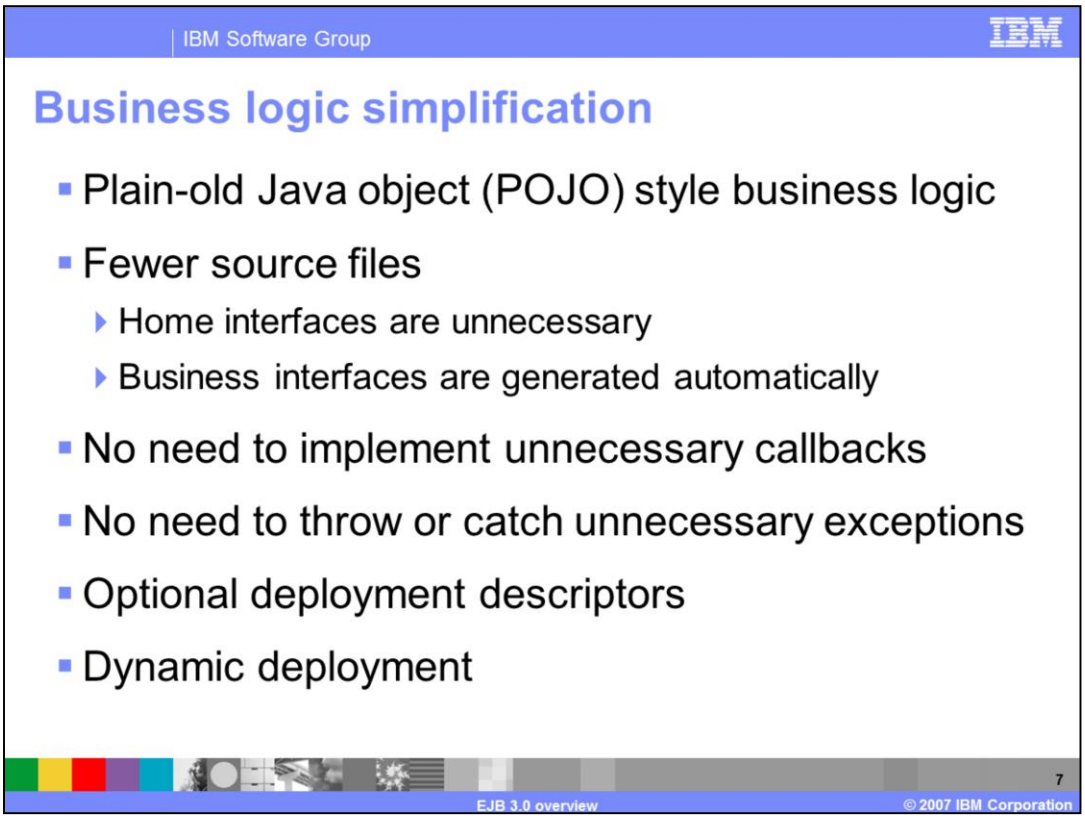
The EJB specification has often been criticized for being overly complex. For example, new developers are often confused about why it is necessary to create several different source files for each EJB. Similarly, you are required to implement several callback methods even if you do not use them, and handle exceptions that may be unnecessary. Deployment descriptors can also be difficult to understand, and can be a bottleneck in a team development environment, since only one person can be updating the deployment descriptor at a given time. EJB 2.1 applications also cannot be tested outside of a container, since some beans are abstract classes that are implemented by the container at runtime.



This section will provide an overview of the changes introduced by the EJB 3.0 specification.



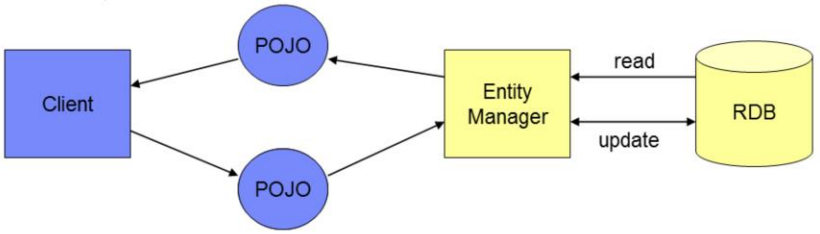
The main goal of EJB 3.0 is to simplify the programming model. The new specification aims to make it as easy as possible to implement simple things, while keeping complex things possible. In many cases this is accomplished by utilizing contextually appropriate default values, allowing you to override the defaults when needed. When it is possible for the container to figure something out, it will do so, rather than requiring a developer to provide unnecessary information. With this simplification, regular Java developers should be able to develop EJBs. That is, if you can write a plain-old Java object (POJO), you can write a business object.



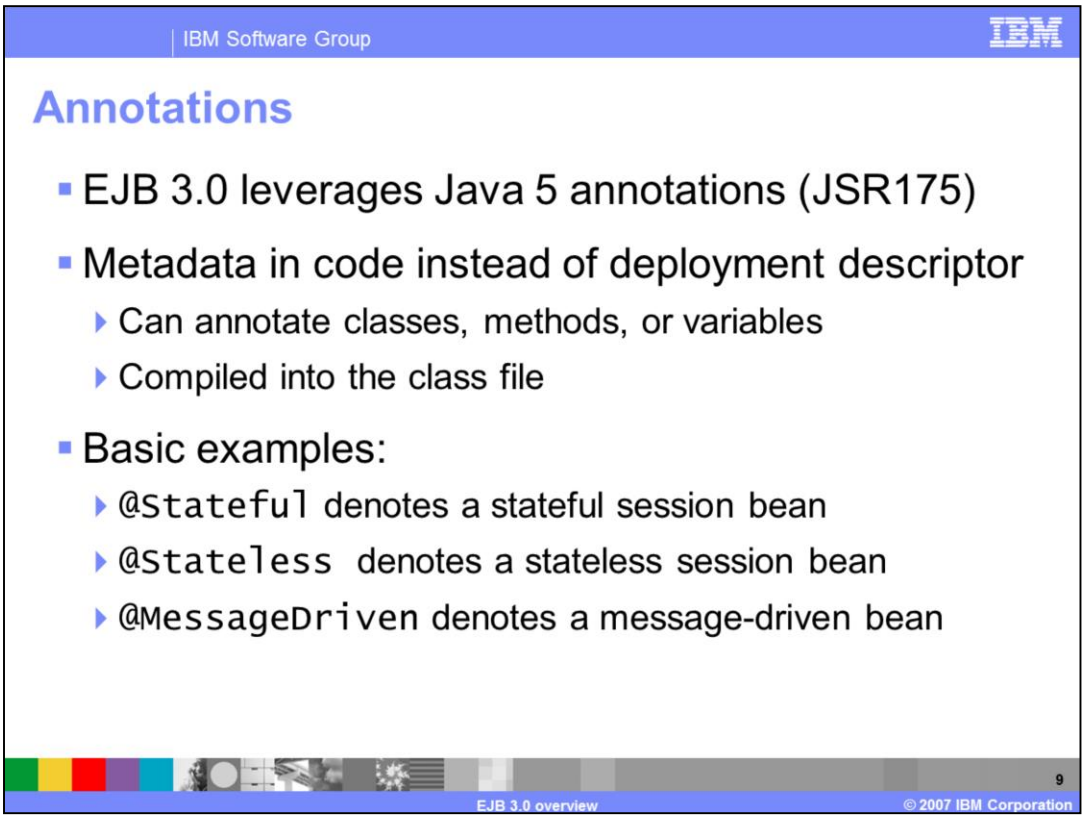
Many changes have been made in EJB 3.0, in an effort to simplify the programming model. EJBs are now developed as plain-old Java objects, rather than their more complex predecessors. Home interfaces are unnecessary in most cases, and business interfaces can be generated automatically by the runtime, so there are far fewer source files to develop and maintain. Along the same line, you now only have to implement life cycle callback methods if you actually want to use them, and the specification no longer requires you to implement unused exceptions. Deployment descriptors are optional in EJB 3.0, because as you will see in this presentation, metadata can now be specified directly in your Java code, using annotations. EJB 3.0 containers also handle deployment dynamically, so you no longer have to manually deploy your EJBs.

Persistence in EJB 3.0

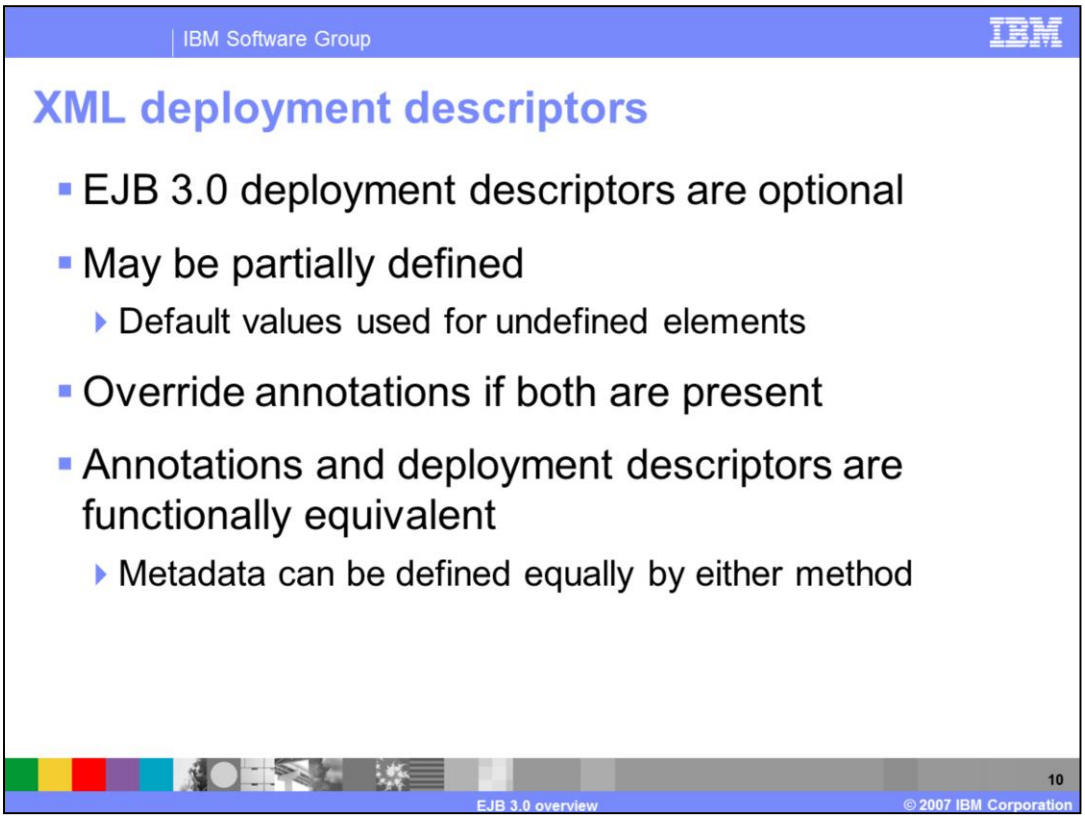
- Java Persistence API (JPA) is a standard persistence and object-relational mapping (ORM) framework for Java
  - Part of the EJB 3.0 specification (JSR220)
  - Represents relational data as plain-old Java objects (POJOs)



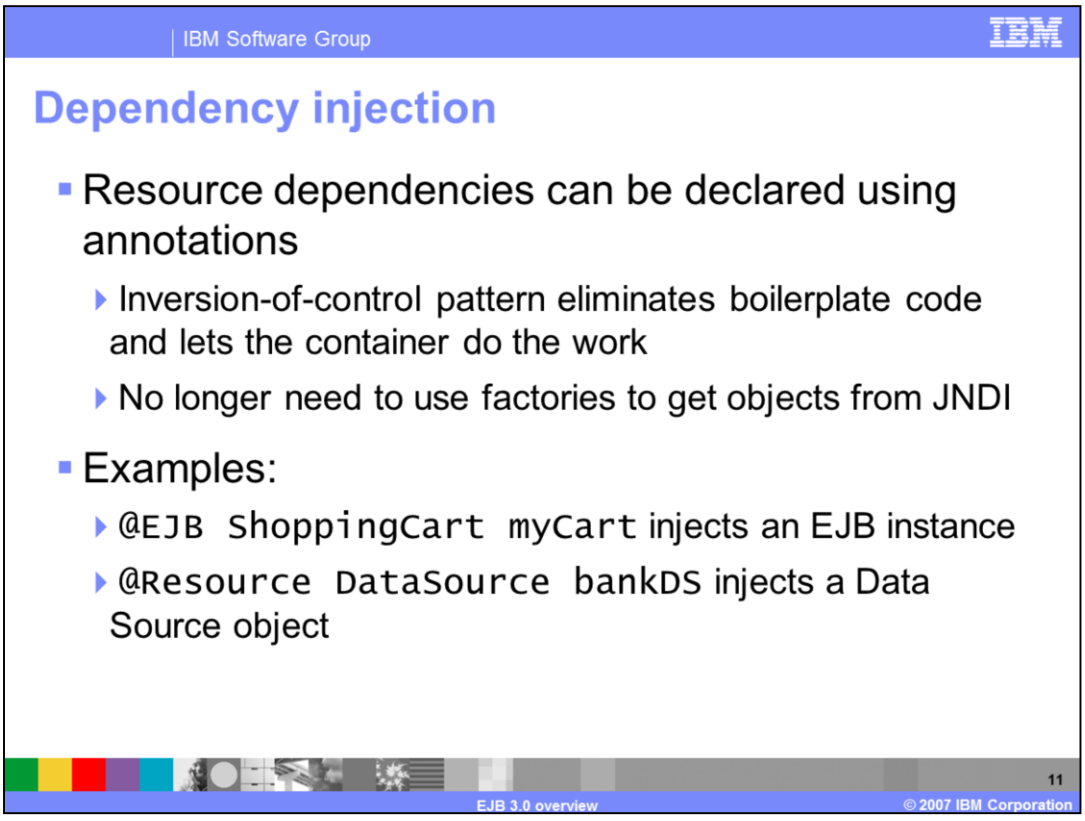
The Java Persistence API, or JPA, is a new framework for Java object persistence and object-relational mapping that was introduced as a part of the EJB 3.0 specification. In JPA, relational data is represented using plain-old Java objects called “Entities”, rather than the more complex “Entity Beans” that are used in Container-Managed Persistence (CMP). JPA is considered both simpler and lighter-weight than CMP. The presentation titled “JPA overview” discusses JPA in more detail.



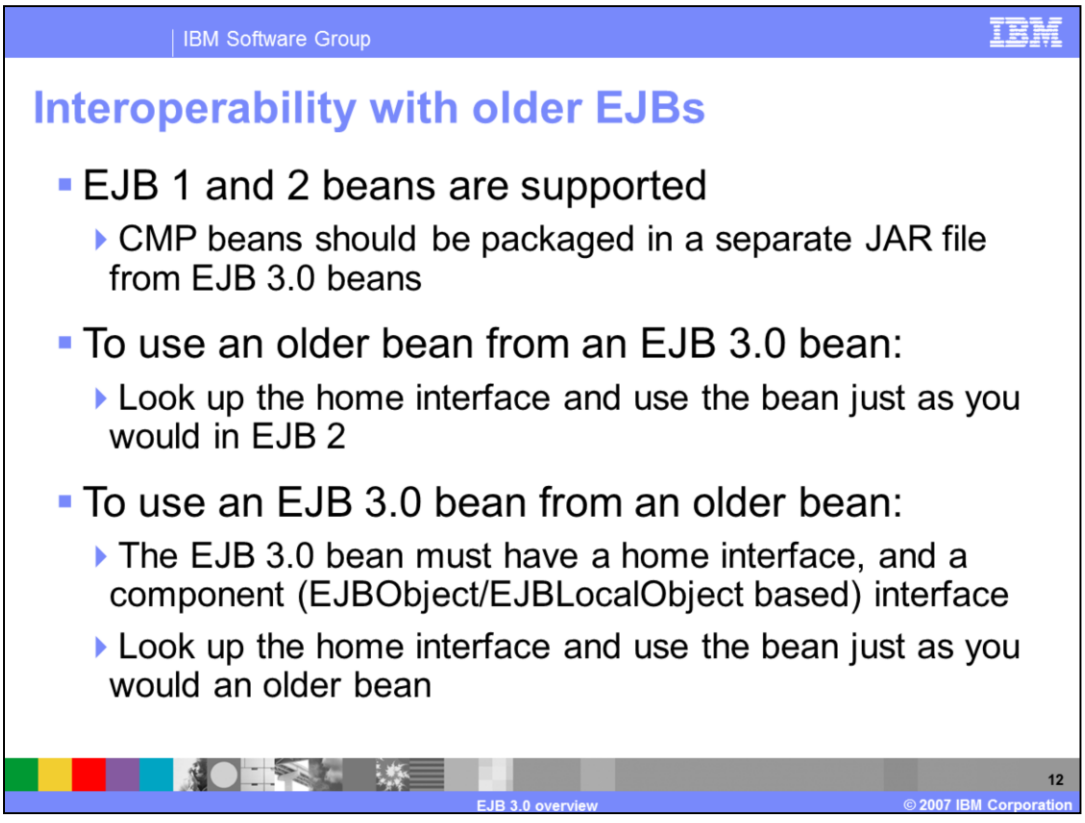
Java annotations were introduced in Java SE 5, under JSR175, and are widely used in EJB 3.0. Annotations are used to specify metadata directly in code, rather than in an XML deployment descriptor, and are compiled directly into your class file. For example, adding "@Stateful" before the name of a class defines it as a stateful session bean, and that no longer needs to be specified in a deployment descriptor. Similar annotations exist for Stateless Session Beans, Message-Driven beans, and JPA Entities. Annotations can be used to provide more complex information as well, as you can see in the presentation titled "EJB 3.0 code examples".



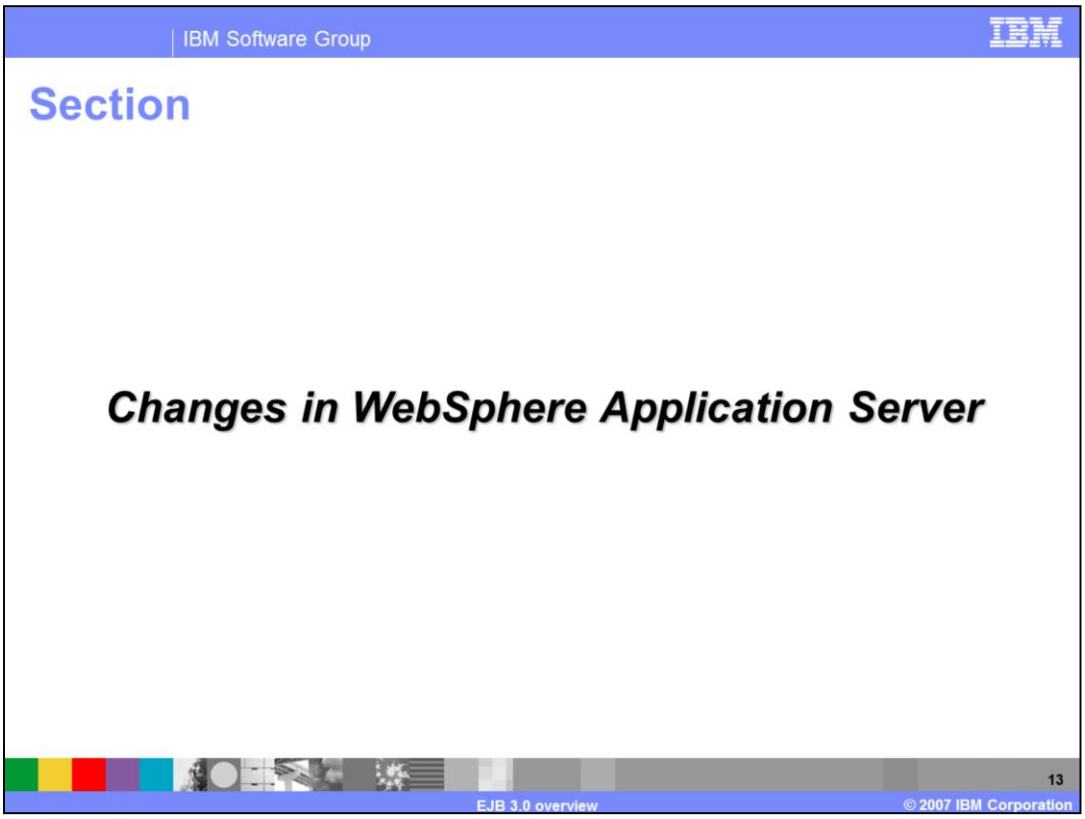
EJB deployment descriptors can still be used in EJB 3.0, if you prefer to use them over annotations. Deployment descriptors can even be partial, if you want to provide some information using annotations, and keep other information in XML. Any values that are not specified will inherit intelligent default values. In the case that the same information is specified by both a Java annotation and the XML deployment descriptor, the data in the deployment descriptor will be used. With experience, you may find that you prefer using annotations for some things, and deployment descriptors for others. It is entirely up to you, as the two are functionally equivalent.



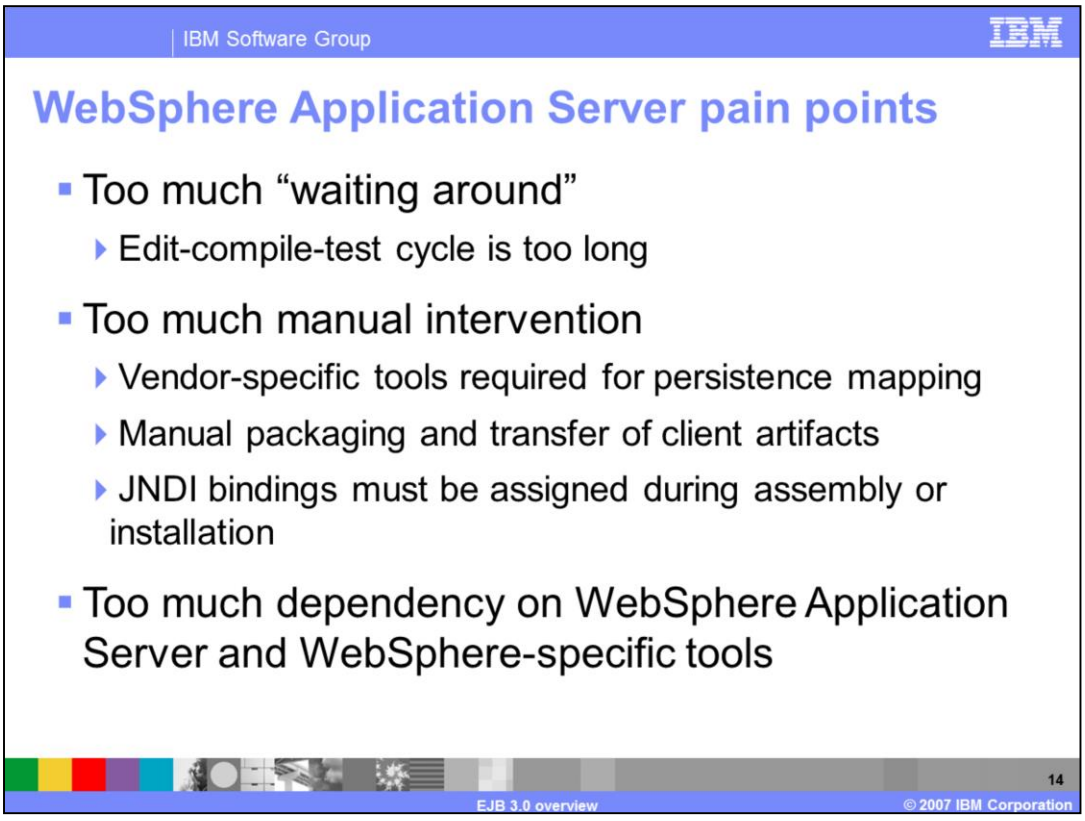
EJB 3.0 also introduces annotations for injecting resource dependencies using the “Inversion of control” pattern. Rather than implement boilerplate code and use a factory to get an object, look it up in JNDI, and then cast or narrow it to make it useable, you now need only to use the @Resource annotation to inject a dependency. The examples shown on this slide illustrate how to inject an instance of an EJB using the @EJB annotation, and a data source using the @Resource annotation.



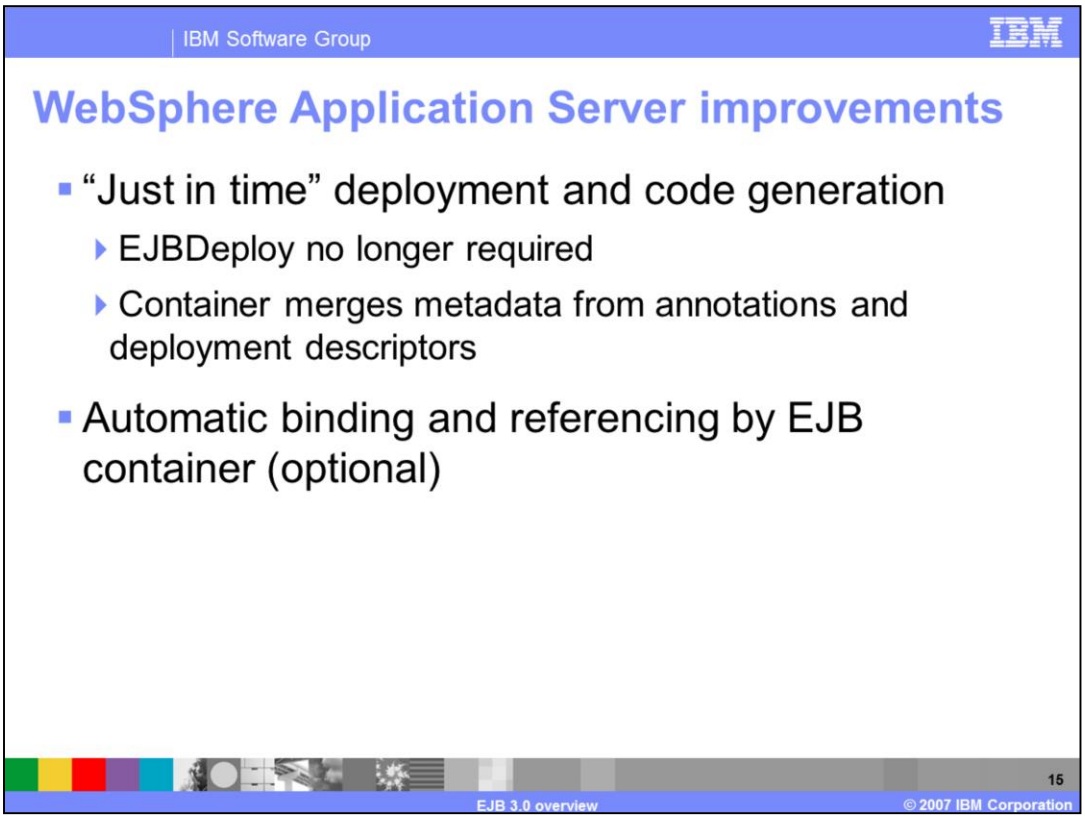
The EJB 3.0 specification requires compatibility with earlier versions, so EJB 1 and 2 beans are still supported when the Feature Pack for EJB 3.0 is installed. You should package any CMP beans in a separate EJB JAR file from your EJB 3.0 beans. If you want to call an older EJB from an EJB 3.0 bean, you only need to look up the bean's home interface, and use it exactly as you would in an EJB 2 bean. To call an EJB 3.0 bean from an older EJB, you need to create a home interface and a component interface for the EJB 3.0 bean, since they are expected by the older bean. Once you have created those interfaces, you can look up the home interface and call it exactly as you would call an older bean.



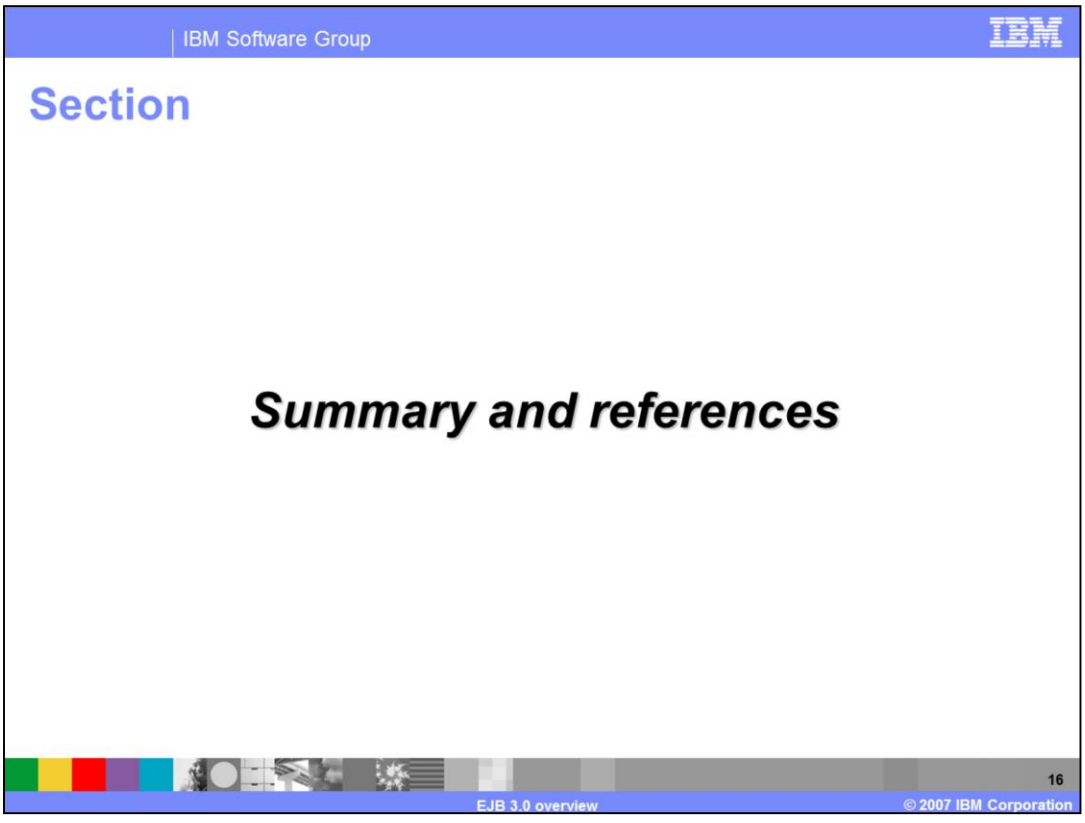
This section covers improvements to WebSphere Application Server when the Feature Pack for EJB 3.0 is installed.



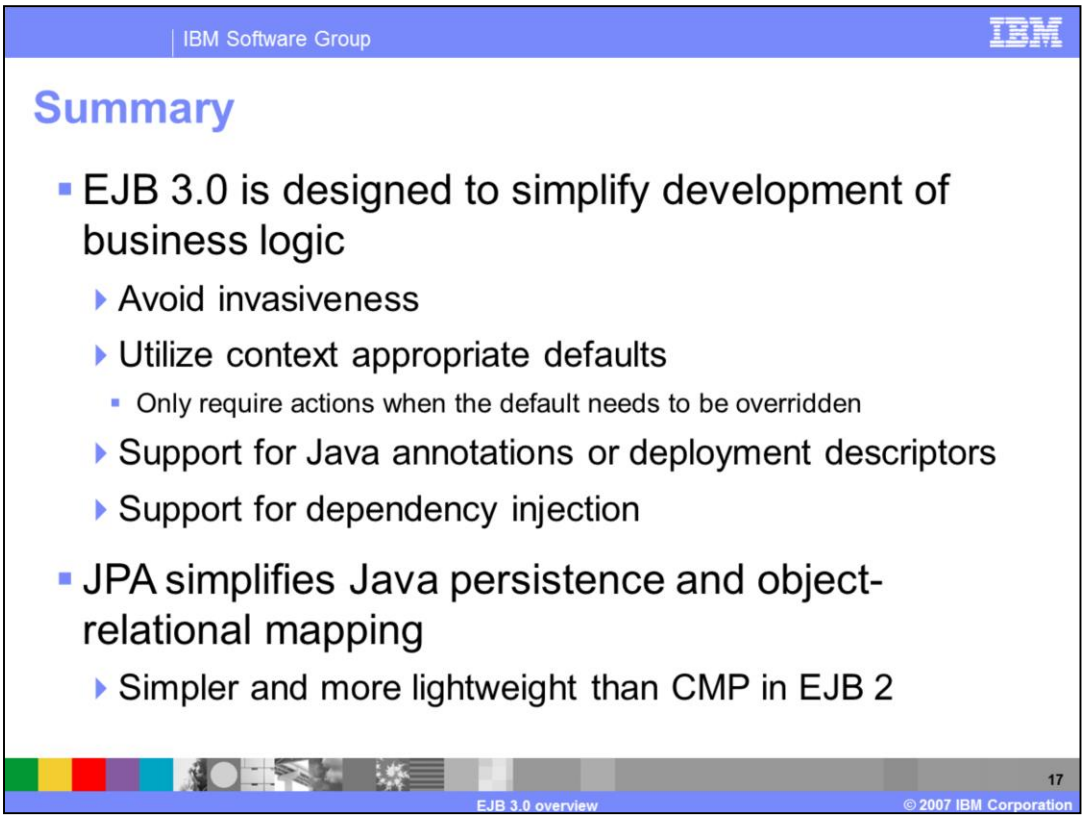
It has often been said that the edit-compile-test cycle is too long when developing applications for WebSphere Application Server. The EJB deployment step had to occur each time you modified an application, and that can take a very long time, depending on application complexity. Vendor-specific tools were also required for persistence mapping. Another frequent pain point is that JNDI bindings had to be specified manually, either during application assembly or installation. There has also been a dependency on WebSphere Application Server or WebSphere-specific tools for some tasks, such as editing WebSphere-specific .xmi files.



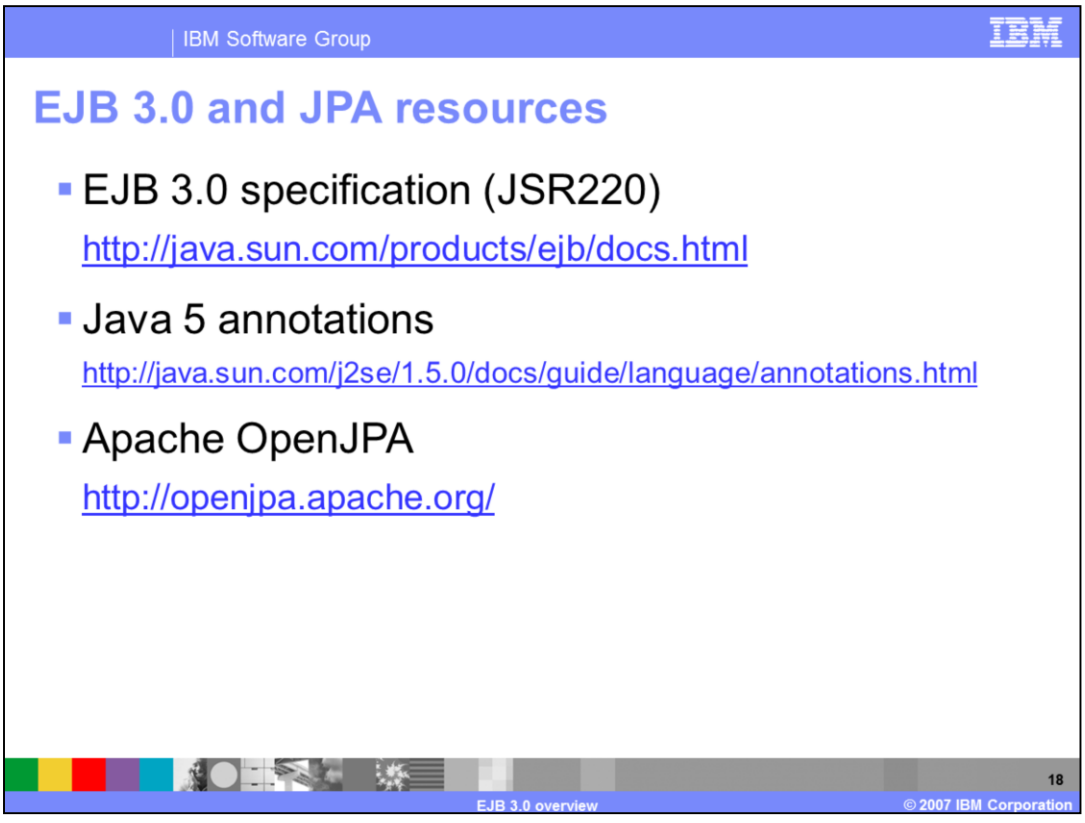
When using EJB 3.0 applications on WebSphere Application Server, many of these pain points are resolved. The Feature Pack for EJB 3.0 provides "just in time" deployment and code generation, so the "EJBDeploy" step is no longer required. The container merges the metadata provided in annotations and deployment descriptors dynamically. Automatic binding is another capability that is provided by the Feature Pack. It automatically binds beans to both a short and long JNDI name, so that you do not have to provide bindings manually. The long-form binding is guaranteed to be unique. Consult the Information Center for more information on automatic binding. You can override the automatic bindings by providing your own bindings if you prefer.



This section will summarize the presentation, and list some resources for further information.



The EJB 3.0 specification is designed to make it faster and easier to develop your business logic. By introducing Java annotations for EJBs, dependency injection support, removing boilerplate code, and making use of intelligent defaults, EJB 3.0 provides a framework that enables anyone with Java skills to develop enterprise beans. The Java Persistence API is also a part of the EJB 3.0 specification, and provides a simple and lightweight framework for Java persistence and object-relational mapping. It is discussed in a separate presentation module in more detail.



This slide lists some resources where you can learn more about EJB 3.0 and JPA. You should also consult the Feature Pack for EJB 3.0 Information Center for more information.

Trademarks, copyrights, and disclaimers

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM WebSphere

EJB, Java, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This document could include technical inaccuracies or typographical errors. IBM may make improvements or changes in the products or programs described herein at any time without notice.

Information is provided "AS IS" without warranty of any kind. THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IBM shall have no responsibility to update this information. IBM products are warranted, if at all, according to the terms and conditions of the agreements (for example, IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products.

IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Note to U.S. Government Users - Documentation related to restricted rights-Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract and IBM Corp.