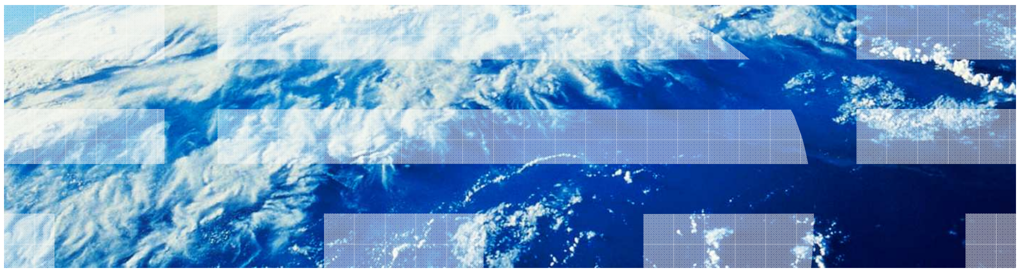
WebSphere Application Server V8

Java EE 6 overview



WebSphere software

This presentation will provide an overview of Java EE 6.

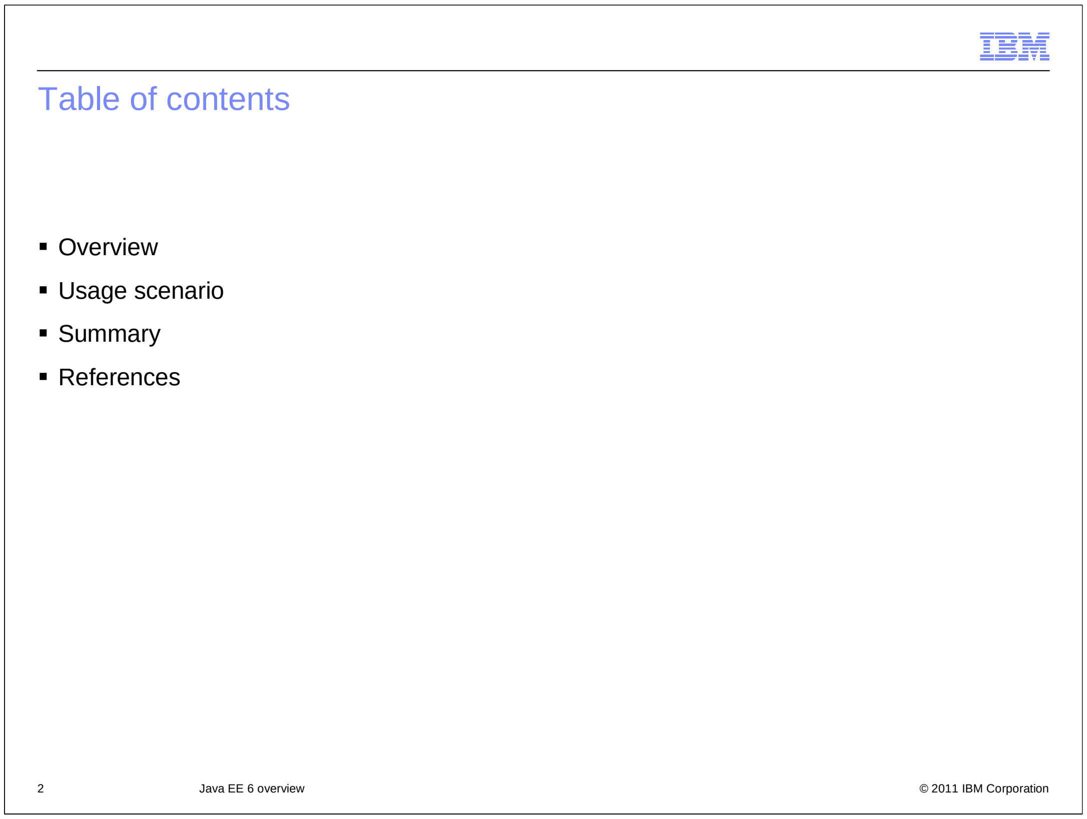


Table of contents

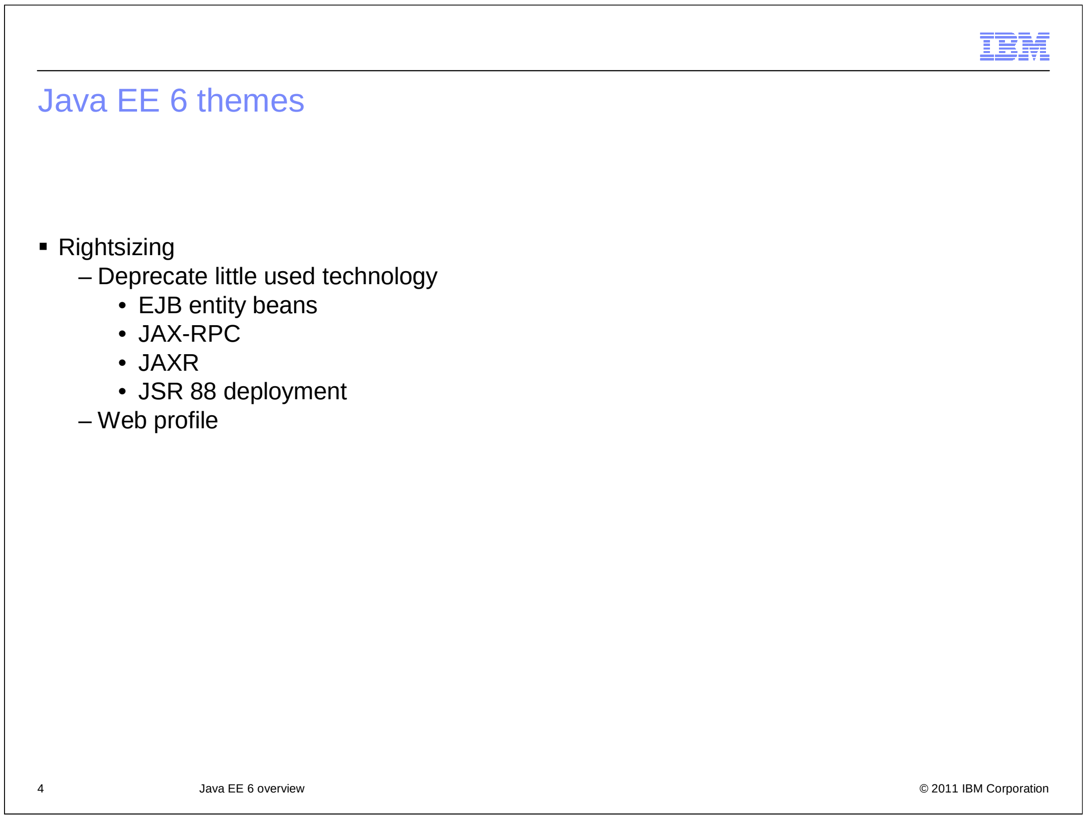
2 Java EE 6 overview © 2011 IBM Corporation

This section will cover what is new in Java EE 6. It begins with the overview of the new content in Java EE 6 followed by a description of how the technologies all work together. However it should be noted that this is not an extensive coverage of all of the new functionality within Java EE 6. It only covers the major additions or changes to the platform.

Section

Overview

This section will introduce Java EE 6.



Java EE 6 themes

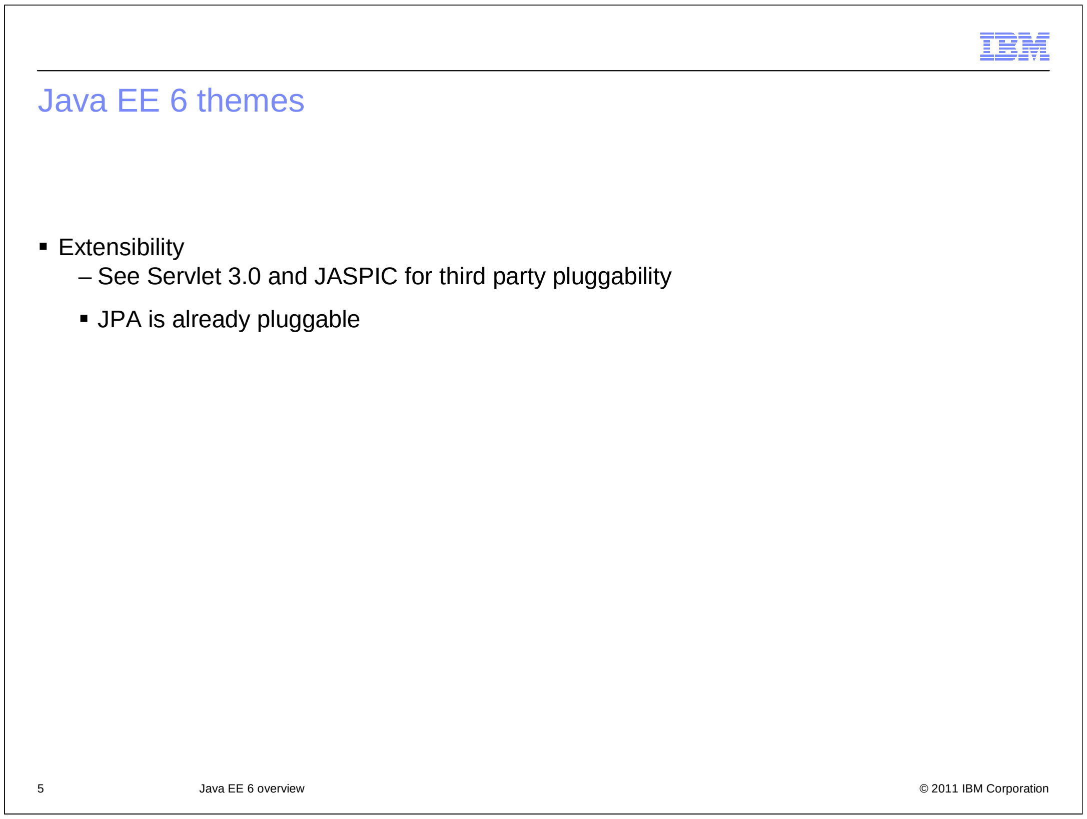
- Rightsizing
  - Deprecate little used technology
    - EJB entity beans
    - JAX-RPC
    - JAXR
    - JSR 88 deployment
  - Web profile

4 Java EE 6 overview © 2011 IBM Corporation

There are several themes in Java EE 6 covering rightsizing, extensibility, and developer productivity improvements. Rightsizing is a way to reduce the footprint size of the EE platform. This covers deprecating technologies that are rarely used by the industry or have been replaced by other technologies and includes EJB entity beans such as CMP and BMP beans, JAX-RPC (which has been replaced by JAX-WS), JAX-R, and Java EE Deployment.

JAX-R is a web services registry technology that is no longer relevant. It has been replaced by other technologies such as UDDI and the WebSphere Web Services Registry and Repository, but there is no Java EE replacement technology at this time. Java EE Deployment is rarely used by customers because it is not rich enough to handle application deployments in most circumstances and therefore proprietary commands for deploying applications are almost always used. There is no replacement Java EE technology for Java EE Deployment at this time.

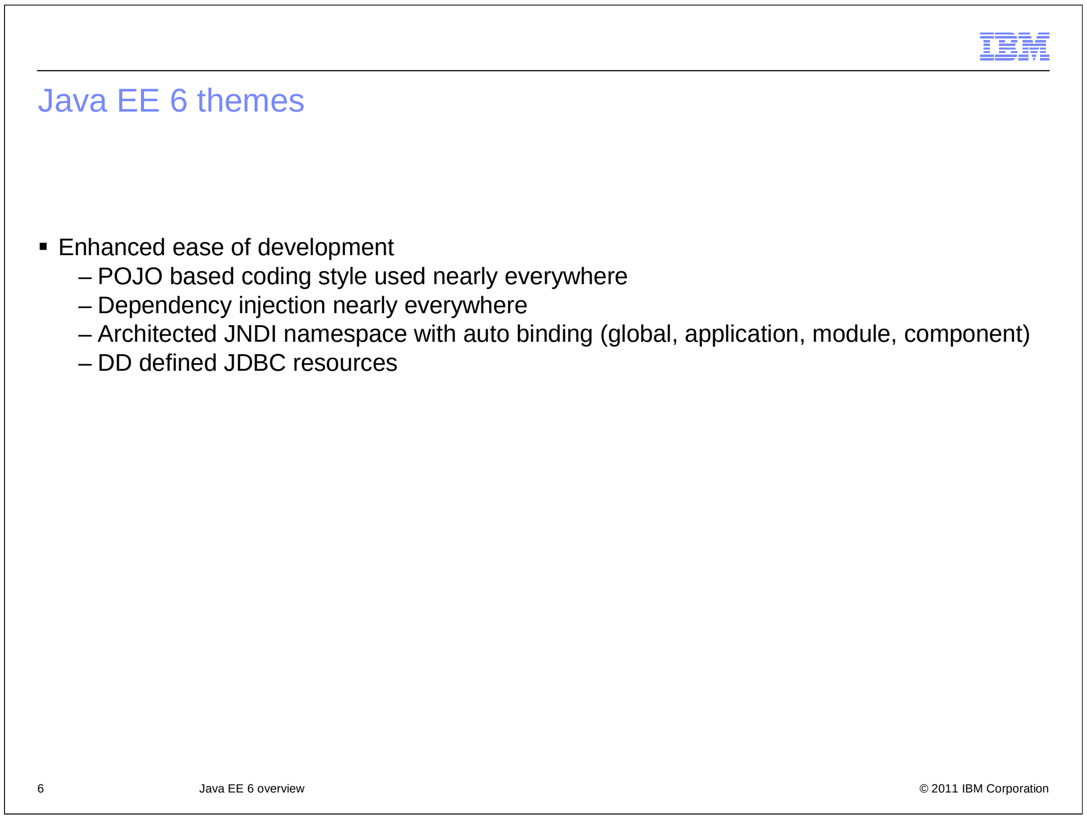
In addition to deprecating little used technologies the Java EE platform has also defined a web profile. This profile is intended to address a smaller footprint requirement for basic web applications.



Java EE 6 themes

- Extensibility
  – See Servlet 3.0 and JASPIC for third party pluggability
  - JPA is already pluggable

Extensibility is supported through several additions for plugability. You can see the Servlet 3.0 and JASPIC specifications for more details on how to plug in third party web application and security frameworks. Be aware that JASPIC makes assumptions that make it difficult to fully integrate with WebSphere security. If you use JASPIC, you may not be able to take advantage of all the WebSphere security functionality.



Java EE 6 themes

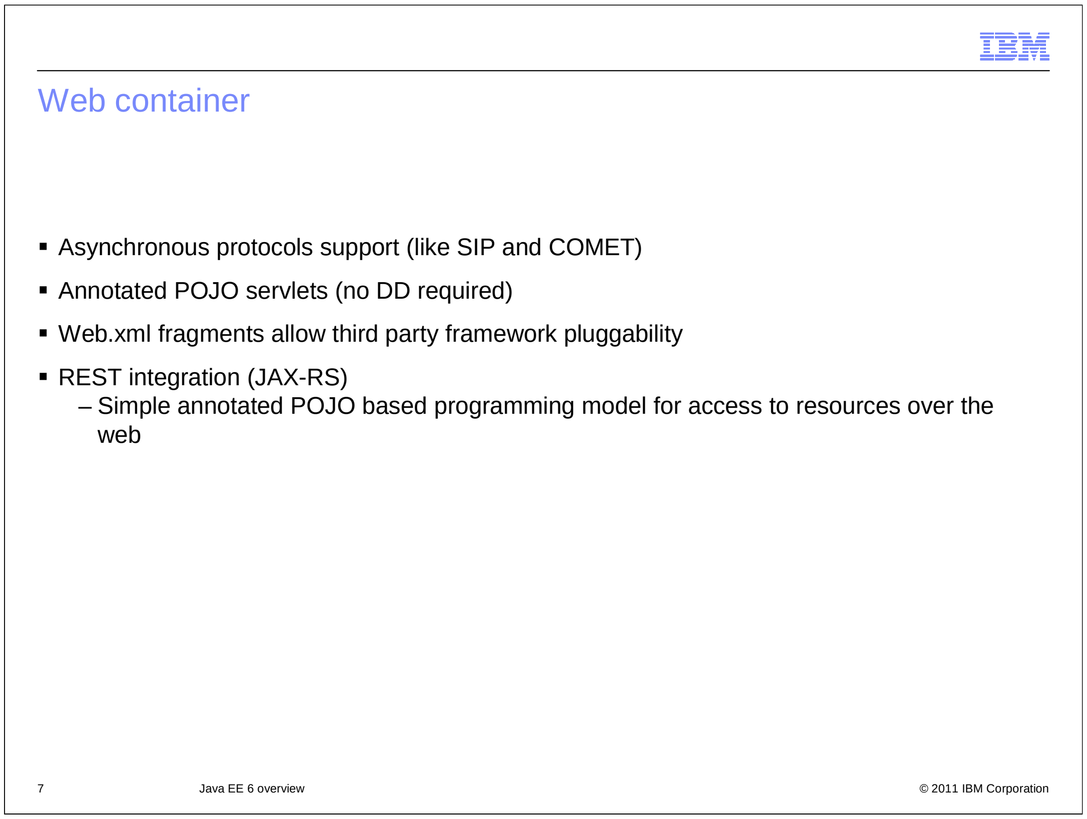
- Enhanced ease of development
  – POJO based coding style used nearly everywhere
  – Dependency injection nearly everywhere
  – Architected JNDI namespace with auto binding (global, application, module, component)
  – DD defined JDBC resources

6 Java EE 6 overview © 2011 IBM Corporation

The most extensive set of changes for Java EE 6 cover simplification of the programming model for Java developers. The POJO, or plain old Java object, coding style is used throughout the platform and makes it very easy for developers to create a class, debug it, and use it within the program. Dependency injection allows developers to get rid of lots of boilerplate code and focus on the business logic or domain logic of the program. For example, a developer used to be required to obtain a JNDI context, perform a JNDI lookup, potentially cast the results, handle exceptions, and configure a deployment descriptor reference in order to obtain a resource dependency. Now, whenever a developer has a dependency on resources or other components they declare the dependency with a simple annotation on a data member and the runtime takes care of satisfying that automatically.

An architected JNDI namespace has been added to extend the scope of bound items from the component level to other scopes including global, application, and module. In addition, all EJBs are automatically bound under the global scope using a well known pattern, making it easy for developers to access EJBs without having to provide vendor specific binding information. This can simplify application deployment as it allows the application to declare a single resource or resource reference that can be shared across multiple modules and bound once by the application deployer. Resource bindings in the architected JNDI namespace can be extended to the global level to share resources across multiple applications.

Developers can also take advantage of defined JDBC resources. These allow developers to define a database resource by annotation or in the deployment descriptor for testing purposes, deploy the application, and test it, without having to configure their JDBC resource through an administration console or script. This improves developer productivity.



Web container

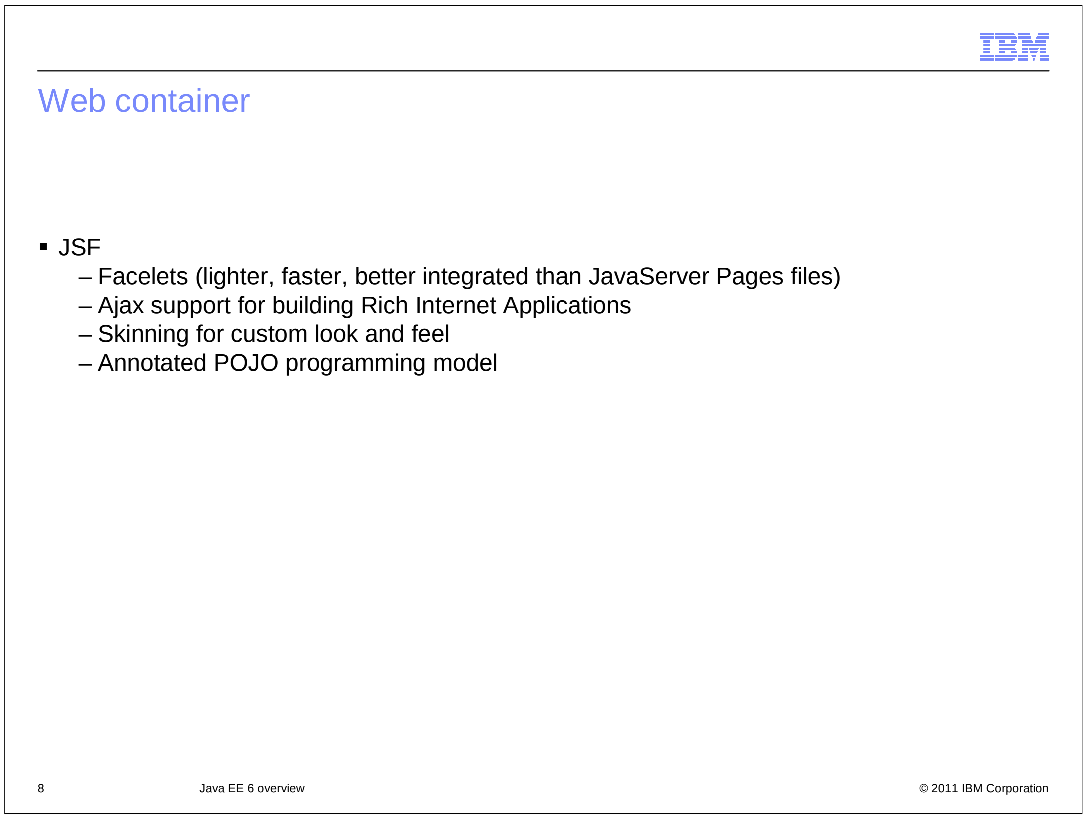
- Asynchronous protocols support (like SIP and COMET)
- Annotated POJO servlets (no DD required)
- Web.xml fragments allow third party framework pluggability
- REST integration (JAX-RS)
  - Simple annotated POJO based programming model for access to resources over the web

7 Java EE 6 overview © 2011 IBM Corporation

The web container now includes asynchronous protocol support, which can be used by higher level protocols such a SIP and COMET. Also, Servlets can now be created using a simple POJO style. All you have to do is write the class, annotate it with a WebServlet annotation, and you are done. There is no need to create an XML deployment descriptor in order to define the Servlet.

Web fragments will simplify the use of third party frameworks. Normally, a web application developer must include in the web framework library in the WAR file and configure the deployment descriptor to use the web application framework. Web fragments allow the developer to drop in a web framework library and the configuration of the deployment descriptor is automatically merged in without the developer having to do anything. To enable this, a web application framework developer must declare the metadata to merge into the deployment descriptor as a web fragment and include this in the web framework library in order for it to be found and automatically merged.

RESTful services, or JAX-RS, is a new technology in the Java EE platform that defines a simple annotated POJO programming model to access resources over the web. It is similar to web services, but without the complex description of the Web Services Description Language. It provides support for mapping URIs to resource identities and HTTP methods, such as GET and PUT, to operations on those resources. The industry seems to be moving to a RESTful service style of interaction for simple remote data access and utilizing JAX-WS Web services for more complex data interaction patterns.

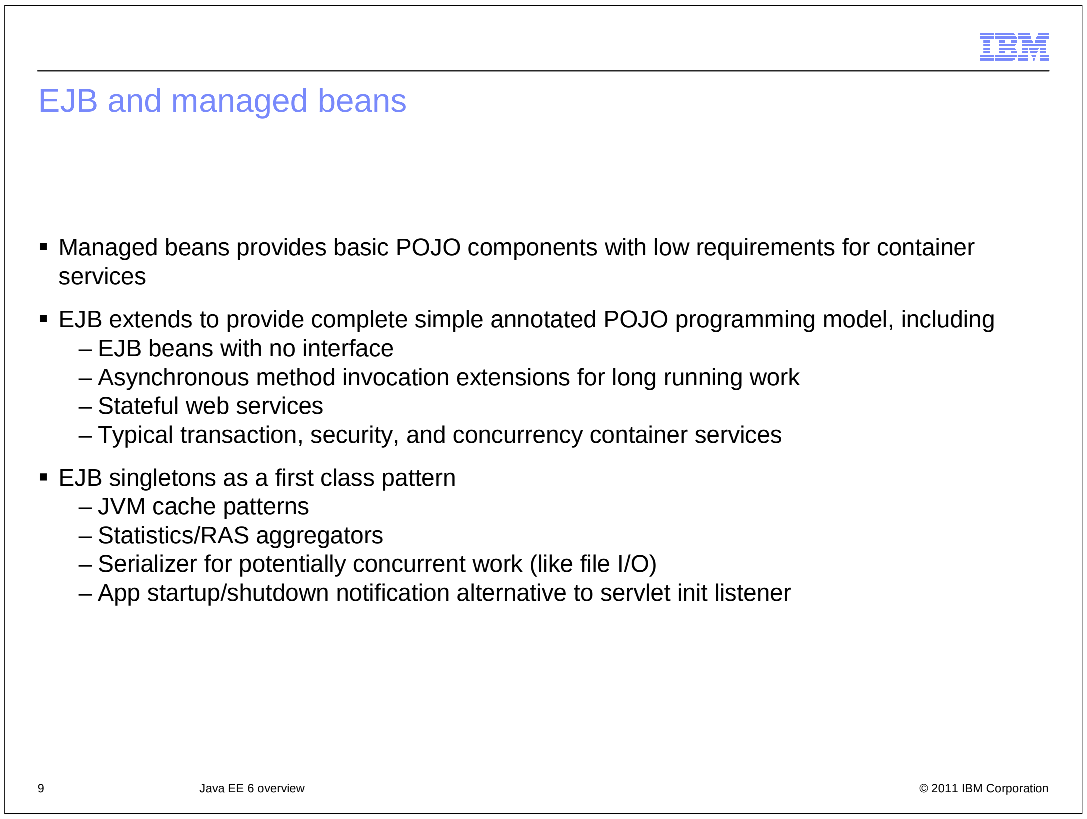


Web container

- JSF
  - Facelets (lighter, faster, better integrated than JavaServer Pages files)
  - Ajax support for building Rich Internet Applications
  - Skinning for custom look and feel
  - Annotated POJO programming model

8 Java EE 6 overview © 2011 IBM Corporation

JavaServer Faces, or JSF, has been significantly enhanced to include support for Facelets, Ajax, skinning, and a simplified programming model. Facelets are similar to JavaServer Pages files, but are lighter weight, more performant, and better integrated with JSF than JSPs and they provide a mechanism for using templates and UI composition. JSF provides life cycle callbacks for Ajax and the rendering model supports custom look and feel selection. Any JSF 2.0 widget library, including those that use Ajax for rich interaction, may be used to simplify UI development. When creating Managed Beans to handle actions and events, developers can now utilize a simple annotated POJO programming model with injection support. It is no longer required to declare Managed Beans in the faces-config.xml file. A new technology called Contexts and Dependency Injection, or CDI, has been integrated with JSF to provide a much richer mechanism for composing Managed Beans. This, in combination with the ability to package EJBs in WAR files and the use of CDI to directly refer to EJBs from Facelets, can greatly simplify a web application developer's life.



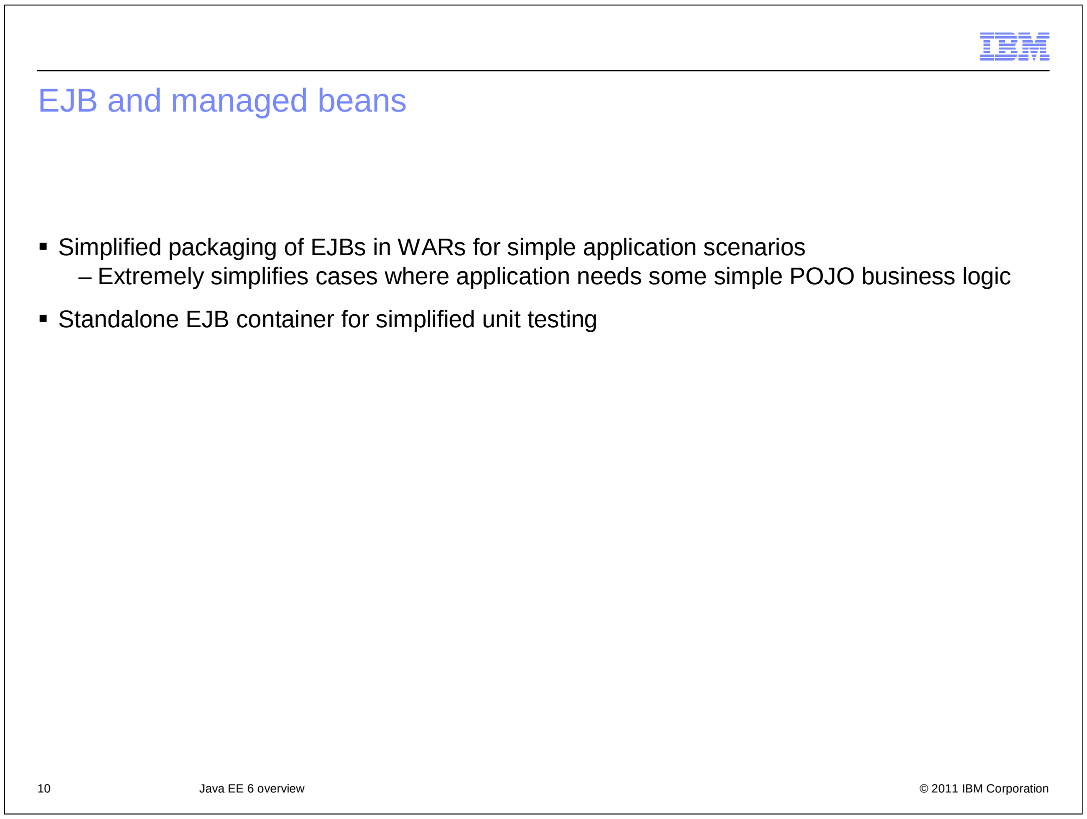
EJB and managed beans

- Managed beans provides basic POJO components with low requirements for container services
- EJB extends to provide complete simple annotated POJO programming model, including
  - EJB beans with no interface
  - Asynchronous method invocation extensions for long running work
  - Stateful web services
  - Typical transaction, security, and concurrency container services
- EJB singletons as a first class pattern
  - JVM cache patterns
  - Statistics/RAS aggregators
  - Serializer for potentially concurrent work (like file I/O)
  - App startup/shutdown notification alternative to servlet init listener

9 Java EE 6 overview © 2011 IBM Corporation

Although Managed Beans have been a part of JSF for some time, the Java EE 6 platform now defines them as a first class component model. Managed Beans provide a simple light weight POJO component model that will serve as the basis for the future of business logic development in the Java EE platform. Managed Beans support basic life cycle services and dependency injection and EJBs extend the simple lightweight programming model to provide enterprise qualities of service such as remote invocation, transactions, security, and concurrency. In the future, you should expect that the EJB container services may be individually selectable and usable with Managed Beans on an as needed basis.

The EJB programming model has been extended in several ways to make life easier for developers. One extension is to allow local beans with no interface. This makes it very easy for a developer to start out with Managed Beans and, with the addition of a simple annotation, obtain the enterprise qualities of service of EJBs without changing any more of the application. Asynchronous method invocation has also been added to allow long running methods to work in parallel. In addition, stateful web services support has been added to provide a consistent programming model with the web container, making it easier to add container managed transaction support to existing web services that may be packaged within a WAR file. There are a number of scenarios that require a singleton pattern such as caching within a JVM, aggregators of data, serializing concurrent work, and application life cycle events. EJBs now support singletons as a first class programming model.

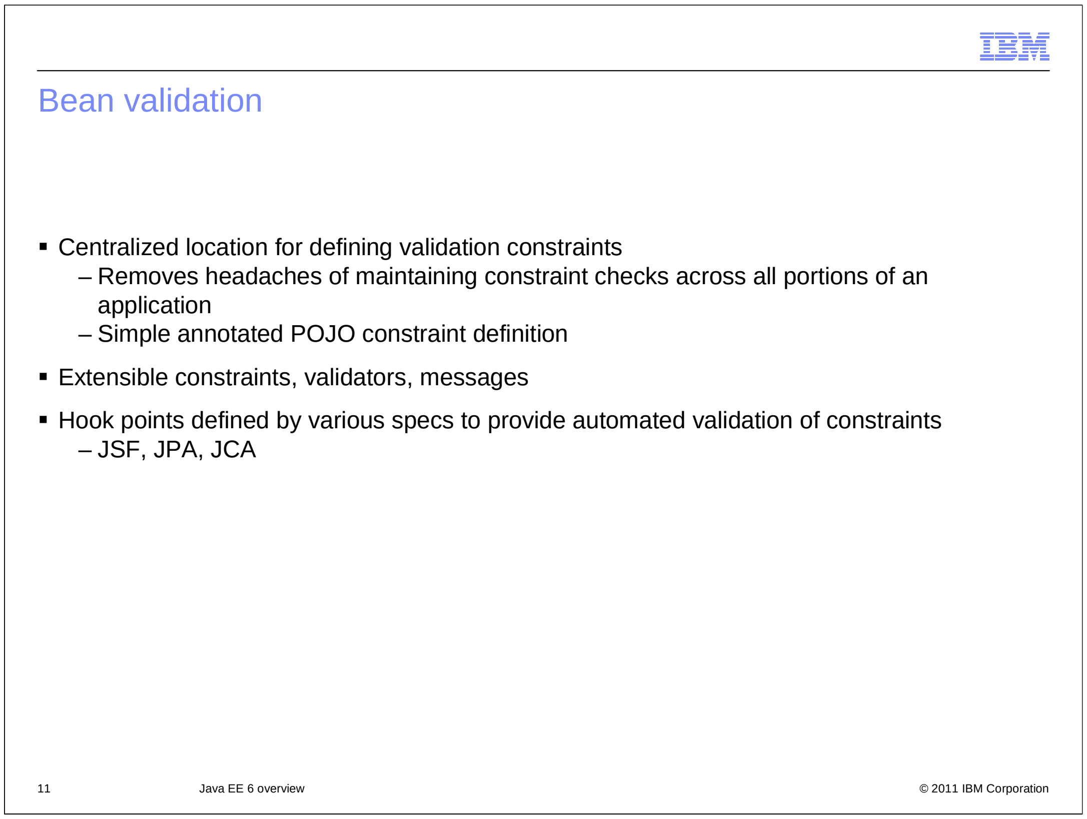


EJB and managed beans

- Simplified packaging of EJBs in WARs for simple application scenarios
  - – Extremely simplifies cases where application needs some simple POJO business logic
- Standalone EJB container for simplified unit testing

10 Java EE 6 overview © 2011 IBM Corporation

One of the more interesting extensions is that EJBs may now be packaged in WAR files. There is no longer an artificial barrier that requires a web application developer to package their EJBs in a separate module. This extremely simplifies cases where web applications need to use transactional session facade components for persistence. In addition, a stand-alone EJB container can now be used to simplify unit testing. This, in combination with CDI injection and JSF integration makes it much easier for developers to write their applications, package, and test them.

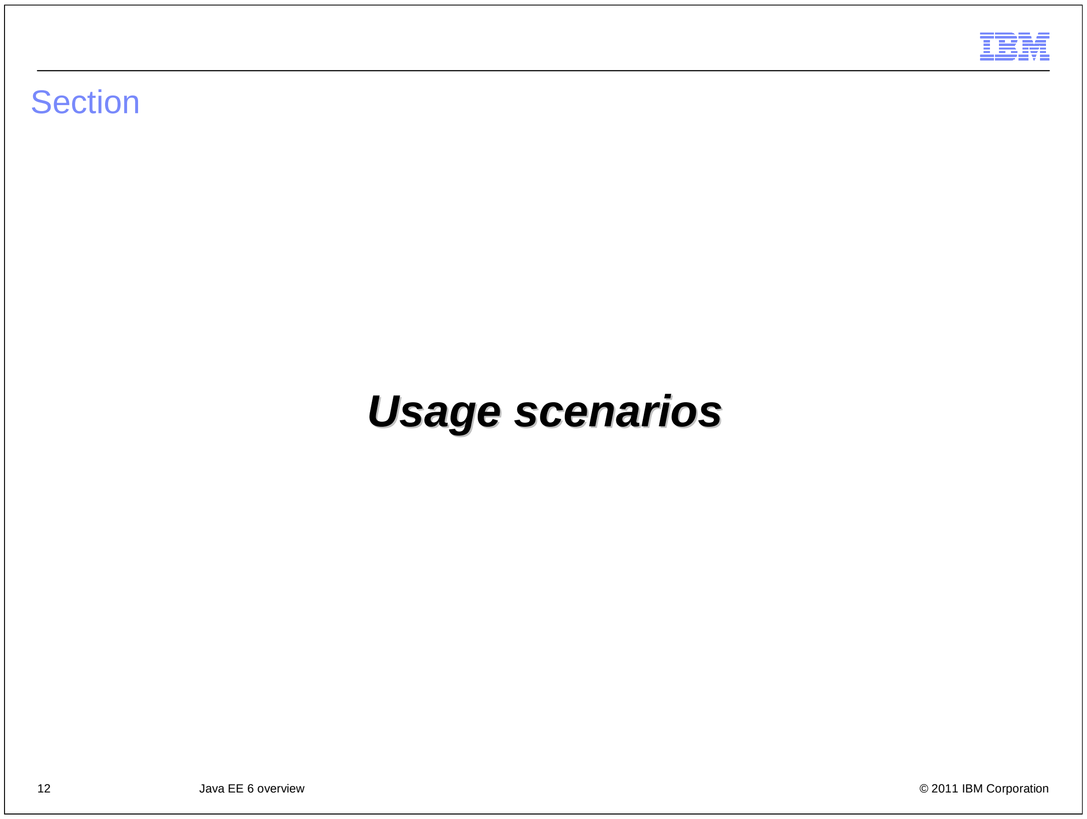


Bean validation

- Centralized location for defining validation constraints
  - Removes headaches of maintaining constraint checks across all portions of an application
  - Simple annotated POJO constraint definition
- Extensible constraints, validators, messages
- Hook points defined by various specs to provide automated validation of constraints
  - JSF, JPA, JCA

11 Java EE 6 overview © 2011 IBM Corporation

One of the problems in developing and maintaining an application is with ensuring valid and consistent data throughout the application. A developer today must write validation logic in the application in many places in order to ensure data integrity is maintained. Keeping all the validation logic in sync can be a problem. To solve this problem, the Java EE platform has added Bean Validation technology that allows the constraint checks for data to be defined in a central location and the logic for validating the data can easily be invoked from anywhere within the application. Simple annotations provide the constraint definition and the constraint can be extended as needed for more complex validations. The platform provides automatic validation of data at various points including JSF form submission, Java Persistence data storage, and connector configuration validation, but this can be extended with specific application invocations as well.

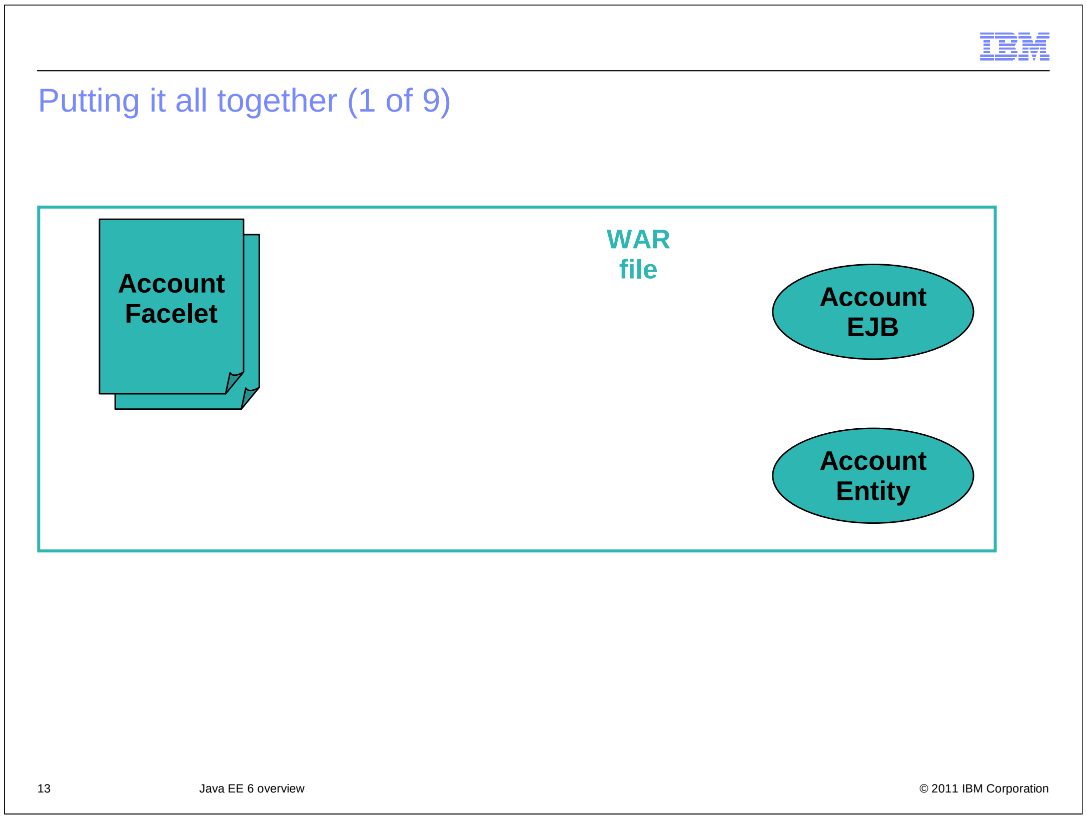


Section

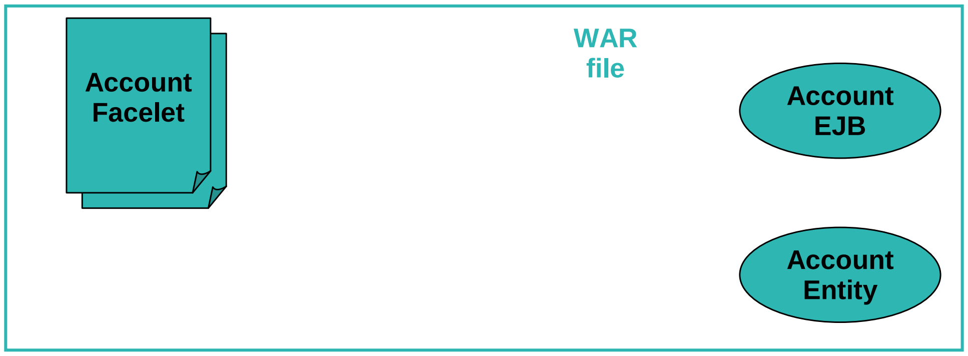
Usage scenarios

12 Java EE 6 overview © 2011 IBM Corporation

This section will cover a usage pattern that shows these technologies working together. The scenario is based on a web application that provides access to account information.

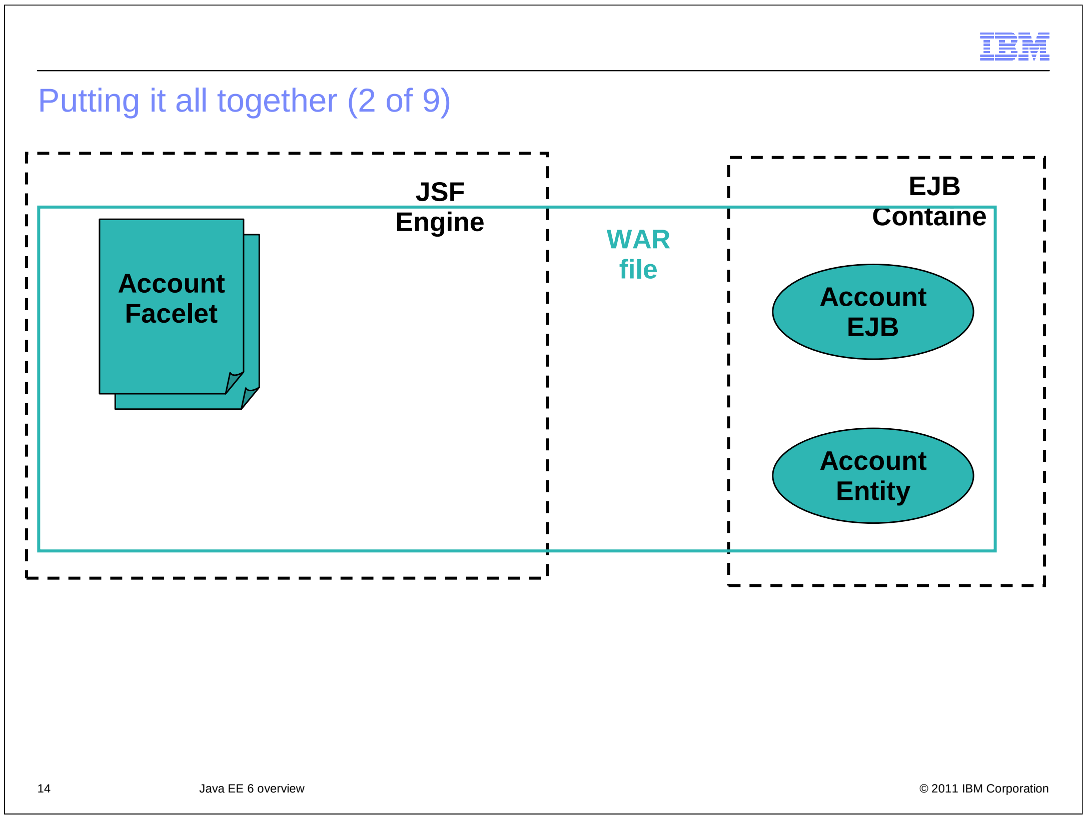


Putting it all together (1 of 9)

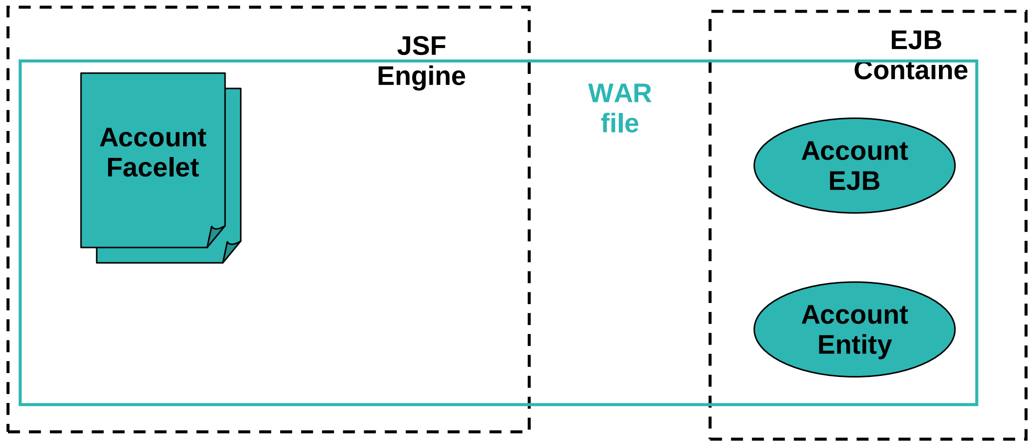


13 Java EE 6 overview © 2011 IBM Corporation

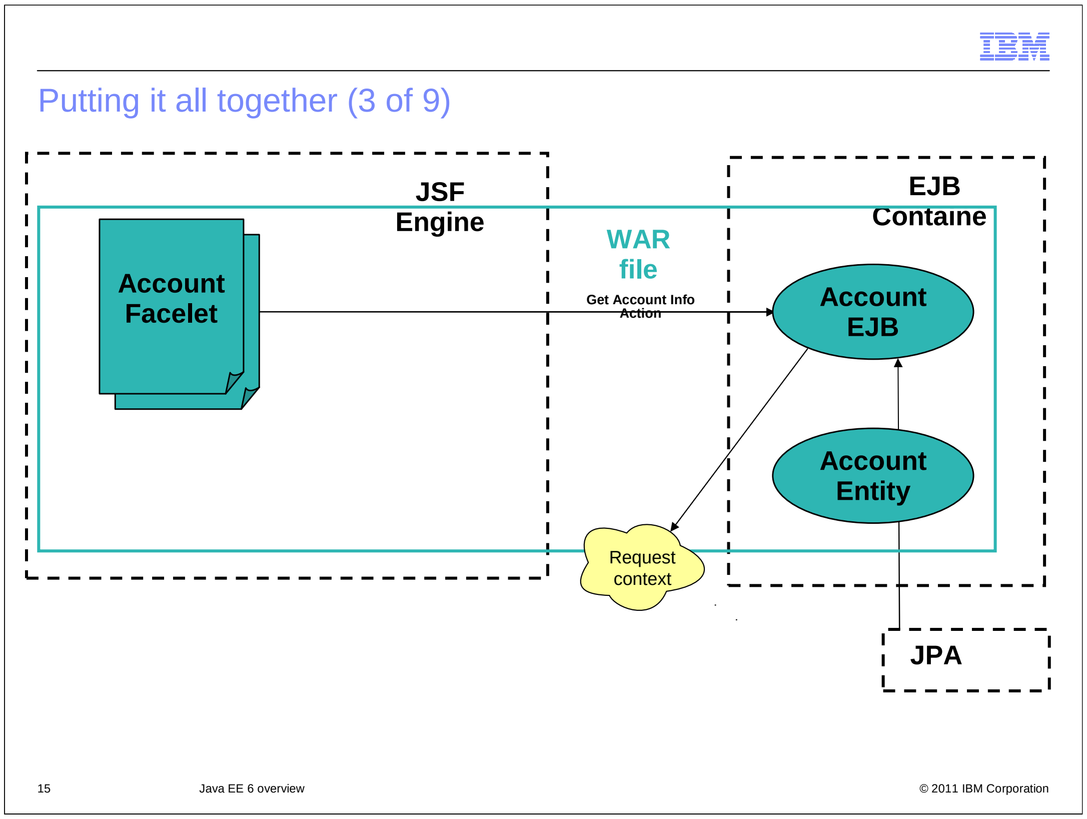
This example begins on the development side where the developer has created an account entity that represents an Object-Relational mapping of a database table to a Java object, an account session EJB that works as a transactional session facade for the account entity, and an account Facelet for visualizing the account entity in a web application. The account entity will be annotated with constraints to ensure the data integrity of the account data. The example will also use the account EJB as a Managed Bean to handle JSF actions. These are packaged in a single WAR file with no deployment descriptor, as all the metadata is represented as annotations in the source code.



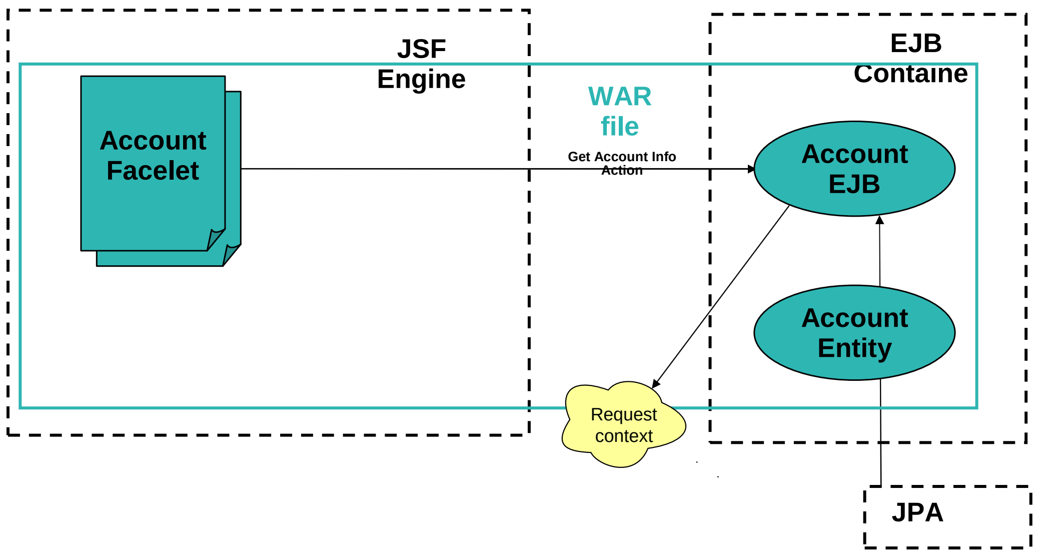
Putting it all together (2 of 9)



When the WAR file is deployed, the JSF Facelets are registered with the JSF engine and the EJB container starts up the session facade EJB.

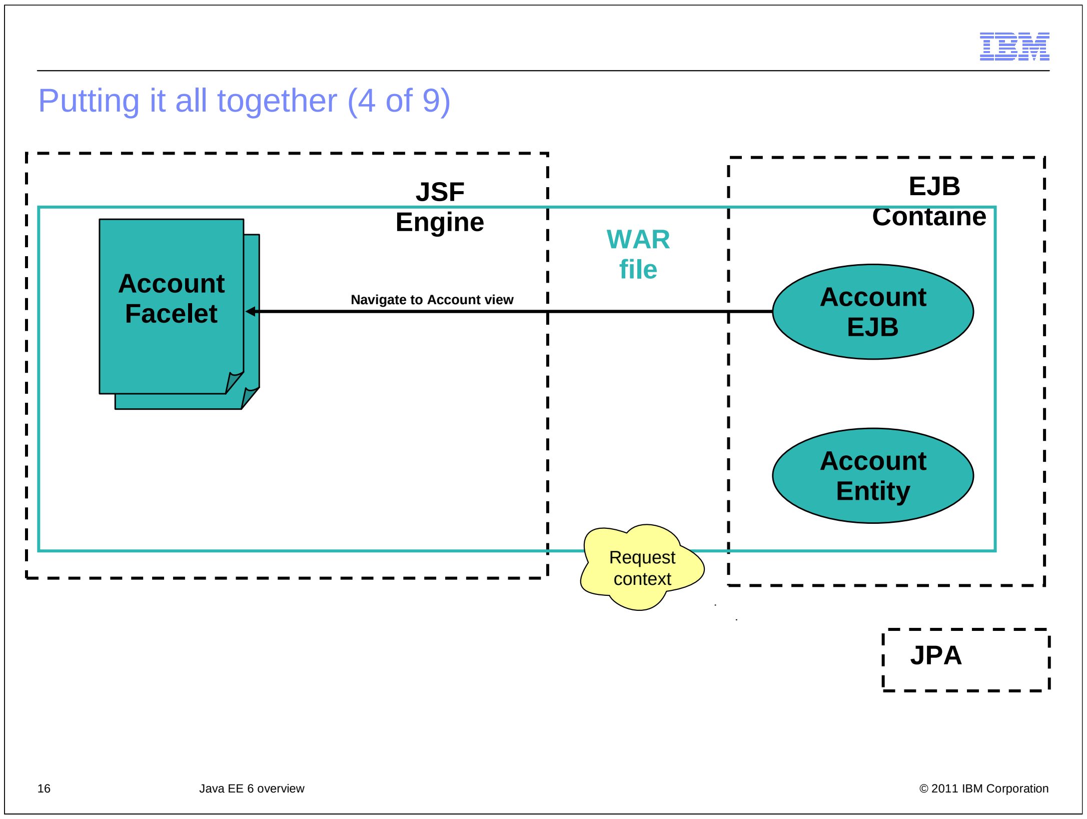


Putting it all together (3 of 9)

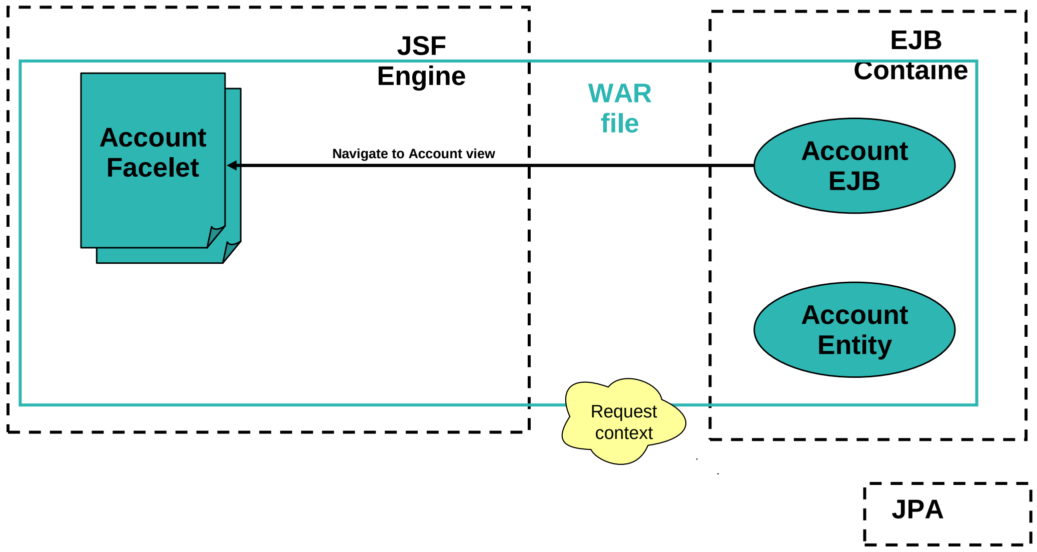


15 Java EE 6 overview © 2011 IBM Corporation

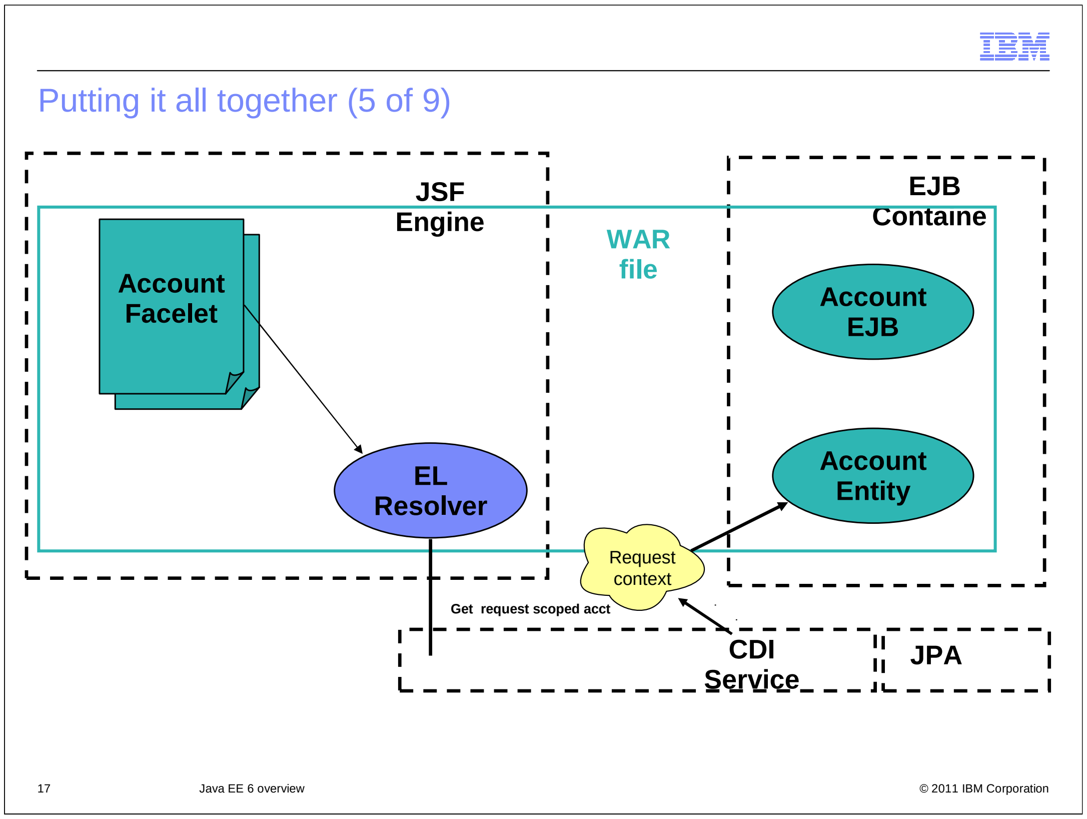
At some point in the execution of the web application the user will press a button to retrieve the account information. This will drive an action event on the account EJB and the session facade will fetch the account information from the database using JPA and store that information as part of the request context.



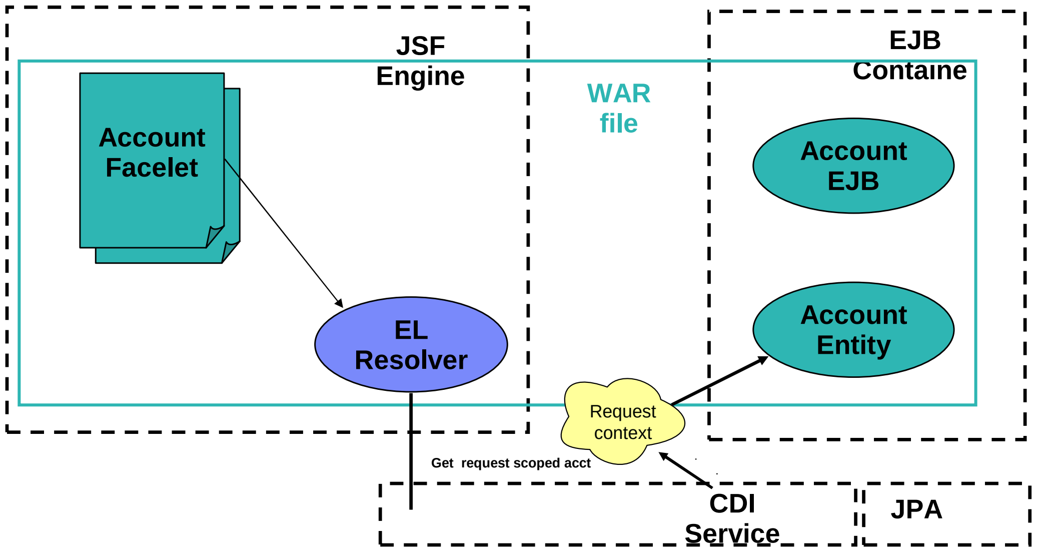
Putting it all together (4 of 9)



It then returns navigation information for the next displayed web page to be handled by the account Facelet.

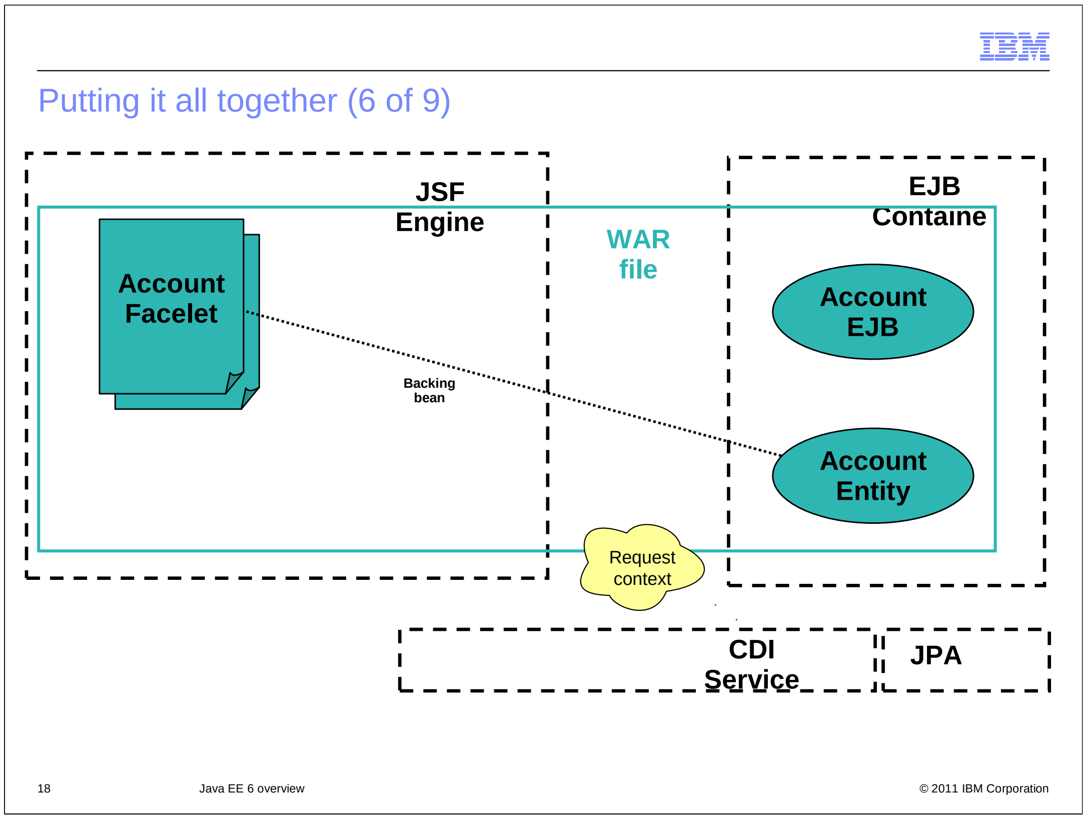


Putting it all together (5 of 9)

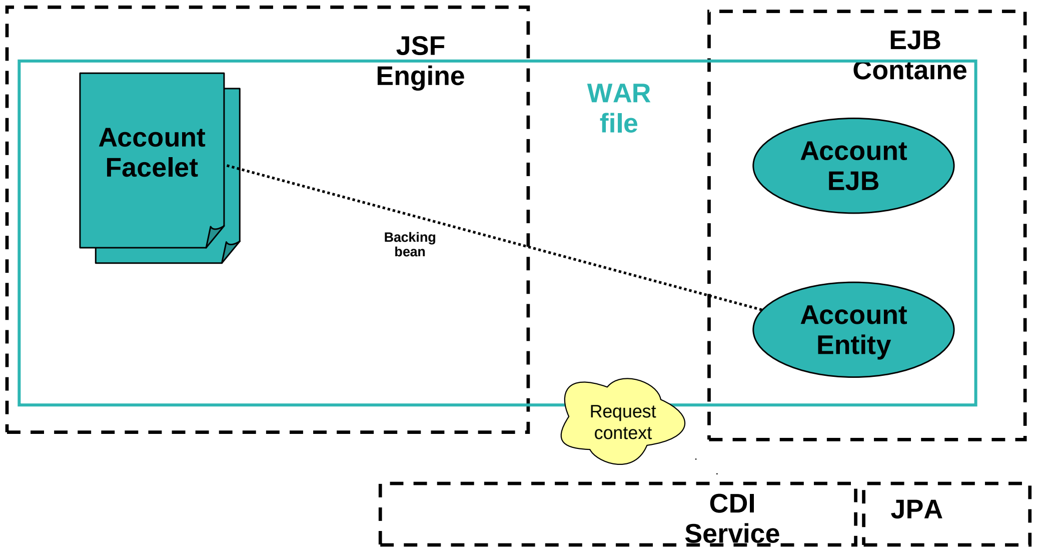


17 Java EE 6 overview © 2011 IBM Corporation

The JSF engine will render the account Facelet, utilizing a template layout, and in the process of doing so will use the EL resolver to obtain values for the account information fields of the Facelet. The EL resolver will then ask the CDI Service for an appropriate instance of the account entity. The CDI Service will look in the request context to see if an instance already exists and since it does it will return that to the EL resolver. The EL resolver will then use the account entity instance to populate the fields of the form and display that to the browser. Note that CDI supports managing other contexts such as session and application contexts as well.

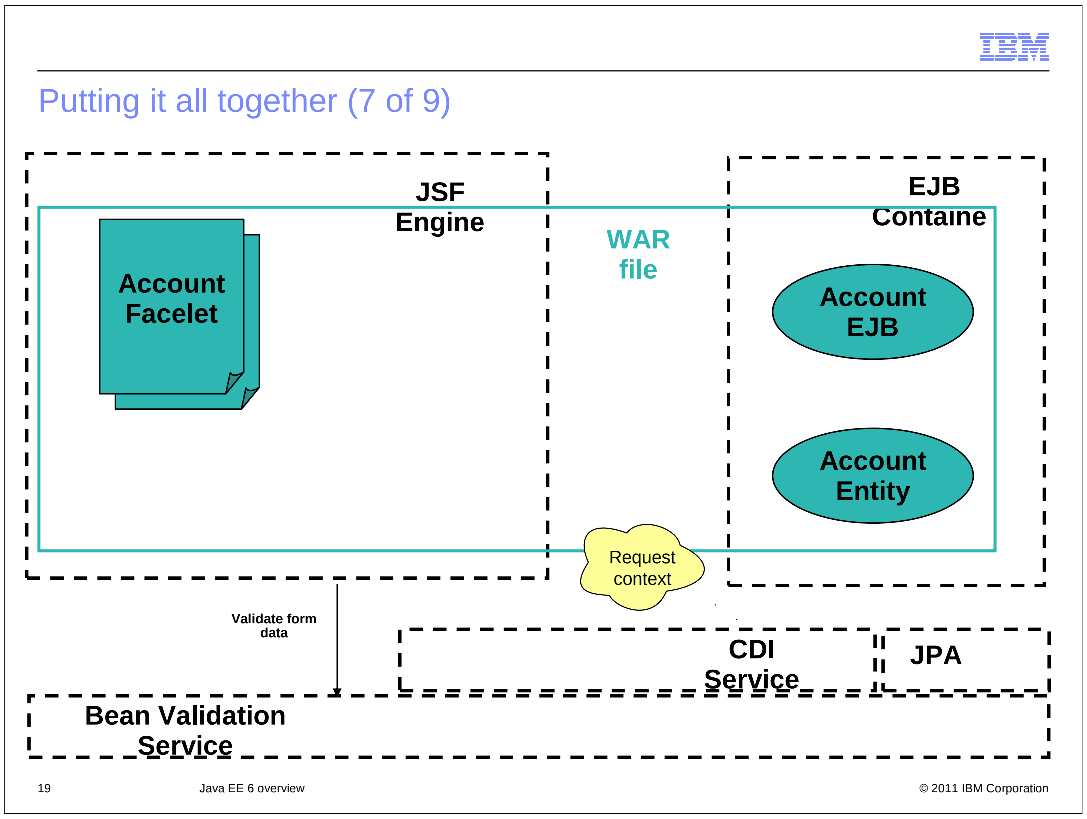


Putting it all together (6 of 9)

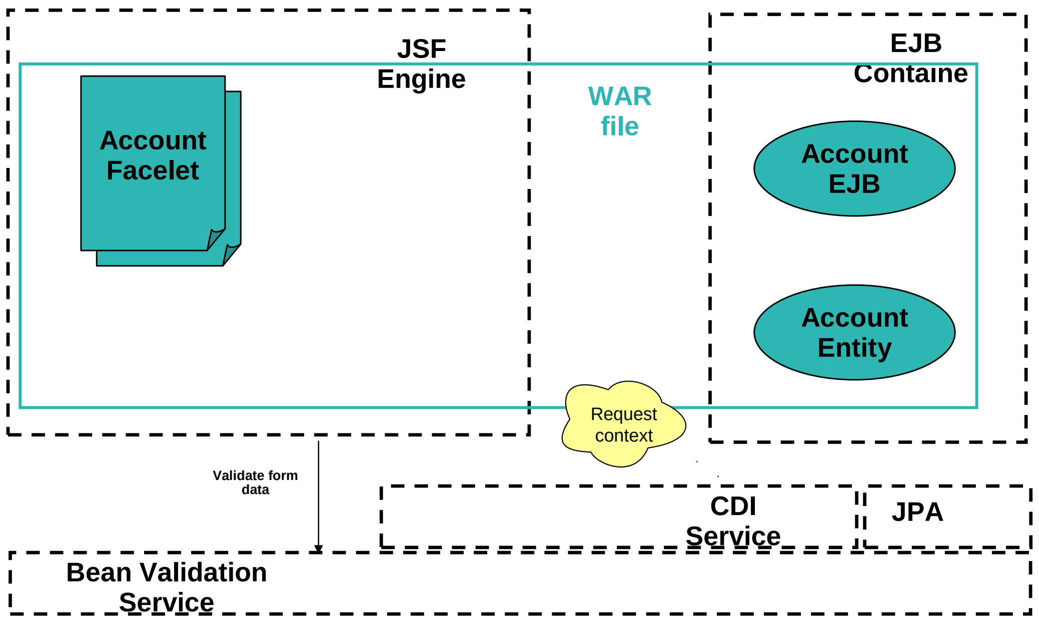


18 Java EE 6 overview © 2011 IBM Corporation

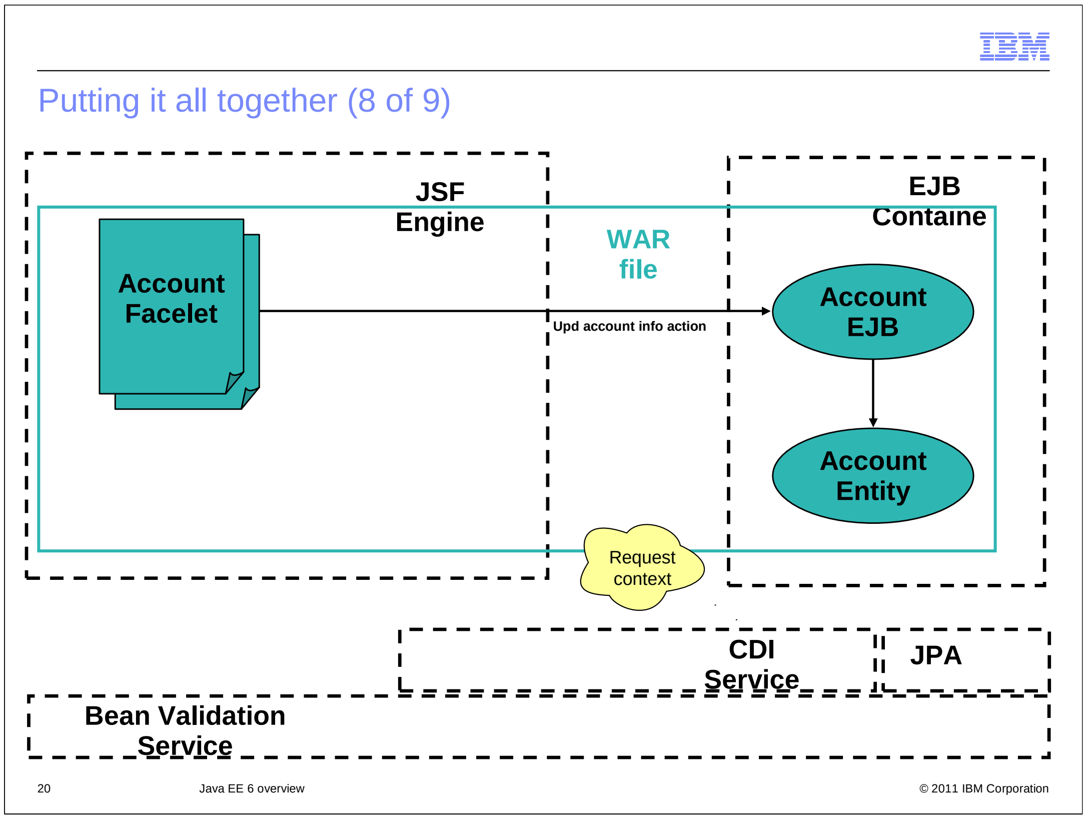
When the user has finished updating the form, they click an update button which submits the form to the JSF engine for processing. The JSF engine creates an account entity instance to use as a backing bean for the request data...



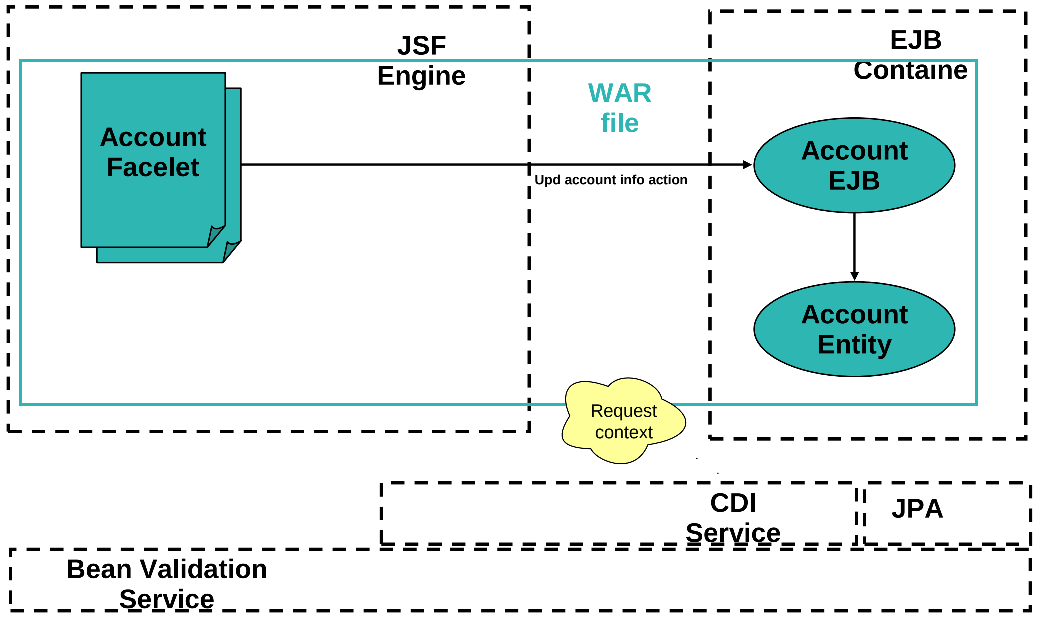
Putting it all together (7 of 9)



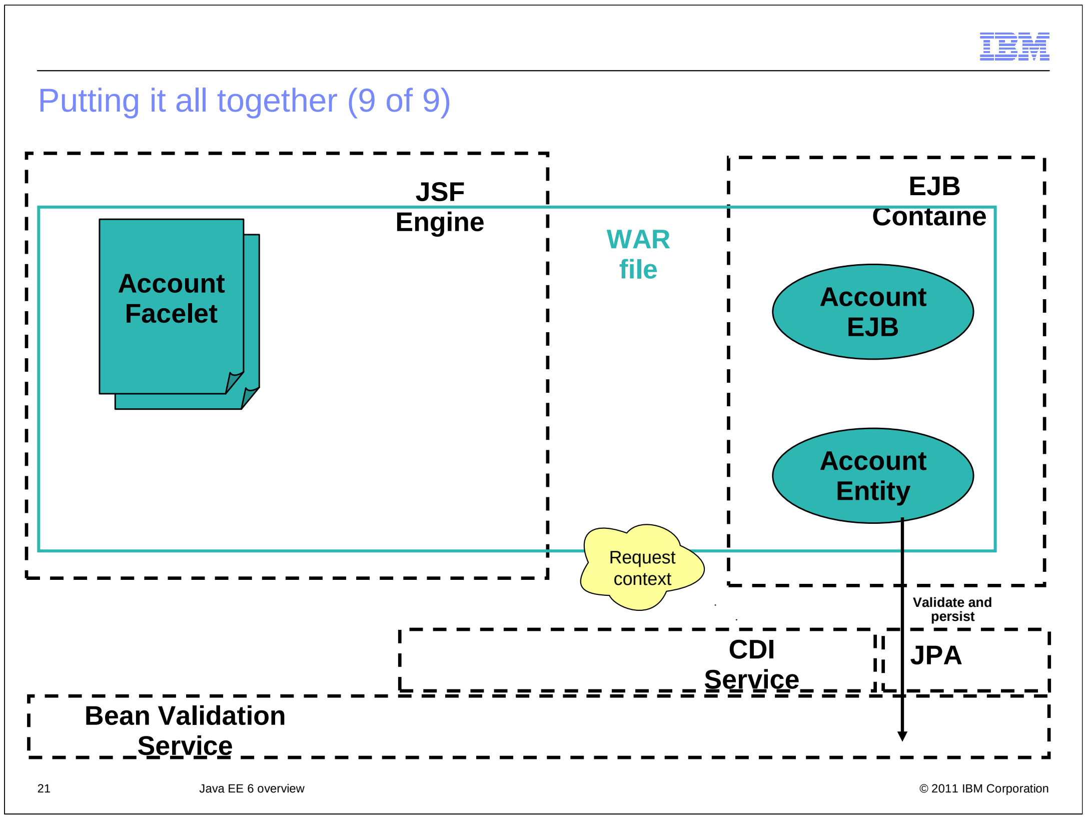
... and because the account entity backing bean has been annotated with validation constraints, it automatically calls the Bean Validation service to validate the form data. If validation fails, constraint specific error messages can be displayed to the user.



Putting it all together (8 of 9)



Once validation has completed, the action bean for the update action is invoked, which happens to be the session facade EJB. The session facade EJB may perform additional business logic to calculate last update times, possibly changing the account entity information.



Putting it all together (9 of 9)

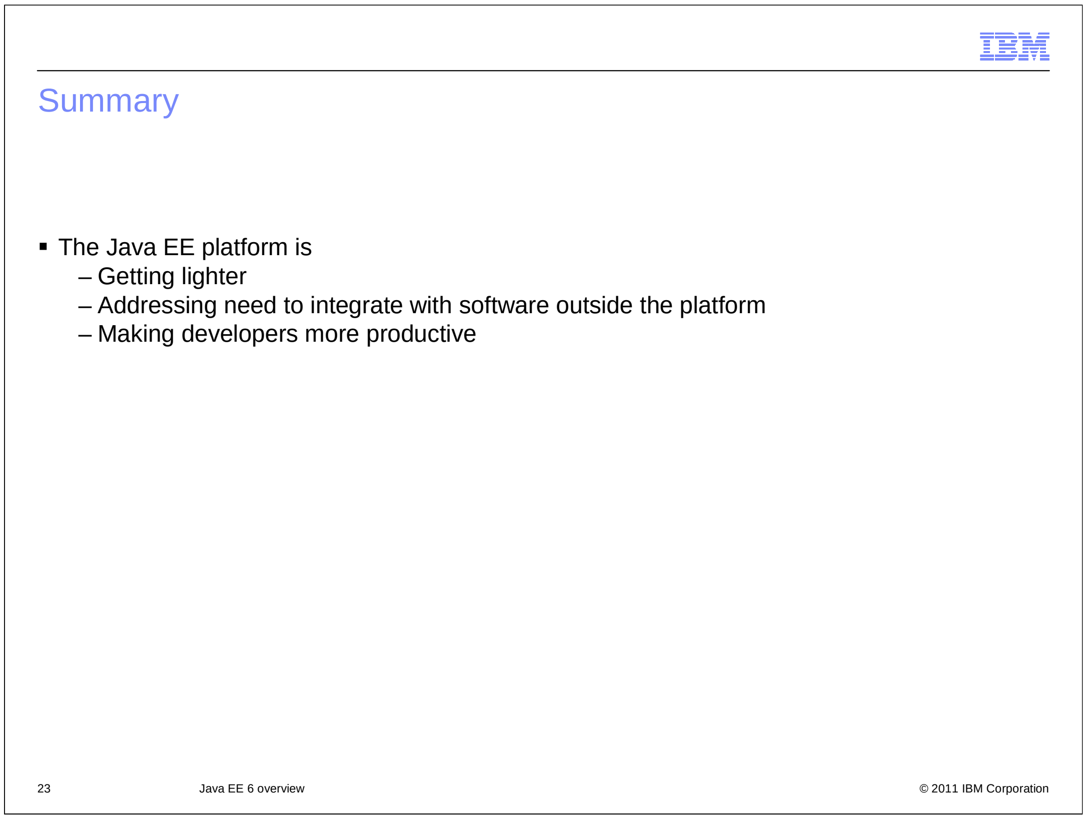
When ready, the session facade EJB persists the account entity information using JPA. JPA notices the validation constraints associated with the account entity and automatically calls the Bean Validation service once again to ensure data integrity. If validation fails, the persistence operation fails. Once validation is complete, JPA will persist the account entity information into the database.

Section

Summary

22 Java EE 6 overview © 2011 IBM Corporation

This section will cover the key take away points.

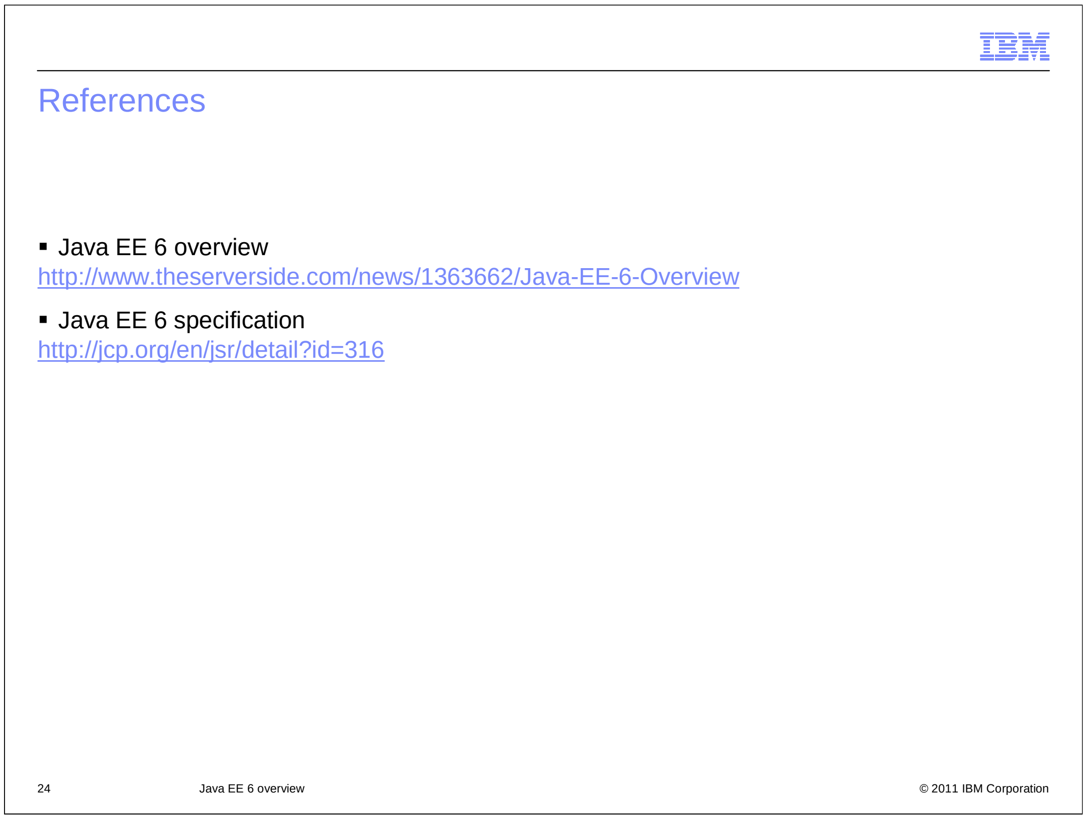


Summary

- The Java EE platform is
  – Getting lighter
  – Addressing need to integrate with software outside the platform
  – Making developers more productive

23 Java EE 6 overview © 2011 IBM Corporation

In summary, the Java EE 6 platform has been extended to make developers much more productive by making it easier to integrate third party software with the platform, providing a lighter weight platform that improves developer edit compile and debug cycles, and reducing the amount of code a developer has to write, debug, and configure.

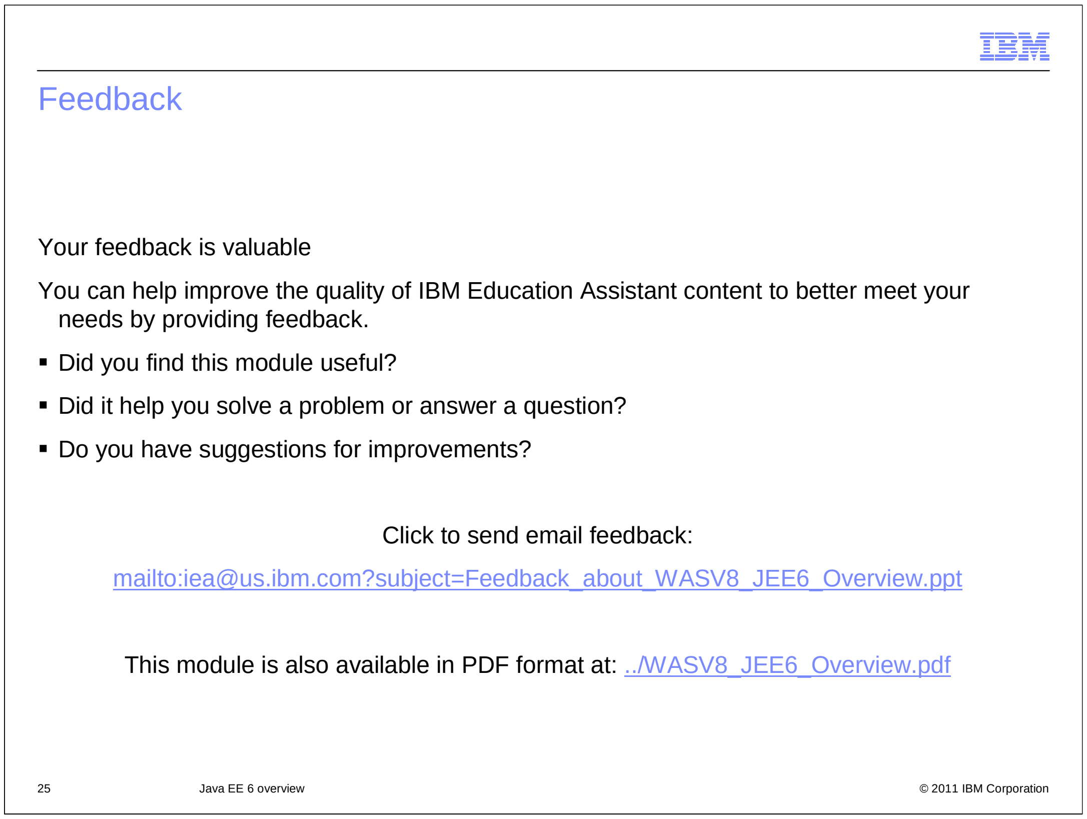


References

- Java EE 6 overview
http://www.theserverside.com/news/1363662/Java-EE-6-Overview
- Java EE 6 specification
http://jcp.org/en/jsr/detail?id=316

24 Java EE 6 overview © 2011 IBM Corporation

This slide contains links to useful information.


Feedback

Your feedback is valuable

You can help improve the quality of IBM Education Assistant content to better meet your needs by providing feedback.

- Did you find this module useful?
- Did it help you solve a problem or answer a question?
- Do you have suggestions for improvements?

Click to send email feedback:

mailto:iea@us.ibm.com?subject=Feedback_about_WASV8_JEE6_Overview.ppt

This module is also available in PDF format at: ../WASV8_JEE6_Overview.pdf

You can help improve the quality of IBM Education Assistant content by providing feedback.

Trademarks, disclaimer, and copyright information