Enterprise Java Overview

* The Java EE platform is designed to help developers create large-scale, multi-tiered, scalable, reliable, and secure network applications. A shorthand name for such applications is "enterprise applications,"
* An application server is the sum of its Java EE containers.
* A Java EE application is packaged into one or more standard units for deployment to any Java EE platform-compliant system. Each unit contains
  + A functional component or components, such as an enterprise bean, web page, servlet, or applet
  + An optional deployment descriptor that describes its content
* Once a Java EE unit has been produced, it is ready to be deployed. Deployment typically involves using a platform's deployment tool to specify location-specific information, such as a list of local users who can access it and the name of the local database. Once deployed on a local platform, the application is ready to run.

1. Tiered Application

* In a multi-tiered application, the functionality of the application is separated into isolated functional areas, called tiers.
* Typically, multi-tiered applications have a client tier, a middle tier, and a data tier (often called the enterprise information systems tier).
* The client tier consists of a client program that makes requests to the middle tier. **The middle tier** is divided into a **web tier and a business tier**, which handle client requests and process application data, storing it in a permanent datastore in the data tier.
* **Java EE application development concentrates on the middle tier to make enterprise application management easier, more robust, and more secure.**
* The client tier consists of application clients that access a Java EE server and that are usually located on a different machine from the server. The clients make requests to the server. The server processes the requests and returns a response back to the client.
* Many different types of applications can be Java EE clients, and they are not always, or even often Java applications. Clients can be a web browser, a standalone application, or other servers, and they run on a different machine from the Java EE server.
* The web tier consists of components that handle the interaction between clients and the business tier. Its primary tasks are the following:
  + Dynamically generate content in various formats for the client
  + Collect input from users of the client interface and return appropriate results from the components in the business tier
  + Control the flow of screens or pages on the client
  + Maintain the state of data for a user's session
  + Perform some basic logic and hold some data temporarily in managed beans
* Following are some of the main Java EE technologies that are used in the web tier in Java EE applications.
  + JavaServer Faces technology
  + Expression Language
  + Servlets - Java programming language classes that dynamically process requests and

construct responses, usually for HTML pages

* + Contexts and Dependency Injection for Java EE
* The business tier consists of components that provide the business logic for an application. Business logic is code that provides functionality to a particular business domain, like the financial industry, or an e-commerce site. In a properly designed enterprise application, the core functionality exists in the business tier components.
* The following Java EE technologies are among those that are used in the business tier in Java EE applications:
  + Enterprise JavaBeans (enterprise bean) components
  + JAX-RS RESTful web services
  + Java Persistence API entities
* The enterprise information systems (EIS) tier consists of database servers, enterprise resource planning systems, and other legacy data sources, like mainframes. These resources typically are located on a separate machine from the Java EE server, and are accessed by components on the business tier.
* The following Java EE technologies are used to access the EIS tier in Java EE applications:
  + The Java Database Connectivity API (JDBC)
  + The Java Persistence API
  + The Java EE Connector Architecture
  + The Java Transaction API (JTA)

1. JAVA EE Servers and Containers

* A Java EE server is a server application that implements the Java EE platform APIs and provides standard Java EE services.

* Java EE servers are sometimes called application servers, because they allow you to serve application data to clients, much as web servers serve web pages to web browsers.
* Java EE servers host several application component types that correspond to the tiers in a multi-tiered application.
* The Java EE server provides services to these components in the form of a container.
* Java EE containers are the interface between the component and the lower-level functionality provided by the platform to support that component.
* The functionality of the container is defined by the platform and is different for each component type. Nonetheless, the server allows the different component types to work together to provide functionality in an enterprise application.
* The **web container** is the interface between web components and the web server. A web component can be a servlet or a JavaServer Faces Facelets page. The container manages the component's lifecycle, dispatches requests to application components, and provides interfaces to context data, such as information about the current request.
* The **EJB container** is the interface between enterprise beans, which provide the business logic in a Java EE application, and the Java EE server. The EJB container runs on the Java EE server and manages the execution of an application's enterprise beans.
* The **application client** **container** is the interface between Java EE application clients (special Java SE applications that use Java EE server components) and the Java EE server. The application client container runs on the client machine and is the gateway between the client application and the Java EE server components that the client uses.
* The Java EE application model begins with the Java programming language and the Java virtual machine
* The Java EE application model defines an architecture for implementing services as multitier applications that deliver the scalability, accessibility, and manageability needed by enterprise-level applications. This model partitions the work needed to implement a multitier service into the following parts:
  + The business and presentation logic to be implemented by the developer
  + The standard system services provided by the Java EE platform
* The developer can rely on the platform to provide solutions for the hard systems-level problems of developing a multitier service.

# Distributed Multitiered Applications

* The Java EE platform uses a distributed multitiered application model for enterprise applications
* Application logic is divided into components according to function, and the application components that make up a Java EE application are installed on various machines depending on the tier in the multitiered Java EE environment to which the application component belongs.
  + Client-tier components run on the client machine.
  + Web-tier components run on the Java EE server.
  + Business-tier components run on the Java EE server.
  + Enterprise information system (EIS)-tier software runs on the EIS server.
* Java EE multitiered applications are generally considered to be three-tiered applications because they are distributed over three locations: client machines, the Java EE server machine, and the database or legacy machines at the back end.
* Three-tiered applications that run in this way extend the standard two-tiered client-and-server model by placing a multithreaded application server between the client application and back-end storage.
* Java EE applications are made up of components.
* A **Java EE component** is a self-contained functional software unit that is assembled into a Java EE application with its related classes and files and that communicates with other components.
* The Java EE specification defines the following Java EE components:
  + Application clients and applets are components that run on the client.
  + Java Servlet, JavaServer Faces, and JavaServer Pages (JSP) technology components are web components that run on the server.
  + EJB components (enterprise beans) are business components that run on the server.
* Java EE components are written in the Java programming language and are compiled in the same way as any program in the language.
* The differences between Java EE components and "standard" Java classes are that Java EE components are assembled into a Java EE application, they are verified to be well formed and in compliance with the Java EE specification, and they are deployed to production, where they are run and managed by the Java EE server.

Java EE Clients

-------------------------

* A **Java EE client** is usually either a web client or an application client.
* A **web client** consists of two parts:
  + Dynamic web pages containing various types of markup language (HTML, XML, and so on), which are generated by web components running in the web tier
  + A web browser, which renders the pages received from the server
* A **web client** is sometimes called a **thin client**. Thin clients usually do not query databases, execute complex business rules, or connect to legacy applications. When you use a thin client, such heavyweight operations are off-loaded to enterprise beans executing on the Java EE server, where they can leverage the security, speed, services, and reliability of Java EE server-side technologies.
* An **application client** runs on a client machine and provides a way for users to handle tasks that require a richer user interface than can be provided by a markup language.
* An application client typically has a graphical user interface (GUI) created from the Swing API or the Abstract Window Toolkit (AWT) API, but a command-line interface is certainly possible.
* Application clients directly access enterprise beans running in the business tier.
* However, if application requirements warrant it, an application client can open an HTTP connection to establish communication with a servlet running in the web tier.
* Application clients written in languages other than Java can interact with Java EE servers, enabling the Java EE platform to interoperate with legacy systems, clients, and non-Java languages.
* Web components are the preferred API for creating a web client program because no plug-ins or security policy files are needed on the client systems. Also, web components enable cleaner and more modular application design because they provide a way to separate applications programming from web page design. Personnel involved in web page design thus do not need to understand Java programming language syntax to do their jobs.
* The server and client tiers might also include components based on the **JavaBeans component architecture** (JavaBeans components) to manage the data flow between the following:
  + An application client or applet and components running on the Java EE server
  + Server components and a database
* JavaBeans components are not considered Java EE components by the Java EE specification.
* JavaBeans components have properties and have get and set methods for accessing those properties. JavaBeans components used in this way are typically simple in design and implementation but should conform to the naming and design conventions outlined in the JavaBeans component architecture.
* The client communicates with the business tier running on the Java EE server either directly or, as in the case of a client running in a browser, by going through web pages or servlets running in the web tier.

Web Components

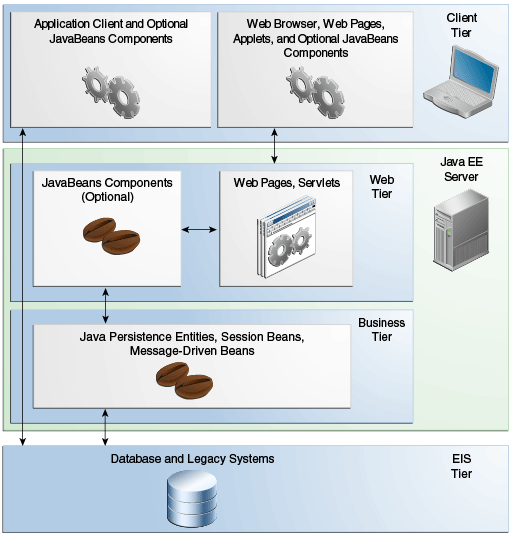
==============

* Java EE web components are either servlets or web pages created using JavaServer Faces technology and/or JSP technology (JSP pages).
* **Servlets** are Java programming language classes that dynamically process requests and construct responses.
* **JSP pages** are text-based documents that execute as servlets but allow a more natural approach to creating static content. **JavaServer Faces technology** builds on servlets and JSP technology and provides a user interface component framework for web applications.
* Static HTML pages and applets are bundled with web components during application assembly but are not considered web components by the Java EE specification.
* Server-side utility classes can also be bundled with web components and, like HTML pages, are not considered web components.

Business Components

=================

* Business code, which is logic that solves or meets the needs of a particular business domain such as banking, retail, or finance, is handled by enterprise beans running in either the business tier or the web tier.



* Enterprise beans are Java Persistence Entities, Session Beans, Message-Driven Beans.

Enterprise Information System Tier

===========================

* The enterprise information system tier handles EIS software and includes enterprise infrastructure systems, such as enterprise resource planning (ERP), mainframe transaction processing, database systems, and other legacy information systems. For example, Java EE application components might need access to enterprise information systems for database connectivity.

# Java EE Containers

* The component-based and platform-independent Java EE architecture makes applications easy to write because business logic is organized into reusable components.
* In addition, the Java EE server provides underlying services in the form of a container for every component type. Because you do not have to develop these services yourself, you are free to concentrate on solving the business problem at hand.

Container Services

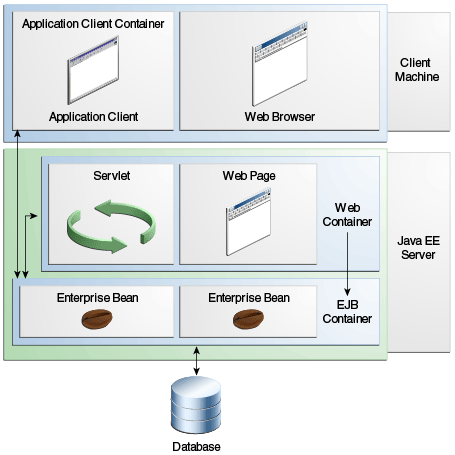
==============

* **Containers** are the interface between a component and the low-level, platform-specific functionality that supports the component.
* Before it can be executed, a web, enterprise bean, or application client component must be assembled into a Java EE module and deployed into its container.
* The assembly process involves specifying container settings for each component in the Java EE application and for the Java EE application itself.
* Container settings customize the underlying support provided by the Java EE server, including such services as security, transaction management, Java Naming and Directory Interface (JNDI) API lookups, and remote connectivity.
* Here are some of the highlights.
  + The Java EE security model lets you configure a web component or enterprise bean so that system resources are accessed only by authorized users.
  + The Java EE transaction model lets you specify relationships among methods that make up a single transaction so that all methods in one transaction are treated as a single unit.
  + JNDI lookup services provide a unified interface to multiple naming and directory services in the enterprise so that application components can access these services.
  + The Java EE remote connectivity model manages low-level communications between clients and enterprise beans. After an enterprise bean is created, a client invokes methods on it as if it were in the same virtual machine.
* The container also manages non-configurable services, such as enterprise bean and servlet lifecycles, database connection resource pooling, data persistence, and access to the Java EE platform APIs

Container Types

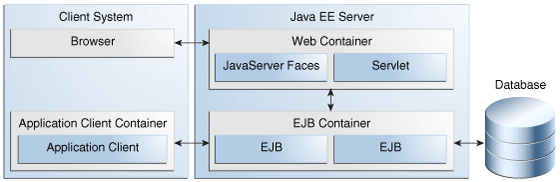
============

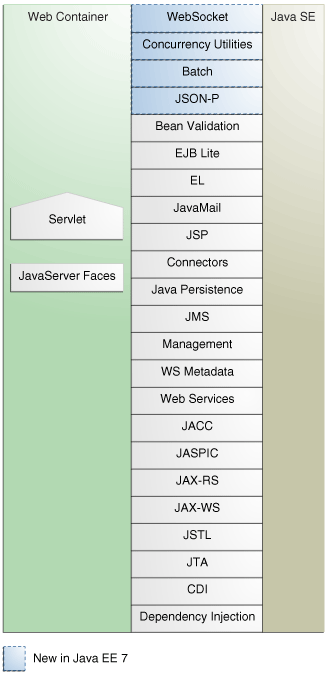
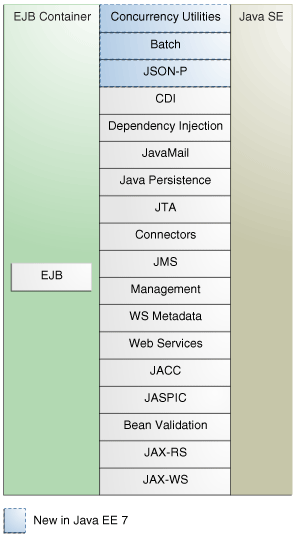
* The **deployment** process installs Java EE application components in the Java EE containers



* The server and containers are as follows:
  + **Java EE server**: The runtime portion of a Java EE product. A Java EE server provides EJB and web containers.
  + **EJB container**: Manages the execution of enterprise beans for Java EE applications. Enterprise beans and their container run on the Java EE server.
  + **Web container**: Manages the execution of web pages, servlets, and some EJB components for Java EE applications. Web components and their container run on the Java EE server.
  + **Application client container**: Manages the execution of application client components. Application clients and their container run on the client.
  + **Applet container**: Manages the execution of applets. Consists of a web browser and a Java Plug-in running on the client together.

# JAva EE 7 apis





The following sections give a brief summary of the technologies required by the Java EE platform and the APIs used in Java EE applications.

1. Enterprise JavaBeans Technology
2. Java Servlet Technology
3. Java Server Faces Technology
4. Java Server Pages Technology
5. Java Server Pages Standard Tag Library
6. Java Persistence API
7. Java Transaction API
8. Java API for RESTful Web Services
9. Managed Beans
10. Contexts and Dependency Injection for Java EE
11. Dependency Injection for Java
12. Bean Validation
13. Java Message Service API
14. Java EE Connector Architecture
15. Java Mail API
16. Java Authorization Contract for Containers
17. Java Authentication Service Provider Interface for Containers
18. Java API for WebSocket
19. Java API for JSON Processing
20. Concurrency Utilities for Java EE
21. Batch Applications for the Java Platform

* An **Enterprise JavaBeans (EJB) component**, or **enterprise bean**, is a body of code that has fields and methods to implement modules of business logic.
* You can think of an enterprise bean as a building block that can be used alone or with other enterprise beans to execute business logic on the Java EE server.
* Enterprise beans are either session beans or message-driven beans.
  + A **session bean** represents a transient conversation with a client. When the client finishes executing, the session bean and its data are gone.
  + A **message-driven bean** combines features of a session bean and a message listener, allowing a business component to receive messages asynchronously. Commonly, these are Java Message Service (JMS) messages.
* In the Java EE 7 platform, new enterprise bean features include the following:
  + Asynchronous local session beans in EJB Lite
  + Nonpersistent timers in EJB Lite
* The Java EE 7 platform requires Enterprise JavaBeans 3.2 and Interceptors 1.2. The Interceptors specification is part of the EJB specification.
* **Java Servlet technology** lets you define HTTP-specific servlet classes.
* A servlet class extends the capabilities of servers that host applications accessed by way of a request-response programming model. Although servlets can respond to any type of request, they are commonly used to extend the applications hosted by web servers.
* In the Java EE 7 platform, new Java Servlet technology features include the following:
  + Nonblocking I/O
  + HTTP protocol upgrade
* The Java EE 7 platform requires Servlet 3.1.
* **Java Server Faces technology** is a user interface framework for building web applications.
* **Java Server Pages (JSP)** technology lets you put snippets of servlet code directly into a text-based document.
* **The Java Server Pages Standard Tag Library (JSTL)** encapsulates core functionality common to many JSP applications. Instead of mixing tags from numerous vendors in your JSP applications, you use a single, standard set of tags. This standardization allows you to deploy your applications on any JSP container that supports JSTL and makes it more likely that the implementation of the tags is optimized.
* **The Java Persistence API (JPA)** is a Java standards–based solution for persistence.
* Persistence uses an object/relational mapping approach to bridge the gap between an object-oriented model and a relational database.
* The Java Persistence API can also be used in Java SE applications outside of the Java EE environment. Java Persistence consists of the following areas:
  + The Java Persistence API
  + The query language
  + Object/relational mapping metadata
* **The Java Transaction API (JTA)** provides a standard interface for demarcating transactions.
* The Java EE architecture provides a default auto commit to handle transaction commits and rollbacks.
* An auto commit means that any other applications that are viewing data will see the updated data after each database read or write operation.
* However, if your application performs two separate database access operations that depend on each other, you will want to use the JTA API to demarcate where the entire transaction, including both operations, begins, rolls back, and commits.
* **The Java API for RESTful Web Services (JAX-RS)** defines APIs for the development of web services built according to the Representational State Transfer (REST) architectural style.
* A JAX-RS application is a web application that consists of classes packaged as a servlet in a WAR file along with required libraries.
* **Managed Beans**, lightweight container-managed objects (POJOs) with minimal requirements, support a small set of basic services, such as resource injection, lifecycle callbacks, and interceptors. Managed Beans represent a generalization of the managed beans specified by JavaServer Faces technology and can be used anywhere in a Java EE application, not just in web modules.
* **Contexts and Dependency Injection for Java EE** (CDI) defines a set of contextual services, provided by Java EE containers, that make it easy for developers to use enterprise beans along with JavaServer Faces technology in web applications. Designed for use with stateful objects, CDI also has many broader uses, allowing developers a great deal of flexibility to integrate different kinds of components in a loosely coupled but typesafe way.
* **Dependency Injection for Java** defines a standard set of annotations (and one interface) for use on injectable classes.
* In the Java EE platform, CDI provides support for Dependency Injection. Specifically, you can use injection points only in a CDI-enabled application.
* The Bean Validation specification defines a metadata model and API for validating data in JavaBeans components. Instead of distributing validation of data over several layers, such as the browser and the server side, you can define the validation constraints in one place and share them across the different layers.
* **The Java Message Service (JMS)** API is a messaging standard that allows Java EE application components to create, send, receive, and read messages. It enables distributed communication that is loosely coupled, reliable, and asynchronous.
* In the platform, new features of JMS include the following.
  + A new, simplified API offers a simpler alternative to the previous API. This API includes a JMSContext object that combines the functions of a Connection and a Session.
  + All objects with a close method implement the java.lang.Autocloseable interface so that they can be used in a Java SE 7 try-with-resources statement.
* **The Java EE Connector Architecture** is used by tools vendors and system integrators to create resource adapters that support access to enterprise information systems that can be plugged in to any Java EE product.
* A **resource adapter** is a software component that allows Java EE application components to access and interact with the underlying resource manager of the EIS. Because a resource adapter is specific to its resource manager, a different resource adapter typically exists for each type of database or enterprise information system.
* Java EE applications use the **JavaMail API** to send email notifications. The JavaMail API has two parts:
  + An application-level interface used by the application components to send mail
  + A service provider interface
* The Java EE platform includes the JavaMail API with a service provider that allows application components to send Internet mail.
* **The Java Authorization Contract** for Containers (JACC) specification defines a contract between a Java EE application server and an authorization policy provider. All Java EE containers support this contract.
* The JACC specification defines java.security.Permission classes that satisfy the Java EE authorization model. The specification defines the binding of container-access decisions to operations on instances of these permission classes. It defines the semantics of policy providers that use the new permission classes to address the authorization requirements of the Java EE platform, including the definition and use of roles.
* The Java Authentication Service Provider Interface for Containers (JASPIC) specification defines a service provider interface (SPI) by which authentication providers that implement message authentication mechanisms may be integrated in client or server message-processing containers or runtimes.
* Authentication providers integrated through this interface operate on network messages provided to them by their calling containers.
* The authentication providers transform outgoing messages so that the source of each message can be authenticated by the receiving container, and the recipient of the message can be authenticated by the message sender.
* Authentication providers authenticate each incoming message and return to their calling containers the identity established as a result of the message authentication.
* **WebSocket** is an application protocol that provides full-duplex communications between two peers over TCP.
* The Java API for WebSocket enables Java EE applications to create endpoints using annotations that specify the configuration parameters of the endpoint and designate its lifecycle callback methods.
* **JSON** is a text-based data exchange format derived from JavaScript that is used in web services and other connected applications.
* **The Java API for JSON Processing (JSON-P)** enables Java EE applications to parse, transform, and query JSON data using the object model or the streaming model.
* **Concurrency Utilities** for Java EE is a standard API for providing asynchronous capabilities to Java EE application components through the following types of objects: managed executor service, managed scheduled executor service, managed thread factory, and context service.
* **Batch jobs** are tasks that can be executed without user interaction. The Batch Applications for the Java Platform specification is a batch framework that provides support for creating and running batch jobs in Java applications. The batch framework consists of a batch runtime, a job specification language based on XML, a Java API to interact with the batch runtime, and a Java API to implement batch artifacts.

# JAVA ee 7 apis in the java platform, standard edition 7

* Several APIs that are required by the Java EE 7 platform are included in the Java Platform, Standard Edition 7 (Java SE 7) and are thus available to Java EE applications.

1. Java Database Connectivity API
2. Java Naming and Directory Interface API
3. JavaBeans Activation Framework
4. Java API for XML Processing
5. Java Architecture for XML Binding
6. Java API for XML Web Services
7. SOAP with Attachments API for Java
8. Java Authentication and Authorization Services
9. Common Annotations for the Java Platform

* **The Java Database Connectivity (JDBC) API** lets you invoke SQL commands from Java programming language methods.
* You use the JDBC API in an enterprise bean when you have a session bean access the database.
* You can also use the JDBC API from a servlet or a JSP page to access the database directly without going through an enterprise bean.
* The JDBC API has two parts:
  + An application-level interface used by the application components to access a database
  + A service provider interface to attach a JDBC driver to the Java EE platform
* The Java Naming and Directory Interface (JNDI) API provides naming and directory functionality, enabling applications to access multiple naming and directory services, such as LDAP, DNS, and NIS.
* The JNDI API provides applications with methods for performing standard directory operations, such as associating attributes with objects and searching for objects using their attributes. Using JNDI, a Java EE application can store and retrieve any type of named Java object, allowing Java EE applications to coexist with many legacy applications and systems.
* **The JavaBeans Activation Framework (JAF)** is used by the JavaMail API.
* JAF provides standard services to determine the type of an arbitrary piece of data, encapsulate access to it, discover the operations available on it, and create the appropriate JavaBeans component to perform those operations.
* **The Java API for XML Processing (JAXP),** part of the Java SE platform, supports the processing of XML documents using Document Object Model (DOM), Simple API for XML (SAX), and Extensible Stylesheet Language Transformations (XSLT). JAXP enables applications to parse and transform XML documents independently of a particular XML-processing implementation.
* **The Java Architecture for XML Binding (JAXB)** provides a convenient way to bind an XML schema to a representation in Java language programs. JAXB can be used independently or in combination with JAX-WS, in which case it provides a standard data binding for web service messages. All Java EE application client containers, web containers, and EJB containers support the JAXB API.
* **The Java API for XML Web Services (JAX-WS)** specification provides support for web services that use the JAXB API for binding XML data to Java objects.
* The JAX-WS specification defines client APIs for accessing web services as well as techniques for implementing web service endpoints.
* The Implementing Enterprise Web Services specification describes the deployment of JAX-WS-based services and clients.
* The EJB and Java Servlet specifications also describe aspects of such deployment. JAX-WS-based applications can be deployed using any of these deployment models.
* The SOAP with Attachments API for Java (SAAJ) is a low-level API on which JAX-WS depends.
* SAAJ enables the production and consumption of messages that conform to the SOAP 1.1 and 1.2 specifications and the SOAP with Attachments note. Most developers do not use the SAAJ API, instead using the higher-level JAX-WS API.
* **The Java Authentication and Authorization Service (JAAS)** provides a way for a Java EE application to authenticate and authorize a specific user or group of users to run it.
* JAAS is a Java programming language version of the standard Pluggable Authentication Module (PAM) framework, which extends the Java platform security architecture to support user-based authorization.
* **Annotations** enable a declarative style of programming in the Java platform.

# WEB Services Support

* Web services are web-based enterprise applications that use open, XML-based standards and transport protocols to exchange data with calling clients.
* The Java EE platform provides the XML APIs and tools you need to quickly design, develop, test, and deploy web services and clients that fully interoperate with other web services and clients running on Java-based or non-Java-based platforms.
* To write web services and clients with the Java EE XML APIs, all you need to do is pass parameter data to the method calls and process the data returned; for document-oriented web services, you send documents containing the service data back and forth.
* No low-level programming is needed because the XML API implementations do the work of translating the application data to and from an XML-based data stream that is sent over the standardized XML-based transport protocols. These XML-based standards and protocols are introduced in the following sections.
* **Extensible Markup Language (XML)** is a cross-platform, extensible, text-based standard for representing data. Parties that exchange XML data can create their own tags to describe the data, set up schemas to specify which tags can be used in a particular kind of XML document, and use XML style sheets to manage the display and handling of the data.
* Client requests and web service responses are transmitted as **Simple Object Access Protocol (SOAP)** messages over HTTP to enable a completely interoperable exchange between clients and web services, all running on different platforms and at various locations on the Internet. HTTP is a familiar request-and-response standard for sending messages over the Internet, and SOAP is an XML-based protocol that follows the HTTP request-and-response model.
* **The Web Services Description Language (WSDL)** is a standardized XML format for describing network services. The description includes the name of the service, the location of the service, and ways to communicate with the service. WSDL service descriptions can be published on the Web.