Java 2 Enterprise Edition (J2EE)

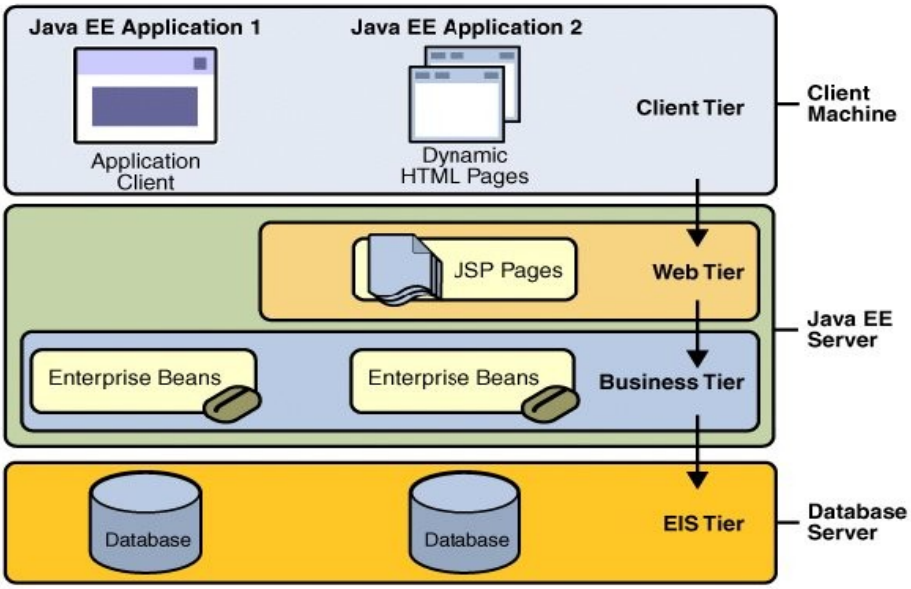
É uma plataforma para o desenvolvimento de aplicações de propósito geral, com foco corporativo centralizado, suportando os seguintes requerimentos:

- Transações
- Segurança
- Escalabilidade
- Integração com outros sistemas

Para que se possa utilizar estes recursos, a plataforma J2EE oferece um conjunto de APIs, de forma que o desenvolvedor pode focar na solução dos problemas da aplicação, ao invés de ter que construir a infraestrutura para a execução da aplicação.

A plataforma J2EE oferece quatro contêineres separados de forma possibilitar o desenvolvimento de aplicações em multi - camada (n-tier), sendo estes:

- Contêiner para Aplicação Cliente
- Contêiner para Applets
- Contêiner para Componentes Web
- Contêiner para Enterprise Java Bean (EJB)



aplicações em multicamada (n-tier)

Todos os contêineres utilizam como runtime o J2SE (Java 2 Standard Edition), sendo que alguns destes adicionam funcionalidades na forma de bibliotecas, que são expostas para as aplicações como APIs.

O contêiner para Aplicação Cliente é uma aplicação J2SE padrão. Uma aplicação cliente se comunicará com o contêiner Web via HTTP ou HTTPS e com o contêiner EJB utilizando RMI, ou SOAP/HTTP. Uma aplicação cliente é normalmente conhecida como um “fat client” (implementado em Swing ou Swt).

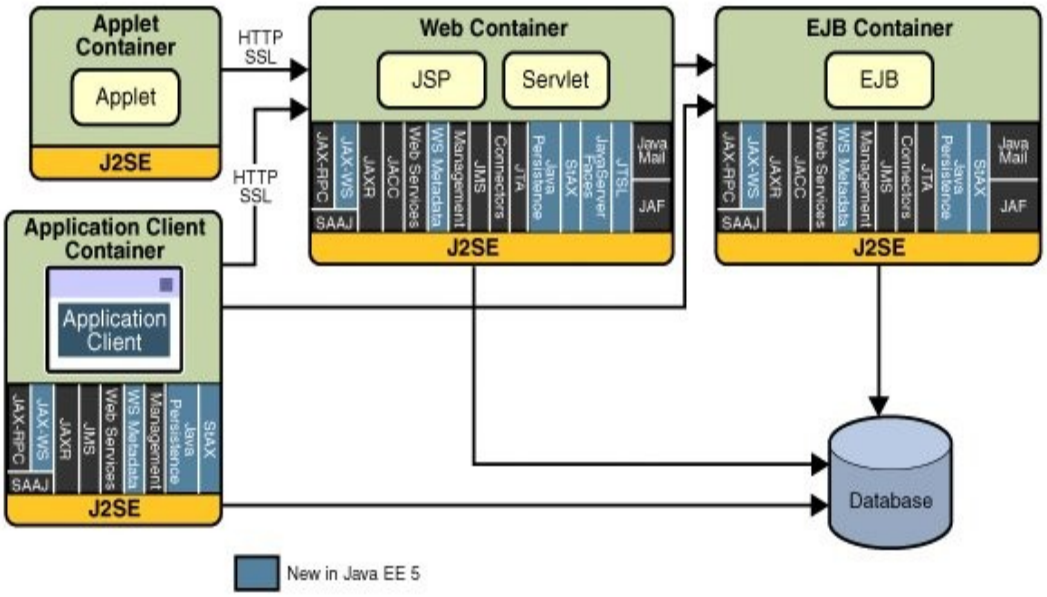
O contêiner para Applets também é uma aplicação J2SE, mas geralmente é executada no contexto de um Navegador Web. A sua comunicação com o contêiner Web ocorre utilizando HTTP e

HTTPS.

O contêiner Web é visto tradicionalmente como sendo J2EE, e este contêiner define as bibliotecas e APIs para o JSP e Servlets.

O último contêiner é o EJB, que bem como o contêiner Web oferece a última ligação para o desenvolvimento baseado em Servidor. Ele define os os mecanismos para os Beans de Sessão e Entidade utilizados no desenvolvimento de componentes distribuídos.

Os contêineres Web e EJB são os mais comumente considerados a Plataforma J2EE.



A Plataforma Java EE, seus Contêineres e APIs

O Contêiner

Um contêiner na arquitetura J2EE é utilizado para oferecer uma visão unificada sobre as suas APIs para os componentes de aplicações. Uma aplicação J2EE nunca interage diretamente com outro componente J2EE; todas estas interações são controladas e gerenciadas pelo contêiner.

Para usar um contêiner, o desenvolvedor necessita informar aos componentes da aplicação um descritor de publicação (deployment descriptor), que declara ao contêiner o componente aplicação em qual contexto eles são utilizados. Por exemplo, na criação de uma aplicação Web, o desenvolvedor criará arquivos JSP com a extensão ".jsp", o que indica de forma implícita ao contêiner Web que estes arquivo deve ser tratado de uma forma particular. Uma classe Java utilizada como Servlet é declarada no "deployment descriptor", e um nome é associado com o servlet assim este pode ser gerenciado pelo contêiner, e acessado por requisições esternas.

O contêiner têm contratos e interfaces ao qual um desenvolvedor deve seguir, para fazer uso deste. Em troca, contêiner oferece serviços conforme segue:

- Gerenciamento do ciclo de vida: Isto envolve iniciar e parar os componentes de aplicações. Se a aplicação é um servlet, isto significa chamar o método "init()" quando o servlet é carregado, e chamar o método service() quando o servlet necessita atender uma requisição.

- Pool de recursos: Isto oferece um ponto único de acesso aos componentes de aplicações que necessitam localizar e utilizar os recursos que são custosos para criar. O benefício adicional na utilização do pool de recursos é que a aplicação não necessita conhecer o recurso concreto para que possa ser utilizado. Este recurso pode ser configurado externamente a aplicação,

possibilitando o contêiner ser responsável pela obtenção deste, criando um cache, e assim permitindo a aplicação obter o recurso do cache. Um exemplo típico é a localização e criação de um pool de conexões a base de dados (database connection pool).

- Gerenciamento de transações: O contêiner oferece gerenciamento de transações declarativas para os componentes de aplicação. Um contêiner é capaz de iniciar um transação ao acessar um componente ou fazer uso das transações já existentes.

- Segurança: Serviços de segurança estão disponíveis aos conte ineres Web e EJB, e estes serviços permitem configurações assim o acesso à recursos podem requerer autenticação antes de qualquer acesso. Isto sem necessitar que seja escrito nenhuma linha de código pelo desenvolvedor.

Estes serviços são infraestrutura, assim eles não necessitam ser criados para cada aplicação, o que possibilita aos desenvolvedores focar na difícil tarefa de solucionar os problemas da aplicação.

As API do ambiente J2EE

A plataforma J2EE necessita de APIs do J2SE além de outras que são específicas do J2EE, abaixo segue uma lista das APIs J2SE utilizadas:

- Java IDL API: É a tecnologia para objetos distribuídos, similar ao RMI, mas para acessar objetos baseados em CORBA. Diferente de RMI, IDL permite a utilização dos objetos por diferentes linguagens de programação.

- JDBC API: O Java Database Connection API oferece um mecanismo independente de Gerenciador de Dados para o acesso a um grande número de fontes de dados, incluindo banco de dados, arquivos, e dados tabulares.

- RMI-IIOP API: Esta API permite aos objetos Java serem acessados por objetos Corba.

- JNDI API: A Java Naming and Directory Interface API oferece os mecanismos necessários a localização de serviços (naming). Por exemplo localização de um pool de conexões JDBC em um J2EE Server. Esta API também oferece os mecanismos para visualização, criação e modificação de atributos localizados em Serviços de Diretório. Por exemplo LDAP.

- JAXP API: Com a Java API for XML Processing é possível processar e transformar documentos XML. JAXP também oferece mecanismos para a substituição do processador de XML utilizado. (Apache Xalan-j e JAXP RI da Sun).

- JAAS API: Java Authentication and Autorization Service API oferece uma interface para os serviços de autenticação e autorização. JAAS permite que diferentes implementações sejam usadas.

Adicionada as APIs anteriores temos a seguir os Packages e as APIs J2EE:

- Enterprise Java Beans (EJB): Especifica os serviços J2EE para componentes distribuídos baseados em servidor. O mais importante é que a plataforma J2EE definiu uma forma consistente para o acesso aos componentes, possibilitando o desenvolvimento de aplicações que podem ser publicadas em uma variedade de contêineres EJB.

- Servlet: Especifica os serviços para a construção de aplicações Web. Muitas das especificações JSP são definidas nesta especificação.

- Java Server Pages (JSP): Esta extensão aumenta a capacidade do Servidor de Aplicações introduzindo uma linguagem script baseada em gabaritos à plataforma de aplicações Web.

- Java Message Service (JMS): O JMS oferece protocolos para a comunicação assíncrona ponto a ponto e publicador – subscritor para as aplicação.

- Java Transaction API (JTA): Define a interface para uma transação de usuário que é utilizada para controlar o ciclo de vida da transação sob controle da aplicação. O controle do ciclo de vida consiste na capacidade de iniciar, executar “commit”, e abortar transações.

- JavaMail API: Oferece um mecanismo padronizado para o acesso de diferentes implementações de protocolos de transporte de email. Estes protocolos incluem a implementação de caixas postais POP ou IMAP e a construção e envio de mensagens MIME usando SMTP.

- JavaBeans Activation Framework API (JAF): Esta APIs oferece uma série de objetos para a manipulação de tipo de dados MIME o que possibilita uma fácil conversão de dados tipo MIME para objetos Java.

- J2EE Connector Architecture (JCA): O JCA oferece uma interface padronizada para sistemas externos que são utilizados pela plataforma J2EE. Os sistemas externos normalmente tomam a forma de Enterprise Information Systems (EIS), mas o sistema externo pode ser somente um arquivo de sistema.

Cada sistema externo define um Resource Adapter que segue um conjunto de interfaces carregadas pelo Servidor de Aplicações J2EE para acessar o recurso.

A facilidade oferecida pelo Resource Adapter inclui o suporte a transações, segurança, e pool de conexões.

- Java API for XML-Based RPC (JAX-RPC): Esta especificação oferece interoperabilidade com Web Services entre diferentes plataformas com protocolos baseados em SOAP sobre HTTP. A especificação define tanto a invocação no lado cliente quanto o protocolo para pontos de conexão (end-points) no Servidor, utilizando Web Services Description Language (WSDL) com o objetivo de habilitar tanto clientes quanto servidores a efetuar a publicação (deploy) em diferentes plataformas de forma que a interoperabilidade ocorra de forma transparente.

Os pontos de conexão (end-points) para Web services podem ser desenvolvidos com componentes Servlets ou EJBs, com a publicação dos Web Services utilizando os seus respectivos mecanismos.

- SOAP whith Attachments for Java (SAAJ): Esta API define um mecanismo padronizado para o envio de documentos XML. SAAJ é utilizado por JAX-RPC para acessar uma mensagem SOAP em um manipulador de mensagens JAX-RPC.

SAAJ também pode ser utilizado independentemente na criação de aplicações que produzem ou consumem mensagens SOAP diretamente, sem a utilização de JAX-RPC.

- Java API for XML Registries (JAXR): JAXR define a interface para os clientes acessarem registros baseados em XML, incluindo ebXML e UDDI.

- J2EE Management: Esta interface oferece um conjunto de ferramentas para consultar o Servidor de aplicações J2EE sobre suas informações internas. Isto inclui examinar objetos que identificam o status do Servidor de Aplicações, as aplicações publicadas (deployed), o status das aplicações, e ainda coleções de estatísticas para monitoração de performance.

O componente de gerenciamento é baseado em JMX, e identifica modelos de objetos concretos que o Servidor de aplicações J2EE expõe através de EJBs tais como o J2EE Management EJB Component.

- Java Management Extensions (JMX): O suporte JMX no Servidor de aplicações é necessário para a interface J2EE Management.

- Java Autorization Contract for Contêineres (JAAC): Esta especificação define a interface entre um Servidor de Aplicação J2EE e um provedor de políticas de autorização. São incluídas as definições para como os provedores de política de autorização são publicados (deployed), configurados, e utilizados na tomada de decisões de acesso.

- Java API for XML Web Services (JAX-WS): JAX-WS é a tecnologia para a construção de Web Services e clientes que se comunicam através de XML. JAX-WS permite aos desenvolvedores escrever Web Services orientados a mensagens bem como orientadas a Remote Procedure Call (RPC).

- Java Architecture for XML Binding (JAXB): Oferece uma forma fácil e conveniente de ligação (bind) entre esquemas XML e objetos Java, criando facilidades aos desenvolvedores a incorporação de dados XML e funções de processamento em aplicações Java. Como parte deste processo, JAXB oferece métodos para decodificar (unmarshalling) documentos XML em componentes, e codificar (marshalling) componentes Java de volta no formato de documentos XML. JAXB também oferece um mecanismo para gerar esquemas XML a partir de objetos Java.

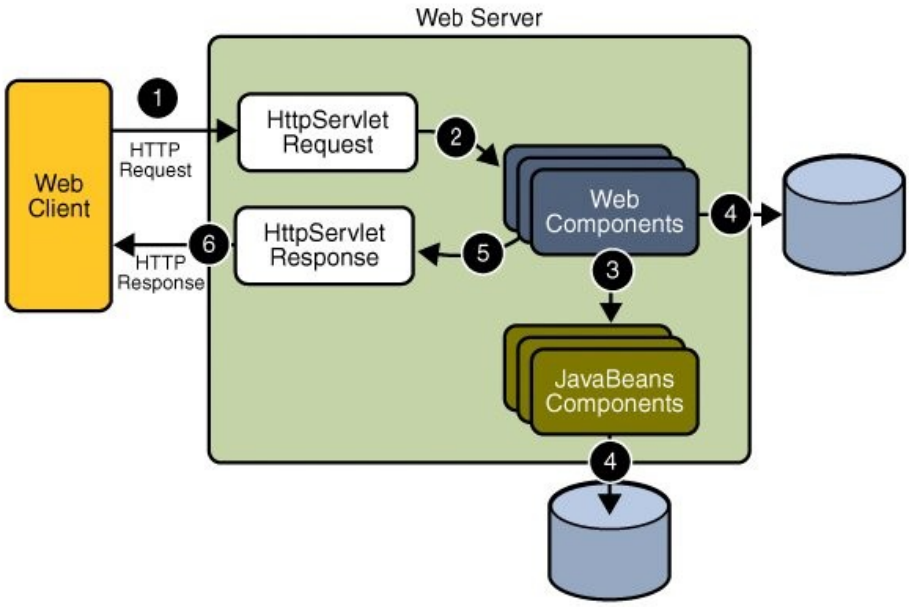
- Streaming API for XML (StAX): Um gerador de stream baseado em Java, orientado a evento, uma API para a leitura e gravação de documentos XML. StAX possibilita a criação de processadores de XML bidirecionais de programação relativamente fácil e rápida, com baixo consumo de memória. StAX é a última API na família JAXP, e oferece alternativas ao SAX, TrAX, e DOM para os desenvolvedores que procuram por um stream filter de alta performance , processamento, modificação, e com facilidade de uso.

- Java Persistence API: É a solução padronizada para persistência de objetos. A Persistência utiliza uma técnica de mapeamento objeto relacional a fim de suprir a necessidade utilização de Banco de Dados Relacionais como repositório de Objetos. Java Persistence consiste de três áreas:

  - A Java Persistence API
  - A query language
  - Os metadados de mapeamento Objeto/relacional

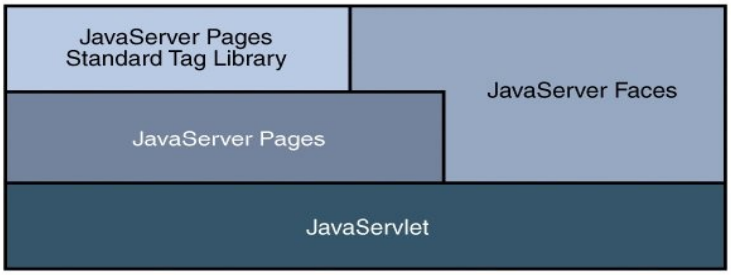
A camada Web

Na plataforma Java, componentes web oferecem capacidades de extensões dinâmicas para um Web Server. Componentes web são Java servlets, páginas JSP, e pontos de conexão (end-points) à Web Services. A interação entre um cliente web e uma aplicação web é ilustrada na figura abaixo. O cliente envia uma requisição HTTP para o Web Server. Um Web Server que implementa a tecnologia Java Servlet e JavaServer Pages converte a requisição em um objeto `HTTPServletRequest`. Este objeto é enviado para o componente web, que pode interagir com componentes JavaBeans ou um banco de dados para gerar conteúdo dinâmico. O componente web pode assim gerar um objeto `HTTPServletResponse` ou passar a requisição a outro componente web. Eventualmente um componente web gera um objeto `HTTPServletResponse`. O web server converte este objeto para uma resposta HTTP e retorna esta ao cliente.



Servlets são classes Java que processam dinamicamente as requisições e constroem as respostas. Páginas *JSP são documentos textos executados como* servlets mas permitem um desenvolvimento mais natural para a criação de contexto estático. Embora servlets e Páginas JSP possam ser utilizadas de formas intercambiáveis, cada uma tem sua própria vantagem. Servlets são mais apropriados para aplicações orientadas a serviço (web service endpoints são implementados como servlets) e com a função de controlados para aplicações orientadas a apresentação, tais como despachar requisições e tratamento de dados não contextuais. Páginas JSP são mais apropriadas para a geração de páginas HTML, Scalable Vector Graphics (SVG), Wireless Markup Language (WML), e XML.

Desde a introdução das tecnologias Servlets e JSP, muitas tecnologias Java adicionais e frameworks para a construção de aplicações WEB interativas foram desenvolvidas. Abaixo temos a ilustração destas tecnologias e seus relacionamentos.



Cada tecnologia adiciona um nível de abstração para a prototipagem e desenvolvimento rápido de aplicações WEB além de tornar estas de fácil manutenção, escalabilidade e robustez.

Componentes Web são suportados por um serviço do servidor de aplicação chamado de Contêiner WEB. Um contêiner Web oferece serviços tais como distribuidor de requisições, segurança, concorrência e gerenciamento de ciclo de vida. Também oferece acesso a APIs tais como gerenciamento de transações e e-mail.

Certos aspectos de uma aplicação web pode ser configurada quando esta for instalada no contêiner web. A informação da configuração é mantida num arquivo texto no formato XML e é chamado de *web application deployment descriptor* (DD).

O ciclo de vida de uma aplicação Web

Uma aplicação web consiste de componentes web, arquivos tais como imagens, classes de auxílio e bibliotecas. O contêiner WEB oferece muitos serviços de apoio que aumentam as capacidades dos componentes web e tornam fácil seu desenvolvimento. No entanto, em razão de que uma aplicação web deve levar em conta estes serviços, o processo de criação e execução de uma aplicação web é diferente de uma classe Java padrão. Este processo pode ser resumido conforme segue:

1. Desenvolver o código do componente web.
2. Desenvolver o web application deployment descriptor.
3. Compilar os componentes da aplicação web e classes auxiliares referenciadas pelos componentes.
4. Opcionalmente empacotar a aplicação em uma unidade de publicação.
5. Publicar a aplicação em um contêiner web.
6. Acessar uma URL que referencia a aplicação web.

Modulos

Na arquitetura Java EE, os componentes web e conteúdo estático tais como imagens são chamados de recursos. Um modulo *web é a menor unidade utilizável de recurso*. Este modulo web Java EE corresponde a aplicação web *definida na especificação do* Java Servlet.

Adicionalmente aos componentes web e recursos, um módulo web pode conter outros arquivos:

- Classes utilitárias executadas no servidor (database beans, carrinhos de compra, etc). Normalmente estas classes seguem o padrão de arquitetura JavaBeans.
- Classes executadas no cliente (applets e classes utilitárias).

Um módulo web tem uma estrutura específica. O primeiro nível do diretório de um módulo web é o ponto inicial para a aplicação (**Document ROOT**). Este é o local onde as páginas JSP, classes e recursos estáticos, tais como imagens são armazenadas.

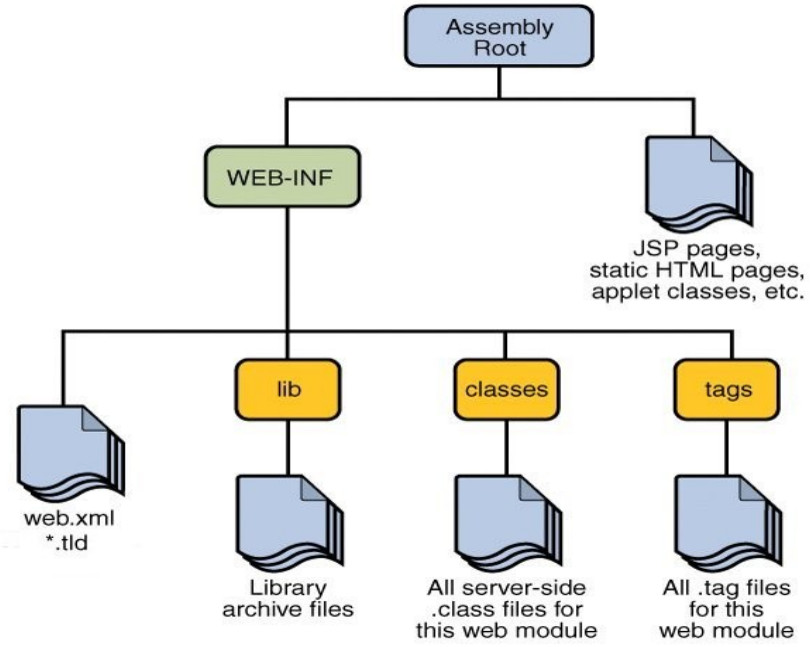
O **document root** contem um subdiretório chamado `/WEB-INF/`, que contem os seguintes arquivos e diretórios:

- `web.xml`: O web application deployment descriptor
- *.tld : Arquivos descritores de Tag library.
- `classes`: Um diretório que contem classes utilitárias, servlets e componentes Java Beans.
- `tags`: Um diretório que contais arquivos de tags, que são implementações de Tag Libraries.
- `lib`: Um diretório que contem arquivos JAR das bibliotecas utilizadas pelas classes da aplicação.

Se seu módulo web não contem nenhum servlet, filtro, ou outro componente não é necessário um web application deployment descriptor, ou seja não é necessária a inclusão do arquivo web.xml.

Você também pode criar subdiretórios específicos tanto no **document root** como em `/WEB-INF/classes/`.

Um módulo web pode ser publicado tanto descompactado como empacotado em um arquivo JAR conhecido como web archive (WAR). Em função do conteúdo diferente entre um Java archive e um Web Achive sempre é utilizado a extensão .war para um Web Archive. O módulo Web descrito acima é portável para qualquer contêiner Web e seque a especificação do Servlet Java.



Empacotamento dos módulos

Um modulo web deve ser empacotado em um WAR em certos cenários e sempre que for necessário distribuir este modulo. Empacotamos o modulo web num WAR executando o comando `jar` num diretório com a estrutura compatível com a de estrutura definida acima, ou pela utilização do utilitário `ant`, ou a ferramenta do IDE de sua escolha.

Declarando referências à recursos

Se seus componentes web utilizam objetos tais como enterprise beans, data sources, ou web services, pode ser utilizado anotações Java EE para injetar estes recursos na sua aplicação. Anotações eliminam muitos códigos e configurações necessárias nas versões anteriores do Java EE.

Embora a injeção de recursos através de anotações pode ser mais conveniente para o desenvolvedor, existem algumas restrições de seu uso nas aplicações web. Primeiro você somente pode injetar recursos em objetos gerenciados pelo contêiner. Isto porque um contêiner deve ter o controle sobre a criação de um componente para que possa executar a injeção de recursos nele. Como resultado, você não poderá injetar recursos em objetos que estão externos ao contêiner. No entanto, Managed Beans do JavaServer Faces são gerenciados pelo contêiner; desta forma eles aceitam a injeção de dependências de recursos.

Adicionalmente, páginas JSP não aceitam injeção de recursos. Isto por que a imformação representada pela anotação deve estar disponível no momento da publicação, mas as páginas JSP são compiladas após esta fase; assim a anotação não será vista quando for necessária. Os componentes que aceitam injeção de recursos estão listado a seguir.

Web Components que aceitam injeção de recursos

Component	Interface/Class
Servlets	`javax.servlet.Servlet`
Servlet Filters	javax.servlet.ServletFilter

Event Listeners	`javax.servlet.ServletContextListener`
	`javax.servlet.ServletContextAttributeListener`
	`javax.servlet.ServletRequestListener`
	`javax.servlet.ServletRequestAttributeListener`
	`javax.servlet.http.HttpSessionListener`
	`javax.servlet.http.HttpSessionAttributeListener`
	`javax.servlet.http.HttpSessionBindingListener`

Taglib Listeners	Igual ao listado acima
Taglib Tag Handlers	`javax.servlet.jsp.tagext.JspTag`
Managed Beans	Plain Old Java Objects

A declaração

A anotação `@Resource` é utilizada para declarar uma referência a um recurso como um data source, um enterprise bean. Esta anotação é equivalente ao declarar um elemento `resource-ref` element no deployment descriptor.

Utilizamos esta anotação na declaração de uma classe, método ou atributo. O contêiner é responsável por injetar as referências dos recursos declarados por esta anotação e mapear apropriadamente as referências JNDI. No exemplo abaixo a anotação `@Resource` é utilizada para injetar um data source num componente que necessita fazer uma conexão utilizando a tecnologia JDBC para acessar um banco de dados relacional:

```
	@Resource
	javax.sql.DataSource catalogDS;

	public List<Categorias> getProductsByCategory() {
		try {
			// Obtêm uma conexão para executar uma consulta
			Connection conn = catalogDS.getConnection();
			// ...
```

O contêiner injeta o data source antes deste componente ficar disponível para a aplicação. O mapeamento JNDI para o data source é determinado a partir do nome do atributo `catalogDS` e pelo tipo, `javax.sql.DataSource`.

Se você tem múltiplos recursos que necessita injetar em seu componente, será necessário utilizar a anotação `@Resources` para contê-las, como no exemplo abaixo:

```
@Resources ({
	@Resource (name="myDB", type=java.sql.DataSource),
	@Resource(name="myMQ", type=javax.jms.ConnectionFactory)
})
```

As aplicações web que utilizam a Java Persistence API para acessar os dados em banco de dados relacional utilizam as anotações `@PersistenceUnit` e `@PersistenceContext` para injetar instâncias de `EntityManagerFactory` e `EntityManager`, respectively.

A tecnologia Java Servlet

Assim que a web iniciou a entrega de serviços, os provedores de serviços reconhecerem a necessidade de oferecer conteúdo dinâmico. Applets são uma das primeiras tentativas para alcançar este objetivo, focado na utilização na plataforma cliente para oferecer ao usuário a experiência de conteúdo dinâmico. Ao mesmo tempo, os desenvolvedores também investigaram a utilização do servidor para este propósito. Inicialmente, scripts utilizando Common Gateway Interface (CGI) como a principal tecnologia para a geração de conteúdo dinâmico no servidor. Embora grandemente usada, a tecnologia de scripts CGI tem muitas restrições, incluindo dependência de plataforma e falta de escalabilidade. Para endereçar estas limitações, a tecnologia Java Servlet foi criada como uma forma portável de oferecer conteúdo dinâmico orientado ao usuário.

Um *servlet* é uma classe java que é utilizada para estender as capacidades dos servidores que hospedam aplicações de acesso através de um modelo de programação de requisição - resposta. Embora os servlets possam responder a qualquer tipo de requisição, eles são comummente usados para estender os aplicativos hospedados em servidores web. Para estas aplicações, a tecnologia Java Servlet define classes específicas para tratar o protocolo HTTP.

Os pacotes `javax.servlet` e `javax.servlet.http` oferecem interfaces e classes para a criação de servlets. Todos os servlets devem implementar a interface `Servlet`, a qual define os métodos para seu ciclo de vida. Quando da implementação de um serviço genérico, você pode utilizar ou estender a classe `GenericServlet` que pertence á API Java Servlet. A classe `HttpServlet` oferece métodos, tais como `doGet` e `doPost`, para tratamento de serviços específicos do protocolo HTTP.

O ciclo de vida de um Servlet

O ciclo de vida de um servlet é controlado pelo contêiner ao qual foi publicado. Quando uma requisição é mapeada para um servlet, o conteiner executa os seguintes passos.

1. Se uma instância do servlet não está em memória, o contêiner carrega a classe do servlet.
2. Cria uma instância de objeto para a classe do servlet.
3. Inicializa a instância chamando o método `init`.
4. Invoca o método `service`, passando os objetos de requisição e resposta.

Se o contêiner necessitar remover o servlet, ele finaliza este chamando o método `destroy`.