***Java***

JSF utilizava JSP como template padrão apenas nas versões 1.x. Versões 2.x, o JSF usa templates facelets em páginas **.xhtm**l.

**ATRIBUTO EXTENDS:**Serve para indicar a super classe que será usada pelo container JSP no momento de tradução da página em um Servlet Java. Exemplo:  
<%@ page extends=”com.taglib.jsp.primeirapagina” %>  
  
**ATRIBUTO IMPORT:** Com o atributo import, diferente do extends, é capaz de estender um conjunto de classes Java que poderão ser usadas nas páginas JSPs. Esta forma é mais econômica de se digitar código, sendo mais prático. Exemplo:  
<%@ page import=”java.util.List” %  
  
**ATRIBUTO ISTHREADSAFE:**Quando uma página JSP é compilada em um Servlet, ela deve ser capaz de atender a múltiplas solicitações. Para isso devemos utilizar o atributo isThreadSafe, caso contrário é necessário defini-lo como “false”. Exemplo:  
<%@ page isThreadSafe=”false” %>  
  
**ATRIBUTO SESSION:**Session é do tipo boolean, indica se a página está participando do gerenciamento de sessão. Por exemplo, se quisermos dizer que uma página é parte de uma sessão, utiliza-se a seguinte sintaxe:  
<%@ page session=”true” %>  
  
**ATRIBUTO AUTOFLUSH:**Semelhante ao Buffer, também é responsável por controlar a saída buferizada, mais exatamente o comportamento do container JSP quando já esteja cheio o Buffer de saída. Neste caso é esvaziado automaticamente o Buffer de saída e o conteúdo enviado para o servidor HTTP que transmite para o Browser responsável pela solicitação. Sendo do tipo boolean, sua sintaxe é dada abaixo:  
<%@ page autoFlush=”true” %>

**Java Server Faces – JSF**

**View**:

•Componentes UI em páginas JSP/XHTML.  
• Kits renderizadores (HTML, WML, XML, etc.).

**Controller**:  
• Faces Servlet (Front Controller).  
• Backing Bean (Page Controller ou Modelo).

**Model**:  
• entidades e regras de negócio.  
• Objetos gerais da aplicação (dados, etc.).

**String getInitParameter(String).** Método usado para ler os parâmetros de inicialização de um servlet. Um **servlet** estende a funcionalidade de um servidor.  
O pacote **javax.servlet** e o **pacote javax.servlet.http** fornecem as classes e as interfaces para definir os servlets.

**O JSF 2.0 possui quatro escopos**

**@RequestScope:** todos os objetos armazenados no escopo request sobrevivem apenas a uma submissão ao ciclo de vida do JSF (os quais irei explicar em outro post). Com isso temos uma duração que condiz com a requisição sendo enviada ao servidor, e este devolvendo a resposta ao usuário que disparou a ação. Possui o menor tempo de vida dentre os escopos, desta forma, os objetos permanecem por pouco tempo em memória sendo esta liberada com maior frequência e com isso temos uma aplicação que tende a escalar melhor.

**@SessionScope:** todos os objetos e atributos vinculados ao ManagedBean sobreviverão durante toda a sessão do usuário. A sessão é definida pelo vinculo do navegador do usuário com o servidor. Desta forma, se usuário abrir dois navegadores, ele estará criando duas sessões com o servidor. Este escopo era muito utilizado nas versões do JSF 1.x, para se trabalhar em casos onde era necessário manter o estado de objetos, atualmente esta necessidade, muitas vezes, pode ser resolvida através do View Scope.

**@ApplicationScope:** tudo armazenado neste escopo permanece enquanto a aplicação estiver executando e é compartilhado entre todos os usuários. É recomendado sempre que for necessário guardar informações que podem ser utilizada por diversas partes da aplicação como parâmetros e também implementar funcionalidades para prover comunicação entre usuários. Este escopo também é interessante para se trabalhar com caches manuais de valores, como exemplo lista de estados.  
 **@ViewScope:** adicionado a partir da versão JSF 2, foi criado para resolver o problema de sempre utilizar session quando era necessário manter os dados entre requisições e que não onerasse tanto a aplicação. O View Scope oferece suporte ao modelo statefull do framework, onde é possível manter os dados durante quantas requisições forem necessárias, desde que todas estas sejam realizadas para a mesma view. Caso seja executado uma requisição para uma pagina e/ou ManagedBean diferente, o escopo é limpo, evitando assim que objetos não utilizados se mantenham vivos por muito tempo (caso que ocorria no escopo sessão).

Características:

* **FacesServlet** – é o servlet principal para a aplicação e, opcionalmente, você pode ter o arquivo de configuração **faces-config.xml**.
* **Renderers** –  são os responsáveis por exibir um componente e traduzir uma entrada de valor realizada por um usuário em componente.
* **Páginas XHTML, JSP** – o JavaServer Faces permite mais de um tipo de arquivo para renderizar seus componentes (PDL – Page Declaration Language), como páginas JSP ou Facelets.
* **Converters** – convertem o valor de um componente (como datas, moedas, porcentagem e outros) dando-lhes novos formatos.
* **Validators** – responsáveis por validar a entrada ocorrida no componente pelo usuário.
* **Managed Bean** – a lógica do negócio é gerenciada pelos managed beans, que controlam inclusive a navegação por entre páginas.
* **Ajax** – Tanto o envio como o recebimento de dados podem ser feitos via Ajax, sem a necessidade de reload na página do usuário.

***Ciclo de vida do JSF:***

* **1-Restore View**: restaura ou cria a visão. Os componentes são armazenados em uma árvore de componentes.
* **2-Apply Request Values**: qualquer dado que for enviado como parte da requisição é passado para os componentes apropriados.
* **3-Process Validation**: os dados que foram submetidos com o formulário são validados.
* **4-Update Model Values**: após todas essas validações terminarem, os objetos de negócio que criam a aplicação são atualizados com os dados validados da requisição.
* **5-Invoke Application**: os métodos de ação de qualquer botão ou link que foi ativado serão chamados.
* **6-Render Response**: essa fase renderizará a página de resposta requisitada pelo usuário.

JSP é considerada obsoleto (deprecated) no Java EE 7, motivo pelo qual é indicado a utilização de XHTML para desenvolvimento de facelets para JSF 2.0.

No Java EE 7, o JSP é considerado obsoleto (deprecated) e não suporta. Aliás no tutorial do Java EE 6 temos:

JavaServer Pages (JSP) technology, previously used as the presentation technology for JavaServer Faces, does not support all the new features available in JavaServer Faces 2.0. **JSP technology is considered to be a deprecated presentation technology for JavaServer Faces 2.0.** Facelets is a part of the JavaServer Faces specification and also the preferred presentation technology for building JavaServer Faces technology-based applications

O **f:ajax** é a tag que adiciona capacidades Ajax a um ou mais componentes JSF. Ela pode ser aninhada dentro de um componente de interface única para permitir Ajax para esse componente, ou pode envolver em torno de múltiplos componentes para habilitar o Ajax para muitos componentes. (TRADUZIDO).

Os “listeners”, por definição, são usados para tratar e processar eventos. Quando um evento ocorre, o “listener” é notificado e recebe um objeto contendo informações sobre o evento ocorrido, e tem a oportunidade de executar e realizar qualquer processamento disparado pelo evento. No *framework* JSF (*Java Server Faces*), os eventos disparados por componentes (UI) são tratados por duas interfaces, que são:

* ***ActionListener***, que é usada para capturar eventos disparados por botões e links.
* ***ValueChangeListener***, que é usada para capturar eventos por checkboxes, radio button e combo.

Interface for receiving notification events about **requests coming into and going out** of scope of a web application.

* public interface **javax.servlet.ServletRequestListener** extends EventListener

***Atributos da diretiva:***

**- Atributo Info**

Usado para inserir informações sumarizadas da página, não havendo restrições ao seu tamanho. Exemplo:

<%@ page info=”Estudo sobre Diretivas JSP, Prof. Cristiano Neves, 2006” %>

**- Atributo Language**

Usado, em geral, para especificar Java como a linguagem de criação de script para a página. Exemplo:

<%@ page language=”java” %>

**- Atributo ContentType**

Este atributo indica qual o tipo MIME (Multipurpose Internet Mail Extensions) da resposta está sendo gerada pela JSP. Os tipos mais comuns são: “text/plain”, “text/html”, “text/xml”. Logo abaixo segue o exemplo usado como padrão para as JSPs.

<%@ page contentType=”text/html” %>

**- Atributo Extends**

Serve para indicar a super classe que será usada pelo container JSP no momento de tradução da página em um Servlet Java. Exemplo:

<%@ page extends=”com.taglib.jsp.primeirapagina” %>

**- Atributo Import**

Com o atribuo import, diferente do extends, é capaz de estender um conjunto de classes Java que poderão ser usadas nas páginas JSPs. Esta forma é mais econômica de se digitar código, sendo mais prático. Exemplo:

<%@ page import=”java.util.List” %>

**- Atributo Session**

Session é do tipo boolean, indica se a página está participando do gerenciamento de sessão. Por exemplo, se quisermos dizer que uma página é parte de uma sessão, utiliza-se a seguinte sintaxe:

<%@ page session=”true” %>

**- Atributo Buffer**

Responsável por controlar a saída bufferizada para uma página JSP. Se for ajustado para “none” o conteúdo de uma JSP é passado instantaneamente à resposta HTTP. O tamanho do buffer é descrito em kilobytes. Exemplo:

<%@ page buffer=”12kb” %> ou <%@ page buffer=”none” %>

**- Atributo AutoFlush**

Semelhante ao Buffer, também é responsável por controlar a saída buferizada, mais exatamente o comportamento do container JSP quando já esteja cheio o Buffer de saída. Neste caso é esvaziado automaticamente o Buffer de saída e o conteúdo enviado para o servidor HTTP que transmite para o Browser responsável pela solicitação. Sendo do tipo boolean, sua sintaxe é dada abaixo:

<%@ page autoFlush=”true” %>

**- Atributo isThreadSafe**

Quando uma página JSP é compilada em um Servlet, ela deve ser capaz de atender a múltiplas solicitações. Para isso devemos utilizar o atributo isThreadSafe, caso contrário é necessário defini-lo como “false”. Exemplo:

<%@ page isThreadSafe=”false” %>

**- Atributo errorPage**

ErrorPage indica uma página alternativa que será exibida caso aconteça um erro não previsto durante o processamento de uma página JSP no container. Exemplo:

<%@ page errorPage=”/trabalho/paginas/erro.jsp” %>

**- Atributo** **isErrorPage**

Responsável por define uma página JSP que servirá como a página de erro padrão para um grupo de páginas JSP. Sendo do tipo boolean, sua sintaxe é descrita abaixo:

<%@ page isErrorPage=”true”%>

CATALINA\_HOME. This represents the root of your Tomcat installation.  
CATALINA\_HOME/conf - Configuration files and related DTDs. The most important file in here is \*server.xml\*. It is the main configuration file for the container.

"O arquivo faces-config.xml é o arquivo principal de configuração de uma aplicação JSF. Localizado no diretório WEB-INF da aplicação e no formato XML, este arquivo é responsável por descrever os elementos e sub-elementos que compõem o projeto, tais como as regras de navegação, os beans gerenciados, configurações de localização, entre outros. " [1].

Em uma página JSP vários objetos estão instanciados e prontos para usar.  
  
Estes objetos são chamados objetos implícitos, pois não é necessário declará-los explicitamente.  
  
Os **nove**objetos implícitos são:

* **request**: representa um objeto do tipo HttpServletRequest e contém a informação do pedido HTTP;
* **response**: representa um objeto do tipo HttpServletResponse e contém a resposta HTTP que vai ser enviada ao cliente. Não é usado com frequência;
* **pageContext**: representa um objeto do tipo PageContext e contém o contexto da página;
* **application**: representa um objeto do tipo ServletContext obtido do objeto de configuração da servlet;
* **out**: representa um objeto da classe JspWriter que escreve no corpo de dados da resposta HTTP;
* **config**: representa um objeto do tipo ServletConfig da página JSP;
* **page**: sinônimo do operador “this” do objeto HttpJspPage. Não é usado com freqüencia;
* **session**: representa um objeto do tipo HttpSession e guarda informações da sessão do usuário;
* **exception**: é o objeto Throwable que é resultante de uma situação de erro numa página JSP.

**Há quatro componentes chaves para JSPs:**

* **Diretivas** - são mensagens para o contêiner de JSP que permitem ao programador especificar configurações de página e bibliotecas de tags personalizadas para utilização em uma JSP.
* **Ações** - encapsulam funcionalidades em tags predefinidas que os programadores podem incorporar em uma JSP.
* **Elementos de script**- permitem aos programadores inserir código Java que interaja com compenentes em um JSP para realizar o processamento de solicitação.
* **Bibliotecas de tags**- permitem que os programadores criem tags personalizadas.
* **Elementos (tags) de diretiva**: Páginas (page) e Inclusão (include)
* **Elementos (tags) de scripts**: Declarações, Scriptlets, Expressões, Comentários
* **Elementos (tags) de ações**: jsp:include, jsp:forward, jsp:useBean, jsp:setProperty, jsp:getProperty, etc.

Objetos Implícitos são criados automaticamente pelo contêiner JSP e depois disponibilizados. São 9 no total:

Request, response, pageContext, application, out, config, page, session, exception.

Os escopos são: application, page, request e session. Não existe o escopo response.

***1) The include directive:***

Static: adds the content from the value of the file attribute to the current page at translation time. The directive was originally intended for static layout templates, like HTML headers.

***2) The standard action***

Dynamic: adds the content from the value of the page attribute to the current page at request time. Was intended more for dynamic content coming from JSPs.

**3) The JSTL tag:**

Dynamic: adds the content from the value of the URL attribute to the current page, at request time. It works a lot like  , but it’s more powerful and flexible: unlike the other two includes, the url can be from outside the web Container!

J2EE define 4 camadas básicas no modelo de aplicação. São as seguintes:

1. **Camada Cliente** – Parte do software que roda no computador do usuário;
2. **Camada Web**   - Parte do software que roda no servidor web, que por sua vez fica normalmente dentro do servidor de aplicações;
3. **Camada de negócios** – Parte do software que roda no servidor de aplicações;
4. **Camada de dados** – Banco de dados e sistemas externos;

**Container JavaEE**

Um contêiner Java EE contém três componentes essenciais: um contêiner web (ou seja, servlet), um contêiner EJB e um provedor de persistência.

* **Provedor de persistência** - a abordagem tradicional consistia no JDBC para comunicação, através interfaces implementadas por seus drivers específicos para cada banco. JPA é semelhante, pois também é um conjunto de classes e interfaces. Este provedor JPA interage diretamente com o JDBC, estabelecendo comunicação com o banco de dados. A vantagem é que o SQL não fica na aplicação Java
* **EJB Container** - responsável pelo **Enterprise Java Beans**. A maioria dos serviços de requisitos não funcionais fica neste container. O acesso aos dados é através dele, visto que JTA não está no Web container. Os componentes Web interagem com os componentes EJB para a lógica de negócios e o acesso aos dados.
* **Web Container -** Java Server Faces, JSP e Servlet. Estas tecnologias disponibilizam a aplicação através de navegador. Sem um webcontainer não é possível publicar páginas html, xhtml e outras, o que impede a criação de aplicações web.

**JMS – Java Message Service**

JMS (Java MessageService) permite que componentes baseados em JavaEE criem, enviem, recebam e leiam mensagens. Viabiliza comunicação **assíncrona**, segura e fracamente acoplada entre componentes de uma aplicação distribuída.

* **O Modelo Point-to-Point -** O modelo Point-to-Point (PTP) de troca de mensagens é baseado no conceito de filas, no qual cada mensagem é enviada por um produtor a uma fila específica, onde ela fica até que seja posteriormente entregue a um consumidor ou até que expire. Este modelo garante que uma mensagem seja entregue a um único destinatário (relação “um para um” entre produtor e consumidor). Portanto, mesmo que a fila tenha mais de um consumidor ativo, apenas um receberá cada mensagem. Neste modelo a mensagem é entregue ao consumidor mesmo que este não esteja ativo no momento do envio da mensagem. É esta característica que garante o comportamento assíncrono em uma troca de mensagens.
* **O Modelo Publish/Subscribe -** O modelo Publish/Subscribe (Pub/Sub) de troca de mensagens é baseado no conceito de tópicos, no qual as mensagens são publicadas pelo produtor em um tópico e são entregues automaticamente a todos os consumidores que assinaram o tópico para recebimento de mensagens. Portanto, ao contrário do modelo PTP, este modelo permite que uma mesma mensagem seja entregue a vários consumidores (relação “um para muitos” entre produtor e consumidor). A garantia de entrega das mensagens aos consumidores dependerá do tipo de assinatura que fazem ao tópico, que pode ser durável ou não. Consumidores com uma assinatura durável têm a garantia de receber até mesmo as mensagens enviadas ao tópico durante um período em que estiveram inativos. Já consumidores com assinaturas não duráveis somente recebem as mensagens enviadas ao tópico enquanto estiverem ativos.

A JMS (Java Message Service) permite a comunicação assíncrona entre aplicações, utilizando dois modelos básicos de conectividade:

* Filas ponto-a-ponto (queues), onde mensagens submetidas por uma aplicação “produtora” são entregues a uma única aplicação “consumidora”. Pode haver vários consumidores conectados à mesma fila, neste caso somente um deles receberá cada mensagem;
* Canais publish/subscribe (topics), onde cada mensagem pode ser recebida simultaneamente por diversas aplicações consumidoras.

**Modelo ponto a ponto -** No modelo ponto a ponto, ou por filas, um "produtor" (*producer*) envia mensagens para uma fila e um "consumidor" (*consumer*) as lê. Neste caso, o produtor conhece o destino da mensagem e a envia diretamente para a fila do consumidor. Este modelo é caracterizado pelo seguinte:

* apenas um consumidor irá ler a mensagem;
* não é necessário que o produtor esteja em execução no momento em que o consumidor lê a mensagem, assim como não é necessário que o consumidor esteja em execução no momento que o produtor envia a mensagem;

Quando lê uma mensagem com sucesso o consumidor envia um aviso (*acknowledged*) para o produtor.

**Modelo publish/subscribe -** Este modelo suporta a publicação de mensagens para um determinado tópico de mensagens (*message topic*). O(s) "assinante(s)" (*subscriber*) podem registrar interesse em receber ("em assinar") mensagens de um tópico. Neste modelo, nem o "publicador" (*publisher*) ou o "assinante" sabem um do outro. As características deste modelo são:

* múltiplos consumidores podem ler a mensagem;
* existe uma dependência temporal entre os publicadores e assinantes de um tópico. Um publicador deve criar uma "assinatura" (*subscription*) para que os assinantes possam receber mensagens. O assinante do tópico deve estar em execução continuamente para receber as mensagens.

***EJB***

***Existem 3 tipos de enterprise beans:***

* **Session Beans:** executa uma tarefa para um cliente
  + Stateless: Sem informação de estado
  + Stateful: Com informação de estado
  + Singleton: Sem informação de estado
* **Message Driven Beans (MDB):** Atua como um listener para a API Java Message Service (JMS), processando mensagens assincronamente.
* **Entity Beans:** Representa um objeto de entidade de negócios que existe no armazenamento persistente.

**JNDI**

A arquitetura JNDI consiste em uma *application programming interface* (API) e uma *service provider interface*(SPI), em que as aplicações Java utilizam a JNDI SPI para acessar os serviços de diretórios, tais como LDAP e **DNS** que são, nessa arquitetura, diretamente ligados a JNDI API. "

**Conceito:**

A API disponibiliza:

* Um mecanismo para ligar um objeto a um nome;
* Uma interface padronizada de busca de objetos no serviço de diretório;
* Uma interface de eventos que permite que um usuário saiba quando uma entrada (nome + objeto) foi modificada;
* Extensões que suportam as capacidades do padrão LDAP.

A SPI permite que a JNDI suporte praticamente qualquer tipo de serviço de diretório incluindo:

* LDAP;
* DNS;
* NIS;
* RMI;
* CORBA serviço de nomes;
* Sistema de arquivos.

***Java Naming and Directory Interface* (JNDI)** é uma interface de programação de aplicativos composta pelos pacotes: javax.naming, javax.naming.directory, javax.naming.event, javax.naming.ldap e javax.naming.spi. O método da classe javax.naming.Context utilizado para recuperar um objeto vinculado a um nome é método **lookup**.