Lição 3

Classes Servlets Avançadas

Objetivos

Ao final desta lição, o estudante será capaz de:

- Redirecionar respostas
- Identificar o escopo dos objetos
- Utilizar sessões e rastreamento de sessão
- Utilizar Filtros

Redirecionamento de Resposta

- Há casos em que o *Servlet* deve realizar algum processamento inicial
- Deixar a geração do conteúdo propriamente dito para alguma outra entidade
- É melhor que o servlet redirecionasse a geração de saída
- Dois modos de se fazer o redirecionamento:
  - Através da utilização de um objeto *RequestDispatcher*
  - Utilização do método *sendRedirect*() encontrado na classe *HttpServletResponse*

RequestDispatcher

- Obter uma instância da classe *RequestDispatcher* e invocar o método:

```
public RequestDispatcher getRequestDispatcher(String
                                              path)
```

- Pode ser encontrado no objeto *HttpServletRequest*
- O argumento String que ele requer é a localização de HTML, JSP ou servlet para o qual queremos processar a requisição
- Uma vez que tenhamos uma instância da classe
- Através do *RequestDispatcher*, podemos invocar um dos seguintes métodos:

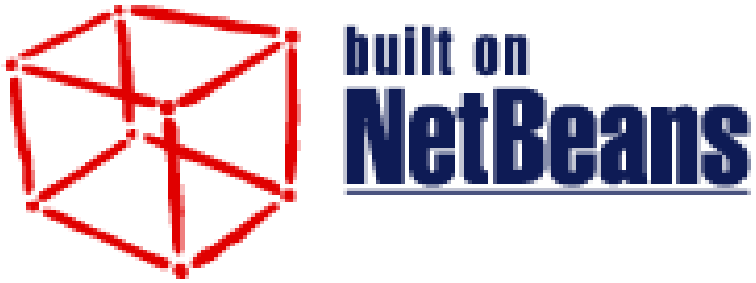
```
public void include(ServletRequest req,
                    ServletResponse res)

public void forward(ServletRequest req,
                    ServletResponse res)
```

include e *forward*

- Recebem o conteúdo produzido pela localização especificada e fazem dele uma parte da resposta do servlet ao usuário
- Diferenças:
  - ***forward*** faz com que o alvo seja a entidade com responsabilidade exclusiva de gerar a resposta
  - ***include*** somente incorpora o conteúdo do alvo
    - Ao usar *include,* poderíamos acrescentar outros conteúdos à resposta, possivelmente até mesmo a inclusão de outro alvo

include e *forward*

- Passaremos agora para o NetBeans



include e *forward*

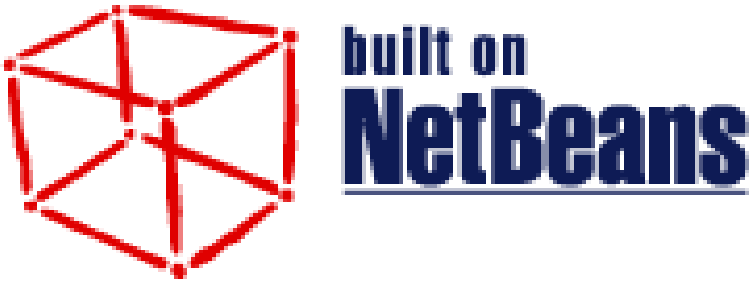
- Dados os códigos virtualmente idênticos, temos duas saídas diferentes:
  - Método *include*, a mensagem String que externamos para o usuário antes de chamar o método é mostrada
  - Método *forward*, a mensagem que adicionamos dentro da resposta antes da chamada do método não faz parte da saída
  - Método *forward*, todo conteúdo do buffer de resposta é limpo antes da chamada ao método, depois do que a resposta é imediatamente consignada; nenhum novo conteúdo pode ser adicionado
  - Método *include*, todo conteúdo colocado no buffer de resposta é mantido antes e depois da chamada ao método

include

- É possível ao servlet apresentar como um todo as saídas de diversas outras fontes
- Permite agregar mensagens a conteúdos que de outra forma seriam estáticos
- Uma vez que o método *include* somente adiciona conteúdo de outra fonte e não se consigna um retorno após sua chamada, podemos usá-lo repetidamente
- *LoginServlet* que se segue é capaz de criar uma resposta contendo três páginas diferentes

include

- Passaremos agora para o NetBeans



sendRedirect

- A outra maneira de redirecionar a saída para uma entidade fora do servlet é o método sendRedirect:

  public void sendRedirect(String relativePath)

  - Pode ser encontrado na classe *HttpServletResponse*
  - O argumento String que ele requer representa o caminho até o alvo para o qual desejamos redirecionar o usuário
  - A chamada a este método instrui efetivamente o navegador a mandar uma outra requisição HTTP, desta vez para o alvo especificado

sendRedirect

- O alvo é a única entidade responsável pela geração do conteúdo
- A requisição reenviada apresenta várias diferenças práticas se comparada com o método *forward*:
  - A URL na barra de endereços do navegador reflete o alvo especificado
  - Dados armazenados no objeto de requisição anterior são descartados
  - Pode ser encontrado na classe *HttpServletResponse*

Redirecionamento de Resposta: Resumo

- Recomenda-se então que:
  - O método *include* seja usado para permitir múltiplas fontes de saída
  - O método *forward* seja usado se redirecionado para um componente que gere conteúdo dinâmico
  - O método *sendRedirect* seja usado quando redirecionando para conteúdo estático

Classes de Escopo

- A especificação do servlet nos possibilita quatro escopos onde podemos armazenar dados, de modo a compartilhá-los entre os nossos componentes
- São eles, em ordem crescente de escopo:
  - Página
  - Requisição
  - Sessão
  - Aplicação

Armazenando e Recuperando Dados de Escopo

- Para armazenar dados dentro de uma classe de escopo:

  public void setAttribute(String key, Object value);

- Para recuperar os dados depois, passe a mesma chave como argumento:

  public Object getAttribute(String key);

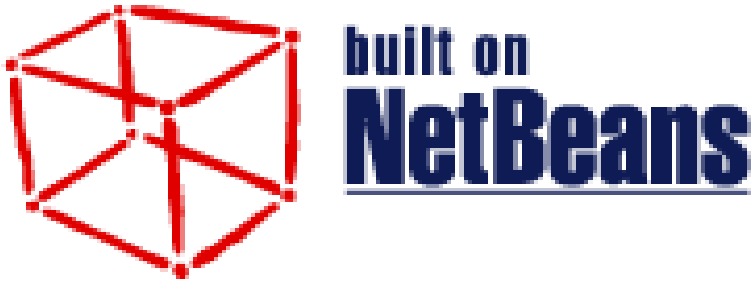
- Para remover um atributo do escopo de um objeto chamar o método removeAttribute() e passar a chave do atributo como seu argumento

Armazenando e Recuperando Dados de Escopo

- Cenário de Exemplo:
  - A aplicação será capaz de mostrar os detalhes sobre uma pessoa dada sua identificação pessoal (*personalID*)
  - Esta identificação pessoal será fornecida pelo usuário
  - Separamos o componente que recuperará os detalhes pessoais do componente que mostrará a informação ao usuário
  - Se comunicam então utilizando as facilidades de armazenamento e recuperação de dados disponível no escopo da requisição

Armazenando e Recuperando Dados de Escopo

- Passaremos agora para o NetBeans



Acompanhamento e Gerenciamento da Sessão

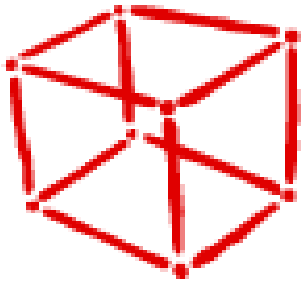
- Lembre-se que o HTTP foi projetado como um protocolo conectado e sem estados
- Um problema se apresenta para aplicações WEB: como manter o estado para a transação WEB de um usuário em particular?
- Uma solução para este problema é fazer com que o servidor mantenha o conceito de “sessão de usuário”
  - Enquanto dentro de uma sessão, o servidor poderá reconhecer o cliente através de múltiplas requisições
  - Há 3 soluções típicas para o problema:
    - Cookies
    - Reescrita de URL
    - Campos ocultos de formulário

Métodos Tradicionais para Manter o Estado da Sessão

- Cookies
  - Pequenas estruturas de dados utilizadas por um servidor WEB para fornecer dados a um navegador cliente
  - Usando cookies, *servlets* podem armazenar “IDs de sessões” que são usados para identificar o usuário como participante de uma sessão em particular

Métodos Tradicionais para Manter o Estado da Sessão

- Passaremos agora para o NetBeans



built on NetBeans

Métodos Tradicionais para Manter o Estado da Sessão

- Reescrita de URL
  - O navegador cliente acrescenta um id de sessão único ao final de cada requisição que faz ao servidor.
  - Esta é uma outra boa solução, e uma que funciona mesmo que o usuário tenha desabilitado a utilização de *cookies*

Métodos Tradicionais para Manter o Estado da Sessão

- Campos ocultos de formulário
  - Um campo oculto de formulário é introduzido em formulários HTML, com o seu valor sendo definido como o ID de uma sessão em particular
  - Este método é muito limitado devido ao fato de que ele só pode ser utilizado quando houver um formulário na página que o cliente está utilizando

Acompanhamento de Sessão em Servlets

- API HttpSession
  - Uma API de alto nível na especificação de servlets para prover acesso ao acompanhamento de sessões
  - Reconhecimento do ID da sessão, detalhes da manipulação de cookies e detalhes da extração das informações da URL são abstraídos do desenvolvedor
  - Ao desenvolvedor é fornecida uma localização conveniente onde armazenar os dados para uma sessão de usuário
  - A utilização correta permite à sua aplicação mudar automaticamente para o método de reescrita da URL se ele detetar que o suporte a cookies está desabilitado no navegador cliente

Acompanhamento de Sessão em Servlets

- Obtendo uma instância da classe *HttpSession*
  - A classe *HttpSession* representando os dados da sessão associados a uma dada requisição de cliente pode ser obtida chamando o método *getSession*() da classe *HttpServletRequest*
  - Passando um valor lógico no método *getSession*() podemos especificar ao servidor se um novo objeto *HttpSession* deverá ser criado automaticamente caso o usuário não esteja atualmente participando de nenhuma sessão

Acompanhamento de Sessão em Servlets

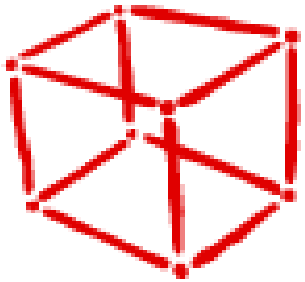
- Armazenando e recuperando dados em uma sessão
  - Com a API *HttpSession*, os desenvolvedores não mais precisam gerenciar explicitamente os objetos para armazenar os dados que precisam ser mantidos dentro de uma sessão de usuário. Tudo que é preciso é chamar um dos seguintes métodos:

public void setAttribute(String key, Object value)

public Object getAttribute(String key)

Acompanhamento de Sessão em Servlets

- Passaremos agora para o NetBeans



built on NetBeans

Acompanhamento de Sessão em Servlets

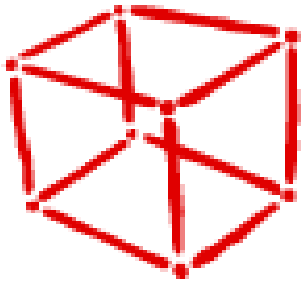
- Removendo os dados armazenados em sessão
  - Para remover os dados colocados no escopo de sessão, chame o método *removeAttribute*(), e passe como argumento a chave String associada com os dados

Acompanhamento de Sessão em Servlets

- Terminando a sessão
  - Término automático por limite de tempo
  - Término programático

Acompanhamento de Sessão em Servlets

- Passaremos agora para o NetBeans



built on NetBeans

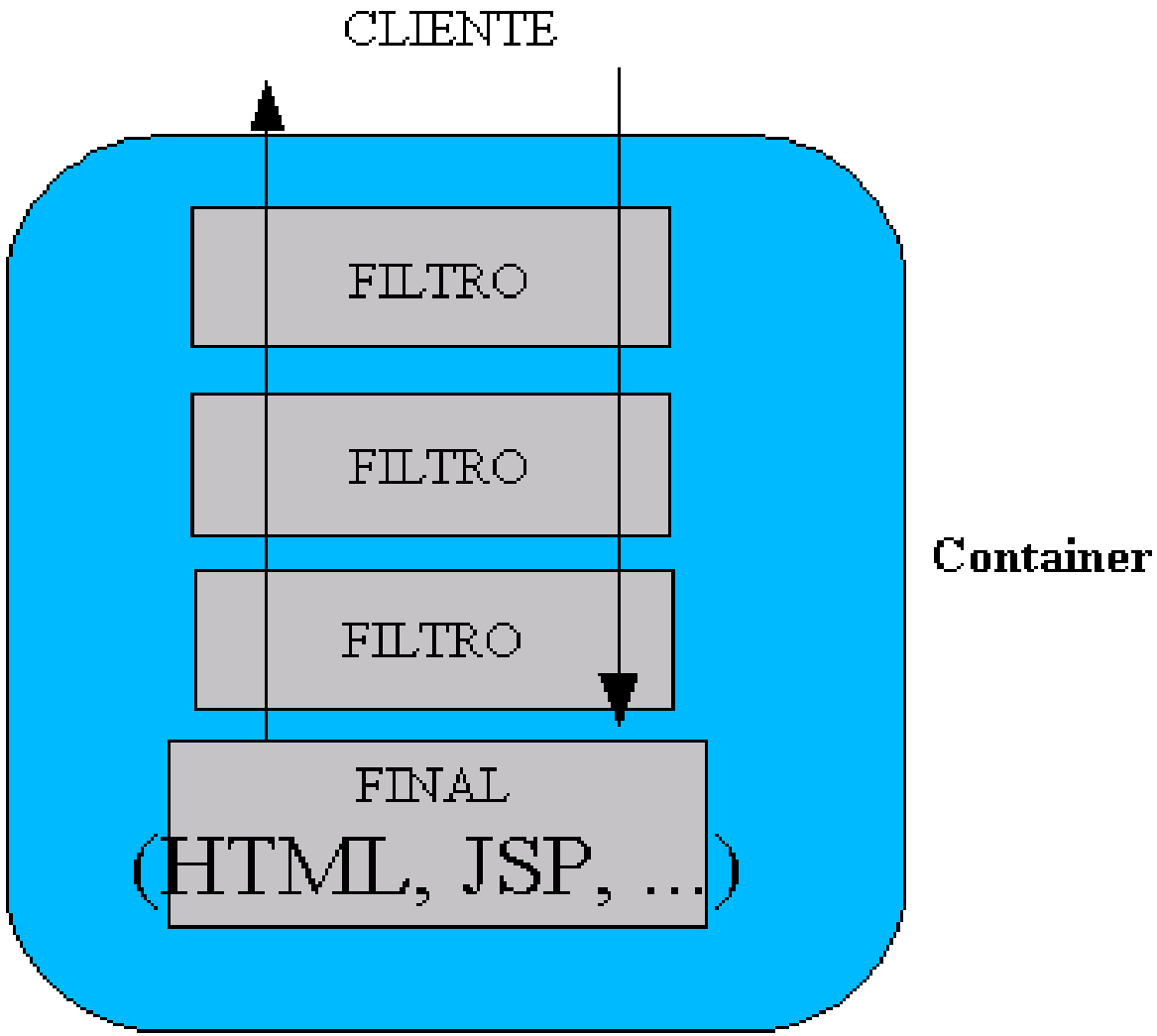
Acompanhamento de Sessão em Servlets

- Fazendo a Reescrita de URL
  - Por padrão, a API *HttpSession* utiliza cookies para acompanhar sessões
  - Podemos adicionar este suporte à reescrita de URL utilizando o método *encodeURL*() encontrado na classe *HttpServletResponse*

Filtros

- Componentes WEB avançados que foram introduzidos desde a Especificação *Servlet* 2.3
- Ficam entre uma requisição de cliente e um recurso em particular
  - Qualquer tentativa de recuperar o recurso alvo tem que passar pelo filtro
  - O recurso pode ser qualquer fonte de conteúdo estática ou dinâmica (HTML, JSP, GIF,...)
- Intercepta requisições do cliente e existe como parte de uma cadeia
  - Há uma sequência definida de filtros através da qual uma requisição passa antes de chegar finalmente no recurso alvo

Filtros



Filtros

- Componentes úteis
- Fornecem uma maneira simples para inserir processamento adicional
  - A funcionalidade é possível utilizando *Servlets* e redirecionamento de resposta
  - Filtros não têm essa limitação

Criando um Filtro

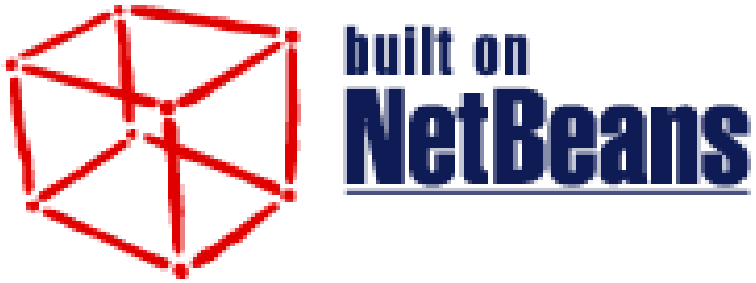
- Para criar um filtro criar uma classe que implemente a interface *javax.servlet.Filter*
- A interface define os seguintes métodos:

```
void init(FilterConfig config) throws
    ServletException void destroy

void doFilter(ServletRequest request,
ServletResponse response, FilterChain chain)
    throws IOException, ServletException
```

- O contêiner cria uma instância da classe Filter
- Utiliza multi-tarefa para tratar múltiplas requisições concorrentes de clientes

Criando um Filtro

- Passaremos agora para o NetBeans



Cadeias de Filtros

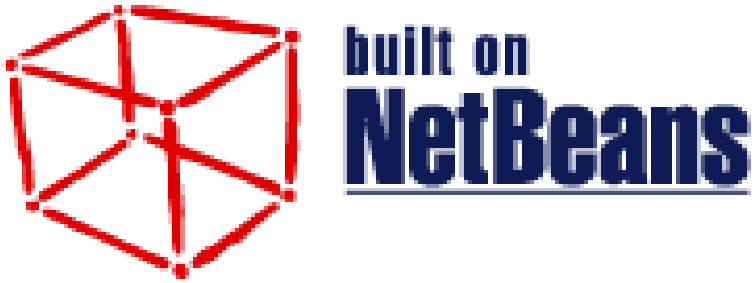
- Permitir que Filtros sejam aplicados em uma certa seqüência
- Representada pela classe *FilterChain*
- Sem esta habilidade, um Filtro é funcionalmente o mesmo que uma *Servlet*
- Permite uma separação entre diferentes camadas de processamento
- A seqüência da cadeia de filtros é determinada pela localização do filtro no descritor de implementação
- Ordem ascendente a ordenação dos elementos <filter-mapping> que representam o mapeamento de cada filtro

Cadeias de Filtros

- Método *doFilter* da classe *FilterChain*
  - O único acesso que um desenvolvedor tem à sequência de filtros
  - Chama o próximo Filter na sequência, ou o recurso alvo se o Filter atual é o último na sequência
  - Requer somente os objetos *ServletRequest* e *ServletResponse* atuais como argumentos
  - Ao chamar este método, os demais filtros na cadeia (assim como o recurso alvo), NÃO serão chamados

Cadeias de Filtros

- Passaremos agora para o NetBeans



Configuração de Filtro

- O arquivo **web.xml** têm um ordenamento particular dos seus elementos
- Garantir que as entradas de elementos de filtro sejam definidas antes de quaisquer *servlets*
- Somente depois de quaisquer entradas no elemento <context-param>
- Por padrão, filtros não são aplicados em componentes WEB que sejam o alvo de chamadas *include* ou *forward* de uma classe *RequestDispatcher*
- Aplicados somente a requisições feitas diretamente pelo cliente
- Este comportamento pode ser mudado pela adição de um ou mais elementos de processamento ao mapeamento dos filtros

Sumário

- Redirecionamento de Resposta
- Classes de Escopo
- Gerenciamento de Sessão
- Filtros

Parceiros

- **Os seguintes parceiros tornaram JEDI™ possível em Língua Portuguesa:**