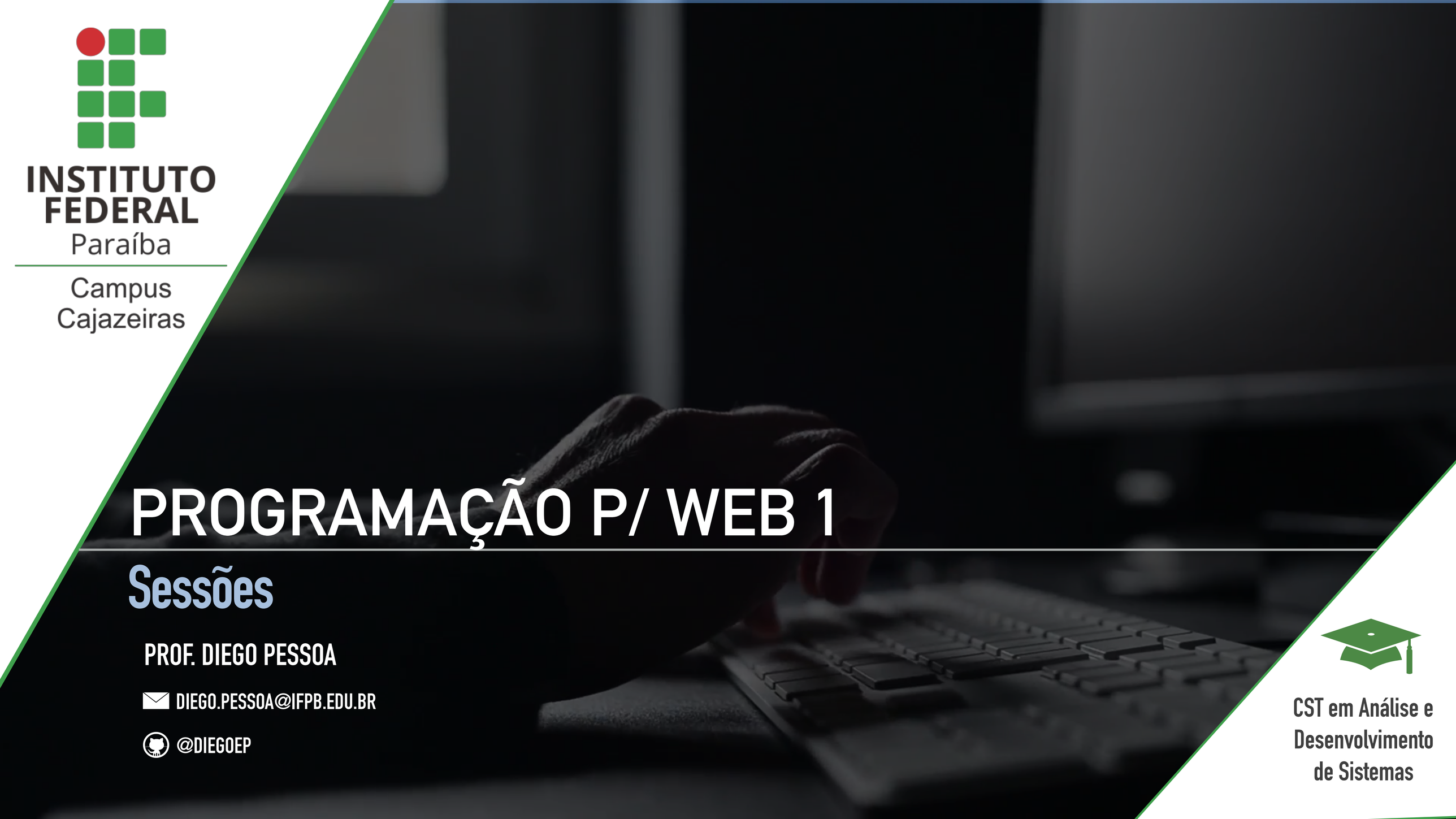


INSTITUTO FEDERAL
Paraíba

Campus Cajazeiras

PROGRAMAÇÃO P/ WEB 1

Sessões

PROF. DIEGO PESSOA

DIEGO.PESSOA@IFPB.EDU.BR

@DIEGOEP

CST em Análise e Desenvolvimento de Sistemas

- Normalmente, ao acessar uma aplicação Web, o cliente permanece conectado a mesma por um bom tempo;

- Durante este tempo, ele pode realizar diversas ações;

Introdução

- Por exemplo, para efetuar uma compra, o usuário passa por várias etapas:
  - Pesquisar os produtos disponíveis;
  - Selecionar os produtos desejados;
  - Informar a forma de pagamento;
  - Informar os dados do cartão de crédito;
  - Informar o endereço de entrega;

Introdução

- Todas estas ações são executadas por meio do protocolo HTTP, que é stateless;
  - Todas as requisições são encerradas após serem processadas;
  - O protocolo não guarda informações a respeito destas requisições, que são tratadas de forma independente;

Introdução

- Para criar aplicações web efetivas, precisamos de uma forma de relacionar as requisições de um mesmo cliente;
- Este relacionamento é feito através de um conceito chamado de sessão;
- Na prática, uma sessão é uma interação entre o cliente e a aplicação;

Introdução

- A sessão pode ser usada na aplicação para muitas coisas, como:
  - Compartilhar dados entre os componentes da aplicação que interagem com o cliente;
  - Obter informações relativas ao cliente;
  - Realizar transações longas, que requerem várias solicitações;

Identificando uma sessão

- Uma aplicação web é normalmente acessada por muitos clientes ao mesmo tempo;
  - Ou seja, ela pode ter várias sessões abertas ao mesmo tempo;
  - Cada sessão é associada a um único cliente;

Identificando uma sessão

- Para que isso seja possível, cada sessão precisa ser identificada unicamente;
  - Pra não correr o risco de se misturar os dados destas sessões;
  - Imagine o que aconteceria se o produto escolhido por um cliente fosse parar no carrinho de compras de outro;

Identificando uma sessão

- Cada sessão é identificada por meio de uma identificação única;
- Esta identificação é gerada pelo servidor web no momento em que a sessão é criada;

Identificando uma sessão

- A identificação gerada pelo servidor é retornada para o cliente por meio de um cookie;
  - Como parte da resposta HTTP;

- Este cookie é enviado pelo cliente nas suas requisições subsequentes;
  - Como parte da requisição HTTP;

Identificando uma sessão

- Reescrita da URL:
  - O identificador da sessão é embutido na URL da requisição;
  - Neste caso, ele é passado como um parâmetro;
  - Exemplo: http://www.myserver.com/catalog/index.html?jsessiononid=1234

Identificando uma sessão

- Sessões SSL:
  - Security Socket Layers implementam criptografia para o protocolo HTTP;
  - Este protocolo já implementa uma infraestrutura que permite agrupar requisições em uma mesma sessão;
  - Containers podem usar esta infraestrutura para gerenciar a sessão do cliente;

Identificando uma sessão

- Quando o cliente se desconecta da aplicação, o container destrói o objeto relativo à sua sessão;

- Quando ele se conecta novamente à aplicação, uma nova sessão é criada, e todo o processo se repete;

Manipulando sessões

- A interface HttpSession:
  - Interface que descreve o comportamento de um objeto que representa uma sessão do cliente com a aplicação;
  - Os objetos que implementam esta interface nos permite gerenciar sessões com o usuário dentro de nossa aplicação;

Manipulando sessões

- A interface HttpSession:
  - Este objeto pode ser obtido por meio do método getSession, disponível no objeto que representa a requisição;
  - Numa aplicação web java com servlets, a sessão é criada no momento em que o método getSession é invocado pela primeira vez;

Manipulando sessões

- A interface HttpSession:
  - Com objetos que implementam esta interface, nós podemos realizar tarefas como:

✓ Recuperar informações relativas à sessão do usuário;

✓ Diferenciar as conexões da aplicação;

✓ Compartilhar informações entre os componentes da aplicação por meio de atributos;

Manipulando sessões

- Alguns métodos definidos na interface HttpSession:
  - getId():
    - ✓ Recupera o valor da identificação única atribuída à sessão;
  - getCreationTime():
    - ✓ Recupera o instante de tempo em que a sessão foi criada;

Manipulando sessões

- Alguns métodos definidos na interface HttpSession:
  - getLastAccessedTime():
    - ✓ Recupera o instante de tempo em que a sessão foi utilizada pela última vez;
    - ✓ Este tempo se refere ao instante em que o cliente enviou a sua última requisição;

Manipulando sessões

- Alguns métodos definidos na interface HttpSession:
  - isNew():
    - ✓ Verifica se a sessão foi recém criada;
    - ✓ Isto acontece quando o cliente só fez o primeiro acesso à aplicação;

- Uma sessão pode ser encerrada de duas formas:
  - Após um timeout;
  - Imediatamente;

Encerrando uma sessão

- Encerrando uma sessão após um timeout:
  - É a forma mais simples de se encerrar uma sessão;
  - Nesta solução, configuramos o tempo que a sessão pode ficar inativa;
    - ✓ Sempre que o cliente usa a sessão (mandando uma nova requisição), este tempo é reiniciado;

Encerrando uma sessão

- Encerrando uma sessão após um timeout:
  - Caso o tempo máximo de inatividade permitido seja extrapolado, a sessão é encerrada;
  - O container é quem se encarrega de verificar o tempo das sessões;
    - ✓ E de encerrá-las, quando for o caso;

Encerrando uma sessão

- Encerrando uma sessão após um timeout:
  - Podemos configurar o tempo máximo de inatividade de duas formas;
    - ✓ Configurando o descritor da aplicação;
    - ✓ Usando o objeto que representa a sessão;

Encerrando uma sessão

- Encerrando uma sessão após um timeout:
  - Podemos configurar o timeout no descritor da aplicação por meio do elemento session-config;
  - A configuração definida neste elemento é aplicada a todos os componentes da aplicação;

Encerrando uma sessão

- Encerrando uma sessão após um timeout:
  - A configuração do timeout é feita por meio do sub-elemento session-timeout;
  - Importante: no descritor, este tempo é configurado em minutos;

Encerrando uma sessão

- Encerrando uma sessão após um timeout:
  - Configurando o tempo máximo de inatividade para 20 minutos:

```
<session-config>
        <session-timeout> 20</session-timeout>
</session-config>
```

Encerrando uma sessão

- Encerrando uma sessão após um timeout:
  - Podemos também configurar o timeout dinamicamente no objeto da sessão, por meio do método setMaxInactiveInterval;
  - No entanto, neste método, o tempo máximo de inatividade é configurado em segundos;

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA PARAÍBA

Exemplo de Servlet Que Redefine o Tempo Máximo deInatividade da Sessão

```java
@WebServlet("/sessionTimeout")
public class SessionTimeoutServlet extends HttpServlet{

    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws IOException{
        doPost(request, response);
    }

    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws IOException{
        /*Configurando o tempo máximo de inatividade da sessão para 20 minutos */

        HttpSession session = request.getSession();
        session.setMaxInactiveInterval(20 * 60);
        /*
            Demais ações do servlet
        */

    }

}
```

Encerrando uma sessão

- Encerrando uma sessão após um timeout:
  - Ainda no objeto que implementa a interface HttpSession, o método getMaxInactiveInterval pode ser usado para se obter informações sobre o timeout;

Encerrando uma sessão

- Encerrando uma sessão imediatamente:
  - Podemos encerrar uma sessão imediatamente através do método invalidate da interface HttpSession;
  - Quando isto é feito, a sessão é encerrada;
    - ✓ Juntamente com todos os seus atributos e todos os componentes que a referenciam;

Exemplo de Servlet Que Encerra a Sessão Imediatamente

```
@WebServlet("/finishSession")
public class FinishSessionServlet extends HttpServlet{

    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws IOException{
        doPost(request, response);
    }

    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws IOException{
        /*Encerrando a sessão imediatamente */

        HttpSession session = request.getSession();
        session.invalidate();
        /*
            Demais ações do servlet
        */

    }

}
```

Reescrevendo URLs

- Quando o browser do cliente está habilitado para não aceitar cookies, o container usa a reescrita de URLs para realizar o controle de sessão;

- Para isso, precisamos reescrever todas as URLs da nossa aplicação;

Reescrevendo URLs

- Para habilitar a reescrita de URL, usamos o método encodeURL do objeto HttpResponse;
  - Passando como parâmetro a URL que deve ser codificada;

- Para manter o controle de sessão, todas as URLs da aplicação precisam ser codificadas;

Reescrevendo URLs

- O método encodeURL pode ter dois resultados:
  - A mesma URL, caso o cliente aceite cookies ou o controle de sessão não esteja ativado;
  - A URL codificada, caso contrário;

Exemplo de Servlet Que Realiza a Reescrita da URL

```
@WebServlet("/reescreveUrl")
public class ReescreveUrlServlet extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws IOException{
           doPost(request, response);
    }

    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws IOException{
        request.getSession(true);
        String encodedUrl = response.encodeURL("sessionServlet");
        PrintWriter out = response.getWriter();
        out.println("<html><head><title> Trabalhando com Reescrita de Urls </title></head>");
        out.println("<body>");
        out.println("<a href=\"" + encodedUrl + "\"> Exemplo de reescrita da URL </a>");
        out.println("</body></html>");
    }

}
```

Reescrevendo URLs

- O método encodeRedirectURL pode ser usado para aplicar a codificação da URL antes de um redirecionamento;

```
String originalURL = "someServlet";
String encodedURL = response.encodeRedirectURL(originalURL);
response.sendRedirect(encodedURL);
```

Trabalhando com atributos

- Podemos usar o objeto HttpSession para associar objetos a uma sessão;
- Este objetos se tornam acessíveis para qualquer requisição referente à sessão;
- A associação destes objetos é feita na forma de atributos;

Trabalhando com atributos

- Podemos manipular atributos de sessões através dos métodos:
  - getAttributeNames();
    - ✓ Recupera uma enumeração com os nomes de todos os atributos associados à sessão;
  - getAttribute(String atributeName);
    - ✓ Recupera o valor de um atributo específico;

Trabalhando com atributos

- Podemos manipular atributos de sessões através dos métodos:
  - setAttribute(String attributeName, Object value);
    - ✓ Cria um novo atributo ou atualiza o valor de um atributo já existente;
  - removeAttribute(String attributeName);
    - ✓ Remove um atributo da sessão;

Trecho de um Servlet Que Implementaum Contador de Requisições do Cliente – Manipulando Atributos

```
HttpSession session = request.getSession();
Integer count = null;
String message = null;
Object countAtr = session.getAttribute("count");
if(countAtr==null){
    count = new Integer(1);
    session.setAttribute("count", count);
    message = "Seja bem vindo!";
}
else{
        count = (Integer) countAtr + 1;
        session.setAttribute("count", count);
        message = "Obrigado por ter voltado!";
    }
```

Trabalhando com atributos

Trecho de um servlet que implementa um contador de requisições do cliente – Imprimindo os dados da sessão

```
out.println("<html><head><title> Imprimindo dados da sessÃ£o </title></head>");
out.println("<body><p>"+message+"</p>");
out.println("<p> <b>Id:</b> "+session.getId()+" </p>");
out.println("<p> <b>Criação da sessão:</b> "+new Date(session.getCreationTime())+" </p>");
out.println("<p> <b>Última requisição:</b> "+new Date(session.getLastAccessedTime())+" </p>");
out.println("<p> <b>Número de requisições:</b> "+count+" </p>");
out.println("</body>");
out.println("</html>");
```

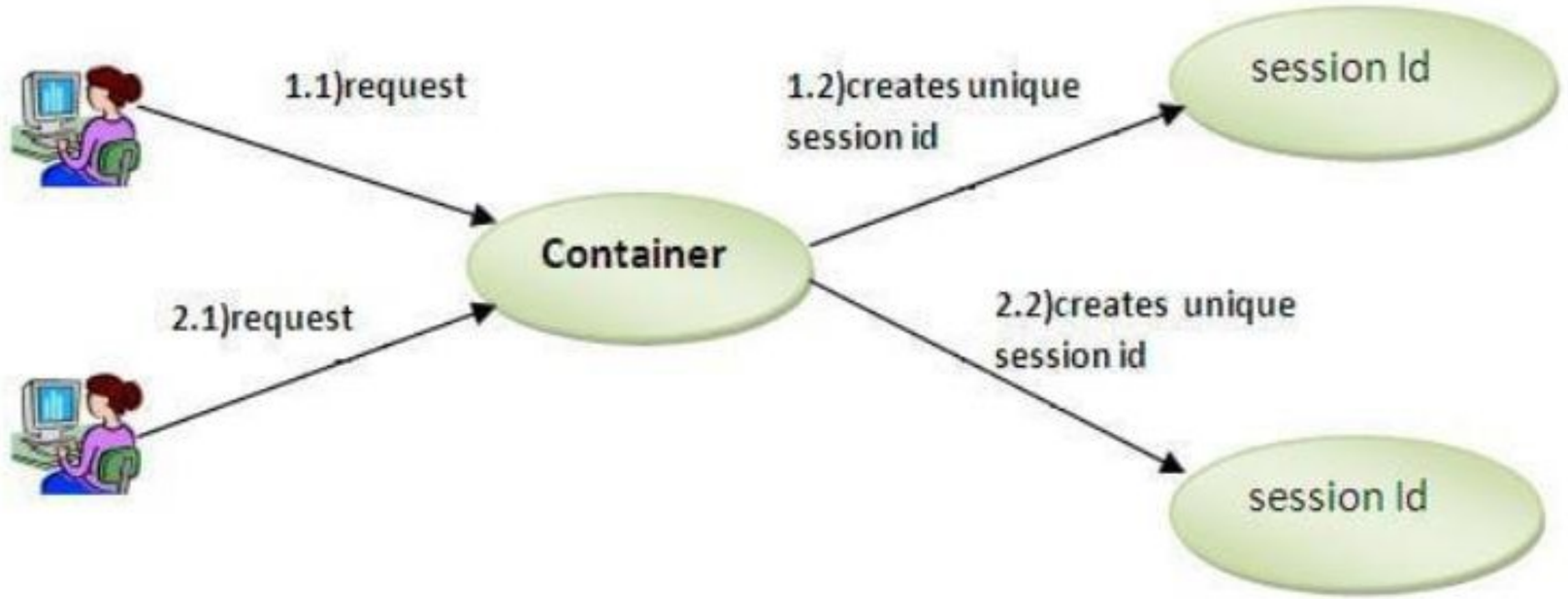
Resumindo a Interface HttpSession

- O contêiner cria um ID de sessão para cada usuário.
  - Usa esse ID para identificar o usuário específico.

- Usado para executar duas tarefas:
  - ligar objetos

  - Visualizar e manipular informações sobre uma sessão:
    - O identificador da sessão, a hora da criação e a hora do último acesso.

Resumindo a Interface HttpSession



Resumindo a Interface HttpSession

- Como obter o objeto HttpSession?
  - public HttpSession getSession ():
    - Retorna a sessão atual associada a essa solicitação
      - se a solicitação não tiver uma sessão, cria uma.
  - public HttpSession getSession (boolean create):
    - Retorna a atual HttpSession associada a essa solicitação,
      - se não houver sessão atual e a criação for verdadeira, retorna uma nova sessão.

Resumindo a Interface HttpSession

- Métodos comumente usados da interface HttpSession
  - **public String getId ():**
    - Retorna uma string contendo o valor do identificador exclusivo.
  - **public long getCreationTime ():**
    - Retorna a hora em que esta sessão foi criada, medida em milissegundos.
  - **public long getLastAccessedTime ():**
    - retorna a última vez que o cliente enviou uma solicitação associada a essa sessão, como o número de milissegundos
  - **public void invalidate ():**
    - Invalida esta sessão e depois desassocia qualquer objeto associado a ela.

Exemplo Servlet HttpSession Login e Logout

- Vamos criar um aplicativo de login e logout do mundo real sem usar o código do banco de dados.
  - Estamos assumindo que o login é “admin" e senha é “secreta". Esses dados são lidos a partir de parâmetros de inicialização.
  - estamos criando 3 links: login, logout e profile.
- Arquivos que devem ser criados:

1. login.html
2. InicioServlet.java
3. LogoutServlet.java
4. LoginServlet.java
5. web.xml

Trabalhando com Listeners

- Podemos implementar listeners oferecidos pela API de servlets para monitorar eventos ocorridos em sessões;

- O container é responsável por instanciar estas classes e notificá-las sobre a ocorrência dos eventos de seu interesse;

Trabalhando com Listeners

- Exemplos de listeners de são oferecidos pela API de servlets:
  - HttpSessionListener;
  - HttpSessionAttributeListener;
  - HttpSessionActivationListener;
  - HttpSessionBindingListener;

Trabalhando com Listeners

- HttpSessionListener:
  - Permite monitorar a criação e a finalização de sessões na aplicação;
  - A sua interface tem os seguintes métodos:
    - ✓ sessionCreated(HttpSessionEvent event);
    - ✓ sessionDestroyed(HttpSessionEvent event);

Trabalhando com Listeners

- HttpSessionAttributeListener:
  - Usado para monitorar o estado dos atributos da sessão;
  - Permite programar ações para quando um atributo de sessão é criado, excluído ou alterado;

Trabalhando com Listeners

- HttpSessionAttributeListener:
  - A sua interface tem os seguintes métodos:
    - ✓ attributeAdded (HttpSessionBindingEvent event);
    - ✓ attributeRemoved (HttpSessionBindingEvent event);
    - ✓ attributeReplaced (HttpSessionBindingEvent event);

Trabalhando com Listeners

- HttpSessionActivationListener:
  - Interface implementada por objetos que desejam saber quando a sessão será migrada para outro container;
  - Recurso usado em aplicações web distribuídas, nas quais a aplicação é distribuída em vários containers;
    - ✓ Por questões de conveniência ou de desempenho;

Trabalhando com Listeners

- HttpSessionActivationListener:
  - A sessão pode ser invalidada no container atual e depois ser ativada em outro container no qual ela também é distribuída;

Trabalhando com Listeners

- HttpSessionActivationListener:
  - Os métodos desta interface são:
    - ✓ sessionDidActivate(HttpSessionEvent event);
      - – Inidica que uma sessão foi ativada em um dos containers;
    - ✓ sessionWillPassivate(HttpSessionEvent event);
      - – Indica que uma sessão foi desativada em algum dos containers;

Trabalhando com Listeners

- HttpSessionBindingListener:
  - Interface que permite que objetos sejam notificados sobre a sua associação a uma sessão;
  - Esta associação acontece através dos atributos da sessão;
  - O evento usado para notificação permite que um objeto obtenha uma referência para a sessão ao qual foi relacionado;

Trabalhando com Listeners

- HttpSessionBindingListener:
  - O objeto é associado a uma sessão quando ele é incluído como um atributo da mesma;
  - O objeto é desassociado da sessão quando o mesmo é removido ou quando a sessão é finalizada;

Trabalhando com Listeners

- HttpSessionBindingListener:
  - Os métodos desta interface são:
    - ✓ valueBound(HttpSessionBindingEvent event);
      - Inidica que o objeto foi associado a uma sessão;
    - ✓ valueUnbound(HttpSessionBindingEvent event);
      - Indica que o objeto foi removido de uma sessão;

Trabalhando com Listeners

- Vamos ver a implementação de um listener de sessão que controla na quantidade de sessões ativas a aplicação;

- O número de sessões abertas é armazenado como um atributo no contexto da aplicação;

Listener Que Controla a Quantidade de Sessões Abertasna Aplicação

```
@WebListener
public class SessionCounterListener implements HttpSessionListener{

    @Override
    public void sessionCreated(HttpSessionEvent event) {
        ServletContext context = event.getSession().getServletContext();
        Integer activeSessions = Integer.parseInt(context.getAttribute("activeSessions").toString());
        activeSessions++;
        context.setAttribute("activeSessions", activeSessions);
    }

    @Override
    public void sessionDestroyed(HttpSessionEvent event) {
        ServletContext context = event.getSession().getServletContext();
        Integer activeSessions = Integer.parseInt(context.getAttribute("activeSessions").toString());
        activeSessions--;
        context.setAttribute("activeSessions", activeSessions);
    }

}
```